UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Master's Degree in Computer Science

Academic year 2024/2025

INTERNET OF THINGS AND SMART CITIES

Prof. Marco Giordani

Written by Michael Amista'

A course offered by the School of Engineering (DEI) –
Master's Degree in ICT for Internet and Multimedia

Table of contents

Internet of Things and Smart Cities course introduction

From a customer perspective IoT is the ability to use sensors and other devices to connect things in a way they can work together. IoT is employed in different domains, such as smart home, smart cities and industrial systems.

Obviously, the main issue when we talk about IoT is the **reliability** of the collected data and this is a big challenge in protecting our privacy, we will deal with that during the course. Another important problem is the **energy consumption**: many sensors and devices have a limited amount of battery and so battery optimization considerations are needed. **Interoperability** is another problem: there are many manufacturers and many different devices, so it is important to have a common standard to make different devices work together. Furthermore, depending on the number of devices, larger networks are created, and it is crucial that these networks are well handled in a way to be able to manage the large number of nodes.

Course schedule:

- Frontal lectures.
- LAB experiences: they start in November and there will be 3(+1) practical lab lectures. We will use Arduino as an emulated sensor to send data to IoT LoRa gateway and we will see how to process those data to return an output.
- Guest lectures (even those part of the written exam).

Slides (and attendance) are enough to pass the exam.

The exam consists of two parts:

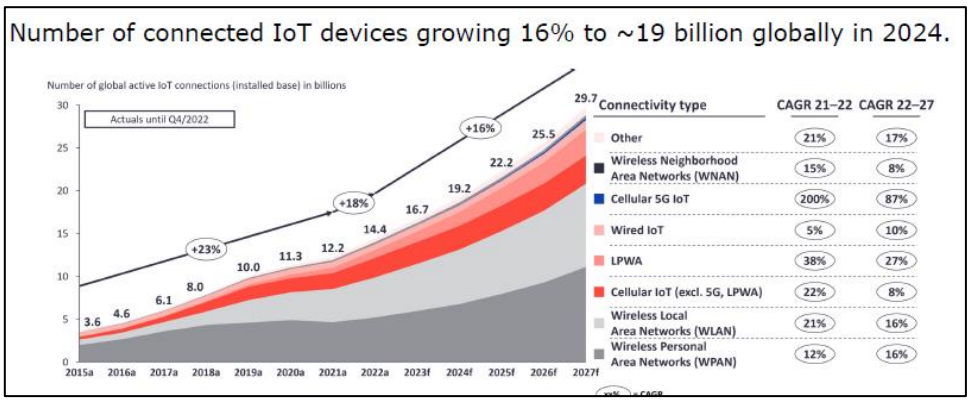
- **PART 1 (up to 28/33 points): Written test (mandatory)** that consists of both multiple-choice questions and open ones with a duration of 1 hour and half (even the guest lectures can be part of the written exam).
- **PART 2 (up to 5 extra points): LAB assessment (optional)** which is “only” a replication of what done during lab lectures so if you follow the lab you do not need to study anything else to work on the assessment. The assessment can be submitted in December, before the written exam session.

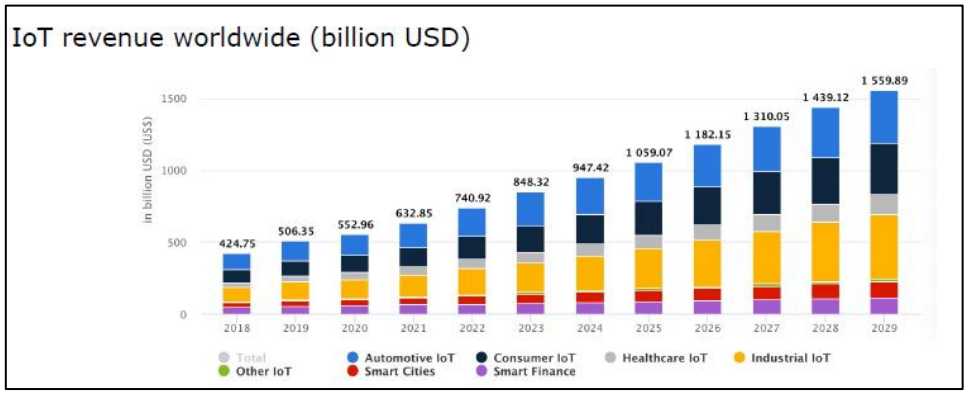
1. Internet of Things (IoT)

The term “IoT” is characterized by two fundamental concepts: **internet** and **things**.

The precursors of the modern Internet (ARPANET, CSNET, NSFNET) were resource sharing networks: computers were bulky and expensive, and researchers used nationwide connections to access them from far away. As personal computers became ubiquitous, and packet-switched traffic was ported to the ubiquitous telephone network, the Internet became the means for people all over the world to communicate with each other. In 1990, Tim Berners-Lee defines HTTP and HTML, leading to the explosion of the World Wide Web.

From interconnecting computers and people now **IoT interconnects personal devices**. The goal is to create a network, an architecture able to sustain all the different interconnected devices that shape the network itself. The basic idea is the interconnection needs to be invisible to users who do not have to do anything, **everything is automatically managed**: data are processed, and the users have an automatic response having so, as result, an **autonomous network that works without human interaction**.





We will see how **IoT can decrease the costs and optimize operations**, that’s why the IoT revenues are so high. Some statistics: the number of Italian enterprises using IoT technologies is around 32% and the number is constantly increasing.

But what is exactly IoT? We need a definition. There are a lot of definitions of what IoT is, all of them are correct but we will use a more formal definition than the ones that appear in the textbooks.

"Internet of things means connecting every thing."

The key part of "Internet of Things" definition is the **Internet**, where Internet means IP protocol: No IP → No IoT. So, the definition we will use during the course is the following one:

"Internet of Things is a paradigm according to which every thing, real or virtual, is assigned an IP(v6) address and can be reached (for example for sensing or actuating purposes) via the standard Internet Protocol stack."

Basic concept: if you assign an IP address to a device, that device is part of an IoT network.

An example of a "Smart" IoT system

- One typical evening planning next working day…
- Tomorrow first office meeting at 8:30 am.
- Typical car trip in these days: 1 hour time.
- 45 minutes to wake up and get ready.
- I decide to set my alarm to wake up at 6:45 am.

What could (will) possibly go wrong?

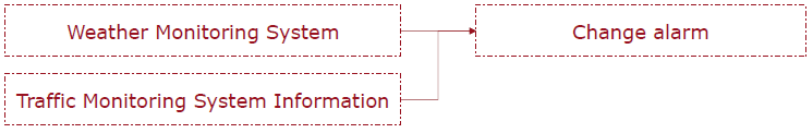
- At 4:30 am it starts snowing
- Truck obstruction along the usual path
- Traffic congestion on alternative paths
- No parking at destination
- Bathroom cold when having shower
- Coffee cold when having breakfast
- Left my car keys at home when in garage
- Elevator busy when leaving my flat

Leaving 10 min. late + 30 min. additional travel time → missed the meeting!

It follows an IoT approach to improve this "critical" scenario…

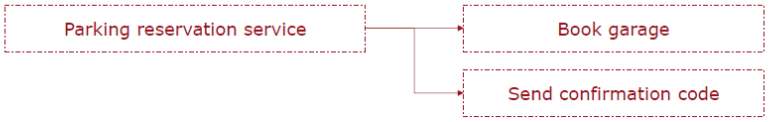
The IoT approach

- **Truck obstruction along the usual path**
- **Traffic congestion on alternative paths**
- **At 4:30 am it starts snowing**
  - Get notified in real time by the Weather Monitoring System or device.
  - Get notified in real time by the Traffic Monitoring System Information.
  - Based on forecasts it anticipates the alarm clock to 6:00 am (30 minutes before)



What could (will) possibly go wrong?

- **No parking at destination**
  - Based on previous experience data and the available parking reservation services decides to reserve a indoor parking slot in a garage.
  - Reservation code uploaded on the mobile phone to access garage at destination

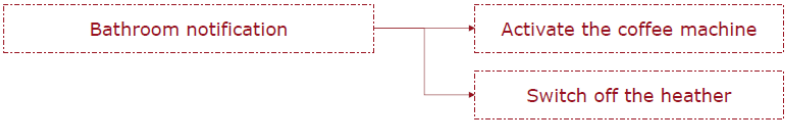


What could (will) possibly go wrong?

- **Bathroom cold when having shower**
  - 10 minutes before at 5:50 am... started warming up the bathroom to 23 degrees

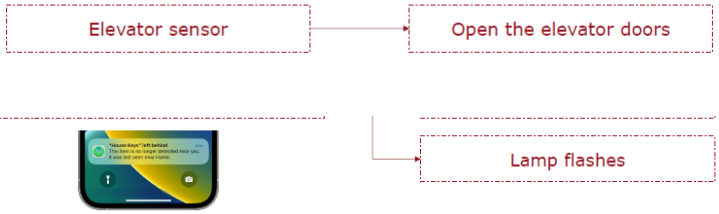
What could (will) possibly go wrong?

- **Coffee cold when having breakfast**
  - The mirror notifies I am leaving the bathroom, and while I get dressed in my bedroom, the coffee machine is activated in the kitchen, and the warming up of the bathroom is switched off.



What could (will) possibly go wrong?

- **Elevator busy when leaving my flat**
  - When I reach the elevator, it has been called and it is waiting for me with open doors.



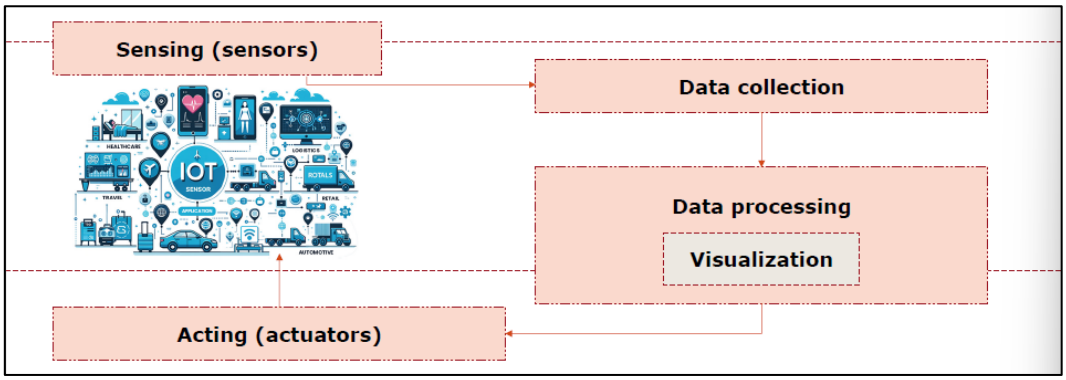
Finally...

- I am leaving with my car right in (planned) time, with my path already set in the navigation system, getting alerted of any problem on the path and need to make detours in real time.
- ... and when at destination I will have my car parked in reserved indoor garage with no delays. Barriers open with my contactless smartphone code...

I participate at the 8:30am meeting RIGHT IN TIME!

We want to have an integrated system made by smart things that make the life easier for the final consumer and so to take actions without human intervention.

IoT high-level architecture



The core part of an IoT system is the set of sensors that produce data to be consumed. Data are collected by a gateway and then they are processed to trigger some actions or only visualized (only to monitor the system making sure everything is working in the proper way). The actuator is another type of device that implements an action based on the result of data processing.

Data collection

Sensors act as a physical-cyber interface that **monitors and reports states of some physical entity** or device. They produce a digital representation suitable for use in the cyberspace. Relatively early in the process, **metadata needs to be captured and used to annotate the data**. In IoT systems, metadata generally describe the nature and context of data capture, such as the sensor type, its location, and in some cases structural relationships to other elements of the system (~ a sort of appendix to real data, used to store other relevant information).

Processing and visualization

There are different types of IoT data processing:

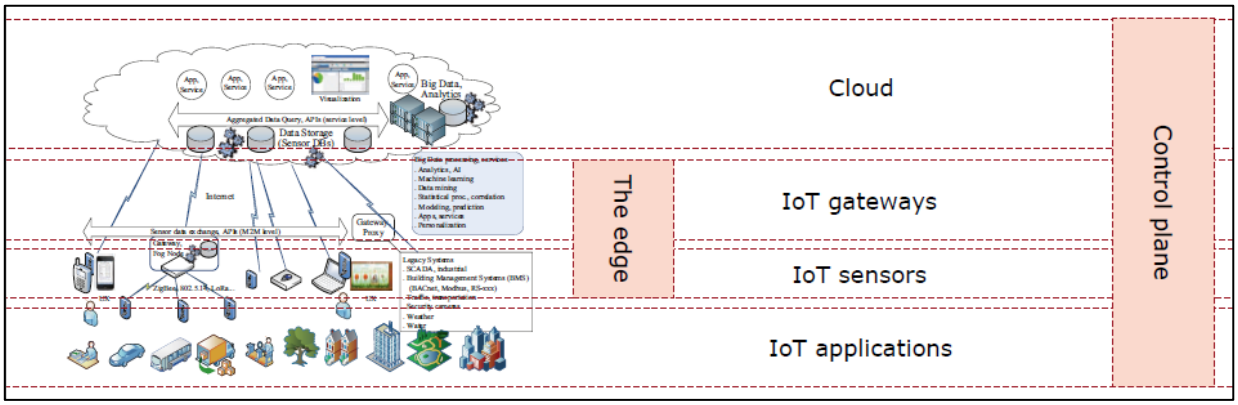
- (Simple) control loop algorithms performed on the incoming data as they arrive.
- Sophisticated forms of analytics and machine-learning algorithms based on past behaviours and observations of the system.

Common data processing steps are: sampling, aliasing, quantization, saturation, hysteresis and non-linearities, calibration, error propagation, optimization and predictions.
In industrial and complex control systems, it is customary to visualize the system state and points of interest to system operators → digital twin / dashboard (e.g., system state, notifications, alarms when faults or anomalous behaviours are detected).

Acting

Acting upon insights and predictions is the output and the ultimate purpose of deploying IoT systems. Common types of actions: from simple remote actuation initiated by operators in response to visualized conditions in a basic monitoring configuration to automated guidance of control points.
Actions can be implemented as **direct** actuation or **indirectly**, in the form of advice to system operators or optimizations resulting in adjustments to the manufacturing process.

A more technical IoT architecture



Let's explore the different elements that shape an IoT architecture.

IoT applications

These are the actual use cases and services that benefit from IoT technology. We need to understand them to design appropriate networks. They span a wide range of industries, including smart cities, health monitoring, industrial automation, transportation, and more. These applications rely on IoT sensors and gateways to collect and process data to offer services like predictive maintenance, asset tracking, or smart energy management.

Examples of IoT applications

A common example is the IoT usage in Industry 4.0 where sensors are used to collect data to optimize industrial operations. Another IoT application is related to smart shops as self-checkout or retail store optimization: sensors can be used to study the shop areas on which the customer is more focused to understand how shop items can be placed to attract customers → look at the example here below where the visors are used to collect data and statistics on customer behaviour.

Another IoT use case is related to healthcare as remote patient monitoring or medical alert systems (sensors are used to detect possible issues). IoT touches other scenarios such as smart cities/homes, transportation and automotive. Autonomous driving is a quiet different use case respect to the others, here the sensors are more complex and due to this is a more challenging use case on which work.

IoT sensors

- **Sensors** can measure some quantity in the environment (e.g., a thermometer). In simple terms they sense/perceive the environment.
- **Actuators** can do something in the environment (e.g., turn on the heating system). Actuators act in response to some measures.

Sensors can be simple or more complex devices. In case of the first they **delegate the major computation to other external nodes** (inside the edge layer - gateways - or even outside) because they are not able to do the operation. In case of the second there are scenarios where the sensors can implement some **local operations to act immediately** because delegating the action to others will take too much time. Basically, most of the sensors are of the first type. Some sensors and things are designed to connect directly to the Internet and communicate with applications and services residing in the cloud. These may require external computation platforms given energy constraints (e.g., security cameras, fire sensors, thermostats, appliances, and power meters, ...).

IoT gateways

Sensors connect to the Internet (Cloud) using intermediaries, such as gateways. Gateways (and fog nodes) are usually more powerful devices. Connection is possible via local network links, often wireless, such as: ZigBee, variants of 802.15.4 networks, Bluetooth, and low-power Wi-Fi, LoRa, NB-IoT.

Gateways usually provide **wide-area connectivity** and **edge processing** for the attached sensors that may come in the form of protocol conversion, data storage and filtering, event processing, and analytics. Gateway is much closer to the end user respect to the Cloud (where processing phase typically happens) and sometimes, as said before, can be useful to process the data in a little amount of time and that's why gateways implement edge processing.

Gateways can also **store data**; this is more convenient, in terms of complexity, instead of saving data on sensors which, again, are simple and resource-constrained devices. This also avoids the problem of data fragmentation since a gateway connects more sensors and it has a full knowledge from more sources (data from different sources stored in a single node).

Communication layers

Enables a vast array of edge devices and things to exchange messages with each other, the rest of the IoT system, and ultimately the Internet.
Communication layers may include a variety of wireless and wired links, spanning local areas and including long-haul connections. Represent a complex infrastructure of links, bridges, and routers that can transport payloads from local point-to-point segments all the way to any endpoint and application on the Internet.

The Cloud

"Back-end" processing is depicted by a generic cloud. Cloud represents a collection of servers and other data storage systems that can be accessed via internet. Sensors are connected to gateways and gateways provide the end-to-end connection with the cloud.

- Data from a variety of diverse sources are **aggregated and processed** for optimization and discovery of global trends and relations.
- Depending on nature and real-time requirements, sensor data may be **processed** "inflight" as streams (a simple check on a value → constantly monitor the condition of a system), **stored** for post-processing and archival purposes, or **both**.
- May also contain some common services such as large-scale storage, analytics processing engines, data visualization and graphing, as well as management functions such as security and provisioning.
- Machine learning (ML) and artificial intelligence (AI) algorithms are usually operated in the cloud where they can work with large aggregations of data.

Control plane

- **Data plane / user plane**: main IoT functionalities: collect, process, and act on data.
- **Control plane**: task of keeping the IoT infrastructure itself running and secure.
    - Service configuration and aggregation.
    - Service problem management.
    - Service quality management.
    - Resource provisioning.
    - Trouble and anomaly management.
    - Performance management.

IoT: why today? A confluence of technological and infrastructure developments centered around the Internet forms much of the basis and impetus for the construction of IoT systems.

- **Industry 4.0** and digitalization.
- **Sensors** – installed base, variety, lower cost, and easier integration.
- Multi-sensors **miniaturization and mass production**.
- **Smart phones** – sensors, gateways, and UI devices.
- **Cloud computing** – global and capacity on demand.
- **AI and ML technologies** – actionable insights with IoT data.
- **Internet** – technology, global infrastructure, and users.

- **Pervasive communications**: mobile devices always connected to the Internet.

MAIN PROBLEMS IN IOT:

1) ENERGY

- Sensors might be placed in hard-to-reach locations: they need to be **battery-powered** and work for months or years.
- Sometimes they have solar panels or other energy harvesting methods, but we need to be careful: size and cost matter!
- **Energy has a significant impact on every IoT operation**: communication, computation, sensing, actuating…
- Due to the energy problem, sensors are often optimized to consume little power:
    - We cannot operate on an always-on basis → **duty cycles** (for instance, having a duty cycle of 1% means that sensor can transmit only for 1% of the time, the rest of the time the sensor does nothing).
      One can also consider a sleeping-mode where sensors are active for a certain amount of time and then they just sleep → this solution is not the best since during the sleeping time sensors cannot detect anything and also the waking-up process can consume a lot of battery if the network is not properly designed.
- We need to consider every aspect: any calculation or long transmission is expensive. For this reason, sensors often transmit **very few bytes** at relatively long intervals (no transmissions for hours or days in some applications).
- Limited processing capabilities (and so latency concerns).

2) SECURITY

- The simplicity of IoT devices also makes them hard to secure.
- Impossible to run strong cryptography.
- Difficult to patch and update to solve vulnerabilities.
- Connected by design to the open Internet.
- There are possible security solutions against hackers, but there is also a privacy issue: what do sensors say about our lives? Who gets to see the data?
- Sensor data can tell a lot about our lives:
    - Location data can identify movements and activities.
    - Domotic systems tell stories about our habits.
- Data are often used by IoT companies and law enforcement agencies.

3) SCALABILITY

- Cellular networks and wireless technologies were designed for few devices with high requirements (e.g., humans streaming videos).
- In the IoT, we have thousands or millions of devices with very little data to send.
    - New access mechanisms.

- Sensor identification problems.
- How do we get the data with the minimum number of transmissions?

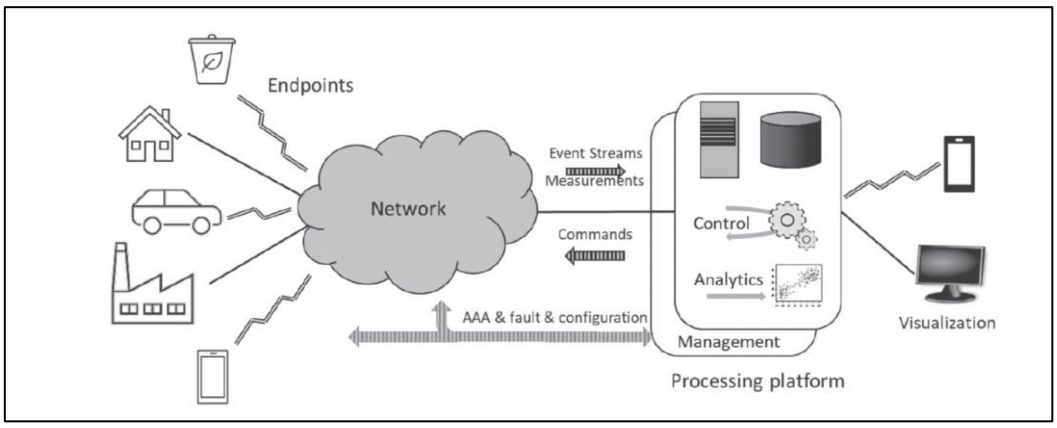
4) RELIABILITY

- The “converse” problem (what do we need to ensure that we get the relevant information) can be crucial in industrial and safety systems:
  - How do we identify anomalous (and potentially hazardous) situations?
  - Anomalies can be single-sensor or multi-sensor.
  - How do we trust that there is an anomaly when sensors are unreliable?

2. IoT system architecture

2.1 Structure and definitions

Now we will focus on the edge layer of the IoT architecture, so on sensors and gateways.



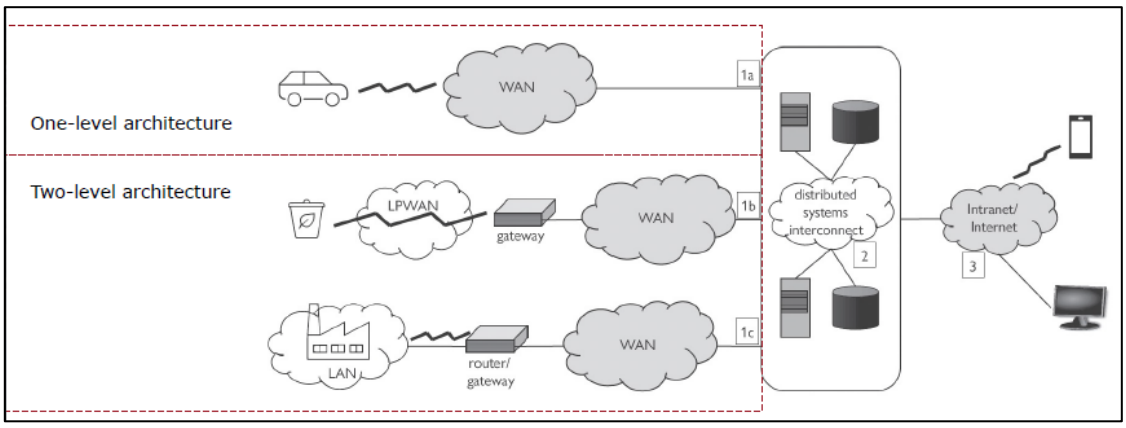
Communication in IoT networks may be achieved using two different approaches:

- **Client/server**: the information sources explicitly know the destination of their messages → not so convenient in IoT networks as the publish/subscribe paradigm.
- **Publish/subscribe**: the sources send (*publish*) their messages on a logical distribution channel, to which zero, one or more interested destinations may *subscribe*, implementing a **one-to-many communication pattern**. In the client/server we need to establish one-to-one communication link. In the publish/subscribe, instead, the nodes that publish data do not publish data for a specific node, so there is a one-to-many communication (many nodes can subscribe to the same list if they are interested in that data). This approach offers more freedom than the client/server one.

Looking at the information carried by the network, we can identify three flows:

- **Streams of asynchronous events** (generated autonomously).
- **Measurements** transmitted periodically or on-demand. Subscribers may request some data to the publisher, which is triggered by the subscriber request, and so the required data has to be sent.
- **Commands** coming from the processing platform.

Network architectures



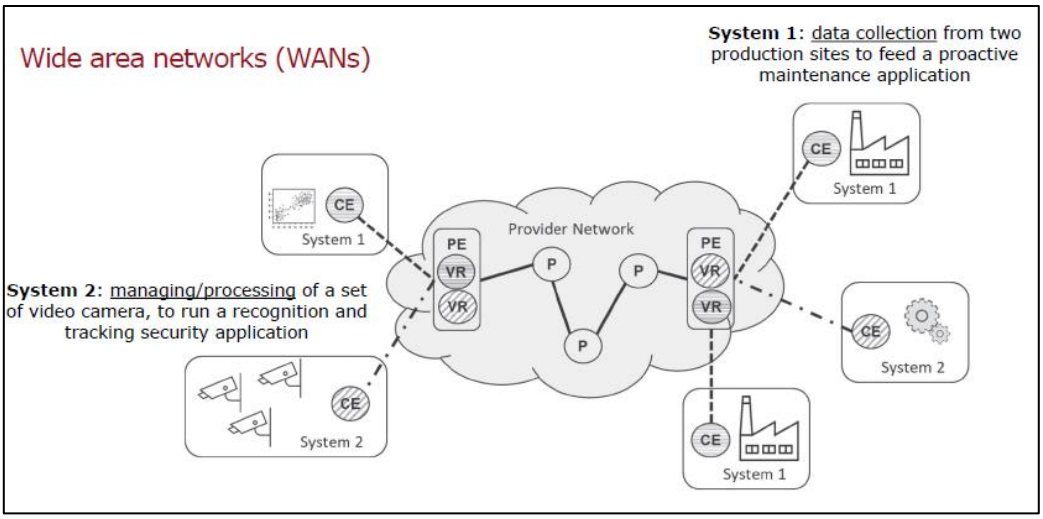
Three possible network architectures:

1. **One-level architecture**: the endpoints know about the addressing scheme. The end-node needs to implement the communication protocol and so it becomes a network device.
    - Direct IoT connectivity to the Internet protocol stack.
    - Typically implemented with cellular network technology.
2. **Two-level architecture**: the last segment is unaware of addressing aspects (endpoints send messages which are not geographically routable) and a gateway handles the communication protocol.
    - End nodes are generally resource and/or energy constrained (e.g., sensors).
    - Typically implemented with specific IoT technologies: Low Power Wide Area (LPWA) technologies (e.g., LoRa, Sigfox, NB-IoT).
3. **Two-level architecture**: the endpoints are not resource-constrained and implement the TCP/IP protocol stack. Due to that, the gateway is no more useful, and the traffic is managed by an intermediate router.
    - Typically implemented with Ethernet LANs or Wi-Fi, etc.

Wide Area Networks (WANs)

WANs are wired TCP/IP networks. The WAN must be shared among many users and applications for economic reasons and due to that potential **security and access issues** may arise. To solve this problem there are mechanisms to segregate and reserve the traffic. One of these mechanisms is VPN.

Virtual Private Network (VPN) is a network that takes resources (bandwidth capacity, switching and routing functions, firewalls and other security appliances etc.) and reserves them to a defined group of users/endpoints. Users external to this group cannot communicate directly with the users of the group. VPNs use **private IP addressing** and **private routing protocol** instances. As a drawback, this setup increases the complexity since we need to manage different VPNs inside the same WAN.

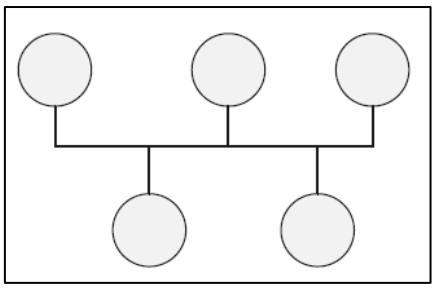


In the above example the WAN is shared by two systems. The system1 collects data from 3 sides. The system2 records videos via cameras. Since both systems share the same network, two VPNs were created and the virtual routers (VR) of both networks provide access to the "real" network. In fact, both systems operate on two separated VPNs, avoiding so possible security and collision problems.

NETWORK TOPOLOGIES

1) BUS

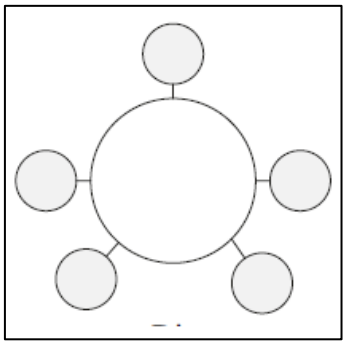
- Every node taps into a common medium (the bus), signals may collide with each other.
- Need to arbitrate who will get the bus.
- The common medium is the bottleneck (to be shared) → **single point of failure** (if the bus is not available for some reason all the nodes are cut off).
- Use CSMA/CD (Carrier Sense Multiple Access with Collision Detection).
- Example: (old, 10BASE2, 10BASE5) Ethernet.



2) (TOKEN) RING

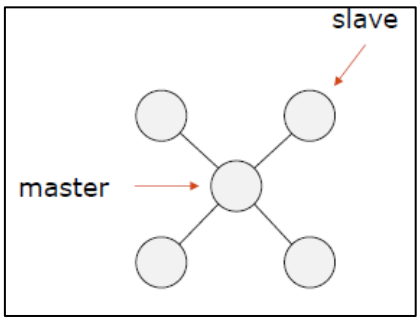
- Nodes are in a ring: receive from predecessor and send to successor.
- Need to arbitrate which node can access the ring (repeaters at every link).
- The common medium is the bottleneck (to be shared) → **single point of failure**.

- **Token-ring topology**:
  - A node wishing to transmit waits for the receipt of a token, removes it from the ring, and places its message.
  - Then, the node holding the token passes it along.
  - Eliminates collisions and can increase throughput.
  - Need a mechanism to preserve token integrity and to regenerate it if necessary (e.g., when nodes are powered off).
  - The **communication cycle can be very long** and there is no coordinator that manages the whole system.



3) STAR

- The **master** is at the center, other nodes are **slaves** linked to the master.
- This approach is better than the ring topology since here we have the coordinator which is the master.
- Slaves communicate via master.
  - Easy to arbitrate among slaves (master decides).
  - Good for small networks or that requires predictable performance.
  - **Not scalable** (the master is the bottleneck).
  - Master failure shutdowns → **single point of failure**.

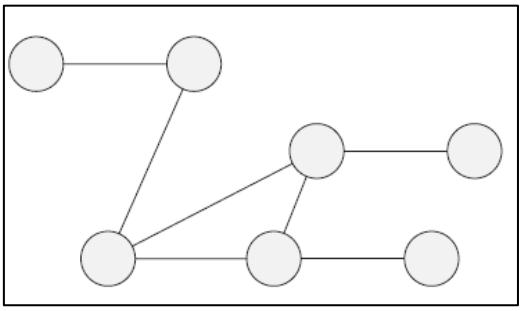


4) (PARTIAL) MESH

- IoT networks use this approach. Every node can communicate with others, deleting so the client/server one-to-one communication approach.
- Nodes are arranged according to **connectivity graph**.
  - Each node has a dedicated point-to-point connection to any other node.

- Number of physical links/connections is n(n-1)/2 (if full-duplex). The problem here is the scalability: if we have many nodes the number of links that we have to establish is really huge and expensive.
- Reduces traffic forwarding issues (+ no shared medium collisions).
- Ensures robust and private/secure network.

- **Partial mesh**: due to the scalability problem of mesh networks this solution represents a better approach. A subset of links representing direct connectivity between a subset of nodes → some nodes are used as intermediaries.



ENDPOINTS

A **sensor** is a device that detects some measurable aspect of the physical world state (stimulus) and converts it to a processable output (e.g., an electrical signal that can be sent to another node). There are thousands of sensor types, that can measure almost anything:

- Pressure, humidity, air quality
- Fingerprints, facial features, iris patterns
- Microphones, touch-sensitive surfaces, radars
- Light, smoke, chemical sensors
- Temperature
- Force
- Positioning
- …

Types of sensors:

- **Fixed**: stable position within the network topology (wireless vs. wired) → so basically sensors that are always in the same physical position (can be connected to a wired network and they do not suffer from battery constraints).
  - No need to re-authenticate, stable address and routing, simple management, etc.
  - Can be mains powered.
  - Can have permanent bad wireless connectivity conditions, if bad deployment.
- **Mobile**: must be connected to a cellular or LPWA network.
  - May experience a degraded service only occasionally.
  - Limited mobility constraints (e.g., handover).
  - Cannot be mains powered.

- **Nomadic**: can change their physical position but stay in the same place for the whole duration of a communication session.
    - Must re-authenticate, but do not require tracking functions.
    - UC: agricultural IoT with drones that move and collect data in different areas.

Another sensor distinction:

- **Active**: need to run a current or send a signal to get a measurement (e.g., clocks, strain gauges, radar).
    - They require an external power source to generate and emit energy (battery).
    - Examples: LiDAR, radar, ultrasonic sensor.
- **Passive**: use the physical phenomenon itself rather than using their own energy or producing dedicated one (e.g., piezoaccelerometers, GPS, thermocouples).
    - This does not mean that passive sensors consume zero energy!
    - Ideal for IoT low energy devices.
    - Examples: temperature sensor, photodetector, passive Infrared (PIR) sensor, RFID tags.
- Main difference: active sensors use their own energy to collect data, passive sensors do not consume their own energy but, instead, collect it through some harvesting techniques.

Actuators are an output part of the IoT system that **performs direct actions**. Actuation provides the means to implement control actions as and when determined by the algorithms or system operators.

- **Digital actuators**: used to implement simple on-off actions. Past the appropriate signal conditioning can trigger relays or change the state of a thing, such as a power switch which can result in turning a light on or off.
- **Analog actuators**: produce continuous signals that can be used to drive devices. For example, controlling the speed of a motor or producing sound in headphones.

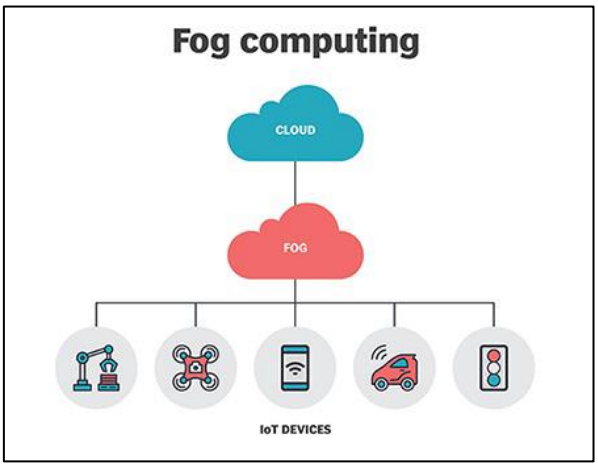
Note: for us sensors and actuators are both end nodes, sometimes we will not distinguish between these two devices when talking about an end node.

Gateways

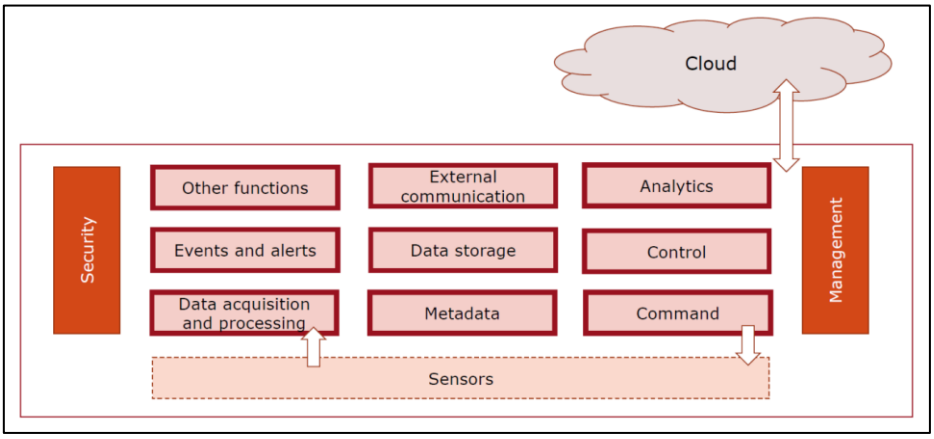
- It **links** sensors and things at the edge with higher levels of system processing hierarchy and the cloud.
- It performs or assists **data collection**.
- It provides **internet connectivity** to transfer of sensor data to other components (e.g., Cloud).
- **Security** boundary between things with varying levels of security.
- Optional: data storage, event and alert processing, and control (automation).
- Advanced: analytics.

Fog nodes

- A simpler version of gateway devices.
- Etymology: “bring the cloud to the ground” → a way to speed up computational processes. The fog node becomes a sort of smaller and more limited cloud component.
- Continuum between the edge and the cloud by placing **additional computation and storage capability closer to the edge**.
- Perform local processing (e.g., data reduction, filtering, and front-end analytics) with potentially **lower latency than in the cloud**.
- Fog nodes tend to be more powerful but are otherwise not architecturally or functionally different from the edge gateways.



2.2 Data plane functions



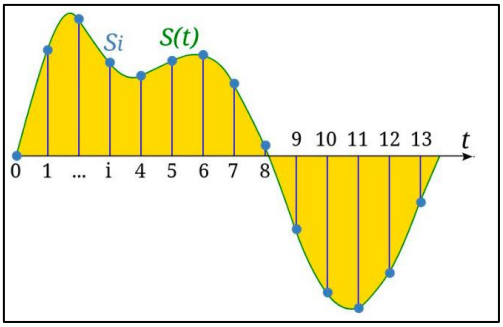
Data acquisition and processing

In order to get a digital signal from a sensor, we need to account for the following processes (consider not all of them are mandatory, it depends on the scenario of interest):

1) *Sampling*

First, we sample: we take periodically measurements over time, and represent the signal with those measurements → ***Nyquist theorem***: if the sampling rate is at least twice as fast as the signal, sampling has no errors so an appropriate sampling frequency should be chosen.

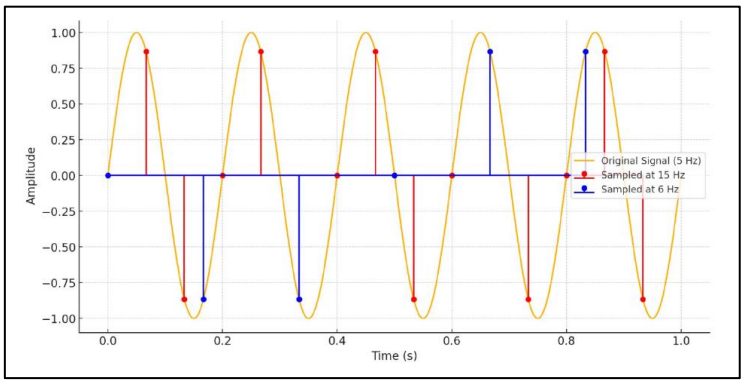
Example: if a signal is thought to have a maximum frequency between 1000 Hz and 4000 Hz, which of the following would be the most appropriate sample rate? 8000 Hz could be a good choice, but we are on the limit since the theorem says "at least twice…" so it is better to have a larger boundary, something as 9000 Hz. (Possible question choices: 500 Hz, 8000 Hz, 9000 Hz, 24000 Hz).



2) *Aliasing*

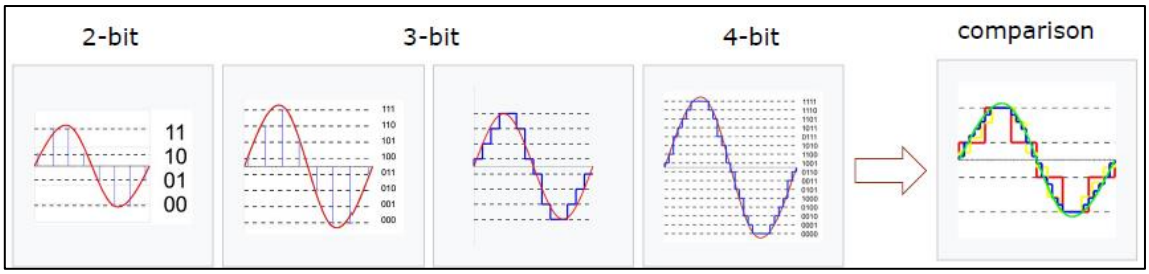
When a signal is not sampled at a sufficient rate, a phenomenon called *aliasing* occurs. This happens when high-frequency components in the original signal's frequency spectrum are misrepresented as lower-frequency components in the sampled signal. As a result, the reconstructed signal becomes indistinguishable from a different signal with an actual lower frequency, leading to distortion and loss of information.

To avoid aliasing, it is essential to recognize it as a potential issue and apply appropriate tools, such as anti-aliasing filters or adhering to the Nyquist sampling criterion. For instance, sampling a 5 kHz signal at 6 kHz may seem sufficient, but it risks aliasing and fails to fully capture the signal's true characteristics. Sampling at 15 kHz, on the other hand, provides a more accurate representation of the original signal.



3) *Quantization*

Digital values are finite: we cannot represent continuous values precisely! Quantization always introduces an error, which is the sensor's natural error. We need to find a trade-off for the number of bits used to represent the signal, otherwise we risk to crowd the network.



4) *Saturation*

Sensors often have an **upper limit**, which is either physical or dictated by the maximum quantization level: in this case, we have saturation → all higher values are mapped to the same output. This is what happens when you take a picture with high exposure: all lighter points are mapped to white!

5) *Hysteresis*

Sometimes, a sensor (e.g., a thermometer) takes a while to track the process because the **physical nature of the measurement is slow → hysteresis error** corresponds to the delay of measurement that may lead to inappropriate sensor behaviour.

Example: imagine we have a sensor that detects the room temperature and when the temperature goes over 21 degrees, we want to turn off the heating system. It may happen that once the room temperature goes over 21 degrees, the sensors is not able to detect it immediately due to the physical nature of measurement and so we have a delay in turning off the heating system. **Hysteresis cannot be eliminated but can be reduced**.

6) *Non-linearities*

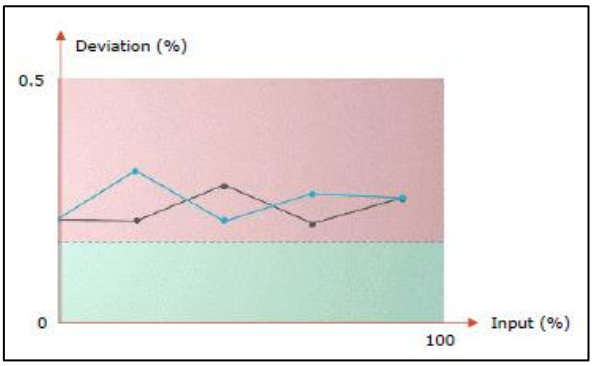
Ideally, sensors should also have linear responses but there are often **nonlinear effects** that we need to adjust for, meaning the sensor output does not change proportionally to the input across the entire measurement range.

Example: a linear sensor's output will change by equal amounts for equal pressure changes across its entire range.

- From 0 to 10 bar
- 0 V at 0 bar pressure
- 5 V at 10 bar pressure
- But at 3 bar pressure, it outputs 4.2 V instead of 3.75 V.
    - **≤ 2% non-linearity of full scale**

7) *Calibration*

- Sensors often require calibration to give proper results:
  - Bias is a constant error term in one direction.
  - Distortion is due to non-linearity.
- By measuring a series of known quantities, we can compensate for these issues (however, external factors such as temperature can throw off calibration!)
- Example: using the data from the calibration sheet, we see that the deviations are all less than the maximum deviation allowed of 0.5%.



8) *Error propagation*

- **Calculations can change the error of a measurement**, as the errors are also considered in the function you compute.
  - Non-linear functions can have weird effects with measurement errors.
  - Derivation (e.g., computing velocity from position measurements) is brittle: since the measurements are close together, noise can have a huge effect.
- **Integration/averaging** are robust (rely on many points, and noise is compensated).
- Example: Calculating heat index
  - Temperature sensor: measured (30°) with uncertainty ±0.5
  - Humidity sensor: measured (70%) with uncertainty ±2%
  - Heat index = T + 0.33×RH − 0.7 = 52.4
  - Uncertainty with error propagation: 0.828 > (0.5 + 0.02) → this happens since the error propagates and the final error is higher than the sum of both single errors, so it is fundamental to reduce the errors as much as possible.

External data communication

Sensors' data may need to be shared with external parties, such as other peer nodes at the edge, fog, and cloud levels. Messages are exchanged between IoT nodes may be delivered using **pull** or modes.

- Pull: a message is **explicitly requested** from the source (server) by its recipient, an IoT client.
- Push: the data source and the gateway send data **when the specified conditions are met**, such as arrival of the data sampling time mark, or exceeding of the measurement threshold.

Publish/subscribe system: data sources "publish" data to a known place, and authorized clients can "subscribe" to data of interest and receive related messages. Publishers do not need to know the identity or number of their receivers (subscribers), and the receivers do not need to know the identity or state of their publishers.

How to share sensor's data?

Some processes (e.g., an electricity meter, a room thermostat) do not need to be sampled very often. If we only care about **cumulative measures** (e.g., how much electricity was consumed in total), we do not need frequent samples (less critical scenarios). On the other hand, faster processes need **frequent samples**, as process can change faster.

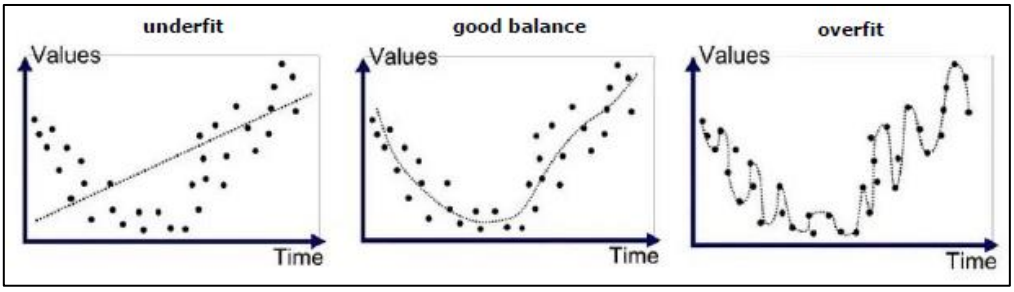
Why don't we transmit all the time?

There are several reasons why we cannot transmit all the time, but we will concentrate on two of them in this first part of the course: **limited bandwidth** and **limited energy**. Each transmission (and, in fact, each activity the sensor performs) costs energy, as well as blocking the wireless channel for other transmissions.

Some solutions to send sensors' data more efficiently and so decreasing the number of sensors transmissions:

1) *Interpolation*

Interpolation is the mathematical tool to **infer missing data from sparse samples**. We have to make some assumptions on the nature of the process to be able to interpolate: if we have the correct model, we can send a lot fewer data samples!
Interpolation can also be dangerous: we can **underfit** (select a model that is too simple) or **overfit** (select a model that is too complex and ends up following random noise) → choose the right interpolation factor!



2) *Spatial correlation*

Data are usually correlated in space:

- Traffic on either side of a bridge.
- Temperatures close together.
- There can be easy natural relations (points close together are similar) or structural relations (connected points are similar).

The main idea is to use sensors' data to estimate data in areas where we do not have any sensor placed. Since measurements have errors, correlations help us **improving the quality of estimates**: we can make statistical calculations and estimates to figure out what is happening in points where we have no sensors or refine our estimates.

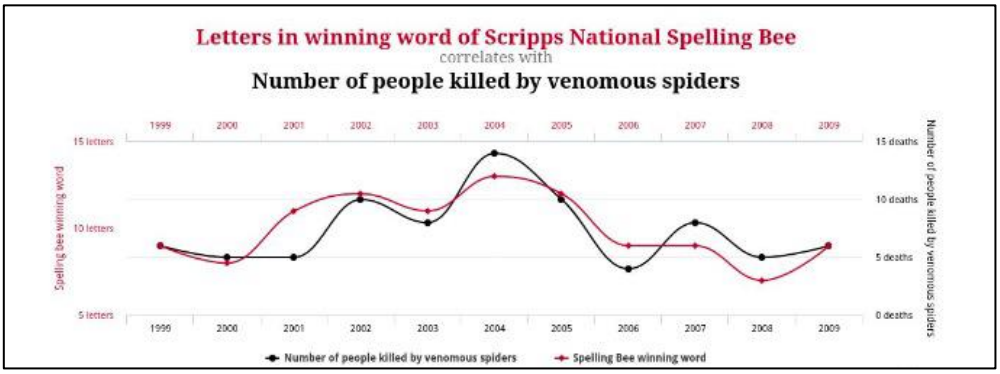
3) *Temporal correlation*

Data are also correlated in time:

- Traffic in one spot.
- Temperature measured by one sensor.

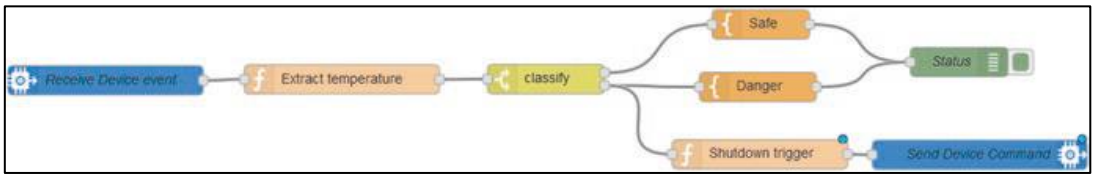
The relation here can be more or less complex, and depend on space as well (e.g., line at a traffic light), but we can also exploit this to get better data with fewer transmissions.

Correlations can be misleading and dangerous! When relying on statistical information to reconstruct missing data or make decisions, we need to be aware that random chance is always a possibility (look at the example here below).



Local control

- Provided in systems where latency is critical and/or edge autonomy is required.
- Local control enables the edge node **to execute pre-defined control sequences** when a specified condition occurs on some combination of local data.
    - **If This Then That**: scenes where lights and heating/cooling are adjusted accordingly when users are leaving or approaching their homes.
- **Node-RED**: provides a graphical user interface to construct data flows with the functional processing modules and actuation output stages.
- In the extreme case of temperature, the script triggers the device shutdown.



Data storage

Data storage for the acquired data may be provided by the **gateway**. The existence of local storage provides the flexibility to conserve network bandwidth and to reduce the load on cloud resources by **selectively reporting data that meet certain conditions**.

- Depending on the nature of the physical thing being measured, only some of the data points may be of interest. E.g., successive minor changes may be of little interest, but those that exceed a comfort guard band or change by a significant percentage or amount from prior readings are.
- Qualifying readings may be defined as events and reported only when they occur, with raw readings stored in the local database should they be required for auditing or subsequent analysis.

Edge analytics

Edge analytics for the data may be provided by the **gateway** or the **fog node**.

- Proximity to data sources: low latency.
- Ability to process high-frequency data samples (at the rate of the sensor).
- Conservation of network bandwidth by not sending data to the cloud.
- Can operate even in disconnected mode (if the connection with the cloud collapse for some reason).
- Cons: analytics generally require AI/ML models:
    - Insufficient computational resources at the edge.
    - No data aggregation of edge/local data.

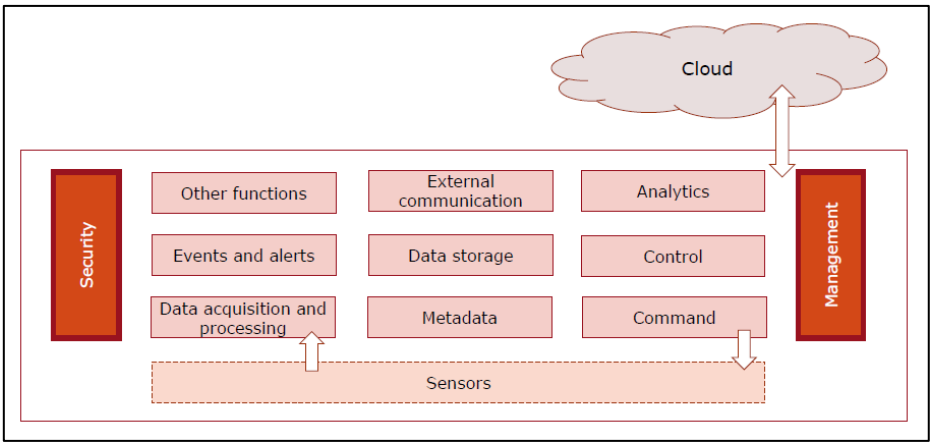
IoT systems should be implemented using design tools and practices that facilitate flexible allocation of functions (in tandem between edge and cloud).

Analytics placement: where?

- Optimal placement of data and processing functions is an age-old trade-off in distributed systems.
- It is based on whether it is more cost-effective to move the data to the computation or to move the computation to the data, depending on:
    - Availability and cost of **bandwidth**
    - **Latency** requirements for time-critical operations
    - Local **autonomy** of operations, including disconnected mode
    - Security and data control or **privacy** concerns
    - Cost and complexity of managing distributed nodes with computing and storage
    - Available **energy** resources
    - Available **computational** capacity
    - ...

2.3 Control plane functions

Control plane functionalities do not deal with data but with the management and security of IoT networks, allowing to keep the whole infrastructure running and secure.



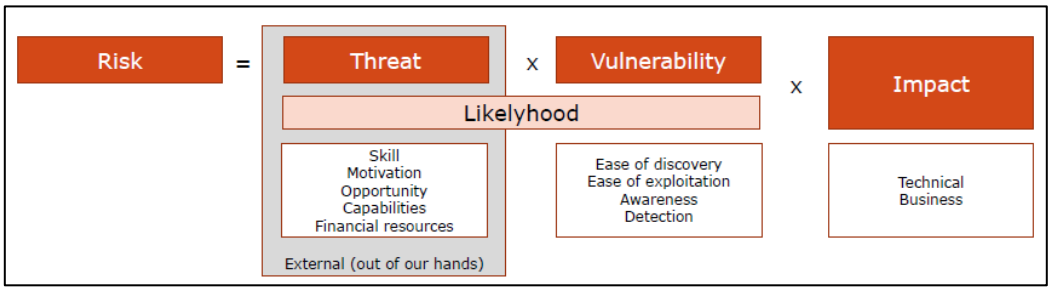
Fully functional gateways may be fairly sophisticated computational and communication nodes that host and execute a variety of services. So, those gateways need to be properly secured and managed.

Main control functionalities:

- Fault management (troubleshooting, error logging, and recovery).
- Remote monitoring, control, administration, and diagnostics.
- Remote firmware and software updates.
- Security updates.
- Metering (network bandwidth and software usage) and supervision.
- Provisioning and authentication.

(Cyber)security

The first step is to determine the security risk: identify, quantify, and prioritize the risks for the system, and find a balance between costs and potential losses from residual risks. There is a model used to quantify risk: **vulnerability, threats, and impacts**.



- **Vulnerabilities**: can originate from the design of the system architecture, technology, and protocols, but more often emerge in the implementation (e.g., coding flaws) and in

the operation phases (e.g., configuration errors, use of weak credential, unpatched devices, unexpert use, etc.).

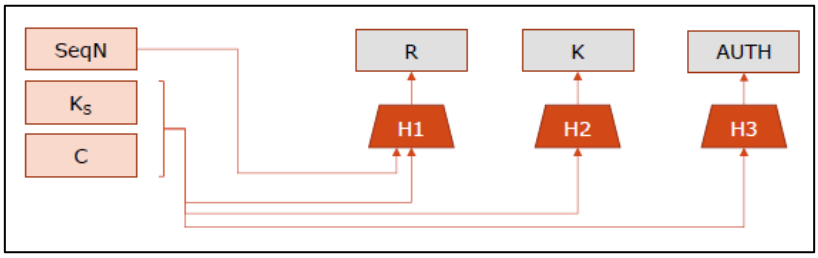
- **Security measures** for the diligent user:
  - Partitioning the system into zones (characterized in terms of security levels).
  - Adopt effective authentication protocols, tailored to the different zones.
  - Provide cryptographic protection of data.
  - Install equipment from truster providers.
  - Make and install frequent patches and workarounds.
- **Threat**: an IoT network is a possible point of attack if the endpoint is unguarded and is positioned in an easily reachable position.
  - **Tampering**: intentional alteration of measurements and event notifications, or intentional installation of malwares on the endpoint.
  - **Software patching** process may be slow (hours or days), making devices vulnerable.
  - Malwares can be injected during **remote firmware updates**.
- End devices must be treated *a priori* as the weak points of the architecture and constitute **a separate security zone**.
- Best approach: anti-tampering self-disconnection operation if any security risk is detected.
- IoT-oriented and industry specific standards have been developed for security (look at slides for more details).

Provisioning and authentication

The lifecycle of an endpoint begins with its **provisioning** in the network: the process of giving the endpoint the configuration data and the ability to present itself.

- **Authorization**: checking the rights of access to the network.
- **Application-level authentication** (if the network is shared among applications).
- **Authentication**: verification of the identity ("I know your name, is it you?").

Auth is performed through a **challenge-response protocol** employing secret keys. The goal is to ensure that both the server and client possess the same secret key K_S, confirming their identity to each other.



- Two parties (end node + gateway) share a secret key K_S.

- The server (gateway) generates a random challenge C, and computes cryptographic operations, using the hash functions H1, H2 and H3 on the following elements:
    - R: the response expected by the client.
    - K: a key used to encode parts of this exchange.
    - AUTH: a code that demonstrates that the server also possesses K_S.
- The server sends E_K{SeqN} (i.e., the ciphering algorithm using K, AUTH, and C).
- The client
    - Computes K, decodes SeqN, and understands if this is a replayed authentication request.
    - If this is a fresh request, it computes AUTH and verifies if the server knows K_S.
    - Computes R and sends it back to the server.
- If the server receives a correct response R from the client, it verifies the identity of the client, and the authentication process is successful.

- Problem: this method is based on the fact that only server and client know K_S.
- However, distributing the secret key is challenging. We cannot assume that endpoints hold the key from the beginning (firmware installation).
- Possible solution: build a **chain of thrust** via a **Trusted Third Party (TTP)**.
    - The endpoint stores the public key of the TTP at the firmware installation.
    - With this key, the endpoint receives the secret key from the authenticated TTP.
    - The server uses different credentials from those used to authenticate the endpoint.

- Problem: a close device, impersonating the real device, distributes credentials.
- Possible solution: **Physically Unclonable Functions** (PUFs).
    - PUFs can generate outputs with the same properties as one-way hash functions, but their input/output function depends on random characteristics of the component occurring in the fabrication process.
    - It works if K_S is embedded in the hardware → impossible to be extracted even if the device is tampered with.
    - PUFs require performing an initial enrolment process, where a set of input/output pairs is generated and stored securely for the upcoming authentication procedure.
    - This is the common approach used in IoT networks.

Metering and supervision

- **Supervision**: set of activities to guarantee that the performance and availability levels required by the IoT applications are maintained.
- The last segment (link connecting the endpoints) is the most critical part (a less strong kind of connection since this is typically a wireless one).
    - The wireless connectivity with the endpoints may be discontinuous due to environmental factors (e.g., rain, obstructions, etc.), or interference.

    - Use metering functions to measure quality of the connectivity (e.g. SNR).
    - Some applications are resource-constrained, and endpoints may be rarely contacted: “keepalive” messages for metering if the state of the node remains unknown for too long.
- The supervision of gateways is less critical. Continuous connectivity is assumed between the gateways and the core network (a strong wired connection via cables and fiber that leads to robustness).
- In case of cellular access (direct-to-endpoint connectivity with no gateway) supervision is not needed since this job is carried out by the standard cellular network.

2.4 QoS and performance

Delay

- **One-way delay (OWD)**: time between the transmission of the first bit of a packet at an interface and the reception of the last bit of the packet at the other interface.
    - Requires perfect time synchronization between the endpoints.
    - Time reference: **Network Time Protocol (NTP)**.
- **Two-way delay (TWD)**: time between the transmission of the first bit of a packet at an interface and the reception of the last bit of the packet at the same interface, after being reflected by the system at the other end, net of the End System Delay - ESD (we don’t consider the processing delay because we are interested in evaluating only the network performances) ➔ it is the time between the transmission of the measurement from the endpoint to the gateway and from the gateway to the same endpoint.
    - No synchronization, but it needs to ESD (processing time at the remote node).
    - Another metric that considers ESD: **Round Trip Time (RTT)** = TWD + ESD.
    - Warning: **OWD ≠ TWD/2** (different routes, different link characteristics, etc.)
- **Packet-delay variation (PDV)** (~jitter): based on previous OWD/TWD measures.
    - PDV = $|OWD_i - OWD_{i-1}|$
    - PDV = $|OWD_i - mean\{OWD\}|$
    - PDV = $|OWD_i - min\{OWD\}|$
    - PDV = $|OWD_i - OWD_1|$
    - ...
- Problem: these delay metrics do not consider the MAC delay for channel access. In fact, in IoT networks many devices can be connected to the same gateway and we need to manage the “channel access” which leads to another kind of delay.
- **End-to-end delay (E2E)**: it consists of (at least) three components:
    - **MAC delay** (under the control of the system designer).
    - **Access link and gateway delay** (under the control of the system designer).
    - **OWD** (depends on the WAN).

Delay vs "real-time"

- The expression "real-time" may have different interpretation:
  - Synchronization to a common time reference.
  - Responding fast.
  - **Responding within predictable response times**.
- **Hard real-time**: if the response exceeds a limit, the system does not operate well. We define a limit, and we check if the system respects that limit or not.
  - Example: respond within 5 ms.
  - CASE 1: The application always responds in 4 ms (Average delay: 4 ms; Failure rate: 0%).
  - CASE 2: The application responds in 1 ms for 99.9% of the times, and in 6 ms in 0.01% (Average delay: 1.005 ms; Failure rate: 10^{-3}). If the delay is of hard real-time type this case is not convenient even if only 0.01% of the communications exceed the limit → we need another type to accept these small percentiles.
- **Soft real-time**: it involves some response time percentiles.

Packet loss

- In IoT networks, loss may occur if:
  - The buffers of a node overflow.
  - Transmission errors (negligible in wired LANs, dominant in wireless LANs).
  - **Delay much larger than expected may be considered as a loss.**
- How to have realistic measures of packet loss? There are some problems:
  - Long-lasting measurements: averaging of data (no indications on the short period). Anyway, this approach may not capture some short important measurements.
  - Short measurements: no statistically meaningful data since we are considering short measurements (opposite problem of long-lasting measurements).
  - Intense bursts: too intrusive, altering the result (not representative of a real solution).
  - Since all the above solutions present some problems, the best approach can be using a combination of all the techniques and then evaluate the metric.

Capacity

Typically not a problem, as applications are not (in general) bandwidth demanding. Capacity depends on how frequently we want the sensors of a network send data; this is not equal in all scenarios (critical vs not critical ones).

- For some high-end applications (e.g., telemedicine, IoT, automotive, etc.), capacity is generally less important than other metrics such as reliability.
- Example: metering application.
  - Collect 1M samples (of 128B) every 15 minutes.
  - Average throughput: 1.000.000 x 128B x 8 (bit) / (15 x 60) = **1.3 Mbit/s**.

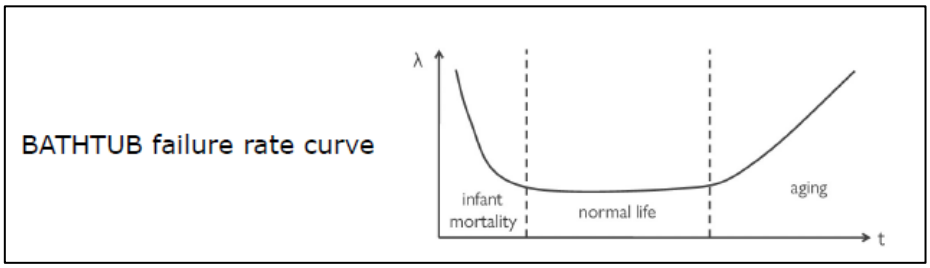
- If the same measurements take place as unsolicited events from the endpoint in the same second, the link capacity shall be around 1.000.000 x 128B x 8 (bit) = **1 Gbit/s**.

Reliability

R(t): probability that a system is working as expected at a certain time t.

- For most systems, the early phase of "high mortality" is short and occurs during tests (e.g. manufacturing defects).
- The system is generally replaced for obsolescence as the aging phase approaches.
- In "normal life", most systems shall be reliable.

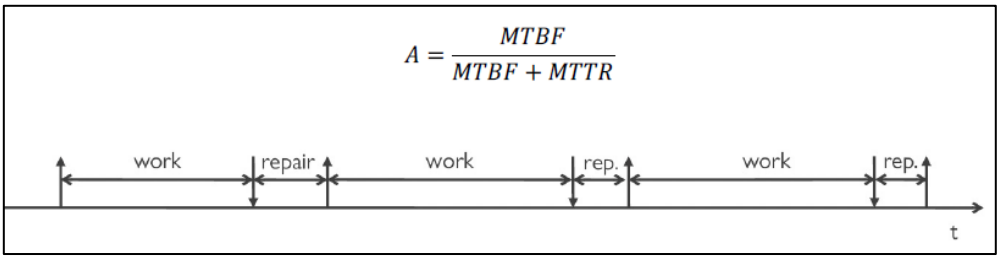
R(t) is **memoryless**: the residual life of failure does not depend on how long the system has been working → the system **does not age during "normal life."**



Availability

Probability that the system is working when observed at a random time instant.

- We assume that the system can be periodically repaired if needed and put back in service. The basic idea is to reduce the maintenance time to have a higher availability indicator.
- **Mean Time Between Failures (MTBF)**
- **Mean Time To Repair (MTTR)**



- For memoryless systems, MTBF = Mean Time To Fail (MTTF) = $1/\lambda$.
- Example: prevention of fires in a forest
  - N sensors that monitor fires in the forest.
  - Sensors fail at rate λ.
  - The prevention works if x% of sensors are active and alive.
  - Period of effectiveness of the system (T): $Ne^{-\lambda T} = xN \rightarrow T = -\ln x / \lambda$

2.5 Energy constraints

Most sensors and components use electrical energy, i.e., they exploit the movements of electrons across a conducting medium. Powering is a very important issue and depends on the endpoint capabilities and the network architecture procedures.

Powering	Wireline	Wireless	Powerline
Mains / manually recharged	Field buses	Smartphones	Smart grid devices
Battery		Most sensors	
Data line	Videocameras		
Harvesting		Environmental sensors	

Sources of consumption

1) **Computing** requires a certain energy for each cycle, corresponding to one basic operation. Some operations (e.g., sinusoidal functions) require multiple cycles. Energy consumption is usually **a function of the number of cycles required**.

- If a device has a certain clock frequency, we can also compute the power it requires to perform calculations.
- Computing energy can be formalized based on the **number of logical operations that are required to perform**: AND, NOT, OR, XOR.

2) **Boards and PLCs** are standard circuits used in industrial automation. Larger IoT nodes also use Arduinos or other pre-made boards. Smaller, more efficient nodes are often designed ad hoc, to use as little energy as possible. In general, IoT nodes have limited computational capabilities.

3) **Digital communication** usually requires more steps than just sending a wave, and each consumes energy:

1. **Encoding** and **error protection**
    - Use known redundant codes: the simplest example is the parity check bit.
    - Decoders are often complex but can correct errors in transmission and recover the original packet. However, complexity is paid for in energy.
2. **Preambles** for synchronization and channel estimation.
3. **Modulation** from the baseband digital signal to the passband analog signal.
4. **Transmission** (power is applied to the antenna) (*not required for reception*). Since IoT devices are quite simple the communication is not optimized, and this leads to more energy consumption.
    - IoT transmissions are usually slow (low bitrate), but distance plays a large role.

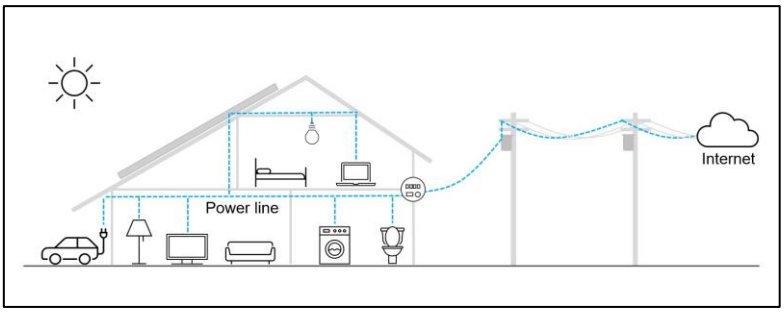
Sources of energy

1) **Mains power**: provide virtually unlimited power, making it possible to employ complex protocols, work at high bitrates, and achieve long-distance communication. There are some drawbacks:

- Only work in wired fixed systems.
- Can be dangerous in environments a risk of fire and explosions.
- Need to lay out an electric line (challenging in hard-to-reach remote/rural areas).
- Need to put transformers (expensive).

2) **Power over Ethernet (PoE)**: employ the wired communication infrastructure also for powering. The idea is to use two of the four cables that compose a connectivity cable to plug the device since in a network communication only two of them are really used → anyway this requires for Ethernet cable to be connected to something. Only one cabled infrastructure needs to be in place. Transformers are not needed. Not optimal for critical areas (e.g. rural).

3) **Powerline communication**: it is the opposite of PoE. It uses the powerline to carry data. Only one cabled infrastructure needs to be in place. Advantageous when the system to be monitored and controlled is the power grid itself. Not optimal for critical areas.



4) **Batteries** use the charge difference between ions to induce a potential difference.

- **Rechargeable** batteries can run the circuit backwards and move the electrons back (by connecting a stronger generator).
- Charging is never perfect: chemical impurities build up.

If the battery runs out completely, it is often impossible to recharge without direct intervention (which may be *expensive*): the objective is **never to let it run out**!

- Apply statistical data and direct information from the meter to define a plan of proactive on-field intervention if needed → carry out with proper timing to minimize the number of substitutions in the lifetime and the number of dedicated "truck rolls" or manual readings.

Energy is the basic constraint of IoT systems. What can we do to improve battery duration?

- Employ battery technology with low self-discharging properties.
- Minimize the duration of active states of the radio components with suitable scheduling algorithms and MAC protocols (duty cycles, sleep mode, alert/notification, etc.)

- Adopt energy-aware routing.
- Introduce endpoint management protocols that communicate the state of batteries.
- Adopt a “let it be off” approach.
- Recharge batteries with **energy harvesting** (use external sources of energy).

5) **Energy harvesting**: recharge it while it is running. Energy harvesting sensors and actuators are equipped with some kind of generator and can recharge their battery from it. This allows them to stay alive much longer than with a single battery charge, even if they get very little energy.

6) **Solar and wind energy**

- Solar panels are now cheap, and solar energy is renewable and clean.
    - The sensor needs to be exposed (shade can be a problem, underground or indoor sensors will not work).
    - Photovoltaic cells will not generate any power at night (the sensor needs to get through the night on a single charge).
- Another possibility is wind energy, but there are several challenges:
    - The sensor still needs to be exposed (underground or indoor sensors will not work).
    - Wind is intermittent and unpredictable.
    - Small turbines are inefficient.

7) **Thermal and vibration energy**

- Thermal energy is cheap when the sensor has access to temperature gradients:
    - The inside/outside or ground/air gradient can generate energy.
    - The power is usually very small but may be the only option for buried sensors.
    - Need for long distances between terminals.
    - Good for wearables: the difference between body temperature and the air is high.
- Vibration energy can be collected using piezoelectric devices with good efficiency.
    - The constant presence of vibrations energetic enough to power the endpoint is uncertain,
    - expect in some industrial environments.

8) **Wireless Power Transfer (WPT)**

- Sensors can benefit from the power of received transmissions, but the benefits are tiny at longer distances: this is only feasible for very low-power sensors.
    - Inductive coupling: uses electromagnetic fields to transfer energy between coils.
    - Capacitive coupling: transfers energy via electric fields between conductive plates.
    - Uses RF signals to transfer power over longer distances (several meters to kilometers).
    - Uses focused laser beams to transmit power over long distances.

9) **Passive sensors**

We can also use the received transmission to power the circuit directly: RFID tags, biometric passports, and contactless cards work this way.

- Power loss is huge: these only work over short distances.
- The sensor cannot actively transmit or record data, only respond to messages.
- There are privacy issues (anyone can read a tag; the only limit is the SNR).

3. IoT technologies and standards

3.1 Short-range technologies

When we talk about IoT technologies we refer to technologies used to connect sensors (and actuators) to gateways. Remember the network architectures already discussed before: with IoT technologies we will focus on 1.b network architecture, where endpoints are connected to gateways, and they do not need to implement any communication protocol since all the work is carried out by the gateway.

Spectrum is a **limited** resource (interference, collision, etc.), this means necessary measures to ensure efficient and fair use of this resource. The spectrum is controlled and owned by the State, that is responsible for these actions and that must provide geography-based regulations for its usage.

- **Licensed spectrum**: allocation of spectrum resources to private companies (e.g., cellular network operators) upon concession fees (auctions). It provides the best quality of service for end customers.
- **Unlicensed spectrum**: free access to spectrum without any centralized control. It is the most used approach for common IoT sensors. There are anyway other cases where this approach cannot be used → more complex devices.
- **"Reserved" spectrum**: spectrum for public safety, emergency forces, police, military.

Unlicensed spectrum

- Also referred to as Instrumental, Scientific, and Medical (ISM) bands.
- It is available to any number of customers, but with no exclusive rights.
- Target: applications with are not lucrative enough to justify the cost of licence but prove valuable if considered collectively. Notable examples: Wi-Fi, Bluetooth, LoRa, ZigBee.
- Unlicensed does not mean unregulated → there are still some rules to avoid/minimize interference, and to promote fair use of resources.
    - Some regulation mechanisms: ERP limits, duty cycle, etc.
    - Regulations are not necessarily aligned in all the Regions of the world, but there are some levels of superimposition (e.g., 2.4 GHz ISM band for Wi-Fi).

Unlicensed spectrum in Europe

At the European level, spectrum is regulated by the European Commission (EC), Electronic Communications Committee (ECC), and the European Telecommunications Standard Institute (ETSI). Different rules/limitations are defined to avoid single device monopolization, especially:

- **Effective Radiated Power (ERP)**: it evaluates the emissions of a device.
    - It is related to the Effective Isotropic Radiated Power (EIRP), defined as the power that would have to be used on an isotropic antenna to get the same field strength that the tested device produces at the same distance.

- We have that ERP = EIRP – 2.15.

- **Duty Cycle (DC)**: it is defined as the ratio, expressed as a percentage, of the maximum transmitter "on" time over one hour.
  - These limitations apply to every receiver, excluding those with LBT capabilities.
  - 1% DC: the maximum transmission time of any device is 1% every hour (i.e., 36 seconds in an hour).

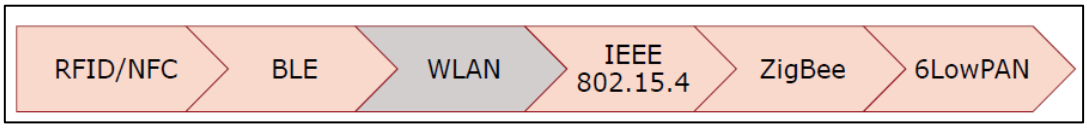
- Some of the most important ISM sub-bands in Europe (focus on IoT use cases).

Range [MHz]	Max. power	Duty cycle	Example	Example IoT scenario
169.4-169.8125	10 mW 500 mW (169.4-169.475)	1% or 0.1% 10% (169.4-169.475)	Wireless M-Bus	Water and gas metering
433.050-434.790	10 mW	10%	RFID, NFC	Metering
863-870	25 mW 500 mW (869.4-869.650)	1% or 0.1% 10% (869.4-869.650)	LoRa, Zigbee	Metering, Smart city, environmental control
2400-2483.5	100 mW	N/A	Bluetooth, Wi-Fi	Smart home
5150-5350	200 mW	Transmission power control	Wi-Fi (802.11)	Smart home
5470-5725	1000 mW	Transmission power control	Wi-Fi (802.11)	Smart home

Short-range: (generally) cell diameter smaller than 100 m.

- Does not support mobility/handover.
- Can be classified according to the reach of the radio link:
  - Vicinity and proximity networks, from a few cm to ~2-3 m.
  - Wireless Personal Area Networks (WPANs), up to ~10 m.
  - Wireless Local Area Networks (WLANs), up to ~100 m.
- Are sometimes referred to as Low-Rate WPAN.

Short-range IoT technologies



3.1.1 RFID and NFC

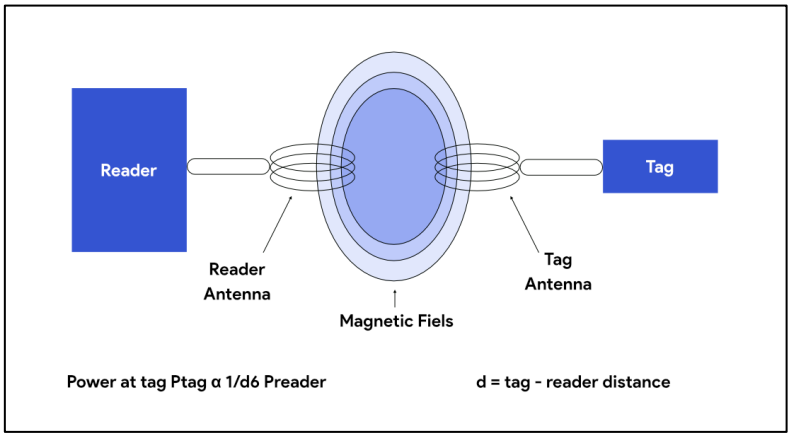
Radio Frequency Identification (RFID) is a method of remotely storing and retrieving data using devices called RFID tags and RFID readers.

- **Tags** store information, that can be retrieved wirelessly in an automated fashion.
- **Readers** can read/write information from/to the tags.
- Many advantages:
  - Data can be stored and retrieved from the tag automatically with a Reader.
  - Tags can be read without line-of-sight restrictions (there is not any constraint on obstacles between readers and tags).
  - Tags can be write-once-read-many (WORM) or rewritable.
  - Other sensors can be combined with RFID.

- Comes with strict critics, e.g., privacy concerns.

How does it work?

- RFID tags are affixed to objects and stored information may be written and rewritten to an embedded chip in the tag.
- Tags can be read remotely when they receive a radio frequency signal from a reader via **inductive (or magnetic) coupling**. Coils in the tag and the reader are coupled together through a magnetic field which is used to give energy to the tag and that energy is consumed by the tag itself to send data to readers → basically tags use the reader energy to send data.
- RFID can operate over a range of distances.



- The coupling occurs in the near-field region.
- Therefore, **the distance between the tag and the reader must be much smaller than the signal wavelength**.
    - Rule of thumb: distance shall be around 15% of the wavelength.
    - It is convenient to operate at low frequency (so the wavelength – and the operational distance – can be larger).
    - At 2.4 GHz (~Wi-Fi), the wavelength is ~0.13 m, so the range would be up to 2 cm.
    - At 13.56 MHz, the wavelength is ~22 m, so the range would be up to 3 m.

Applications

- Readers monitor entering and exiting a closed region.
    - Security (RFID in identification cards and/or passports).
    - Access control (e.g., Telepass).
    - Merchandise in stores.
    - Logistics.
    - Ticketing.
    - NFC in phones.
- Readers tracking an RFID-tagged object.
    - Business process monitoring (RFID tags on pallets).

- Tags marking a spatial location.
    - Sport match/competition.

Applications: Passport

- RFID tags are used in passports.
- RFID tags have been used in new British and American passports since 2006.
- **The tags will store the same information printed in the passport** and will also include a digital photograph of the owner.
- To this day, RFID tags consist of the following elements:
    - A microchip that contains information RFID readers can interpret.
    - A tag antenna to send and receive signals.
    - A substrate to hold all components together.

Applications: Metro card

- Balance is maintained on the card à Cryptographically secured.
- The "reader" updates the balance as you enter/leave the metro station.
    - Enter: record when and where you boarded.
    - Leave: update balance on the card based on the trip.
    - These operations are entirely at the reader.
- Readers record all trips and send updates to a server about the balance of cards.
    - Auditing trail, lost cards, etc.
    - Riders can check their balance online.

Applications: Merchandise in stores

- **Data storage**: store more data than barcode labels (can be changed at any time).
- **Improved quality**: identify returned or counterfeit products by tracking goods from manufacture to storage.
- **Efficiency and speed**: automatically read many tags from a distance.
- **Better stock control**: locate product in a shop's stock. It allows the inventory to be taken as and when sales or returns.
- **Durability**: resistant to handling and external aggression.
- Tags and readers can serve for **anti-shoplifting** and **authenticity verification**.
    - If you walk through a shop doorway without paying for something, the radio waves from the transmitter (hidden in the door gates) are picked up by the coiled metal antenna in the tag. This generates a tiny electrical current that makes the label transmit a new radio signal of its own at a very specific frequency. The receiver (also hidden in the door gate) picks up the radio signal that the tag transmits and sounds the alarm.
    - Why are checkout alarm gates so high? As much the wavelength is larger, larger will be the covered area between readers and tags → so we are basically placing

"big antennas" on shop entrance to emit larger wavelengths, increasing the probability to capture the tag signal.

- **Theft protection**: the RFID chips are automatically deactivated at the checkout and thus reduce the risk of theft or human error.
    - Checkout assistant passes your item over or through a deactivating device. This destroys or deactivates the electronic components in the tag, so they no longer pick up or transmit a signal when you walk through the gates.

RFID vs Barcode

Feature	RFID	Barcode
Rate	Thousands	One at a time
Range	Up to a few meters	~6 meters
Speed	Very fast (ms)	Slow
Line-of-sight	Not necessary	Necessary
Identification	Can **uniquely** identify items	Only identifies the type of item

RFID standard: Electronic Product Code (EPC)

- In 2003 EPCGlobal was formed to promote RFID standards.
- It defined a standard for the Electronic Product Code (EPC), and standards for coding and modulation.
- Designed to be unique across all physical objects in the world, over all time, and across all categories of objects.
- Intended for use by business applications that need to track all diverse physical objects.
- **EPC data is stored on the RFID tag** (can be accessed using reader).
- Locate EPC Information Services (EPCIS), using Web Services like SOAP and WSDL.

Types of RFID tag

- **Passive**: rely on an external energy source to transmit.
    - In the form of a reader that transmits energy.
    - Relative short range.
    - Very cheap – used everywhere today!
- **Semi-passive**: have their own battery, but it is only activated when detecting the reader's field (expected lifetime duration: 3-5 years). Anyway, if the sensor is battery powered it is not convenient to use RFID technology because we are losing the main advantage (inductive coupling).
- **Active**: have a battery to transmit.
    - Has longer transmission range.
    - Can initiate transmissions and transmit more information (it is more like a sensor).

Passive RFID tag

- **Operational frequency**
    - *Low Frequency (LF)*: tag/readers at 120-140 kHZ.
    - *High Frequency (HF)*: tag/readers in the 13.56 MHz ISM band.
    - *Ultra-High Frequency (UHF)*: tag/readers at {433, 860, 960} MHz, based on the region.
    - *"Microwave"*: tag/readers at 2.4 and 5.8 GHz, and millimeter wave.
    - Note: all these frequencies are unlicensed.
- **Types of tag**
    - *Vicinity*: ISO/IEC 15693 standard, with distance up to 1-1.5 m.
    - *Proximity*: ISO/IEC14443, with distance up to 2-10 cm.
        - Inherent robustness against interception, interference, and unwanted readings.
        - Ticketing, access control, contactless payment, ...
- Transmission consists of a stream of bits (plus CRC). CRC allows reader to verify the value it reads.

Passive RFID tag: PHY Layer

- Different modulations used by reader and tag.
- Tags are cheap and dumb, so only **changes in amplitude of the reader signal** can be used (advanced modulations like PSK or QAM are not available).
- Passive tag modulations differ from typical radio communication schemes because the reader signal also powers the tag, so it is **useful to have the signal be at its maximum value most of the time**. The transmission power of readers needs to be maximized, otherwise tags may not receive enough power to work properly (no inductive coupling).

1. The reader sends a series of sync pulses to help the tags set their clock.
2. A binary '0' is transmitted by turning the reader power down or off for a brief time, after which the power is turned back up for the rest of the symbol.

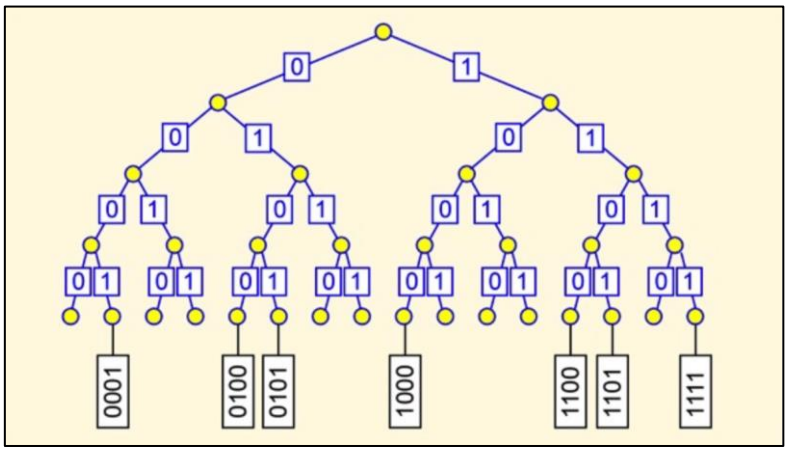
3. A binary '1' is transmitted by turning the power down for a longer time.
4. The “null” symbol is used to signal tags to change their state.

Passive RFID tag: MAC Layer

- MAC for tags is a challenge (very high concentrations of tags).
- If only one reader is present: no need for MAC to handle interference with other readers.
- In case of multiple readers:
  - Tags are simple and cannot run sophisticated protocols (carrier sense, RTS/CTS).
  - May also deal with multiple readers operating in the same environment.

Two types of schemes used (standard):

1) **ALOHA**: tags transmit with a random backoff but if there are a lot of tags the probability to have a conflict exponentially increase with the number of tags in the network.
2) **Binary tree resolution**: reader explores a tree of tag values (every tag is a leaf node).
   - Send requests to tags with IDs that start with a certain string.
   - Narrow down search until one tag responds.
   - At each step:
     - If no tags respond: skip the subtree (it does not contain any tag).
     - If multiple tags respond: continue the breadth first search. When more than one tag responds, the reader knows that there is a collision in that part of the tree (prefix group), and it must continue exploring this subtree to resolve the collision. This involves further splitting the group by adding more digits to the prefix and querying again.
     - If one tag responds: you have found a tag! In case of collisions this is the result of exploring a subtree, reaching the leaf node which corresponds to this tag.
     - For example, in the tree shown below, the reader might go down 0001 but wouldn't bother with 001… because no tag responded with a '1' at that bit.



Possible problem: the structure of three may change over time due to tags that go out of range, so we should rebuild the three in real-time to have a more realistic structure BUT, as said before, RFID tags are simple and so this kind of sophisticated mechanisms cannot be implemented and it is not convenient → if you want to reach that kind of performance you should choose another technology.

NFC

Near Field Communication (NFC) is a device that combines the functionality of an RFID reader and a tag.

- Integral part of mobile devices (e.g., mobile phones reading tickets / bank cards).
- Reader mode: NFC devices can access data from an object with an embedded RFID tag.
- Peer-to-peer: allows two-way communication between NFC devices, can act as smart tag.
- Passive communication: one device acts as a reader and the other as a tag.
- Active communication: both devices alternatively act as readers.
- Since NFC devices can read and write, they must check for collisions.
- NFC operates at 13.56 MHz (HF) and is compatible to international standards.

RFID vs NFC

Feature	RFID	NFC
Network type	Point-to-point	Point-to-point
Communication	Unidirectional	Bidirectional (two-way)
Range	Up to 100 m	<0.2 m
Frequency	LF/HF/UHF/Microwave	HF (13.56 MHz)
Bitrate	Varies with frequency	Up to 424 KHz
Power consumption	Varies with frequency	<15 mA
Continuous sampling	No	Yes

Note: the RFID operating range is larger since with RFID we can increase/decrease the wavelength to increase/decrease the communication range between tags and readers while with NFC we are forced to operate using only the 13.56 MHz frequency.

3.1.2 Bluetooth Low Energy (BLE)

Bluetooth is a wireless technology designed to connect devices of different functions such as telephones, notebooks, computers (desktop and laptop), cameras, printers, when they are at a short distance from each other. Originally thought as a solution to replace wired connections within computers. Standardized as IEEE 802.15.1.

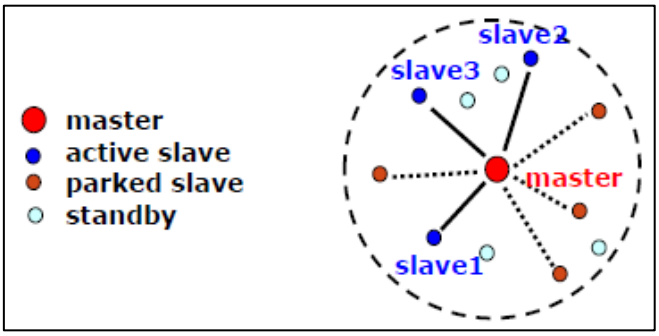
There are 3 classes of Bluetooth:

1. **Bluetooth Classic (Basic Rate / Enhanced Data Rate – BR/EDR)**
    - Audio streaming (headphones, speakers, car audio systems, ...).
    - Peripheral devices (keyboards, mice, printers, ...).
    - File transfers (sending files between phones, computers, and other devices).
2. Bluetooth High Speed
    - (High-resolution) video streaming.
    - Tethering (for sharing internet connections between devices).
3. **Bluetooth Low Energy**
    - Wearables (use Bluetooth to connect to the smartphone/gateway).
    - Beacons (utilized in proximity marketing and location-based service).
    - Smart Home devices.

BR/EDR

Bluetooth classic shape networks using a **star topology**: units connect in small networks called **piconets** (that can be further combined to form what is called a **scatternet**).

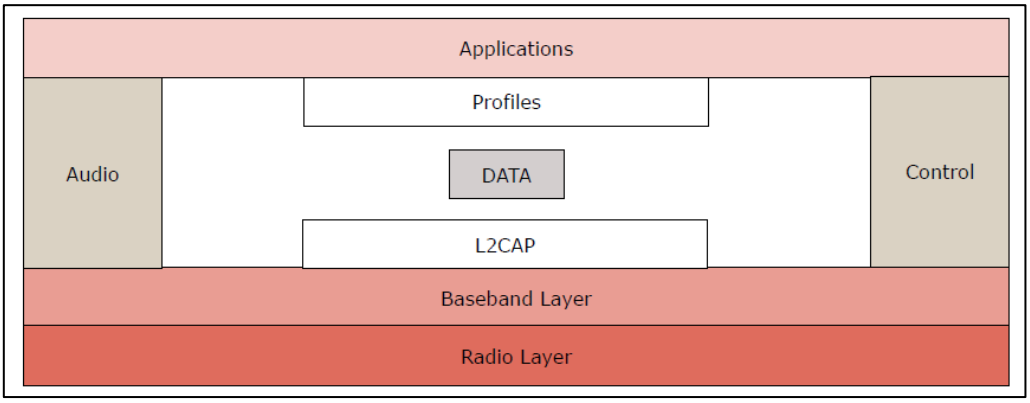
- Each piconet can host from 2 to 8 **active** devices (maximum 8 active sessions).
- Up to 255 in sleep (parked) state: synchronized but cannot take part in communication until it is moved to the active state.
- One unit acts as Master, the others as Slaves.
- Master manages the channel access by using a polling algorithm.



Frame

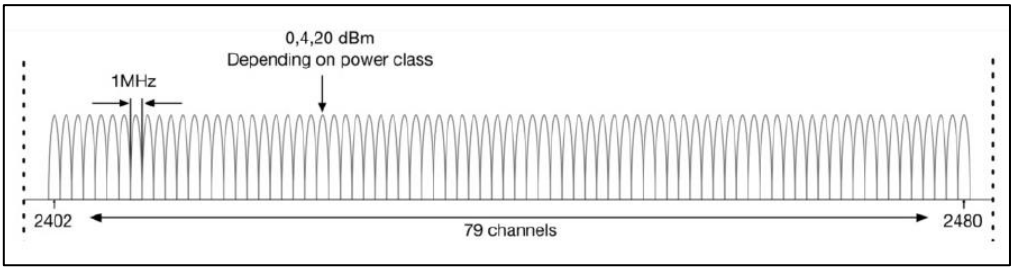
- **Access code**: synchronization bits and piconet ID (sync between master and slaves).
- **Header**: 18-bit pattern repeated 3 times.
    - Address (3 bits for 23-1=7 possible destinations in the piconet).
    - Type of message.
    - F: Flow (=1 is receiver cannot accept packets).
    - A: Acknowledgment (Stop and Wait, 1 bit is enough).
    - S: Sequence number (Stop and Wait, 1 bit is enough).
    - HEC: Header Error Correction.

BR/EDR Layers



Radio Layer

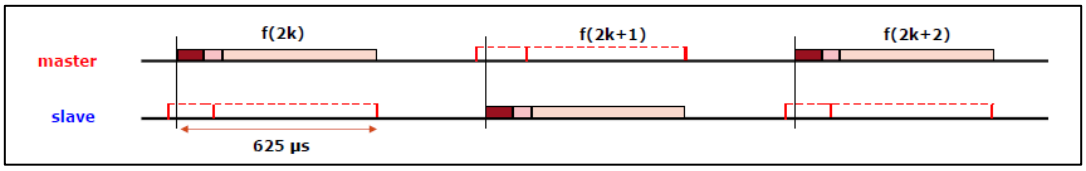
- Roughly equivalent to the PHY layer in LANs.
- Three different modulation schemes:
  1) Gaussian Frequency Shift Keying (**GFSK**) with Gaussian Filtering: 1 Mbit/s.
  2) **π/4-DQPSK**: 2 Mbit/s.
  3) **8-DQPSK**: 3 Mbit/s.
- Operates in the **2.4 GHz ISM band**, divided into **79 channels of 1 MHz** (bandwidth) each.
- The range depends on the transmission power (up to ~100 m).
- Frequency-Hopping Spread Spectrum (FHSS): Bluetooth hops 1600 times/s (times Bluetooth changes channel among the 79 available; hopping among different frequencies constitutes the robustness of this technology) → each frequency used for only 625 μs (dwell time – slot time) before hopping to another frequency. This is also to reduce interference → ISM bands are very crowded since they are unlicensed frequencies. The basic idea is to hop to another frequency that may be less crowded.
  - Problem: where/how to hop? How can two devices know what is the next frequency to hop to? Bluetooth needs **coordination** and **link establishment**.



Baseband Layer

- Roughly equivalent to the MAC layer in LANs.
- **Time Division Duplex (TDD)**, with slots of 625 μs.
  - Bi-directional data transmission is realized by alternating slots in the two directions.
  - Master to slave (**EVEN** slots) or slave to master (**ODD** slots) → no transmission overlapping master-to-slave and vice versa.

- Each transmission occurs on a different 1-MHz RF channel, according to a frequency-hopping pattern that is different for each piconet (this is similar to walkie-talkies using different carrier frequencies) → WHERE and HOW?



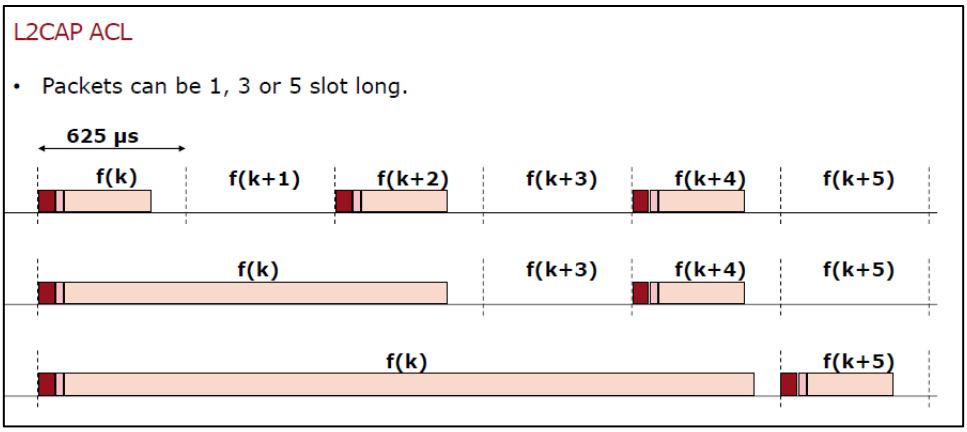
Baseband Layer: Connection procedure

- We need to agree on the list of hopping frequencies to avoid interference and collisions.
- **PHASE 1: Inquiry**
  - Master initializes the communication link.
  - Master establishes the pseudo-random hopping sequence for the piconet.
  - Terminals go in sleep mode.
- **PHASE 2: Paging (it may take a lot of time)**
  - The Master pages another Slave.
  - The Slave sends a reply to the source (Device Access Code (DAC)).
  - The Master sends the list of future planned hops of frequency.
  - The Slave sends a second DAC to the source.

L2CAP

Logical Link Control and Adaptation Protocol (L2CAP) is used for multiplexing, segmentation, reassembly, QoS and group management → network management. It supports two types of communication:

1. **Synchronous Connection Oriented (SCO)** (latency more important than integrity).
   - Synchronous, symmetric, connection-oriented service. Every slave has a dedicated connection to the master and only that slave is authorized to use that link → resource-dedicated communication.
   - MAC is deterministic: slots reserved to voice traffic at regular time intervals.
   - Used for real-time applications: if a packet is lost retransmission is not possible → here we are prioritizing latency instead of integrity.
   - Examples: audio, video.
2. **Asynchronous Connection-Less (ACL)** (integrity is more important than latency).
   - Packet oriented, asymmetric, asynchronous.
   - If a payload encapsulated in the frame is corrupted, it is retransmitted.
   - Packets can be 1, 3 or 5 slot long.
   - Carrier frequency does not change during the transmission in a multislot.
   - Multislot packets reduce overhead due to header and guard time (~259 µs) → overhead < 10% vs the previous solution that leads to an overhead of 40%. The drawback of this approach is that if we lose a packet, we need to retransmit the entire slot, and this takes more time (integrity > latency).
   - Example: data transmission.



Profiles

- Profiles: different application-specific protocol stacks.
- Nearly 40 profiles.
- Some profiles have a quite narrow application score (e.g., headset profile).
- Some profiles support more flexible and general applications (e.g., PAN).
- The networking capability of BR/EDR depends on the profile.

BLE

Bluetooth Low Energy (BLE) is a more recent extension meant specifically for sensors or low-power IoT devices. There are some differences with BR/EDR:

- **Fewer channels**: 40 (2 MHz) instead of 79 (1 MHz).
  - 3 (out of 40) advertisement channels for special use.
- **GPSK modulation only** (other types consume too much energy).
- **Data rate up to 2 Mbit/s** (vs. 1 Mbit/s in GPSK-BR/EDR).
  - Higher data rate results in shorter transmission times → **less energy consumption**.
- Lower power consumption.
- Different codes for error protection.
- Dwell time for frequency hopping is only determined during connection establishment.

Communication modes

In BLE the nodes can assume 4 roles (vs. 2 in BR/EDR):

1. **Broadcaster**: a node which periodically transmits advertisements but does not allow connections to be established (e.g., iBeacon).
2. **Observer**: a node that just listens for advertisements and does not attempt to open connections (e.g., smartphone with an active localization App).
3. **Peripheral**: a node which transmits advertisements and may accept connection requests, acting as a **Slave**.
4. **Central**: a node which may open connection towards a peripheral, acting as the **Master** once the information relative to a peripheral has been received through advertisements.

New profiles

- New profiles: Heart Rate, Internet Protocol Support Profile, **Mesh Profile**...
- **Mesh Profile**: it allows the creation and management of mesh networks.
  - Some nodes may act as relays.
  - **Low Power Nodes**: sleep for some time and wake up periodically to talk with Friend Nodes nearby, which store the message that they could not receive while sleeping.
  - **Managed flooding mechanisms** to avoid loops.

BR/EDR vs BLE

Feature	BR/EDR	BLE
Power consumption	Higher	Lower
Data rate	Up to 3 Mbps (8-DPSK)	Up to 2 Mbps (GFSK)
Latency	Lower	Higher
Connection Type	Point-to-point	Point-to-point, broadcast, mesh
Use case	Audio, peripherals, ...	IoT, wearables, sensors. ...
Compatibility	Classic Bluetooth devices	Bluetooth 4.0 and newer

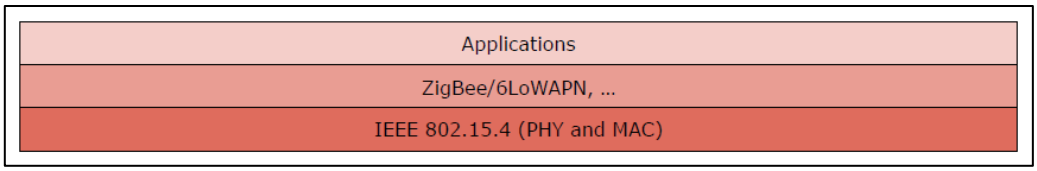
3.1.3 IEEE 802.15.4 technologies: ZigBee and 6LowPan

The project IEEE 802.x has been promoted by a consortium of companies under the support of **IEEE (Institute for Electrical and Electronics Engineers)**. IEEE 802.x represents the general family of all wireless network technologies, from the most popular to IoT specific ones.

- ISO (International Organization for Standardization) and IEC (International Electrotechnical Commission) identify every IEEE 802.n standard under the designation ISO/IEC 8802.n
- Why 802? The number 802 was simply the next free number IEEE could assign, though "802" is also associated with the date the first meeting was held: February 1980 → 80/2

IEEE 802.15.4: **Low-Rate Wireless Personal Area Network** (LR-WPANs or LoWPAN – IoT specific ones). Communication between personal devices and gateways is usually provided by this kind of networks.

- **It defines the PHY and MAC sublayers** of LR-WPANs. All the other layers are defined by the specific employed technology.
- Pros: easy installation, reliable, extremely low cost, reasonable battery life, relaxed throughput.
- Support of a large number of nodes (unlike BLE).
- Support for asynchronous communication and transmissions of small data packets.
- Used by **ZigBee**, **6LoWPAN**, and others.



IEEE 802.15.4

Operational frequency

Min. power to decode a message. For 2.4 GHz, it is harder to decode.

Band	868.3 MHz	915 MHz	2.4 GHz
Channels	1	10	16
Bandwidht/channel	600 KHz	2 MHz	2 MHz
Bitrate	20÷250 Kbit/s	40÷250 Kbit/s	250 Kbit/s
Sensitivity	-92 dBm		**-85 dBm**
Availability	Europe	America	Worldwide
Transmission power	<1 mW (low-power)		
Range	10-75 m (typically 30 m)		
Interference	Few		Severe (very croweded)

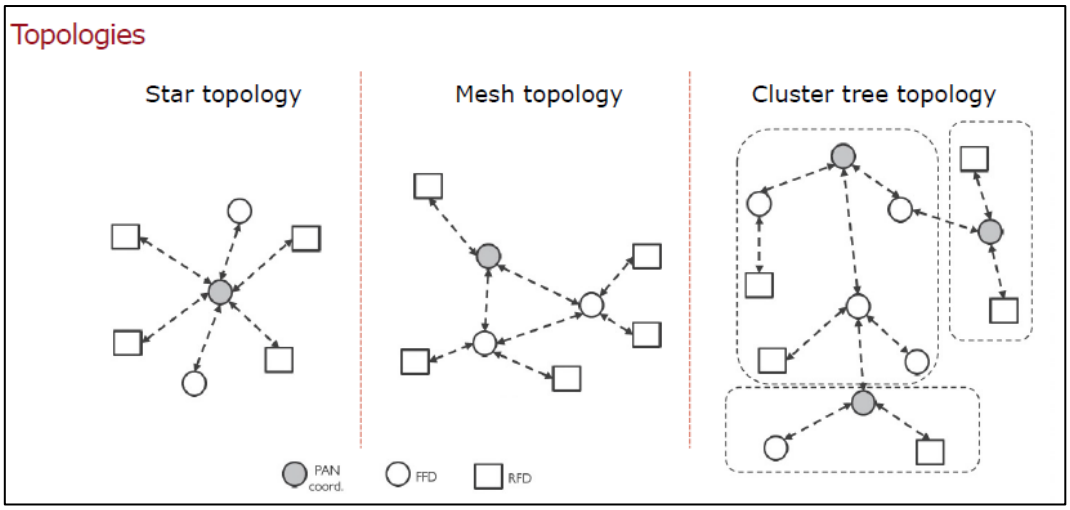
Note: IEEE 802.15.4 uses only **unlicensed spectrum** frequencies.

Note on *sensitivity*: it corresponds to the minimum channel quality to guarantee the signal can be received/detected. As we increase the frequency the propagation is worst and the signal is more difficult to detect, anyway having a higher frequency means having a higher data rate → as always, we need to find a balance.

IEEE 802.15.4 defines two different types of nodes in the network:

1. **Full Function Device (FFD)**
   - Full PHY and MAC capabilities.
   - Can relay messages.
   - Can act as a **PAN coordinator**, i.e., it can manage operations in the PAN network. A sort of gateway, it can act as central node (master if we want to find an analogy).
   - Any topology.
2. **Reduced Function Device (RFD)**
   - Simpler implementation with reduced functions.
   - Can only talk to an FFD and cannot become PAN coordinator. “A sort of slave”.
   - Cannot relay data (can only act as end nodes).
   - Limited to star topology.

⇨ How many nodes for each type should we have? If FFD number is too low a lot of RFD may be managed by the same FFD and this can lead to more traffic to handle and possible network congestion, again we need to find a balance.

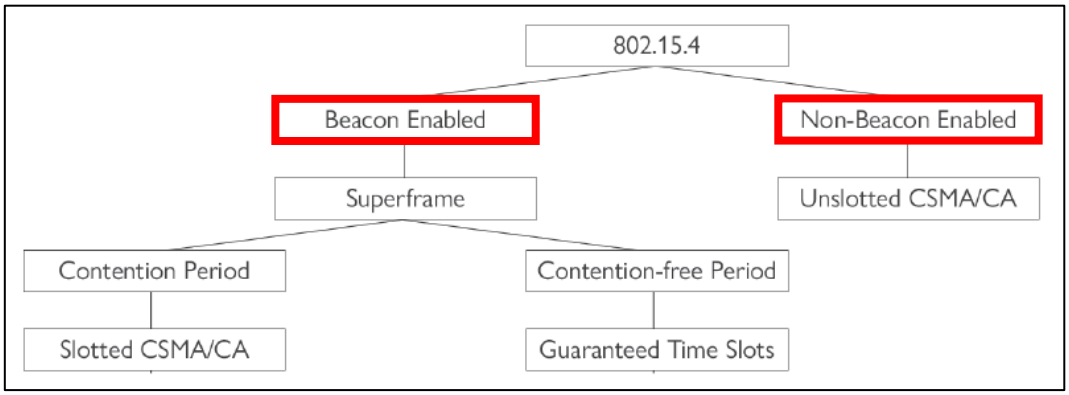


Cluster tree topology: very popular in IoT networks. In fact, routing protocols are fundamental to find a path to two end nodes and it is more convenient, as we will see later, to have a tree structure topology, as the one provided by the cluster tree approach, instead of a star or mesh.

Frame structure

- Preamble (32 bits): for synchronization.
- Start of Packet Delimiter (8 bits): a simple flag that highlights the start of the frame.
- PHY Header (8 bits): PSDU length.
- PSDU (0 to 1016 bits): Data field.
- Note the simple structure of this frame: as we said many times IoT networks are energy constrained networks and we do not want to create complex protocols/structures, we transmit a minimum number of bits so more sophisticated structures are not so convenient in these scenarios.

MAC Layer



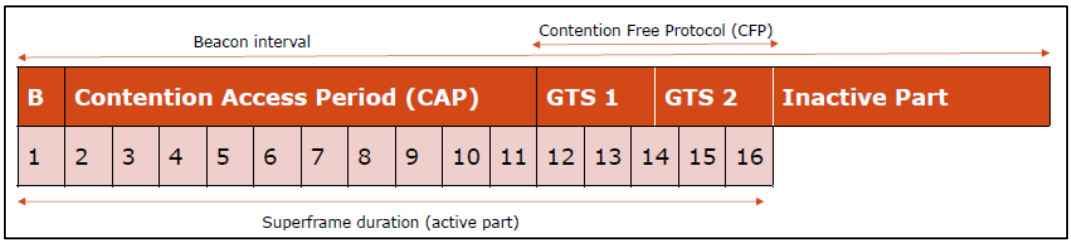
First MAC implementation: Beacon-enabled mode

PAN coordinator periodically exchanges **beacon frames (B)** to handle communications.

- B field is sent periodically to provide synchronization in the IoT network. As long as a device receives the beacon, the communication can start. If the beacon is lost the device needs to wait for the next packet and so for the next beacon B to start the

communication. Beacon is relayed automatically, there is no need to ask for it because it is transmitted periodically.

- Provide superframe transmission period (for synchronization).
- Nodes are discovered by listening to beacon frames.
- Time is divided into **beacon intervals**, that start with a beacon frame B.



Beacon interval consists of an **active part**, organized as a **superframe.**

- Data frames can be sent.
- Divided into 16 time slots, each of which carries 60 symbols.
- Divided into 2 parts:
    1) **Contention Access Period (CAP)**: data transmission using CSMA/CA for the channel access.
    2) **Contention Free Protocol (CFP)** ***(optional)***: time slots are allocated to traffic with specified throughput and latency requirements. The PAN coordinator allocates a node with such requirements a **Guaranteed Time Slot (GTS)**, which consists of a set of slots. This node will have dedicated resources, so there is no risk of contention.

CAP vs CFP

Within CAP data exchange is managed by CSMA/CA routing protocol, there is no coordination with master nodes and, due to that, messages collisions may happen. CSMA/CA is a random-access protocol designed to avoid collisions, even if there is anyway a small probability of collisions.
Using instead CFP, communication of some specific nodes is scheduled, the coordinator decides who is/are the slave/s that can communicate to the master and this obviously without risk for collisions. In the above frame example: GTS1 defines the slots on which only node1 can communicate and GTS2 defines the slots on which only node2 can communicate.
In GTS, nodes allowed to transmit are decided a priori → if we use only CAP every node can communicate. It is fundamental to find a trade-off between the two (remember: less CAP slots less data a node can transmit).

After the active part there is an **inactive part**, where nodes are asleep.

- The ratio between the superframe duration and the beacon interval is the **duty cycle**.
- Duty cycle ranges between 100% and 0.006%.

Second MAC implementation: Non-beacon-enabled mode

In this approach there are no beacon frames: a device shall **explicitly request the transmission of beacon frames** and wait for replies from the PAN coordinator. In fact, this MAC layer approach is much easier to manage but at the same time collisions here may happen since beacon frames transmission are not scheduled.
The channel access is managed by **unslotted CSMA/CA** protocol: nodes can start transmissions at any time rather than at the beginning of a slot that is what happens with slotted CSMA/CA.

The IEEE 802.15.4 MAC layer provides different security services:

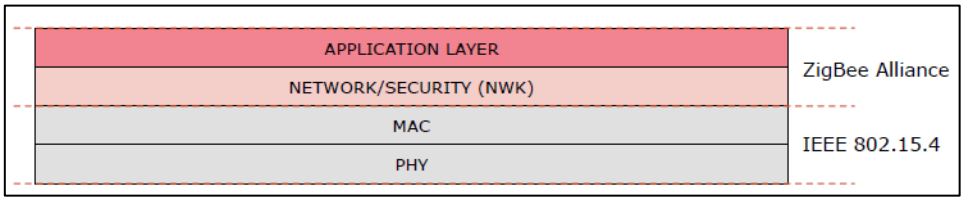
- **Unsecured**: no security mechanisms.
- **Access Control List**: accept/reject frame based on the source node.
- **Secured**: implement many security mechanisms:
  - Access control.
  - Data confidentiality using symmetric encryption.
  - Frame integrity.
  - Sequential freshness protection against frame reply.
  - …

The IEEE 802.15.4 standard underwent several **amendments** (new 802.15.4 releases that incorporate new features). Amendments generally concern the modulation and coding scheme and the introduction of new frequency bands. Some interesting new features/extensions:

- **Timeslotted Channel Hopping (TSCH)**: designed for industrial processes that require real-time response. It works in the non-beacon enabled mode using slot frames.
  - TSCH defines cells, i.e., combinations of slot frames.
  - Dedicated cells can be used by a single transmitter.
  - Shared cells can be use by multiple transmitters, and channel access is ALOHA.
  - Channel hopping using a different cell if previous transmissions failed.
- **Deterministic and Synchronous Multi-channel Extension (DSME)**: implements the use of multi-channels in the same superframe → robustness via diversity.

ZigBee

ZigBee is an industrial standard promoted by the ZigBee Alliance, targeting IoT, built on top of the IEEE 802.15.4 PHY and MAC layers. It defines three layers: **Network**, **Application Support**, **Application Layers**, the ones not implemented by IEEE 802.15.4 family.

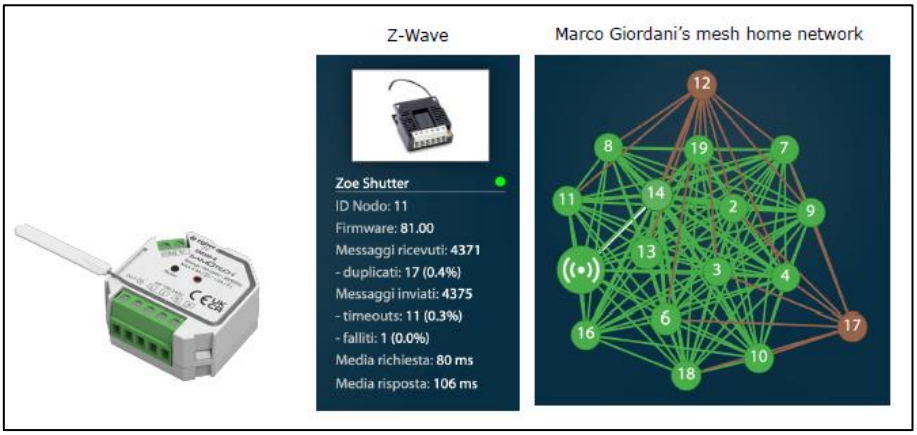


Applications

- Building automation
  - HVAC (heating), AMR (metering), lighting control, access control, garden irrigation, …
- Personal healthcare
  - Patient monitoring, fitness monitoring (some specific gym equipment may be connected to the gym controller via ZigBee).
- Industrial automation
  - Asset management, process control, environmental/energy management
- Consumer electronics
  - TV, VCR, DVD/CD, remote
- PC and peripherals
  - Mouse, keyboard, joystick, … control

Application: Wireless lighting control.

- Light switches anywhere.
- Customizable lighting schemes.
- Energy savings on bright days.

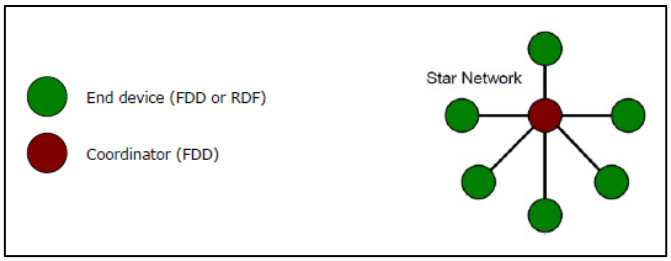


Network layer: functionalities

- **Starting a network**
- **Joining and leaving a network**: gain (join) or relinquish (leave) membership to a node of the network.
- **Configuring a new device**: sufficiently configure the stack for operation as required.
- **Addressing**: assign addresses to devices joining the network.
- **Routing**: rightful node/route discovery and maintenance operations, and routing to destinations.
- **Security**: applying security to outgoing frames and removing security to terminating frames.

Network layer: topology

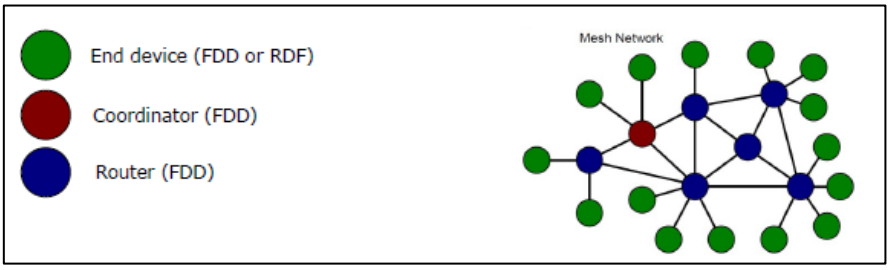
In a star topology, the FFD (coordinator) initiates and maintains communication with other FFD or RFD nodes.



In a mesh topology the network is extended beyond the nodes directly connected to the coordinator using ZigBee **routers** that forward/relay data.

- Routers are FFDs connected to the mains energy, and never go to sleep mode.
- **Spanning-tree**-like structure, with the coordinator at the root. Every node can establish communication links with any other node (not only parents and children).

Router is a new type of FFD node that manage the connection between a certain node and its coordinator (different from the previous star topology) and that's why here routing protocols are needed: we need to find the best path to reach the coordinator node defining some rules and metrics that help in taking a decision. Routers in fact may extend the range of a coordinator.



Network layer: addressing

Each ZigBee device gets two types of identifiers (ID):

- **Personal Area Network Identifier (PAN ID)**: 16-bit ID selected by the PAN Coordinator while setting up the network and communicated to the End Devices.
  - Also known as Short Address.
  - 16 bits → 2^{16} = ~64'000 nodes (>> BLE). Here we can see why ZigBee is a more specific IoT technology (the number of devices can be very huge).
- **Extended PAN ID (EPID)**: 64-bit ID assigned to the device during its production.
  - Must be unique and universal.
  - Also known as MAC address.

⇨ These two kinds of addresses are similar to the concepts of IP address (changes depending on the network) and MAC address (never change). The combination of PAN ID and EPID make ZigBee connections and communications possible.

Network layer: routing

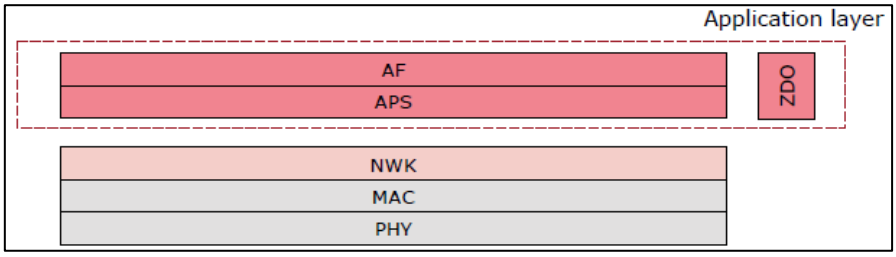
- **Hierarchical**: messages follow the tree established during network creation.
- **Dynamic "best"**: search for the best available path:
  - Based on the **Ad-hoc On-demand Distance Vector (AODV)** protocol (a special kind of Distance Vector Protocol). Routing is on demand, only if a node wants to transmit something the routing protocol is triggered. It seems obvious but in fact there are other IoT technologies that continue to run routing protocols (as 6LowPAN).
  - Routers are discovered on demand, when needed.
  - Algorithm searches for the minimal cost path, computed at the sum of the **link costs**:

$$C\{l\} = min(7, \lfloor 1/p_l^4 \rfloor)$$

p_l = prob. of packet delivery on link l
Depends on the **quality** of the link.

  - The best route is created incrementally from the source.
  - To reduce delay, suboptimal routers are filtered out when propagating.
  - The routing algorithm also considers the **link quality**: if we have two possible path, we select the one with the highest link quality. Recall the mesh network schema here above: I want to reach a coordinator node using the minimum number of "hops", so passing through a minimal number of routers also considering the quality of the link, so we pass via more robust routers. Consider that link quality may change over time and a node1 may reach coordinator1 in different ways at different time intervals.

Application layer



Application layer: APS

It supports several functionalities:

- **Binding**: match 2 devices together based on their services and their needs.
- **Address mapping**: manages the mapping between 64-bit and 16-bit network addresses.
- **Fragmentation and reassembly.**
- **Service Discovery**: allows devices to find and communicate with other devices that offer the services they require.
- **Group Addressing**: enables communication with multiple devices simultaneously for broadcast, e.g., *turning on all lights in a room*.

- **Security**: manages encryption and decryption of data. Note that security in IoT is not provided only in one way: there are different types and layers of security, one of them is provided by ZigBee right here in the application layer.

Application layer: ZDO

It is a special application which employs NWK and APS primitives. It supports several functionalities:

- Defining the **role** of the device: Coordinator, Router, End Device.
- **Device discovery** and **network formation.**
- **Network management**: manages joining and leaving of devices in the network.
- **Security management**: distribution and management of cryptographic keys.
- **Network address management**: it resolves conflicts that may arise from address duplication within the network.
- **APS management**: manages the communication between the application layer and the lower layers of the Zigbee stack through the APS.

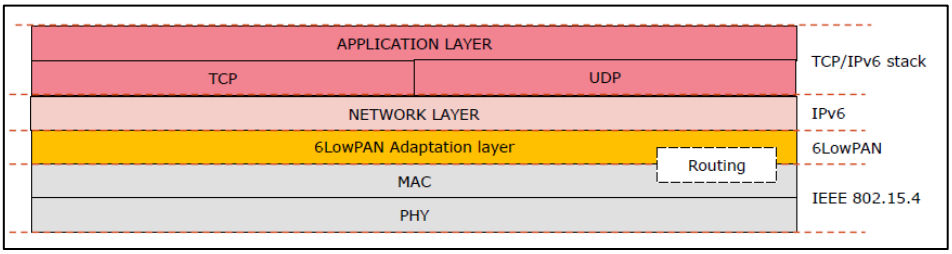
Application layer: AF

It provides application models for applications developers (in fact, engineers that need to implement some ZigBee application typically act here). It supports several functionalities:

- **Hosting** for the application objects.
- Provides some **application models** for the actual application of ZigBee technology to facilitate the development and applications from different manufacturers.
- ZDO Functions used via the Public Interface.
- Control and management of the protocol layers in the ZigBee device.
- Initiation of standard network functions.

6LowPAN

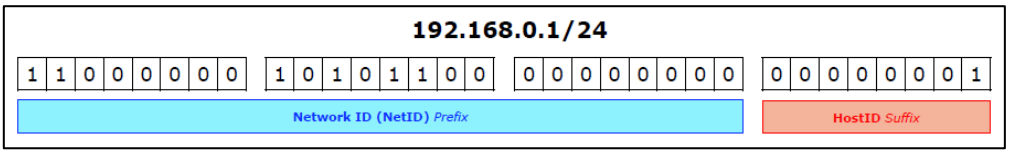
IETF standard aimed at implementing **IPv6 over IEEE 802.15.4** LP-WPAN → idea: transform any device into an endpoint (host) of an IPv6 internetwork.



Note: 6LowPAN implements a sort of adaptation layer that allows to aggregate layers defined by IEEE 802.15.4 family and IPv6.

Let's recall how IPv4 works:

- An IPv4 address is a 32-bit address (2^{32} or 4'294'967'296 addresses → enough for the internet environment).
- IPv4 addresses are **unique** in the sense that each address defines one, and only one, connection to the Internet.
- IPv4 addresses are **universal** in the sense that the addressing system must be accepted by any host connecting to the Internet.
- IPv4 can be **public** (i.e., routable over the Internet) or **private** (special IP addresses) → there can be two users that use the same private IP address as long as both users belong to two different private networks. NAT: the way we can convert private to public IP addresses to increase the number of IP addresses that can be used in internet.
- IPv4 has two parts: **NetID** (defines the name of the network) and **HostID** (defines the name of the device contained within the network defined by Network ID).



IPv4 allows to reach ~4B addresses. Anyway, with the growth of the Internet, it was clear that a **larger address space** was needed as a long-term solution. How to solve this issue?

- Classless addressing.
- Private addressing and NAT.
- **IPv6**.

IPv6

- 128 (> 32) bits: → 2^{128} (1 Undecillion = 1036) possible combinations.
- 8 groups of 16-bit **hexadecimal** numbers separated by ":"

3FFE : 085B : 1F1F : 0000 : 0000 : 0000 : 00A9 : 1234

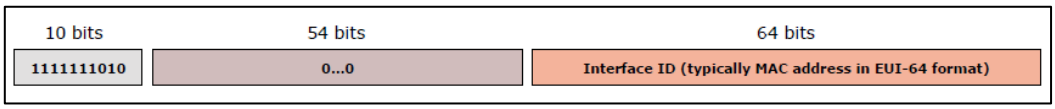
- Leading zeros can be removed: 3FFE:85B:1F1F::A9:1234
- Only CIDR notation for the netmask.
- Three types of IPv6 addresses: **Unicast**, **Multicast**, **Anycast**.
- **Broadcast is not supported.**
- A network interface can have **multiple addresses**: Link-local, Site-local, Global.

IPv6 addresses

1) **Link-Local Address**

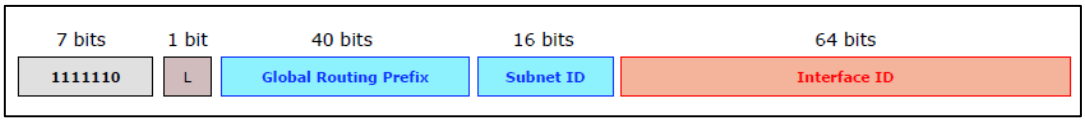
- Starts with a prefix **FE80::/10** (first 10 bits equal to 1111 1110 10) and proceeds with 54 zeros, the only thing that changes is the last part formed by 64 bits.
- Contains the MAC address of the node in the EUI-64 format.

- **Can be used to reach nodes attached to the same link without a globally unique address** (this means we cannot use it to communicate with a node of another network). If we connect several IPv6-enabled nodes to a switch, they will **auto-configure** their interfaces with link-local addresses, will discover each other, and be able to communicate.
- IPv6 routers must not forward packets having link-local source/destination address.
- All IPv6 enabled interfaces have a link-local unicast address.



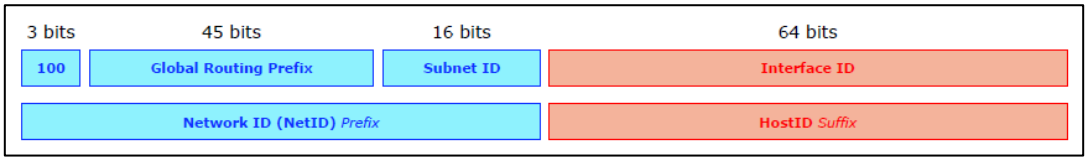
2) **Unique Local Address (ULA)**

- Starts with a prefix **FC00::/7** (first 7 bits equal to 1111 110).
- It is an Internet Service Provider independent address space. Therefore, these addresses won't overlap with any other ISP assigned range.
    - It allows sites to be interconnected without creating any address conflict.
    - **Similar properties as IPv4 private addresses**. Obviously, this address is not usable for communication over public internet, to do that NAT is required to translate the private address to a public one.

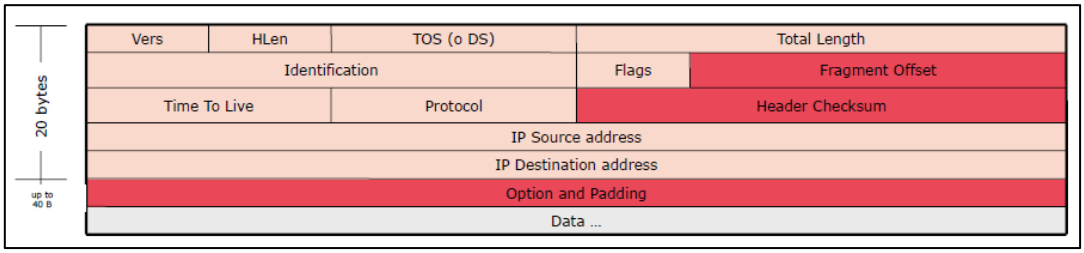


3) **Global Unicast Address**

- Starts with a prefix 2000::/3 (first 3 bits equal to 100).
- Can be used to route IP datagrams over the Internet → the IPv6 counterpart of a common IPv4 address.
- The structure consists of a 48-bit global routing prefix and a 16-bit subnet ID also referred to as **Site-Level Aggregator (SLA)**.
- Variable prefix, defined from router advertisements. Some IP addresses can be reserved.
- The final 64 bits may be chosen randomly (instead of the MAC address), this is done to preserve the device privacy. Using random bits means having a small probability that two nodes in the network choose the same 64-bits sequence, having so the same final address. Even if this probability is low, some protocols that check whether the random sequences are equal are needed.



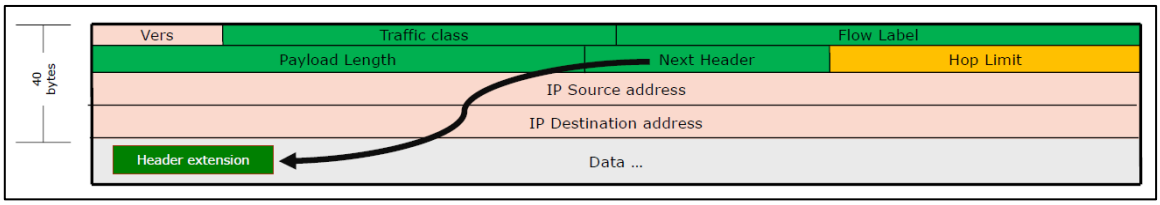
IPv4 header



The above figure shows the fields that have been removed from IPv4 header to implement IPv6:

- **Checksum**: replicated in MAC and TSP header, not needed at the IP layer.
- **Fragmentation**: it is performed by end points and may not be supported by routers.
- **Options**: replaced by pointer to **Next Header Extension** in IPv6

IPv6 header

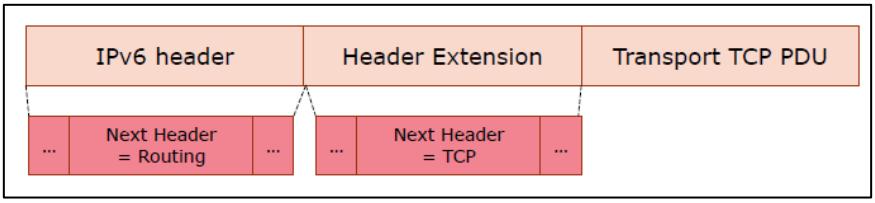


Fields that are added from IPv4 to IPv6

- **Traffic class (8-bit)**: identify possible QoS requirements.
- **Flow label (20-bit)**: identify a source-destination traffic flow.
- **Payload length (16-bit)**: length in bytes of data including any IPv6 Extension Headers.
- **Next header (8-bit)**: specifies the type of the **Header Extension**. If there is no Header Extension, the next header field points directly to data field.
- **Hop Limit** (= TTL in IPv4).

Header Extension: ≥0 extension headers

- The **Next Header** field in the IPv6 header indicates the next Header Extension.
- In each Header Extension there is a Next Header field that indicates the next Extension header.
- The last Extension Header indicates the upper layer protocol (such as TCP, UDP, or ICMPv6) contained within the upper layer protocol data unit.
- It replaces the IPv4 Options fields.
- Header Extension may contain specific information, for instance, about routing (e.g., if we want two nodes follow a predefined route to communicate each other, it is possible to write this in the header extension), hop-by-hop options, etc.



IPv6 assignment: 3 possible ways

1) Manual configuration (similar to "ifconfig" for IPv4).
2) Stateful configuration using **DHCPv6 protocol**.
3) Stateless autoconfiguration without DHCP → IPv6 nodes can connect to a network and **automatically generate a Global IPv6 address** with no dedicated server. IPv6 address can be simply autogenerated, in fact if we look at global unicast address scheme the MAC address is already known and so the only part to generate is the blue one that depends on the network itself.
    1) The node auto-configures itself with a Link-Local Address. This address is only valid for communication within the local network segment.
    2) The node performs **Duplicate Address Detection (DAD)** to ensure the address is not already in use by another device on the same link.
    3) The node sends a **Router Solicitation** (on IP FF02::2) to trigger the Routing Advertisement.
    4) The router responds with a **Router Advertisement** which contains the Global Prefix Information (prefix address, length, and default gateway) → the blue part of global unicast address.
    5) The host uses this information to generate an IPv6 address (Global Address) for itself.
    6) The node performs DAD.

Automatic generation of IPv6 addresses does not create any problem of traffic in assigning the IP addresses (DHCP was used in IPv4 to do that but not in an automatic way).

There are several **benefits** of using IPv6 protocols in IoT scenarios:

- Address/manage/access any IoT device from the Internet (no saturation).
- Easily connect to other IP networks without the need for translation gateways or proxies.
- Use well-known socket API for the deployment of the network applications.
- Easily re-use tools for managing, commissioning, and diagnosing IP-based networks.
- Can connect a potentially infinite number of IoT devices using IPv6 addressing.

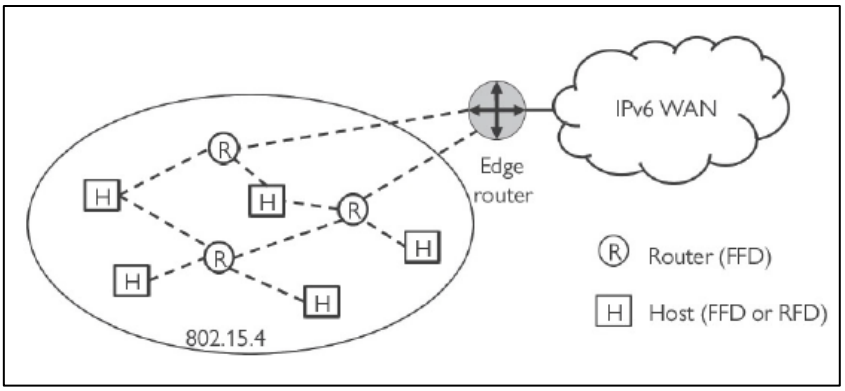
There are also some **disadvantages** of using IPv6 protocols in IoT scenarios:

- IPv6 assumes that a link is a single broadcast domain, while the assumption does not hold in multi-hop wireless sensor networks.
- IPv6 includes optional support for IP security (IPsec), authentication and encryption, but these techniques might be too complex for IoT-devices.
- IPv6 datagrams are not a natural fit for IEEE 802.15.4 networks.
    - MTU (Maximum Transmission Unit) size of an **IEEE 802.15.4 frame is 127 bytes**, while the minimum for **IPv6 is 1280 bytes**.
    - The IPv6 header size (40 bytes) can occupy 1/3 of the MTU, having so only few bites for data. This does not seem to be a real problem since IoT devices do not generate many data, but this is not completely true for all IoT applications.

6LowPAN is most used in **large-scale** IoT deployments, so as to fully exploit **IPv6 network compatibility**.

- Smart lightning system (e.g., Philips Hue).
- Industrial automation (e.g., Bosch XDK development kit).
- Smart agriculture (e.g., Meshlium Gateway from Libelium).
- Healthcare monitoring (e.g., Fitbit Wearable).
- Home automation (e.g., Tado).
- Smart grid enabling smart meters (e.g., Enexis (The Netherlands) uses 6LoWPAN to communicate energy consumption data wirelessly to a central system).

6LowPAN architecture



6LowPAN uses the IEEE 802.14.5 (for the lower layers) and IPv6 architecture (for the network layer). There is no need to define application gateways to interconnect end nodes to the Internet, since standard **IPv6 edge routers** can do the job.

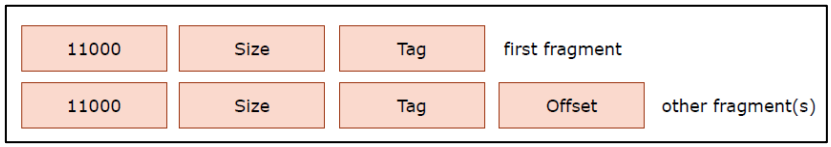
Addressing

- Recall: IPv6 should support 1280-bytes packets, while for IEEE 802.15.4 the maximum payload is only 127 bytes. Considering 40 bytes of header for IPv6, plus at least 8 bytes of UDP, plus some other headers for security protocols, most of the payload is consumed by the overhead.
- To solve this problem, 6LowPAN uses **Extension Headers** to carry optional data.
    - **Header compression:** exploit redundancy (e.g., the payload length can be inferred from the data-link layer header, so no need to replicate this information in the IPv6 header). Another approach is to shorten the length of addresses. Compression can reduce the size of the header from 40+8 to 7-12 bytes.
    - **Fragmentation** (even though it shall be avoided in the first place). In fact, fragmenting means also mix all the fragment together but what if a fragment is lost?

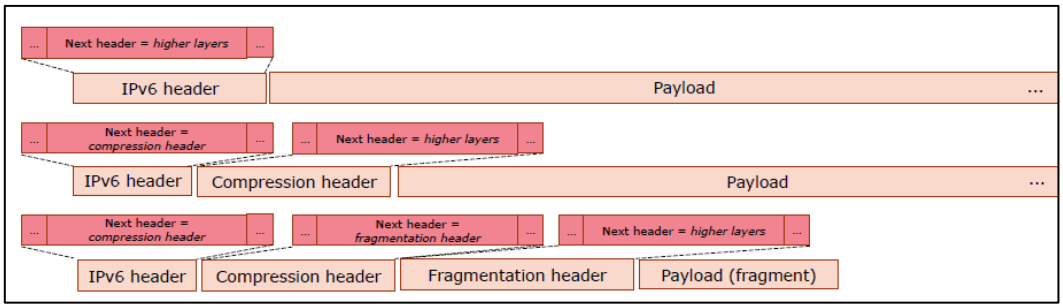
Fragmentation

- **Fragment header**: used in case of packet fragmentation.
- All IPv6 subnetworks have a **minimum MTU of 1280 bytes**.

- Fragmentation in order to fit the size of IEEE 802.15.4 MTU (**127 bytes**).
- Fragment information is carried in the **Fragment Header**. When a new fragment is created a tag field is added to that fragment, this allows to understand which fragments belong to the same resource.
- All fragments carry the same tag value, assigned sequentially by the source of fragmentation.



- 6LowPAN makes used of **stacked headers**.
  - First, we should apply **compression header**.
  - If the size of the packet is still too large, we apply **fragmentation**.

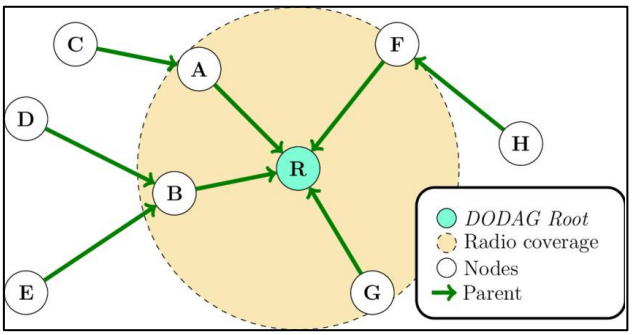


How can we understand if a packet consists of more fragments? We can see it from the *next header* field → if the option is "higher layers" there are no fragments and next header points directly to the payload. If the option is "fragmentation header" there are fragments instead of a single frame. The above figure compares three different cases: standard header; compressed header; compressed and fragmented header.

IPv6 routing protocol (RPL)

- In IoT, the network **topology may change quite rapidly**. Mobility, changing link quality, disconnections, sleep periods (nodes are unreachable), …
- IPv6 routing is resource-consuming, which is not compatible with IoT.
- 6LowPAN requires a **new routing scheme**.
- One solution: **Routing Protocol for Low Power and Lossy Networks (RPL).**
  - De-facto standard routing protocol for IoT applications.
  - It separates packet processing and forwarding from the routing optimization objective.
  - It can be used to disseminate IPv6 or 6LoWPAN specific info (e.g. neighbour discovery).
  - It does not necessarily rely on link-layer protocol.
- The network topology is created **proactively** (not on demand such as in ZigBee), regardless of the specific cost metric being used.

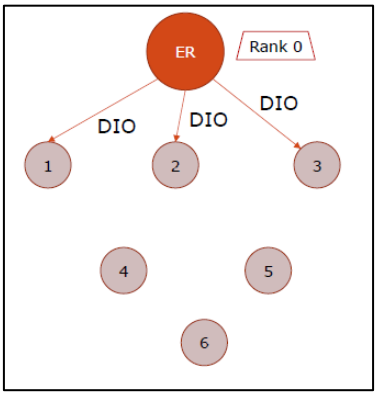
- Routing topology: **Destination-Oriented Directed Acyclic Graph (DODAG)** → directed graph without cycles, oriented towards a root node (the edge router). Note: cycles lead to a higher transmission time and so to a higher energy consuming which is not optimal for IoT networks, so it becomes crucial to not consider them in choosing routes.
- All the nodes have a **rank** which measures the cost to reach the root.
- In case of multiple edge routers, RPL creates multiple disjoint DODAGs. Traffic is distributed across different edge routers.



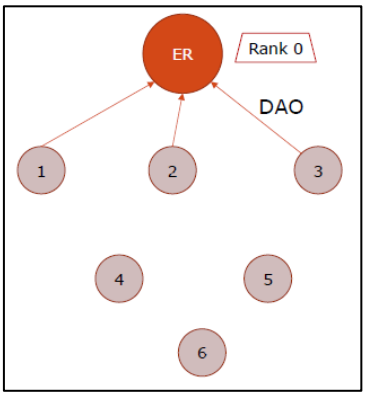
- The rank roughly represents "the node's individual position relative to the other nodes with respect to a DODAG root."
- The rank may depend on some other metrics/constraints, called **Objective Function.**
  - Example: Shortest route (METRIC) by avoiding low-energy nodes (CONSTRAINT).
  - Example: Lowest end-to-end delay (METRIC) by avoiding low-quality links (CONSTRAINT).
- The rank stringently increases in the DODAG's downward direction (root to leaf).
- The rank concept is used to:
  - Detect and avoid loops.
  - Build permament relationships.
  - Provide mechanisms for nodes to differentiate between parents and siblings.
  - Enable nodes to store a set of "preferred parents" to "climb" the DODAG in case one is unavailable.
- To create and maintain the DODAG, the RPL protocol requires each node to send the following control packets, together with their own IPv6 address:
  - **DAO (Destination Advertisement Object)**: establish the downlink path (towards leaf nodes).
  - **DIO (DODAG Information Object)**: establish the upward path (towards roots).
  - **DIS (DODAG Information Solicitation)**: solicitate the transmission of DIO messages.
  - **DAO-ACK (Destination Advertisement Object Acknowledgement).**
- There are two modes of operation:
  - **Storing**: nodes keep a routing entry for all the destinations reachable via the sub-DODAG. Requires intermediate nodes to send complete routing data.
  - **Non-Storing**: the root is the only network node maintaining routing information. In this way, routing always passes through the root.

RPL – Example: creation of the DODAG

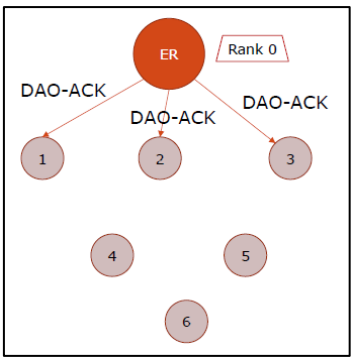
1. The ER (router) creates the DIO message with its rank and ID and sends it in multicast (to the nodes within reach).



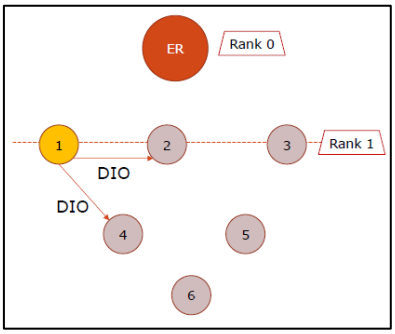
2. The receiving nodes respond with a DAO message in unicast.
   - In the **non-storing mode**, DAO is sent to the root, by using the upward path established through the DIO message. All the intermediate parents (if any) extend the DAO message by adding their IPv6 addresses.
   - In the **storing mode**, DAO is sent to the parent nodes. Each parent maintains additional routing tables for all the nodes of its sub-DODAG.



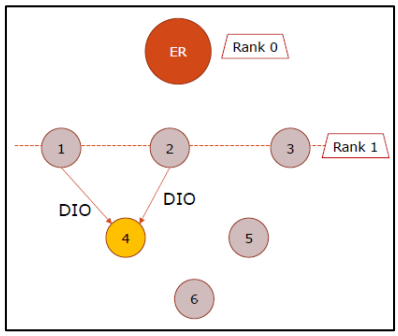
3. The node sends a DAO-ACK to acknowledge the reception of the previous DAO.



4. The receiving nodes establish the upward link toward the ER. and compute the rank value based on the Objective Function.
5. Each node rebroadcasts the DIO message with its own rank following **Trickle algorithm** (it avoids having too many broadcast messages), which strikes a trade-off between reactivity to topology changes and energy efficiency.
    - Trickle ensures that DIOs are advertised aggressively when the network is unstable, and instead at a slow pace when it is stable. The network is stable when the nodes are fixed and the network topology does not change, otherwise is unstable. Depending to that we increase/decrease the DIO message rate.



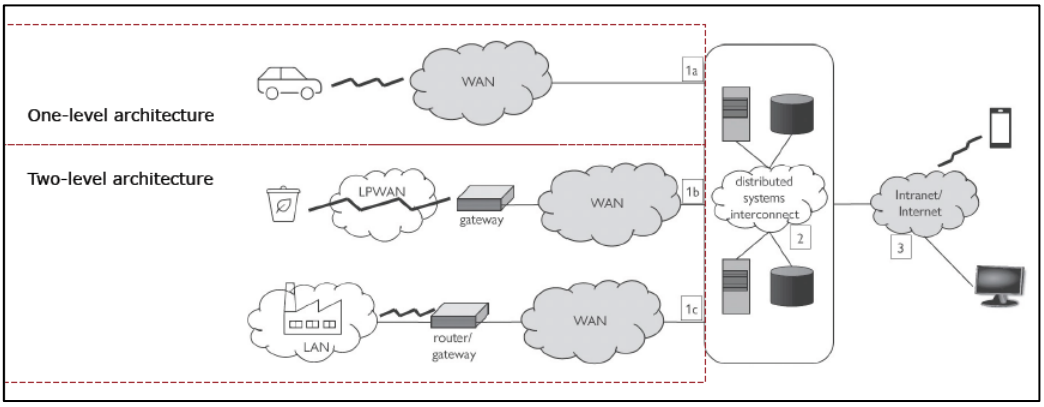
6. If a node receives multiple DIO messages from multiple nodes, a preferred parent is selected based on the lowest rank.
7. If the node has already its rank, and the received one is ≥ than the local rank, the DIO message is discarded (loop avoidance).
8. The process continues iteratively, and the routing procedure ends when reaching the leaf nodes.



Short-range IoT technologies summary

Feature	RFID	BLE	ZigBee	6LowPAN
Frequency	LF/HF/UHF/Microwave	2.4 GHz	868/915 MHz	868/915 MHz
Bitrate	Varies (tpically ≤600 bps)	0.7-2.1 Mbps	250 Kbps	250 Kbps
Range	Up to 100 m	10-15 m	10-1000 m	10-100 m
Modulation	ASK	GFSK, CPFSK, 8-DPSK	D-BPSK, O-QPSK, QPSK	BPSK, O-QPSK, ASK
Topology	P2P	Star	Star, Mesh, Cluster Tree	Star, Mesh, P2P
Network size	Large	8	65536	~100
Bandwidth	Varies	1 MHz	0.3/0.6 MHz; 2-5 MHz	2-5 MHz
Complexity	Simple	Complex	Simple	Medium
Applications	Asset tracking, inventory management	Wireless Sensing, Peripheral	Wireless Sensing, Monitoring	Home, Automation, IoT

3.2 Long-range technologies



Coming back on network architectures, we are dealing with 1.b two-level network architecture. Short-range technologies are involved when the distance between end nodes and gateways is not so large, also remember in that case we assume the bottleneck is on the edge part since we consider the connection provided by the gateway robust. Anyway, when the distance and covered area augment, new long-range technologies are needed.

Long-range: cell diameter of (several) kilometers. Generally, this is enabled by cellular networks (one-level network architecture 1.a):

- High complexity and infrastructure cost.
- Support for very high data rates.
- Not designed for energy efficiency (recharging batteries of mobile terminals is normal).

IoT long-range approach: **Low-Power Wide Area Networks (LPWAN)**: jointly optimize a combination of energy efficiency, data rate, communication range, cost per device.

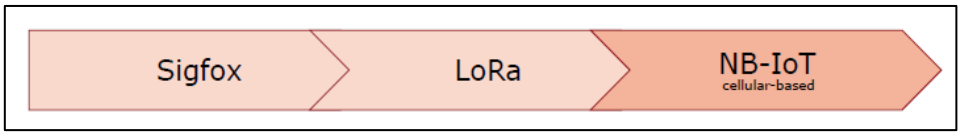
Parameter	WSN	WLAN	Cellular networks	LPWA
Range	Very short (~1-3 m)	Short (~10-100 m)	Long (~1-5 km)	Very long (~5-20 km)
Energy efficiency	Very high (tx. power < 1 mW)	Medium (tx. power ~80 mW)	Low (tx. power ~500 mW)	Very high (tx. power ~ 20 mW)
Data rate	Low	Very high	High	Very low
Cost	Low	Medium	Medium	Very low

LPWAN features:

- Long-range coverage (up to 10 km).
- Low infrastructure cost.
- Very high energy efficiency (duration of the battery up to 10 years).
- Low data rate per device, but very large number of devices. That's obvious due to the new network design → increasing the range means having a lower data rate.

- Mostly static (simple core network, with limited support for handover and mobility).
- Easy network implementation and deployment (no need for detailed radio planning).

Long-range IoT technologies

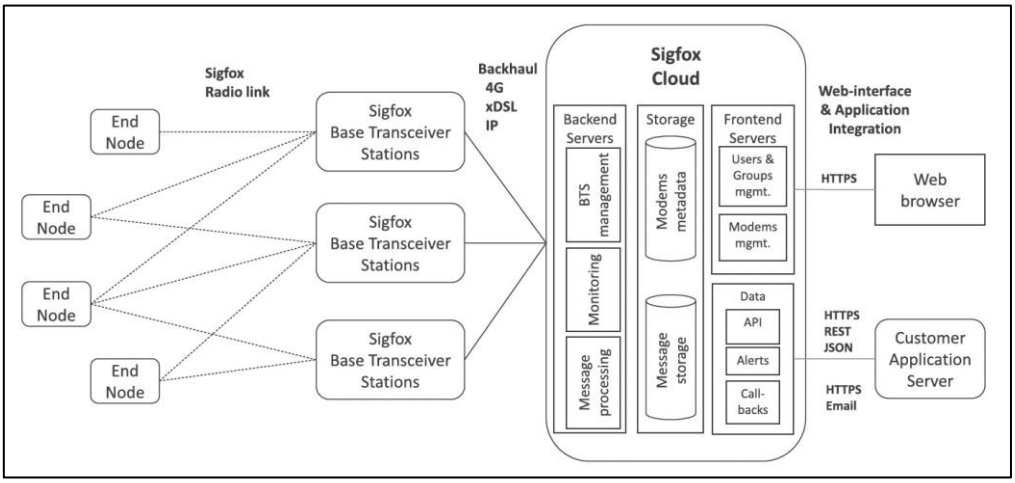


NB-IoT (Narrow Band IoT) is designed to work with cellular network which is in some way the opposite of what we discussed up to now. Anyway, it is possible to modify the way in which cellular networks are designed to make them work properly in IoT environment. Also remember that using cellular networks means paying for using those frequencies (licensed spectrum), that's why it is interesting to make the comparison between [Sigfox, LoRa] and NB-IoT.
Out of the three, Sigfox is the one that provides the lowest quality of the service. Anyway, for some applications we do not care about network efficiency and so it becomes useful to consider even this technology.

3.2.1 Sigfox

Sigfox is the first LPWAN technology, developed by the French company Sigfox. The idea is to provide a **Sigfox cloud service** to end nodes via a subscription-based business model. Sigfox offers for free the connectivity service between end nodes and gateways but to process data in their cloud you must pay a fee.
Communication between the end nodes and base stations is managed by local operators. Base stations forward data to the Sigfox cloud for storage, processing, etc.

Sigfox network architecture



Long-range large-scale IoT applications:

- Connected dumpsers (e.g., OnePlus Systems, Sayme).
- Gas tank remote monitoring (e.g., Silicon Controls, Ijinus).
- Street lightning (e.g., Kawantech). The idea is to equip streetlights with IoT sensors and turn on/off them according to the presence of someone in the street to save energy. The sensors make measurements and with Sigfox it is possible to send data to the cloud, and a response is returned to actuators that turn on/off the lights.
- Smart parking (e.g., IoTMalta, Libelium, Sterela). Idea: find parking more easily to alleviate traffic down the streets.

Frequency range

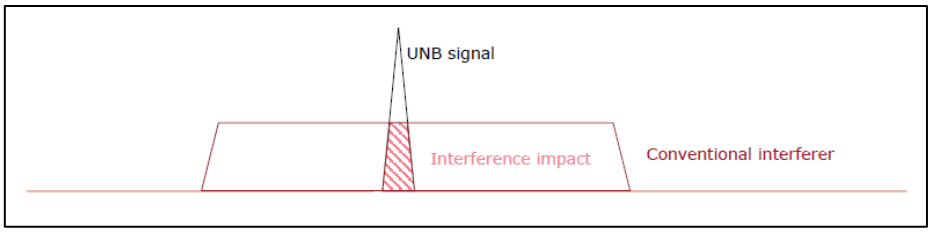
Parameter	RC1	RC2	RC3	RC4	RC5	RC6	RC7
UL frequency (MHz)	868.130	902.200	923.200	920.800	923.300	865.200	868.800
DL frequency (MHz)	869.525	905.200	922.200	922.300	922.300	866.300	869.100
UL data rate (bit/s)	100	600	100/600	600	100/600	100/600	100/600
DL data rate (bit/s)	600	600	600	600	600	600	600
EIRP (dBm)	16	24	16	24	14	16	16
Specifics	DC 1%	FH	LBT/DC 1%	FH	LBT	DC 1%	DC 1%

As said before, Sigfox works with unlicensed frequencies, but these are limited due to regulations that change country to country (DC, etc.). Seven frequencies according to the different regulations (for instance, Italy uses the RC1).

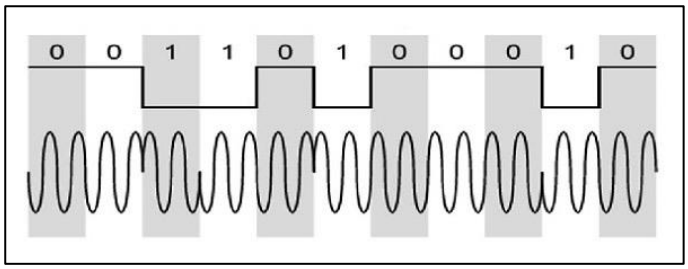
Specifics are some regulations about DC (Duty Cycle) and LBT ("Listen Before Talk") to avoid interference between end nodes. The MAC layer implementation is quite simple (a sort of ALOHA → when end nodes have to communicate, they just send data) and a way to manage channel contention is using DC and LBT (listen the channel before transmitting). **LBT and DC are mutually exclusive**: only one can be used depending on the specific country regulations (look the frequency range table → if LBT/DC is present we can decide which one to use). In this course we will focus on RC1 regulations, basically the ones valid for our country.

PHY layer

- The keyword is **simplicity**: no connection, no configuration, no signalling.
- Sigfox defines a proprietary **Ultra-Narrow Band (UNB)** wireless technology.
  - Energy is concentrated into a very tiny portion of the bandwidth, as we can see from the figure below.
  - **Interference-immune**. Interferences are usually large bands, instead UNB signals are ultra narrow band, so the interference does not affect much the transmitted signal.
  - Small amount of noise power.



- **Bandwidth is limited**.
- Modulation: Differential Binary Phase Shift Keying (DBPSK) and Gaussian Frequency-Shift Keying (GFSK). Bits are encoded as changes in phase of the signal.
- **Repetition scheme**: each message is transmitted **3 times**:
  - More robustness, it is more likely that at least one Sigfox station receives the signal.
  - The Sigfox cloud/backend will take care of possible duplicates.



If the data bit is Low i.e., 0, then the phase of the signal is not reversed, but continued as it was. If the data is a High i.e., 1, then the phase of the signal is reversed.

Frame structure

- Max. frame size: 26 bytes.
  - **Uplink frame**: payload up to 12 Bytes.
    - UL is more common than DL (e.g., for sensors to report data measurements).

Preamble	Sync.	Device ID	Payload	Auth.	FCS
4 bytes	2 bytes	4 bytes	0÷12 bytes	variable	2 bytes

  - **Downlink frame**: payload up to 8 Bytes.

Preamble	Sync.	Flags	FCS	Auth.	Error codes	Payload
8 bytes	13 bits	2 bits	1 byte	2 bytes	variable	0÷8 bytes

Note: since we are forced to use UNB and implement the repetition scheme, the final data rate is highly reduced. Consider the following example.

Transmission times (example)

- Let's consider Sigfox Europe.
  - Duty cycle: 1% → I can transmit 36 s/h.
  - UL data rate: 100 bit/s → With the DC, I can transmit 3600 bit/h = 450 byte/h.
  - Repetition code: 3 → For «new» data, it is 450/3 = 150 byte/h.
  - Frame size: 26 bytes → I can transmit 460 / (26·3) ≃ **6 frames/h.**

    - Max. payload size: 12 bytes → I can transmit up to 12 · 6 = **72 bytes/h → very limited!** This highly reduce the range of possible applications.
- Applications have (limited) requirements, and benefit from the simple design.
    - GPS coordinates (lat x long): 6 bytes
    - Temperature: 2 bytes
    - State reporting: 1 byte

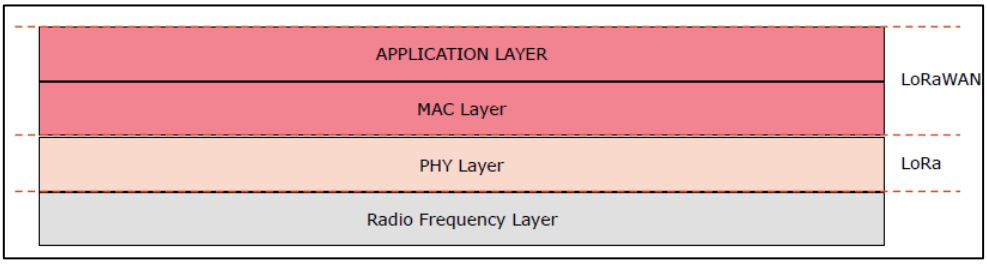
MAC layer

Transmission is **initiated by end devices**:

- End devices periodically wake up to transmit UL data.
- Then, it briefly listens for messages to be received (e.g., ACKs or commands).
- Then, it goes to sleep mode.
- This approach is good for regular data collection and monitoring, but **not for command-and-control applications** (command always follows sensors-based transmissions).

3.2.2 LoRa

LoRa was developed in France in 2009 and it stands for "Long Range". LoRa implements the PHY layer while LoRaWAN constitutes the rest of the protocol stack (MAC + higher layers).



As Sigfox, also LoRa covers only specific areas around the globe since an infrastructure is needed to implement long-range communications (look at slides for more details).

LoRa is more aggressive in network performances compared to Sigfox. This in fact increases the range of possible applications:

- Smart agriculture (Cattle management, Health of soil monitoring)
- Smart buildings
- Smart cities (City network, Bus Schedule)
- Smart environment (Fire monitoring)

Frequency range

Regional specifications establish rules on preamble, channel frequencies, allowed spreading factors, maximum payload size, receive windows, join procedures.

Frequency range (Europe)

- From 24 to 80 channels, of 125 KHz each.
- The network operator can decide how many channels are employed.
- Must implement at least **3 channels**: 868.10, 868.30, 868.50 MHz.
  - For sending join-requests, so all gateways should listen on these channels.
- The maximum MAC payload size is **230 bytes** (application: **222 bytes**)

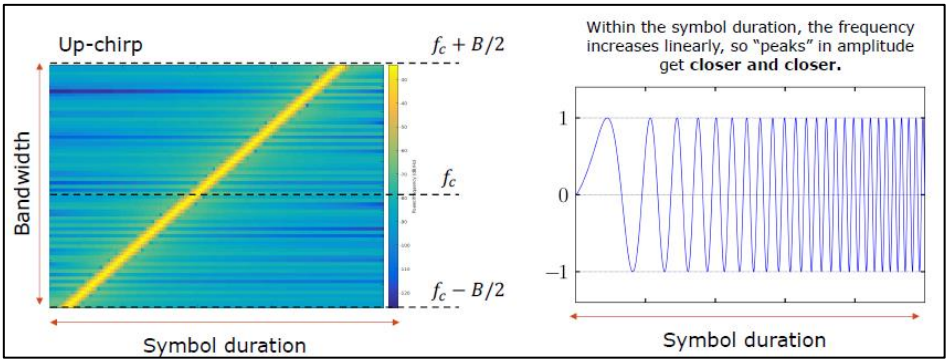
Range	Power	Duty cycle	Bandwidth(*)
863 MHz - 865 MHz	25 mW (**14 dBm**)	0.1%	125 KHz
865 MHz - 868 MHz	25 mW	1%	125 KHz
868 MHz - 868.6 MHz	25 mW	1%	125 KHz
868.7 MHz - 869.2 MHz	25 mW	0.1%	125 KHz
869.4 MHz - 869.65 MHz	500 mW (27 dBm)	10%	125 KHz
869.7 MHz - 870 MHz	5 mW	No requirements	125 KHz
	25 mW	0.1%	125 KHz

Note: select the same channel (among the three mandatory ones) for both end node and gateway to make the communication possible. Having only three channels make network joining procedures for sensors simpler: talking about the first connection, the sensor cannot know which channel is used by the gateway and this simplifies things, helping the synchronization between end nodes and gateways (the highlighted row in the above table refers to this, these three "mandatory" channels must be implemented).

The keyword of Sigfox was simplicity (we use the network as it is without caring about performances). Instead, for LoRa, the keyword is **flexibility**: there is more freedom in setting the communication aspects.

PHY layer

- **Chirp Spread Spectrum (CSS)**
  - Symbols are encoded by modulating a carrier that changes frequency **linearly in time**. Frequency is changed to prevent interference (similar to Bluetooth hops).
  - Up-chirps and down-chirps.



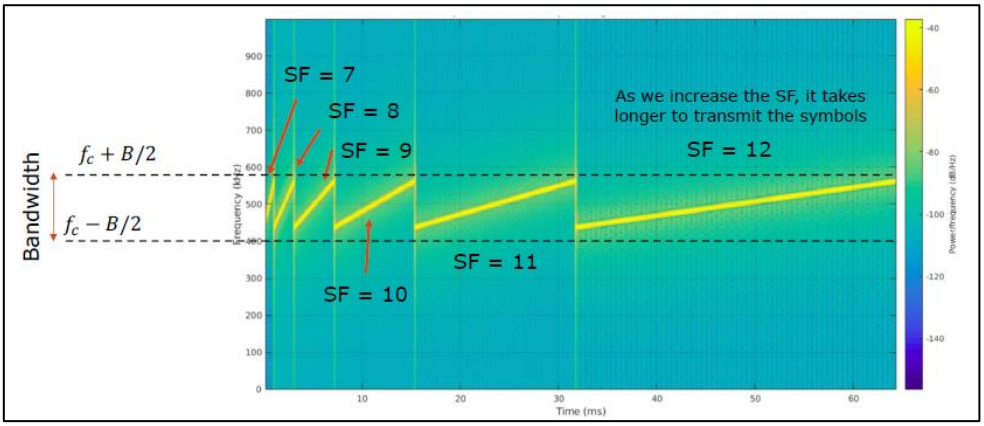
  - The signal is centered in f_c (the signal exists between f_c-*B/2* and f_c+*B/2*) and, as we can note, the frequency changes in time, it increases, having so more oscillations in time. This is done to prevent interference and to reduce the impact of noise in signal transmission (more distance means more noise). We cannot

change bandwidth, but it is possible to change symbol duration → in this way LoRa is flexible, "**we can select how much the symbol transmission is fast**".

- **Spreading Factor (SF)**: number of bits per symbol.
  - 2^{SF} possible combinations of symbols (also called **chirps**).
  - Example: SF = 2 → 2 bits to represent symbols → 2^2=4 possible symbols: {00,01,10,11}.
  - Example: SF = 3 → 3 bits to represent symbols → 2^4=8 possible symbols: {000,001,010,...111}.
  - In LoRa, SF can assume values between 7 and 12.
  - **Given the SF value with LoRa we can influence values as data rate, Time-on-Air (ToA), battery life, and receiver sensitivity** → a flexible mechanism.

PHY layer: spreading factor

Select a different spreading factor influence the chirp shape, meaning the symbol takes less time to go through the spectrum or more time. **Smaller is the SF faster is the transmission** but then we are limited on the number of symbols we can transmit (and not only).



- **Symbol rate**: number of symbols/s → $R_S = B/2^{SF}$.
  - Since there are 2^{SF} possible symbols, the whole **bandwidth is split into 2^{SF} parts**.
  - If we increase the SF, we have more combinations of symbols.
- **Symbol duration**: how much time it takes to transmit a symbol → $T_S = 1/R_S = 2^{SF}/B$.
  - If we increase the SF, it takes longer to transmit a (longer) symbol. Increasing SF by 1 means doubling the number of possible symbols, leading so to longer time transmissions.
- **Bit rate**: number of bits/s → $R = SF \cdot R_S = SF \cdot B/2^{SF}$
  - Each symbol is represented by SF bits.
  - Increasing the SF means to **increase linearly the number of bits per symbol** but also **increase exponentially the symbol duration**.
  - The increase of symbol duration is dominant → **the bit rate decreases by increasing SF**.
- **Energy consumption** increases as $SF/2^{SF}$ increases (more ToA = more energy).

- Increasing the SF means increasing symbol duration, energy consumption and decreasing the bit rate, so what is the right trade-off? Keep the SF as low as possible but if all end nodes use the same SF interference problem arises.
- SFs are **pseudo-orthogonal**.
    - Multiple signals at different data rates (e.g., using different SFs) on the same channel (i.e., using the same time/frequency resources) can be received simultaneously.
    - Signals can arrive concurrently with no collision, as long as they have a different SF values.
    - In practice, no collision even with the same SF if the power difference is **≥6 dB**.
- **Code rate (CR)**: LoRa applies **Forward Error Correction** by adding a number of CR extra bits every 4 bits, with CR = {1,2,3,4}. The actual bit rate can be computed as:

$$R = SF \cdot \frac{B}{2^{SF}} \cdot \left(\frac{4}{4+CR}\right)$$

- Higher SF have a longer transmission duration ("**Time on Air**" - ToA).
    - The receiver can accumulate energy for a longer time, meaning the signal may reach further destinations, covering more sensors (first benefit of a high SF).
    - **Better sensitivity**: the receiver can effectively receive more corrupted signals.
    - **Longer range**: the receiver can receive on a more attenuated (i.e., longer) link. The sensitivity (S) depends on the bandwidth, the min. SNR, and the circuitry noise:

SF (bits/symbol)	SNR_m
7	-7.5 dB
8	-10 dB
9	-12.5 dB
10	-15 dB
11	-17.5 dB
12	-20 dB

$$S = -174 + 10\log(B) + NF + SNR_m$$

- Lower SF → Higher rate → Lower range.
- Higher SF → Lower rate → Higher ToA → Higher range.

SF	Bandwidht (KHz)	Bit rate (Kbit/s) (CR = 1)	Range (km)	ToA (ms) (10-Byte)	Sensitivity (dBm)	Max. payload (B)
7	250	10.9	2	28	-120.5	222
	125	5.5	2	56	-123	222
8	125	3.125	4	103	-126	222
9	125	1.758	6	205	-129	115
10	125	0.977	8	371	-132	51
11	125	0.537	10	741	-134.5	51
12	125	0.293	12+	1483	-137	51

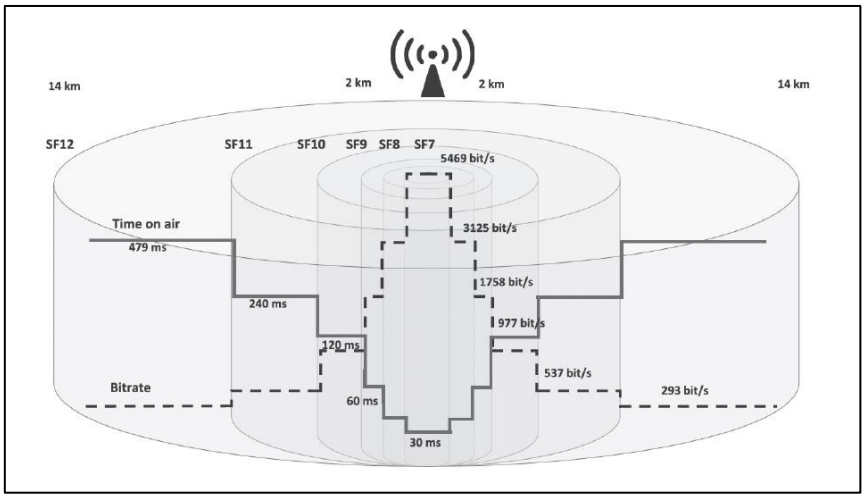
Note: if there are end nodes far away from the gateway it is convenient to increase their SF, in a way the signal can reach the gateway and overtake the SNR threshold to make the signal

recognizable. Obviously, the data rate will be lower but that's the price to pay to make the signal at least detectable.

How to select the optimal SF?

- It must be large enough to reach the end device, but not too large to avoid energy waste.
- **Adaptive data rate**: mechanism used by the gateway to adapt the SF based on the link conditions.
- The objective is to jointly optimize the quality of the communication, the network capacity, and the power consumption.

Rule: if the link margin is high, the data rate can be increased (i.e., the SF can be reduced). On the other hand, if the link budget is low, the data rate should be reduced (i.e., the SF should be increased). In other words, **as the distance of the end device from the gateway increases, the SF increases to cope with the progressively worse SNR**. Correspondingly, the ToA increases, together with the power consumption. It follows an example of what said up to now.

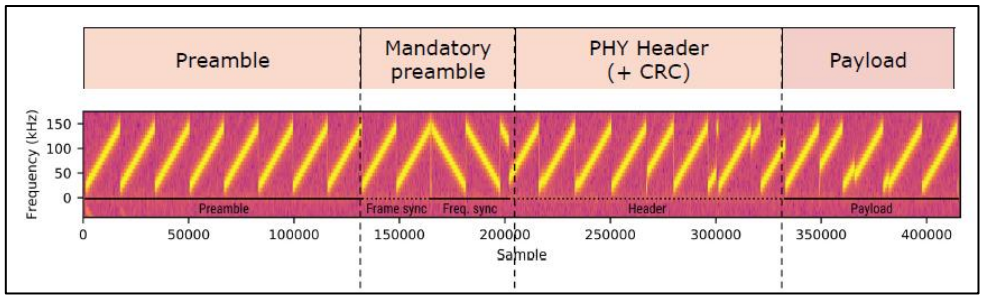


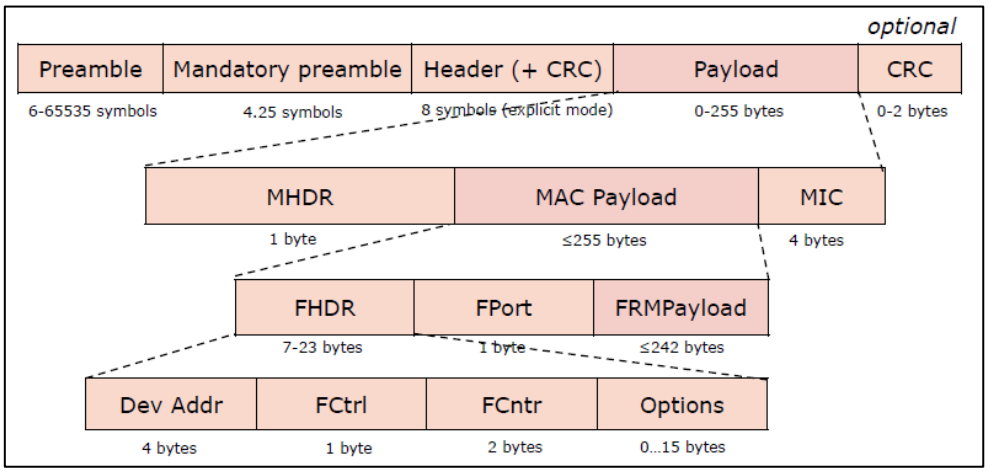
Frame structure

- **Preamble**: well-known sequence of up-chirps and down-chirps.
  - The longer, the more likely the detection of the packets by the receiver.
  - Typically, 8 symbols.
- **Mandatory preamble**: for synchronization.
- **PHY Header**: payload length, data rate configuration, …
- **Header CRC**: indication of the CRC in the payload.

Preamble	Mandatory preamble	PHY Header (+ CRC)	Payload	CRC
6-65535 symbols	4.25 symbols	8 symbols (explicit mode)	0-255 bytes	0-2 bytes

- It follows a spectrogram of an example LoRa signal transmitted with the RN2483 LoRa transceiver using **SF 11** and **CR 5** and received with a USRP B210 SDR (every chirp corresponds to one symbol transmission).





- **MHDR** (MAC Header): specify the message type.
- **MIC** (Message Intregrity Code): digital signature (for security).
- **FPort** (Frame Port): if the payload has some application data, then Fport = 1.
- **FHDR** (Frame Header): contains several subfields:
    - Dev Addr (32 bits).
    - Frame Control (FCtrl): network control information, such as whether to use the data rate specified by the gateway for uplink transmission, whether this message acknowledges the reception of the previous message, whether the gateway has more data for the mote.
    - Frame Counter (FCntr): sequence numbering.
    - Frame Options: commands used to change data rate, power transmission, etc.

Some frames are mandatory, others are optional. Typically, the gateway decides that according to the application needs.

Time on Air

- The "**Time on Air**" is the time to transmit a frame, which depends on the number of symbols in the preamble and in the payload:

$$ToA = t_{preamble} + t_{payload}$$

- $t_{preamble} = (n_{preamble} + 4.25)T_S$.
- $t_{payload} = (n_{payload})T_S$ → The computation of $n_{payload}$ depends on many parameters:
  - The Spreading Factor (SF).
  - Number of payload bytes (PL).
  - Whether the PHY Header is enabled (H).
  - Whether low data rate optimization is enabled (DE). This mode consist in deleting the top two rows of the interleaving matrix, because they are more prone to errors.
  - The number of CRC bits (CR).

- We have that:

*Min. payload overhead is **12 (+1) bytes**: MHDR (1)+Dev Addr (4) + FCtrl (1) +FCnt (2) + MIC (4) (+ FPort (1) if app data).*

$$n_{payload} = 8 + max\left(\left\lceil \frac{8PL - 4SF + 28 + 16 - 20H}{4(SF - 2DE)} \right\rceil (CR + 4), 0\right)$$

Formula based on SX1276 Semtech datasheet: https://www.mouser.com/datasheet/2/761/sx1276-1278113.pdf

- Given that $T_S = 1/R_S = 2^{SF}/B$, we have that:

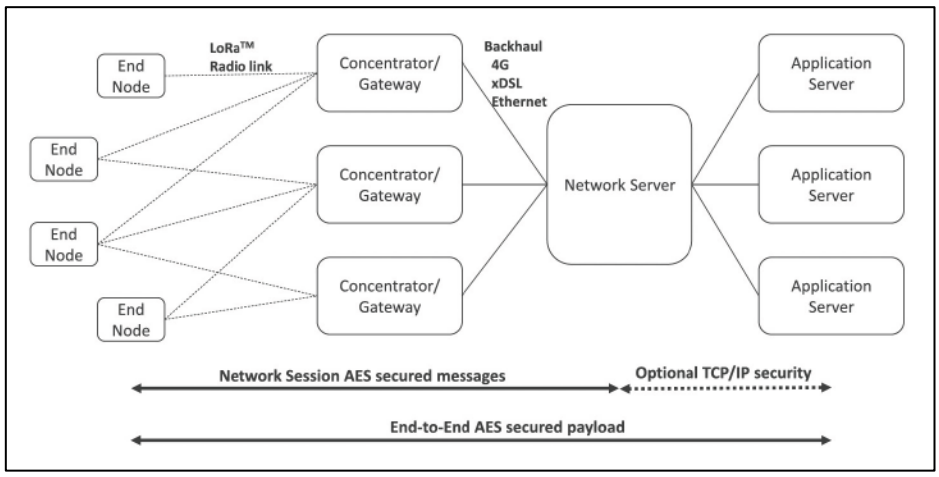
$$ToA = \frac{2^{SF}}{B}\left((n_{preamble} + 4.25) + 8 + max\left(\left\lceil \frac{8PL - 4SF + 28 + 16 - 20H}{4(SF - 2DE)} \right\rceil (CR + 4), 0\right)\right)$$

Note: if we have a high ToA and we combine it with DC regulations, the number of messages that is possible to transmit is very limited. Sensors should be stay silent for a long time after a transmission. In these scenarios a trade-off should be found, for instance reducing the SF which is one of the main constraints that leads to a higher ToA.

Fair Access Policy (FAP) and Duty Cycle (DC)

The correlation between the FA and DC in LoRaWAN networks lies in their shared goal of managing network access and ensuring equitable resource allocation among devices. Duty Cycle limits the percentage of time a device can spend transmitting within a given period (e.g., 1% per hour), effectively controlling airtime and preventing any single device from monopolizing the network. The Fair Access Policy, often set by network operators, builds upon DC by further managing the frequency, data rate, and duration of transmissions each device can make. This policy ensures that even within DC limits, devices do not overuse network capacity, maintaining fair access and balanced airtime distribution across all users.
Together, FAP and DC form a complementary framework that both restricts individual device transmission and actively balances the load across a large number of devices, promoting network scalability and reliable performance in crowded, shared-spectrum environments.

LoRaWAN architecture



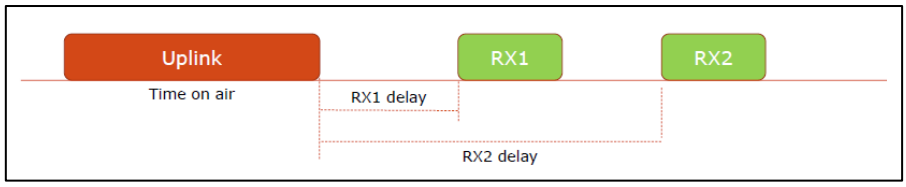
LoRaWAN defines the components and the recommended protocols:

- **End Nodes**: end devices carrying sensing and actuating tasks.
  - DevEUI: End Node identifier (64 bits).
- **Gateway**: wireless base stations, to which End Nodes communicate based on the LoRa physical layer. Star-of-stars topology.
  - Dev Addr: Gateway identifier (32 bits).
- **Network Server**: eliminates duplicate messages, routes data to the relevant **Application Server**, and selects the best Gateway to reach the End Node in DL (based on the link quality)
  - End Nodes transmit messages that can be received by all gateways within reach.
  - The gateways relay the message at the Network Server.

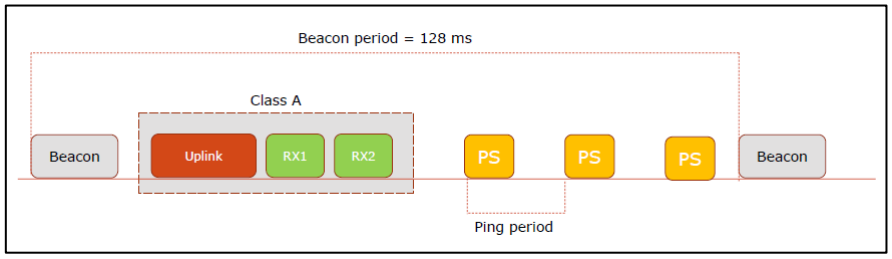
MAC layer

Three different classes of End Nodes.

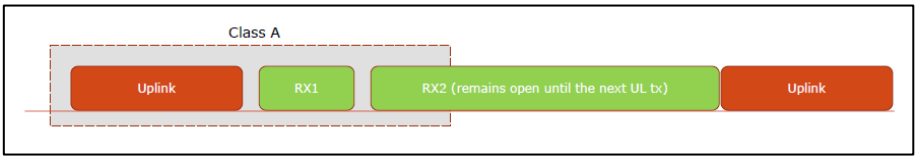
- **Class A: communication is always initiated by the End Node**.
  - End Node sends an UL message at any unspecified time.
  - Completely unsolicited and asynchronous.
  - After a certain delay, the End Node opens two short received windows (RX1 and RX2), that can be used to implement DL communication (e.g., commands or ACK).
  - **Very low energy consumption** (End Node is most of the time in sleep mode).
  - **Very large DL latency** (DL communication is only triggered after UL transmissions).
  - Use case: periodic sensor measurements.
  - RX2 is opened only if the network server does not respond during RX1.
  - If the network server does not respond during both RX1 and RX2, the next DL opportunity will be scheduled immediately after the next uplink transmission → long delay due to DC.



- **Class B: bi-directional communication**.
  - It operates as Class A, with RX1 and RX2.
  - In addition, it uses time-synchronized beacons, sent by the Gateway, to open ping slot, i.e., extra receive windows opened by the Gateway itself to initiate DL transmissions.
  - Beacons are sent every 128 ms, for 160 ms. The duration of the ping slot is 30 ms.
  - The interval between two beacons is divided evenly into 4096 ping slots.
  - **Lower (bounded) DL latency** (DL communication does not necessarily depend on UL).
  - **Higher energy consumption** than Class A (due to ping slots).
  - Use case: utility meters (electrical meters, water meters, etc.), streetlight.



- **Class C: continuous receive mode**.
  - RX2 stays open until the next uplink transmission, unless there is activity on RX1.
  - **Very low DL latency** (End Nodes can receive DL messages at almost any time via RX2).
  - **Very high energy consumption** (since RX2 is always on).
  - Class C devices shall be connected to the mains (no battery power), not so compatible with most of IoT sensors.
  - Use case: beacon lights, alarms.



3.2.3 NB-IoT

As previously discussed, the idea behind NB-IoT is to use cellular networks as the base infrastructure for IoT networks (ubiquitous public wireless infrastructure). Also, NB-IoT provides good features for IoT: localization, flexibility (5G), slicing and virtualization. Anyway, attention

being paid on higher and higher bitrates, which is not compatible with IoT (drain the sensor battery and not only).

There are two options to exploit cellular networks:

1. **Adapting existing cellular solutions** (3G, 4G, etc.) for IoT via new user profiles. Use the cellular network "as it is" with limited investments. Radio protocols remain inefficient (e.g., from a power consumption point of view). A fast approach but not efficient.
2. **Specifically designed cellular-based solution for IoT**. For example: Extended coverage GSM; LTE-M, **NB-IoT**. NB-IoT can work based on both 4G-LTE and 5G-NR and is the most promising solution.

NB-IoT stands for "Narrow Band IoT", standardized by the 3GPP (that typically addresses cellular mobile standards). Target low complexity and small resource (power and spectrum) usage. It is **based on the design principles of 4G-LTE**, even if not yet compatible with LTE: an NB-IoT radio access is provided **with its own control signals and radio channels**. Also, note that even if NB-IoT works with 4G, a sensor that belongs to a NB-IoT network cannot communicate with a smartphone, for instance, since the 4G used by NB-IoT is quite different from the common one we use every day on our smartphones → two separated 4G networks.

In terms of NB-IoT world coverage, this technology is not so popular as the previous discussed long-range technologies but only because it is quite new. One of the most covered areas is Italy.

Applications

- Smart agriculture: soil and environmental data collection:
- Healthcare monitoring: remote monitoring devices to transmit vital health data in real-time
- Retail: manage inventory, enhance customer experiences, and ensure asset security.
- Public safety
- Supply chain and logistics
- Energy management: smart meters to obtain real-time data on electricity consumption, enhancing efficiency and reducing waste.

Frequency range

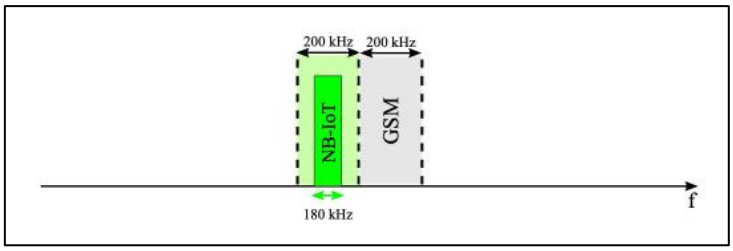
3GPP has defined a set of frequency bands for which NB-IoT can be used. It uses the same frequency bands as in LTE, with a subset defined for NB-IoT → **licensed bands**. Generally, in the lower range of existing LTE bands, to increase the range.
Consider the flowing table that defines frequency range in Europe and note the large bandwidth available which is much more than the one commonly used by IoT sensors. Anyway, we will see that only a small portion of those large bands is used, also to avoid interference between devices.

Frequency range (Europe)

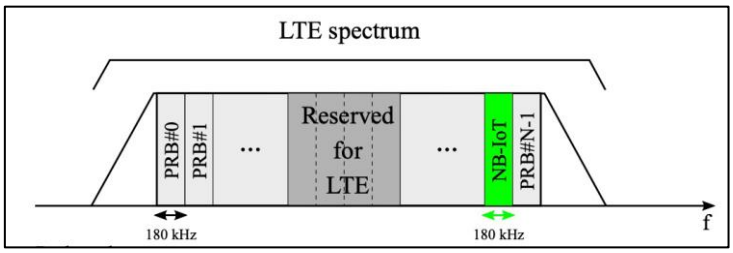
NB-IoT band	Uplink band	Downlink band	Bandwidth	Duplex mode
3	1710 - 1785 MHz	1805 - 1880 MHz	75 MHz	HD-FDD
8	880 - 915 MHz	925 - 960 MHz	25 MHz	HD-FDD
20	832 - 862 MHz	791 - 821 MHz	30 MHz	HD-FDD

Operation modes

- **Standalone**: the NB-IoT signal is intended to occupy the liberated spectrum of the GSM (Global System for Mobile communications) system.
  - The NB-IoT signal still occupies **180 kHz** from the 200 kHz GSM carrier, with 10 kHz of band-guard on both sides of the spectrum.
  - **Dedicated spectrum**, without competing with regular mobile broadband services → we are using the whole bandwidth of GSM.
  - Suitable for areas where there is no existing cellular network (refarming of GSM).

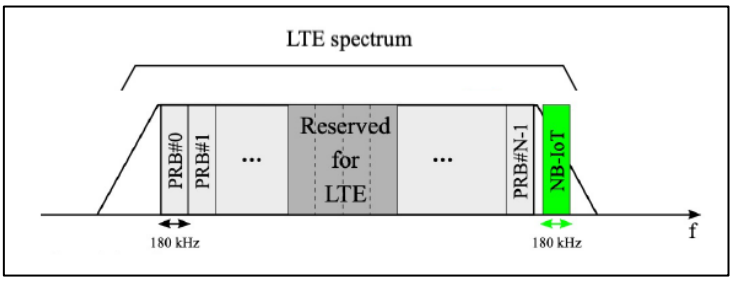


- **In-band**: the NB-IoT signal occupies one 180-KHz PRB from the LTE bandwidth.
  - It is the most privileged mode due to the benefits in terms of cost savings and ease of integration over the legacy LTE networks.
  - The NB-IoT anchor carrier can take **only a predefined set of possible PRBs (Physical Resource Blocks) to avoid overlapping with LTE transmissions**.



- **Guard-band**: the NB-IoT signal occupies one PRB from the unused guard band PRBs of the LTE bandwidth. This means that NB-IoT does not take away resources from the primary LTE service, allowing it to coexist without impacting LTE performance. The main

idea is to conduct NB-IoT operations during guard periods which are essentially buffer zones between LTE carriers that are not actively used for LTE transmissions.

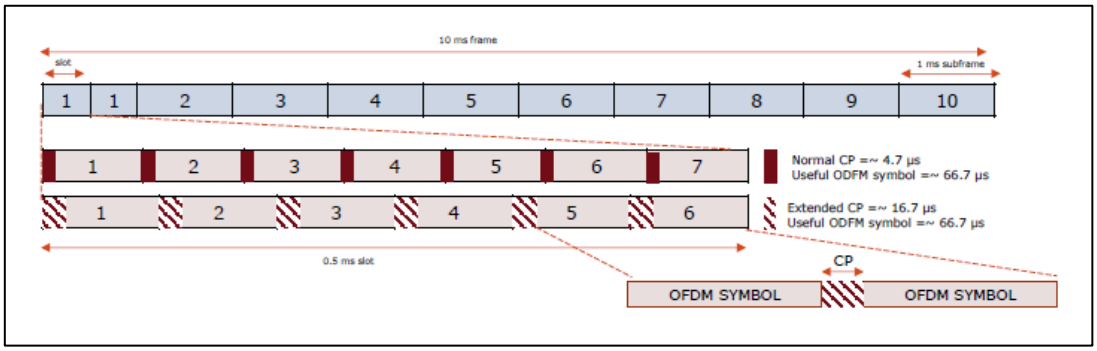


PHY layer

NB-IoT operates on principles similar to those used in LTE, utilizing **Orthogonal Frequency Division Multiple Access (OFDMA)**, which allows efficient frequency management. The modulation schemes employed are **QPSK** or **BPSK**, offering reliable data transmission for low-data-rate IoT applications. The technology supports a maximum transport block size (TBS) of **680 bits for downlink and 1000 bits for uplink**, which aligns with its low-bandwidth requirements. The transmit power for NB-IoT devices is capped at 23 dBm, providing sufficient signal strength while conserving battery life. It operates in **Frequency Division Duplexing (FDD)** mode but is limited to **half-duplex**, meaning a device can either transmit or receive at any one time, rather than both simultaneously. This design is tailored to meet the low-power and low-complexity needs of IoT deployments.

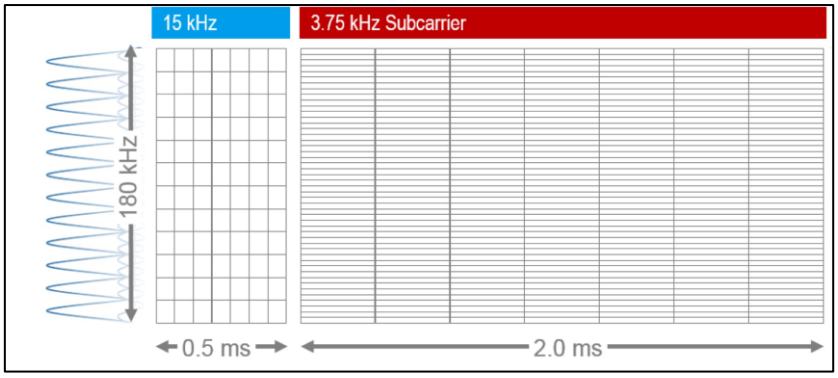
Frame structure

- Time: fixed slot duration of **10 ms**
  - Within each frame, 10 subframes of 1 ms.
  - Within each subframe, 2 slots of 0.5 ms.
  - Within each slot, 7 (normal CP) or 6 (extended CP) OFDM symbols to prevent interference (similar to the guard period seen before).
- Frequency: **Physical Resource Blocks (PRB)** of 12 subcarriers.
  - Subcarrier spacing of 15 kHz: the PRB has a bandwidth of 12*15 = **180 kHz**.



- In **uplink**, NB-IoT supports allocating one or more of the 12 subcarriers (aka **tones**) of a single PRB to different UEs (User Equipment → any device connected to the network).

- **Multi-tone transmission**: SC-FDMA with 15 kHz subcarrier spacing and 0.5 ms slot. It increases data rates and reduces transmission delays and power consumption. Multi-tone (3 tones, 6 tones, and 12 tones).
- **Single-tone transmission**: 15 kHz (0.5 ms slot) or 3.75 kHz (2 ms slot). With 3.75 kHz, we have 48 (vs. 12) subcarrier within the 180-KHz PRB.



- NB-IoT defines a new **uplink** resource mapping via **Resource Unit (RU)**.
  - RU is a combination of a number of subcarriers (in the frequency domain) and a number of slots (in the time domain).
  - RU can take several different types of **configurations** where a trade-off between transmission time and frequency used should be found (look at slides for more details).

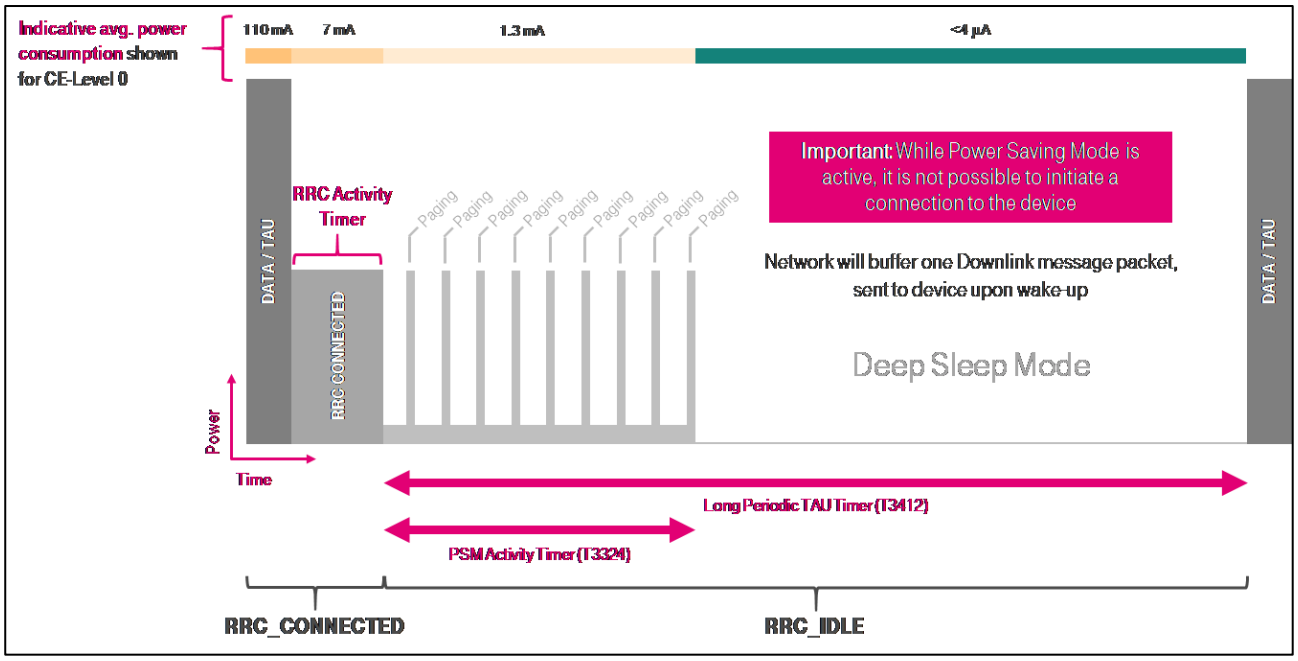
Subcarrier spacing	# tones in RU (subcarriers)	# slots in RU	# symbols in slot	Slot duration	RU duration
3.75 KHz	1	16	7 normal CP	2 ms	32 ms
15 KHz	1	16		0.5 ms	8 ms
	3	8			4 ms
	6	4			2 ms
	12	2			1 ms

MAC layer

It introduces different modes with the aim of saving energy, we focus on two of them.

1) **Power-Saving Mode (PSM)**: it helps conserve battery power.

- Disable parts of the protocol stack and drop power consumption into the micro-Ampere range **while remaining attached to the network** (no reconnection).
  - PSM Activity Timer: time during which the device remains in Idle Mode.
  - Long-Periodic TAU Timer: time between two Tracking Area Updates.
- During the deep-sleep mode, the network retains the state information and the IoT device remains registered with the network.
- If a device awakes before the expiration of the time interval to send data, a reattach procedure is not required, and energy is saved.



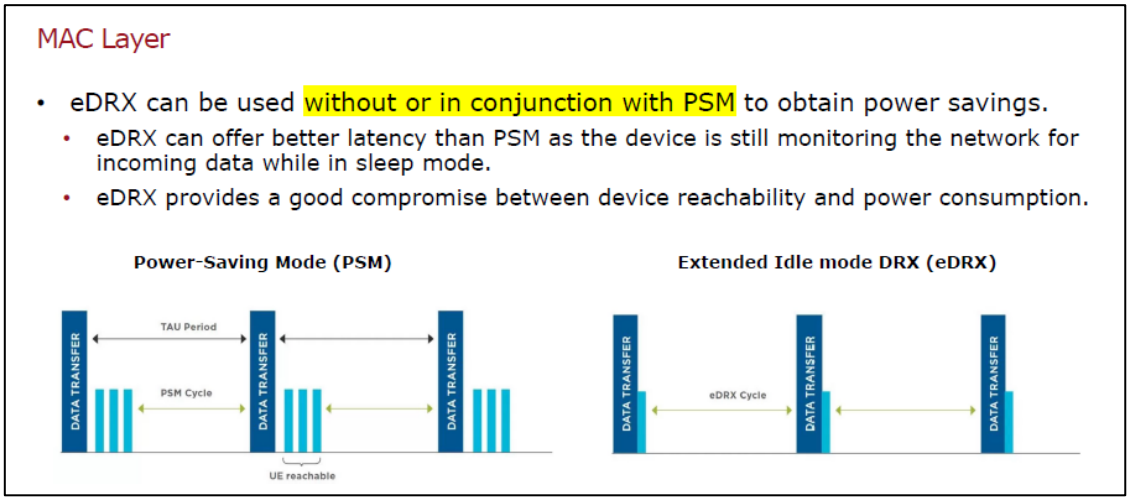
Note: during light-sleep mode the device energy is only dedicated to receiving **paging signals**, signals that notify about incoming messages, in a way the device can wake up and receive them.

2) **Extended Idle mode DRX (eDRX)**: it allows the device to enter a sleep mode while still monitoring the network for incoming data.

- This approach reduces UE power consumption and prolongs UE battery life.
- The device can extend the time it remains in the sleep mode before waking up to check for incoming data. “eDRX cycle” from 1.28 seconds to 40.96 seconds, depending on the configuration.
- eDRX is suitable for use cases where the device needs to receive data more frequently than in PSM, but still not constantly (e.g., in smart parking systems where the device needs to receive data periodically to check for parking spaces).

The main difference between the two modes is that if we are in deep-sleep mode (such as in PSM), the “wake up” process is slower, even if it consumes less energy. On the other hand, in eDRX, the device can wake up more quickly to check for incoming data, but it consumes slightly more energy compared to PSM due to more frequent monitoring intervals.
This distinction makes PSM better suited for devices that need long periods of inactivity (like sensors that rarely need to send updates), while eDRX is better for applications requiring intermittent communication, such as devices that need to receive periodic updates from the network.
It is also possible to combine both modes together, adopting so a hybrid solution, trying to get the pros of both modes.

Combine the two modes together



MAC layer: repetition

Repetition: NB-IoT adjusts data repetition counts based on the signal strength.

- Up to 2048 (128) repetitions in downlink (uplink).
- Achieves **coverage** extension up to 10 km (at cell edges, repetition count is increased).
- It improves **sensitivity**: each doubling of the repetitions improves the sensitivity by **~3 dB** due to coherent addition of the symbols and incoherent addition of thermal noise.
- Accumulating repetitions takes time: trade-off between latency and sensitivity.

The appropriate number of repetitions is **adjusted dynamically**, which assigns an Enhanced Coverage Level (ECL) to each device based on the received uplink and reported downlink signal level (somehow similar to SF in LoRa).

- Higher ECL → more problematic radio conditions → higher number of repetitions.
- Computed based on the **Maximum Coupling Loss (MCL)** parameter.

Link budget

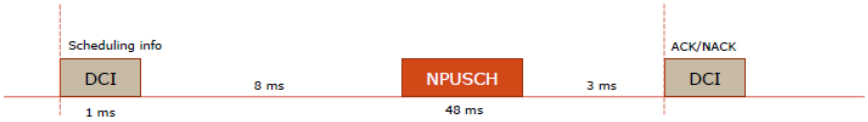
Maximum coupling loss (MCL) of **164 dB**, chosen by 3GPP as a metric to evaluate coverage performance of wireless networks: the maximum loss that we tolerate to define the communication reliable.

MCL is a critical parameter in the planning and optimization of wireless networks, as it determines the coverage area, the cell size, the antenna height, the transmission power, and the frequency allocation.

Radio Access Technology	Maximum Coupling Loss (MCL) [dB]
E-GPRS	~ 164 dB
LTE	~ 144 dB
LTE-M	~ 160 dB
NB-IoT	~ 164 dB
Sigfox	~ 162 dB
LoRa	~ 157 dB

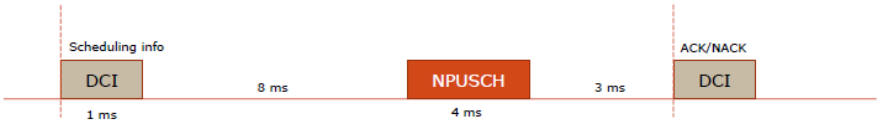
Data rate (uplink single-tone)

- Maximum transport block size (TBS): 1000 bits.
- Maximum MCS: 10.
- Minimum number of RUs: 6.
- Minimum RU duration: 8 ms (obtained with 15 KHz subcarrier spacing).
- The min. time to transmit 1000 bits is $N_{rep} \cdot N_{RU} \cdot T_{RU} = 1 \cdot 6 \cdot 8 = 48$ ms.
- Peak data rate is 1000 bits / 48 ms = **20.8 Kbps.**
- Considering the overhead for scheduling: 1000 bits / (48+12) ms = **16.7 Kbps.**



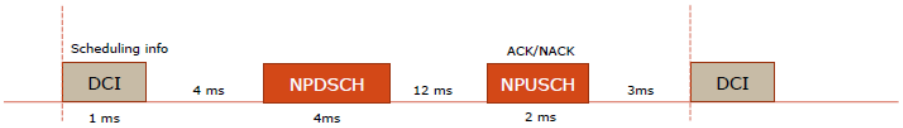
Data rate (uplink multi-tone)

- Maximum transport block size (TBS): 1000 bits.
- Maximum MCS: 12.
- Minimum number of RUs: 4.
- Minimum RU duration: 1 ms (obtained with 12 tones).
- The min. time to transmit 1000 bits is $N_{rep} \cdot N_{RU} \cdot T_{RU} = 1 \cdot 4 \cdot 1 = 4$ ms.
- Peak data rate is 1000 bits / 4 ms = **250 Kbps.**
- Considering the overhead for scheduling: 1000 bits / (4+12) ms = **62.5 Kbps.**



Data rate (downlink)

- Maximum transport block size (TBS): 680 bits
- Maximum MCS: 10.
- Min. number of subframes: 4 (or 3, if MCS 12 is used).
- Minimum subframe duration: 1 ms.
- The min. time to transmit 680 bits is $N_{rep} \cdot N_{SF} \cdot T_{SF} = 1 \cdot 4 \cdot 1 = 4$ ms.
- Peak data rate is 680 bits / 4 ms = **170 Kbps.**
- Considering the overhead for scheduling: 680 bits / (4+22) ms = **26.2 Kbps.**



Note: uplink indicates the communication from the IoT device to the network while downlink represents the communication from the network to the IoT device. Performances are obviously better for uplink rather than downlink. If you think about it, it has sense. Uplink is often more critical than downlink because the primary role of many IoT devices is to send data to the network rather than receive it.

Long-range IoT technologies summary

Parameter	Sigfox	LoRa	NB-IoT
Spectrum	Unlicensed	Unlicensed	Licensed
Modulation	UNB	CSS	QPSK
Bandwidth (UL)	100 Hz	125 kHz (Class A)	180 KHz
Peak data rate (UL)	100 bps	6 kbps (SF7)	250 Kbps
Max. range	50 km	12+ km (SF12)	10 km
Energy consumption	Very low	Low	Low
Tx. power (UL)	14 dBm	14 dBm	23 dBm
Interference immunity	Immune (UNB) + Repetition	Spreading Factor	Repetition
Sensitivity threshold	-140 dBm	$-127 - 2.5(SF_k - 7)$	-141
Duty cycle	Yes	Yes	No
MCL [dB]	162 dB	157 dB	164 dB
Device cost	Low	Moderate	Moderate

3.3 Wired technologies

We have seen many wireless technologies used to connect end nodes to gateways and now, we move our attention on a critical analysis of wired approaches which in general introduce more complexity.

Ethernet has become the dominant wired communication technology, extending its reach beyond office and home networks into the **industrial domain**. Known for its cost-efficiency, driven by large-scale production, Ethernet offers industries the advantage of a unified and homogeneous network infrastructure. Consider also that industrial machines usually operate always in the same place, so we can assume that sensors equipped with the machine are stationary, meaning they can be connected to gateways via cables.

However, to meet the specific demands of industrial environments, such as real-time communication and robustness against harsh conditions, modifications are required, leading to the development of "Industrial Ethernet" or **Real-Time Ethernet (RTH)**. Key examples of RTH standards include Powerlink, **EtherCAT**, and Profinet, each designed to provide the low latency, high reliability, and synchronization needed for industrial automation and control systems.

Note: real-time is crucial in industry. There may be scenarios where employees are near a machine and if the machine is not notified about the presence of someone near it, bad things may happen. Real-time behaviour plays a crucial role in dangerous scenarios (and not only).

3.3.1 Ethernet

Ethernet is the name given to the data-link layer communication protocol described in the **IEEE 802.3 standard**. Ethernet has gone through four generations, following traffic demand:

- Standard: 10 Mbps
- Fast: 100 Mbps
- Gigabit: 1 Gbps
- 10-Gigabit: 10 Gbps

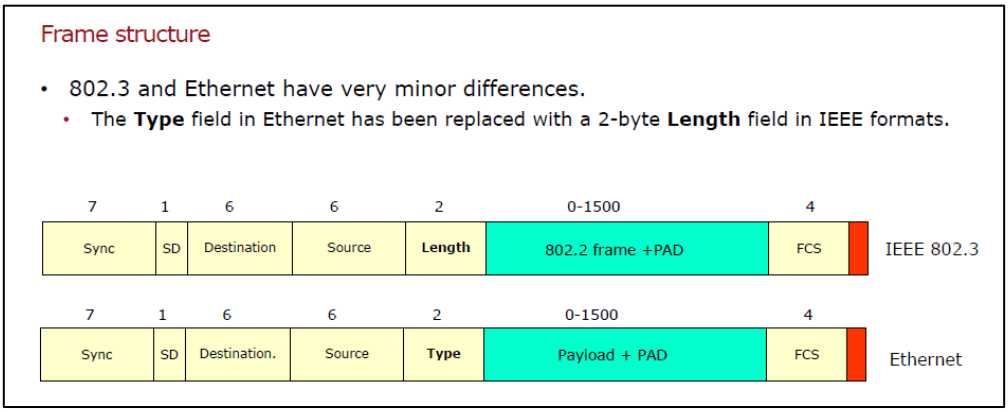
Ethernet operates as a **connectionless communication protocol**, meaning that it transmits frames as soon as they are ready, without establishing a dedicated connection between sender and receiver (the connection is in act when the two parts are connected to the same cable). This setup, while efficient for fast data transfer, comes with a few limitations:

- **Lack of Flow Control**: ethernet lacks flow control mechanisms, allowing the sender to transmit frames without checking whether the receiver is prepared to process them. As a result, there is a risk that the sender could overwhelm the receiver with excessive frames.
- **Frame Dropping**: if a receiver cannot handle the volume of incoming frames, it silently drops any excess (this approach has been chosen to make the protocol simple and fast). Additionally, here there are no ACKs, and ethernet frames include a 32-bit Cyclic

Redundancy Check (CRC) field for error detection. However, if a frame is found to be corrupt (due to a failed CRC check), it is dropped without notifying the sender.

- **Loss Recovery**: ethernet does not implement an Automatic Repeat reQuest (ARQ) mechanism at the MAC layer, meaning that loss recovery is managed by higher-level protocols, like TCP.

In scenarios with shared transmission media, ethernet uses CSMA/CD to handle potential data collisions.



Frame size

The frame size of ethernet is limited between 64 bytes and 1518 bytes.18 bytes are for header and trailer: payload is **between 46 bytes and 1500 bytes**.

Why a maximum frame length?

- Avoid one station monopolizes the bus for a long time.
- Bound the packet error probability: if I lose the packet, just retransmit it.
- Reduce buffering requirements (back in the ’80, memory was expensive!).

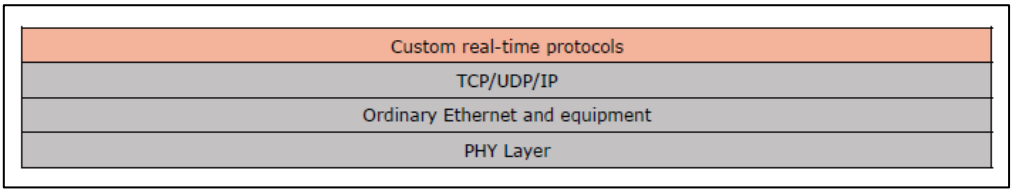
Why a minimum frame length?

- Reducing overhead? But then what if just one byte is transmitted? With the standard implementation we are **forced to send frames of 46 bytes at least**, maybe inserting some 0s in the payload → a waste!
- MAC mechanism: **CSMA/CD**. Sensing the channel before transmitting uses 2-time propagation time (send a signal to the destination and wait for the returning signal to understand if the channel is idle). This means that we need to wait 2-time propagation time every time we want to transmit. This led to a specific minimum number of bytes for the frame, to make CD work properly (more details on slides).

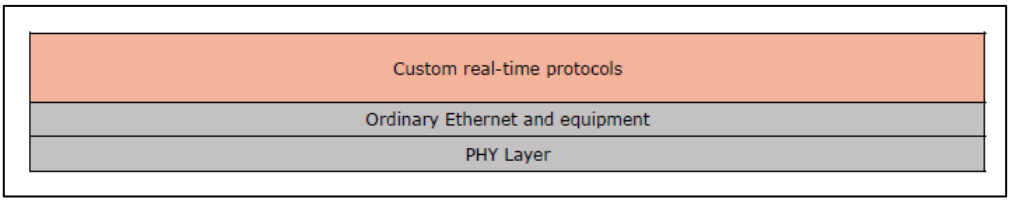
3.3.2 Industrial Ethernet

Some modifications are needed to enable ethernet in the industrial domain, also referred to as **Real-Time Ethernet (RTH)**. In particular, ethernet is not deterministic and may not support real-time operations. Therefore, it is possible to classify ethernet-based technologies into 3 classes:

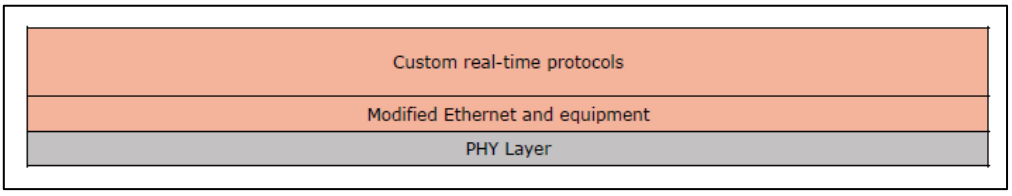
- **Class 1: Ordinary Ethernet + TCP/IP stack**
  - Full compatibility with Ethernet/IP networks.
  - "Soft" real-time (with loose requirements: latency ≥ 100 ms).
  - Unpredictable delay due to other possible traffic flows.



- **Class 2: Ordinary Ethernet + custom apps**
  - Runs a specially designed real-time protocol stack on the Ethernet layer (latency ≥ 10 ms).
  - Some protocol modifications are required to support real-time (e.g., IEEE 1588 PTP).
  - Cannot route packets through the IP network (TCP cannot be used).



- **Class 3: Modified Ethernet + custom apps**
  - Support hard real-time requirements (latency ≥ 1 ms).
  - Some protocol modifications are required to support real-time (e.g., IEEE 1588 PTP).
  - Protocol modifications to the original Ethernet implementations.
  - Requires some custom Ethernet equipment.
  - Cannot route packets through the IP network.



Class 3 will be our focus since we want to satisfy industrial hard real-time constraints.

Applications

- **Manufacturing and logistics**: when capturing data in the industrial environment, the Industrial Ethernet carries out the management of the data to the IT network in real-time for the tracking control.
- **Outdoor applications**: when using motion sensors, it is common to use the network to deliver Power over Ethernet (PoE), as this simplifies wiring and makes it easier to control the device.
- **Electricity substations**: network devices are vulnerable to electrical networks by levels of electromagnetic interference. Cables used in Industrial Ethernet can simplify installation.
- **Vehicles and trains**

Most widely adopted Industrial Ethernet technologies

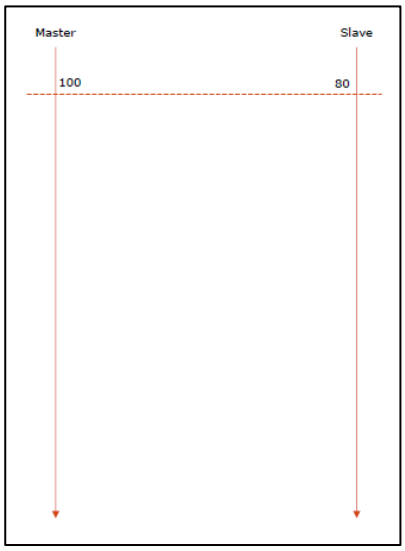
Technology	Technical approach	Class	Standard
Profinet-IO	Ethernet with priorities and VLANs Modified Ethernet with deterministic scheduling (IRT)	2/3	IEC 61158 IEC 61784 Communication Profile 10
EtherNet/IP	TCP/IP over Ethernet	1	IEC 61158 IEC 61784 Communication Profile 2
EtherCAT	Modified Ethernet with master-slave ring topology and summation frame	3	IEC 61158 IEC 61784 Communication Profile 12
Modbus TCP	TCP/IP over Ethernet	1	IEC 61158 IEC 61784 Communication Profile 15
Powerlink	Ethernet with priorities and VLANs	2	IEC 61158 IEC 61784 Communication Profile 13

IEEE 1588 Ethernet

The lack of real-time capabilities of early Ethernet was the primary barrier for the adoption of this technology in the industrial field. The main problem was the **lack of a common timing reference among the sub-systems**, which allows to precisely guarantee synchronization (very important for some apps). GPS can be used to take the reference and set clocks accordingly, but this is not a good solution, because it does not work indoors, and may be inaccurate.

Solution: use communication network to distribute timing signals, which allow clocks (at the transmitter and receiver) to become synchronized → via the **IEEE 1588 Precision Time Protocol (PTP)**.

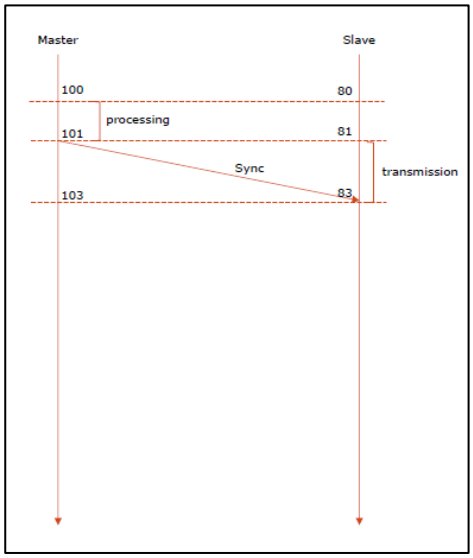
Example of IEEE 1588 over Ethernet



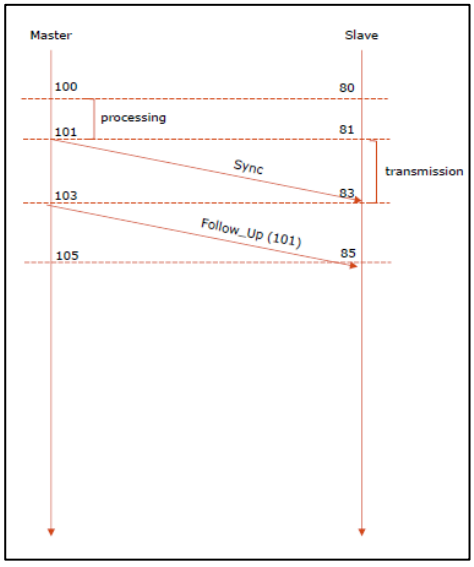
Goal: synchronize a master and a slave

1) Assume that the clock of the Master and the clock of the Slave are not synchronized (at the beginning). Assume that the time offset is -20 s.

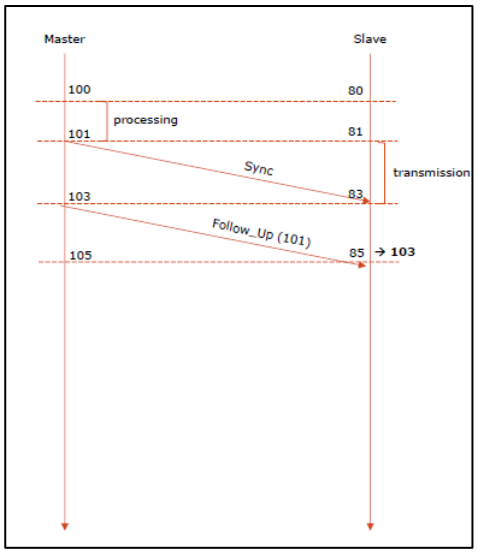
2) PTP starts at 100 s for M (= 80 s for S).



3) M sends a **Sync** message to S.
- Processing time of 1 s
- Transmission/network time of 2 s.
- The message is received at S at time 83 (= 103 for M).



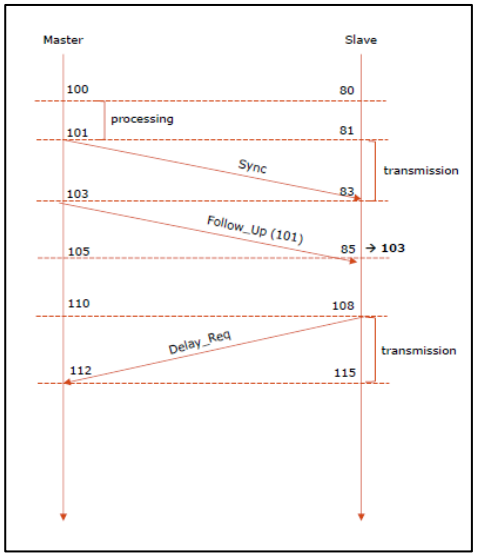
4) At 103 s, M sends a **Follow_Up** message to S, which contains the value of the Master's clock when the Sync message was generated after the processing (= 101 s).



5) S compares the value of the Follow_Up (101) with the value of its clock when the Sync was received (83).

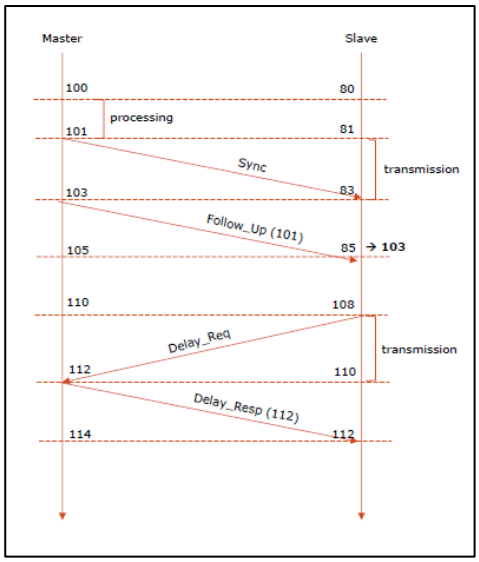
6) S subtracts the two values: 101-83=18.

7) S adjusts its current clock (85) with the new offset (18).
- The new clock becomes 85+18=103.
- The Slave's new clock (103) is still not synchronized with the Master's clock (105).
- The difference is due to the transmission time.

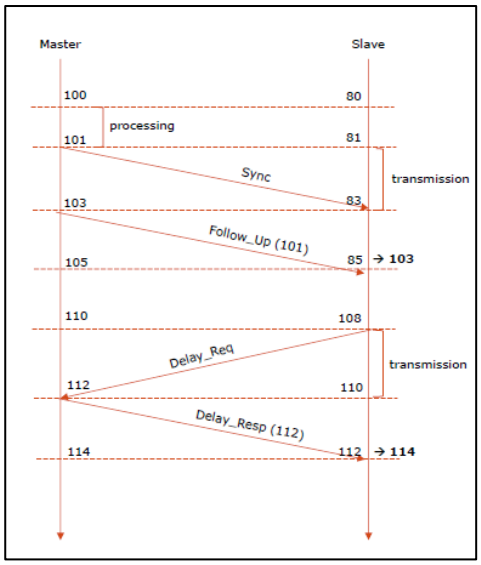


8) S sends a **Delay_Req** at time 108 s (= 110 at M as they are not yet fully synchronized).

9) M receives the message at time 112 s, due to the transmission and network delays.



10) M sends a **Delay_Resp** message to S, which contains the value of the Master's clock when the Delay_Req message was received (=112).



11) S compares the value of the Delay_Resp (112) with the value of its clock when the Delay_Req was generated (108).

12) S subtracts the two values: 112-108=4.

13) S adjusts its current clock (112) with the new offset divided by 2 (4/2=2), since it is due to 2 message transmissions (Delay_Req + Delay_Resp).
- The new clock becomes 112+2=114.
- The Slave's new clock is **fully synchronized**.

SINGLE PAIR ETHERNET

Another problem: ethernet is not compatible with the constraints of the industrial scenario, including power consumption and the limited space for the wires. This led to a new version of Ethernet for industry: **Single Pair Ethernet (SPE)**.

- Compared to ordinary Ethernet, which typically uses four twisted pairs of wires (eight wires in total), SPE uses a **single twisted pair** of wires (two wires in total vs eight in ethernet). A single twisted pair is enough to connect small and simple sensors, decreasing also the energy consumption.
- **(Lower) data rate**: from 10 Mbps to 1 Gbps (vs. up to 100 Gbps).
- **(Higher) range**: up to 1 km (vs. up to 100 m).
- **(Lower) cost**: generally cheaper for applications that do not require high bandwidth, with reduced cabling complexity and lower material costs.

Applications

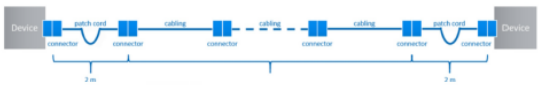
- **E-transportation**
  - Visual and acoustical passenger information systems (PIS); Seat reservation; CCTV; Passenger counting visual (APC); Infotainment.
  - SPE offers weight reduction up to 55%: 3 kg/100 m vs. 4,6 kg/100 m with standard Eth.
  - For battery-operated vehicles, weight directly translates into better battery efficiency.
- **Automation**
- **Monitoring and reporting**
  - Example: smart monitoring of low shelf-life goods. A SPE system can use sensors embedded within the bread case itself that monitor fill level of the product on the shelves.

Some more details about Single Pair Ethernet follow. The main idea is that as we increase the data rate, we decrease the range. The goal becomes finding a trade-off to achieve the desired performance.

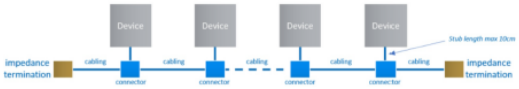
Single Pair Ethernet

- 10BASE-T1S (IEEE 802.3cg)

Data rate	Duplexing	Range
10 Mbit/s	Half-duplex	15 m (point-to-point) 25 m (multidrop topology)



POINT-TO-POINT



MULTIDROP
No switches, less cable, reduced power.

- **Physical Layer Collision Avoidance (PLCA) scheme**
  - Prevents collision in a multidrop topology.
  - A Master periodically sends beacon frames that start an ordered cycle of transmissions.
  - When it's its turn, a device can transmit or "pass the turn" to the next device.
  - Similar to a ring token-passing topology.
- Use cases: applications connecting sensors and actuators, e.g., in cars targeting compact implementations.

- 10BASE-T1L (IEEE 802.3gc)

Data rate	Spectrum	Range
10 Mbit/s	20 MHz	1000 m (point-to-point)

- Use cases: industries where remote sensing and control is required, e.g., building automation, elevators, factory automation, energy supply and monitoring, automation of waterworks, wastewater treatement.

- 100BASE-T1 (IEEE 802.3bw)

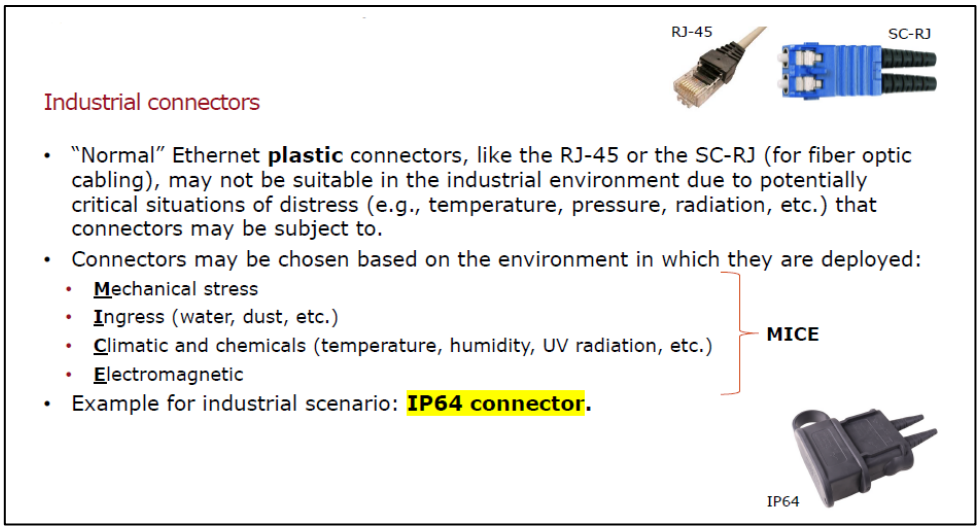
Data rate	Spectrum	Range
100 Mbit/s	166 MHz	15-40 m (point-to-point)

- Use cases: automotive industry towards autonomous driving for cooperative perception, where transmissions of large amounts of sensory data (e.g., from LiDAR and camera sensors) are required.

- 2.5/5/10GBASE-T1 (IEEE 802.3ch)

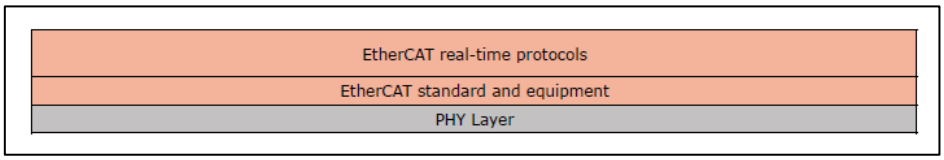
Data rate	Spectrum	Range
Up to 10 Gbit/s	4-5 GHz	15 m (point-to-point)

- Use cases: Advanced Driver Assistance Systems (ADAS), high-performance computing in vehicles, high-definition sensor fusion, real-time control of robotic systems with high-speed data transfer between industrial machines.



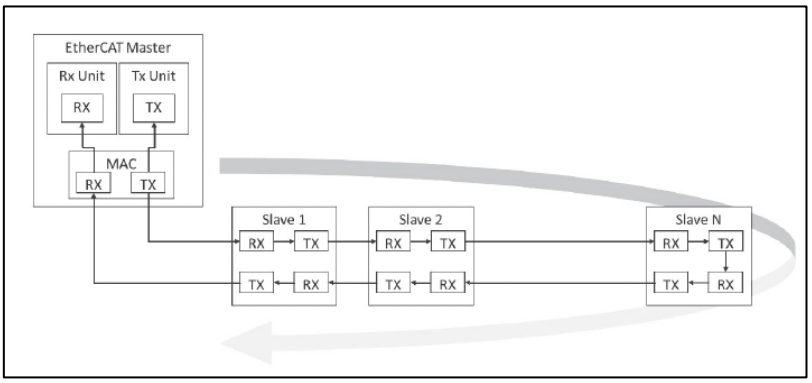
3.3.3 EtherCAT

EtherCAT is a class 3 Industrial Ethernet standard that modifies traditional ethernet to support **ultra-fast "on-the-fly" processing** speeds of up to 1 Gbit/s, making it ideal for hard real-time applications like robotics and human prosthetics, where latency is critical. This high-speed performance is enabled by custom hardware designed for rapid data processing within a **daisy chain ring topology**, allowing frames to be processed as they pass through each node with minimal delay. EtherCAT also incorporates a **Distributed Clock** mechanism, a variant of IEEE 1588, which provides high-precision time synchronization among all devices (slaves) on the network, ensuring coordinated and deterministic communication.

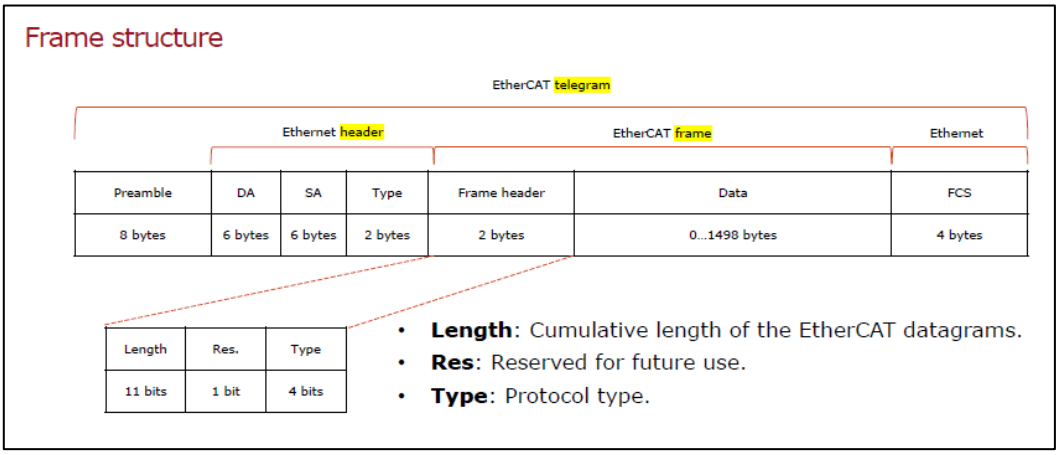


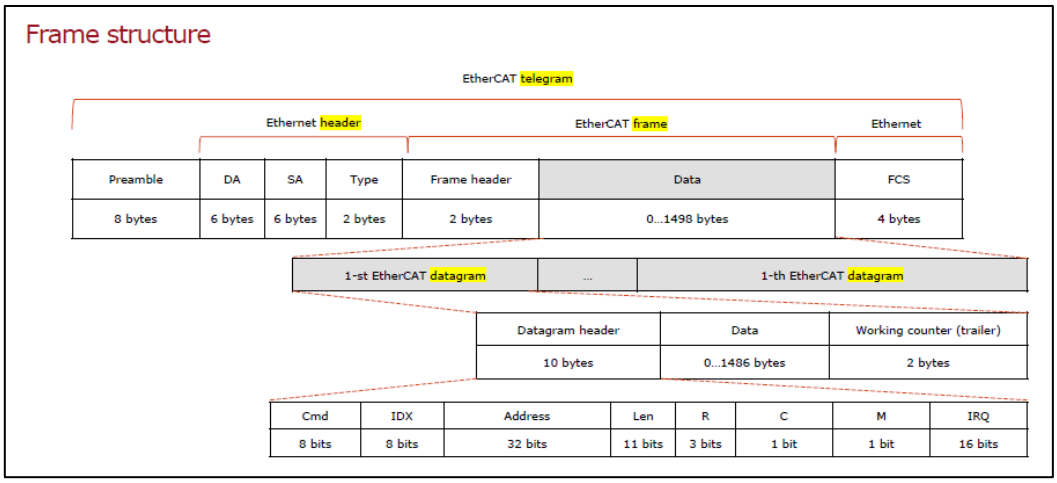
Master-slave daisy chain ring (aka bidirectional line).

- Deterministic communication (cycle time ≤100 µs).
- The Master sends frames that pass along the slaves.
- Slaves read and process the information in the frame "on the fly", inserting responses if requested (summation-frame).



A token ring MAC protocol is used to avoid collisions among different slaves. Anyway, remember all the drawbacks of using a ring topology, including the single point of failure.





- **Cmd**: EtherCAT command type.
- **IDX**: The ID used by the Master for the identification of duplicates/lost datagrams.
- **Address**: Auto-increment (configured logical address).
- **Len**: Length of the data within the same datagram.
- **R**: Reserved.
- **C**: Flag to avoid a frame to circulate forever. 0: Frame does not circulate; 1: Frame has circulated once.
- **M**: Flag to indicate that there are more EtherCAT datagrams in the same frame.
- **IRQ**: EtherCAT Event Request register of all slave devices combined with a logical OR.
- **Working Counter**: Incremented if an EtherCAT device was successfully addressed and a read/write operation was executed successfully. The Master can check whether a datagram was processed successfully by comparing the value to be expected for the Working Counter with the actual value of the Working Counter.

Why etherCAT frame is so complex? Since there is not the standard internet stack, as the IP protocol, all the functionalities are “implemented” directly in the frame, leading to a more articulated structure.

4. IoT Applications

So far, we discussed about IoT technologies mainly focusing on physical and MAC layers. Now, we move our attention on the upper layers: **transport** and **application**.

While UDP and TCP serve as the backbone of Internet transport layers, their intrinsic limitations often fall short in addressing IoT-specific challenges such as real-time data exchange, scalability, and reliability in large-scale deployments. UDP, being connectionless and simple, lacks reliability, while TCP offers guaranteed delivery but at the cost of increased complexity. To bridge these gaps, IoT systems employ **middleware**, additional protocols to realize the full support of applications. Middleware becomes a middle layer that connects the bottom layers (PHY and MAC) with the upper layers (transport and application) to support IoT requirements.

Why middleware for IoT?

- **Heterogeneous** types of data (notifications, measurements, commands, etc.).
- Both (hard-)**real-time** and non-real-time traffic (note that in a real-time environment data that arrives too late should be discarded).
- Both client-server and **publish-subscribe** models.
- **Constraints** of endpoints (complexity, powering, bandwidth, etc.).
- **Large-scale** cyber-physical system.
- Complexity of **storage**. In IoT networks data cannot be stored in the way they are stored in a common network. Storing more bits means more energy consumption!

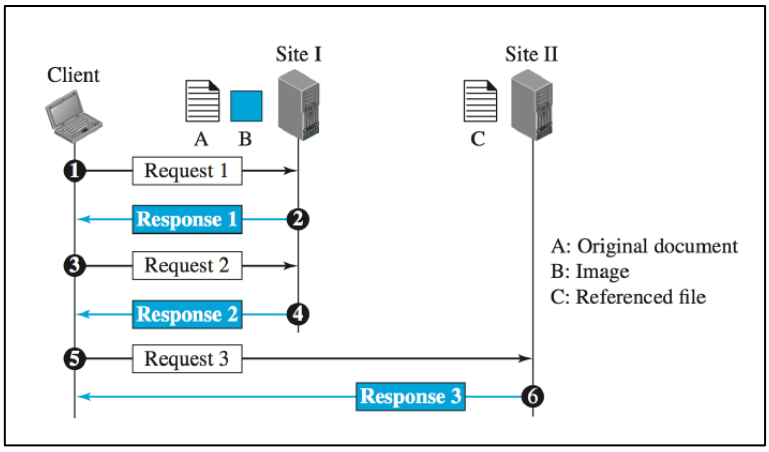
No single solution exists satisfying the requirements of all IoT networking scenarios. In this chapter, we are going to discuss about different middleware implementations considering different protocols that can be employed depending on the application requirements.

4.1 HTTP / WebSocket

The idea of the Web was proposed by Tim Berners-Lee in 1989 at CERN, to allow several researchers at different locations in Europe to access each other's research.
The Web today is a repository of information in which the documents (web pages), are distributed all over the world and documents are linked together. The linking of web pages was achieved using hypertext.
The WWW today is a **distributed client-server service**, in which a client using a browser can access a service using a server. The service is distributed over many locations called sites and each site holds one or more web pages.

World Wide Web (WWW) example

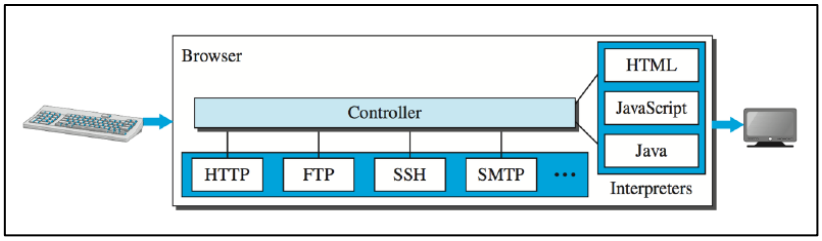
Assume we need to retrieve a document (file A) that contains one reference to another text (file C) and one reference to a large image (file B). File A and file B are in the same site; file C is stored in another site. Since we are dealing with 3 different files, we need 3 transactions to see the document.



Web client and server

A variety of vendors offer commercial browsers that interpret and display a web page, and all of them use nearly the same architecture. Each browser usually consists of three parts: a controller, client protocols, and interpreters.

The web page is stored at the server. Each time a request arrives, the corresponding document is sent to the client. To improve efficiency, servers normally store requested files in a cache in memory. A server can also become efficient through multithreading multiprocessing. In this case, a server can answer more than one request at a time.



Uniform Resource Locator (URL)

Web page has a unique identifier to distinguish it from other web pages. We need three identifiers: host, port, and path. However, we also need to tell the browser **what client-server application we want to use** to get to the path, i.e., the protocol. Hence, we need four identifiers to define the web page: host, port, path, protocol.

The uniform resource locator (URL) combines these four pieces together:

- protocol://host/path → used most of the time.
- protocol://host:port/path → used when port number is needed.
- Example: http://www.dei.unipd.it/en/department/map-department

1. **Protocol**: abbreviation for the client-server program that we need in order to access the web page, usually **HTTP** (**HyperText Transfer Protocol**).
2. **Host**: it can be the IP address of the server, or the unique name given to the server (the name is normally the domain name that uniquely defines the host → ".../www.unipd.it").
3. **Port**: a 16-bit integer, it is normally predefined for the client-server application. (e.g., HTTP uses port 80). However, the number can be explicitly given.

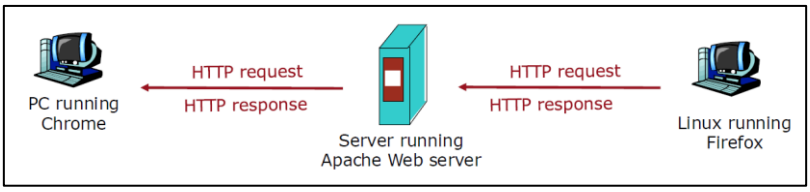
4. **Path**: the path identifies the location and the name of the file in the underlying operating system. The format of this identifier normally depends on the operating system. In UNIX, a path is a set of directory names followed by the file name, all separated by a slash (e.g., /top/next/last/myfile).

HTTP

The **HyperText Transfer Protocol (HTTP)** is used to define how the client-server programs can be written to retrieve web pages from the Web. An HTTP client sends a **request**; an HTTP server returns a **response**.

- Server uses port number 80; client uses a temporary port number.
- HTTP uses the services of TCP (more recently, some applications use QUIC). Connection establishment and termination phases are needed. Client/server do not need to worry about errors in messages exchanged or loss of any message, because the TCP is reliable and will take care of this matter.
- If Transport Layer Security is added on top of TCP: **HTTPS** (HTTP Secure).

- The client issues requests (called methods) towards a server hosting the objects.
- Depending on the method, the server's response may contain the resource data or the metadata only.
- Requests and responses may contain a large number of parameters, managing the many options available in HTTP.



Sometimes a client does not simply want to get the webpage. The requests may be different, the most popular ones are following reported:

HTTP methods

Method	Action
GET	asks for a resource to be delivered to the client
HEAD	asks only for the delivery of a header containing metadata about the resource
POST	appends data to an existing resource
PUT	writes a resource into the server's store at the specified URL location
DELETE	removes a resource from the store
TRACE	asks that the server transmits back the request as received (for debugging purposes)
CONNECT	asks for the establishment of a direct connection (a tunnel) to the resource original site, for instance passing through a proxy
OPTIONS	requests the list of methods and options supported by the server
PATCH	requests the modification of parts of the target resource (used for convenience when small modifications to a large resource must be made).

Problems for IoT traffic

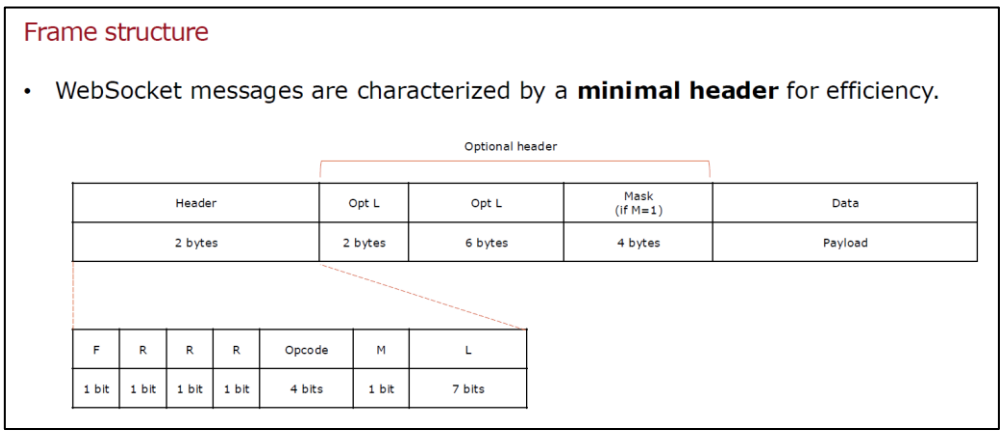
- HTTP is a very **generic protocol**, which is used for many different applications and to retrieve different types of data. In turn, IoT traffic has very specific characteristics and constraints.
- HTTP is a strict **client-server model**. In fact, it cannot accommodate event-based updates: a specific request should be submitted by the client to return something.
    - A client issues a request, and the server gives a response, but cannot receive unsolicited updates (e.g., when the state of the system changes), as in the IoT scenario.
    - Connection always starts from the client, so it cannot support **asynchronous** events.
- HTTP header is textual (using UTF-8 encoding), so it is "**verbose**" (size ~100 KB).
- HTTP implements **sophisticated options**, which may not be used for IoT traffic.
- Some solutions: HTTP/2, HTTP/3. Still, "normal" HTTP does not seem to be the ideal solution for IoT gateway-based nets.

WebSocket

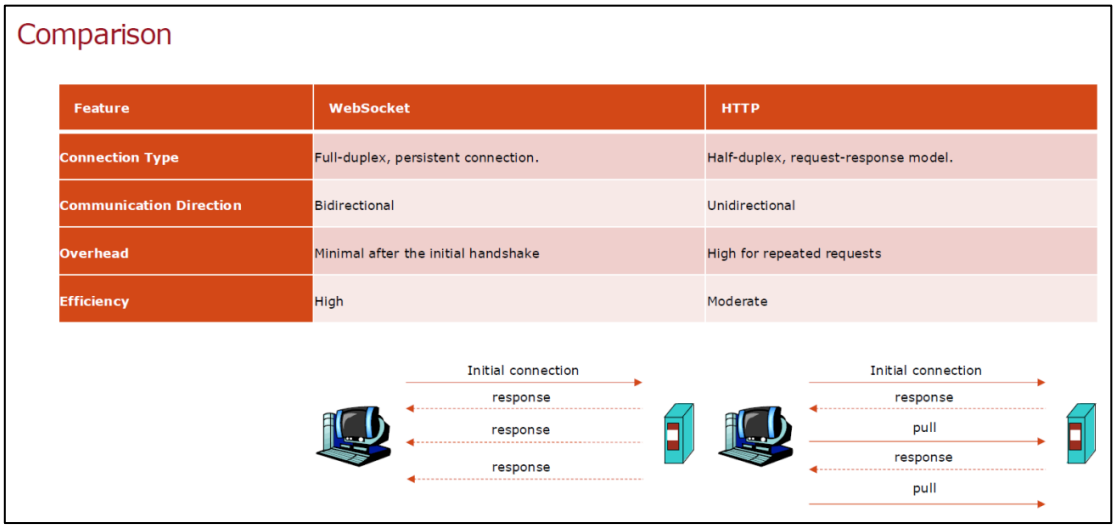
WebSocket is a new protocol used at the application layer for IoT traffic. WebSocket is defined as the combination of two elements:

1. a mechanism to open a bidirectional TCP connection for data exchange using HTTP;
2. a minimal set of messages which allow data transfer and maintenance of the connection.

WebSocket does not replace HTTP which is used by WebSocket only to open the connection between the client and the server. Data, instead, is retrieved by specific WebSocket methods. To open the connection, the client uses an HTTP GET request. The server agrees by sending a response and WebSocket communication can start. If the server does not support WebSocket, it responds with an error code.
WebSocket connection is **persistent**, so it can support unsolicited event transmissions from both sides. The main limitation is that it is still a client-server model. As already discussed, there are more effective solutions for IoT, such as the publish-subscribe paradigm.



- Note that the frame structure is much simpler compared to HTTP frame, trying to meet IoT requirements even if still based on HTTP protocol which remains the main bottleneck.
- **F** (FIN): if 1, it indicates that this is a complete message or the last fragment.
- **R**: reserved.
- **Opcode**: it indicates the type of message.
- **M**: if 1, a "mask" is included in the payload.
- **L** (Length): it indicates the length of the message. The total length of the frame becomes dynamic according to the size of the message which influences the header size:
  - If L < 126 → Payload length.
  - If L = 126 → The following 2 bytes (OptL) are the payload length (up to 65KB).
  - If L = 127 → The following 8 bytes (OptL + OptL) are the payload length (up to 9.22 EB).
- **Mask**: mandatory only for client-to-server communications. It is used to "mark" packets with a key to avoid poisoning attacks.

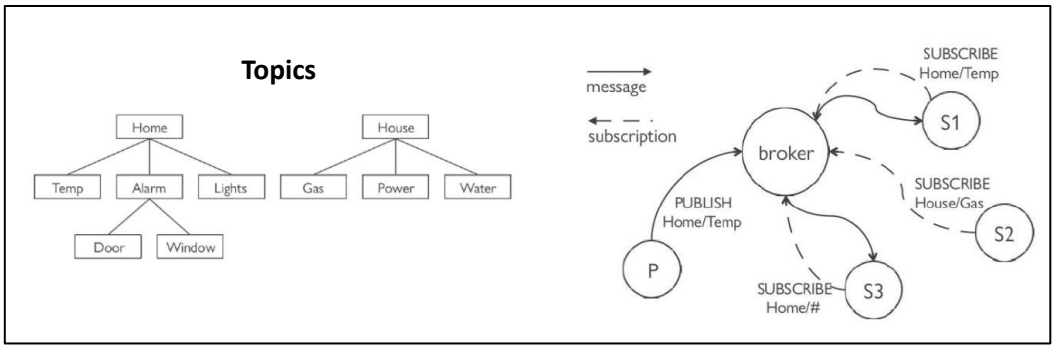


4.2 MQTT

Message Queuing Telemetry Transport (MQTT) is an example of an IoT-specific application protocol. It was specifically designed for the collection of events from sensors. It is normally layered on top of TCP or TLS, and it is implemented based on the publish-subscribe paradigm which operates in this way:

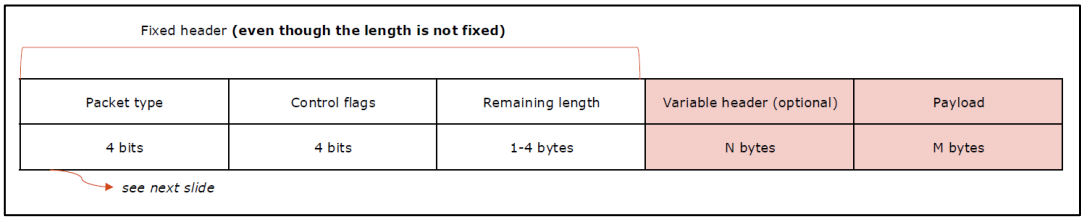
- A central **publisher** distributes messages to a **broker**.
- The broker forwards these messages to the **subscribers**.
- Subscribers never create direct connections among them.
- Subscription to **topics**.

Consider the below example where "home" data is sent by node P to the broker which then forwards it to all "home" subscribers.



FRAME STRUCTURE

- MQTT messages are characterized by a **fixed header** for efficiency.
- **Remaining length**: it represents the sum of the lengths of the Variable header and Payload fields. It is encoded using a data type called **Variable Byte Integer**, which employs a number of bytes [1-4] depending on the represented integer.
- **Variable header / Payload**: it depends on the type of message. In this way it is possible to choose the size of the message to send, depending on the quantity of measurements.



Variable Byte Integer: utilize the lower 7 bits of each byte to encode data, while the highest bit indicates whether there are more bytes to follow (1 if another byte follows, otherwise 0).

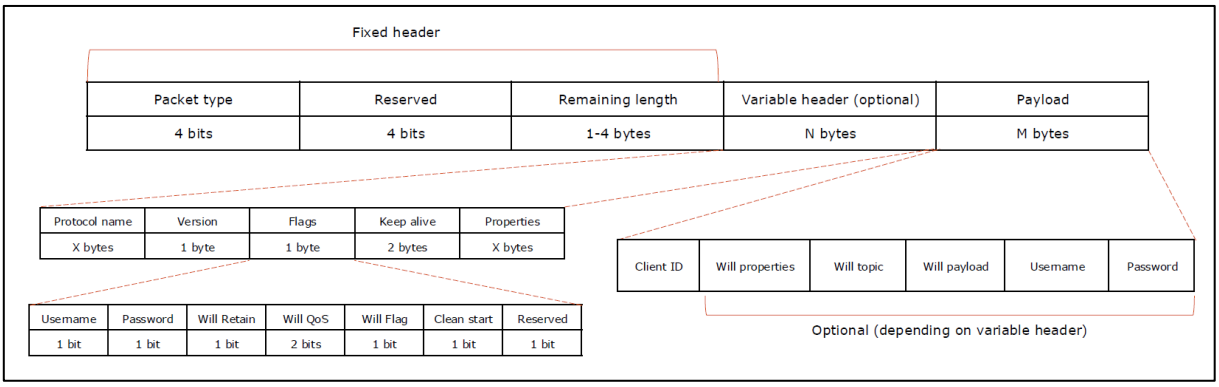
- Packet length <128 bytes: we only need 1 byte.
- Maximum length (4 bytes): up to (2^{28}-1) bytes, i.e., 256 MB of data.

MQTT messages

Group	Message	Meaning
Connection Management	CONNECT	Connection open
	CONNACK	Connection acknowledgment
	AUTH	Authentication message
	PINGREQ	Keepalive
	PINGRESP	Keepalive response
	DISCONNECT	Connection close
Subscriptions	SUBSCRIBE	Subscribe to a subject
	SUBACK	Subscription acknowledgment
	UNSUBSCRIBE	Remove a subscriprion
	UNSUBACK	Removal acknowledgment
Publishing	PUBLISH	Message with topic and data
	PUBACK	Message acknowledgment (only for QoS=1)
	PUBREC	Message received (only for QoS=2)
	PUBREL	Release pending publish (only for QoS=2)
	PUBCOMP	Release acknowledgment

CONNECT

- It deals with connection **establishment**, **control**, and **removal**.
- Aim: open/reopens a session with the broker and carries the ID of the client.



- **Username**: used to indicate whether the Payload contains the Username.
- **Password**: used to indicate whether the Payload contains the Password.
- **Will Retain**: used to indicate if the Will Message is a Retained Message.
- **Will QoS**: used to indicate the QoS of the Will Message.
- **Will Flag**: used to indicate if the Payload contains fields of the Will Message.
- **Clean Start**: used to indicate whether the current connection is a new session or a continuation of an existing session, which determines whether the server will directly create a new session or attempt to reuse an existing session.
- **Reserved** (for future use).
- **Keep Alive**: used to indicate the maximum time interval (in seconds) between two adjacent control packets sent by the client.

Retained message

MQTT has the disadvantage that subscribers cannot actively fetch messages from publishers (similar to HTTP client-server paradigm). New subscribers can get the latest data immediately without waiting, using **Retained Messages**. Consider the following network setup: two nodes S1, S2 with a broker B. Assume S2 is not already subscribed to the data published by S1. Since S2 will start receiving messages once the subscription is submitted, for S2 it may take time to receive new messages (a new publish request should be submitted by S1 to receive something). Due to that, if S1 message contains the **Will Retain bit set to 1**, the broker B stores the message which will be then delivered to new subscribers, a sort of first update.

- Upon receiving a message with the **Retain** flag set, the MQTT broker must store the message for the topic to which the message was published (store only the last message).
- Subscribers of that topic can go offline and **reconnect at any time to receive the latest message** instead of having to wait for the next message after the subscription.
- Example: sensor monitoring a value that rarely changes. The typical implementation is regularly publishing the latest value, but a better implementation is sending it only when

the value changes in the form of a Retained Message. This allows any new subscriber to get the current value immediately, without waiting for the sensor to publish again.

Will message (or "Last Will and Testament" - LTW)

Will messages, if enabled, will be published by the client if its connection terminates "ungracefully" (e.g., without a DISCONNECT or in case of protocol errors). This is done to inform other nodes about the change in the network topology and also to "mark" that node with a certain level of reliability. When the client unexpectedly disconnects, the server sends a Will Message to other clients who have subscribed to the corresponding topic.
Once the server publishes the Will Message, the problematic node will be removed from the session. If the client who cares about this Will Message is offline, it will miss this Will Message. To avoid this situation, we can set the Will Message **as a Retained Message**, so that after the Will Message is published, it will be stored on the server in the form of a Retained Message, and the client can get this Will Message at any time.

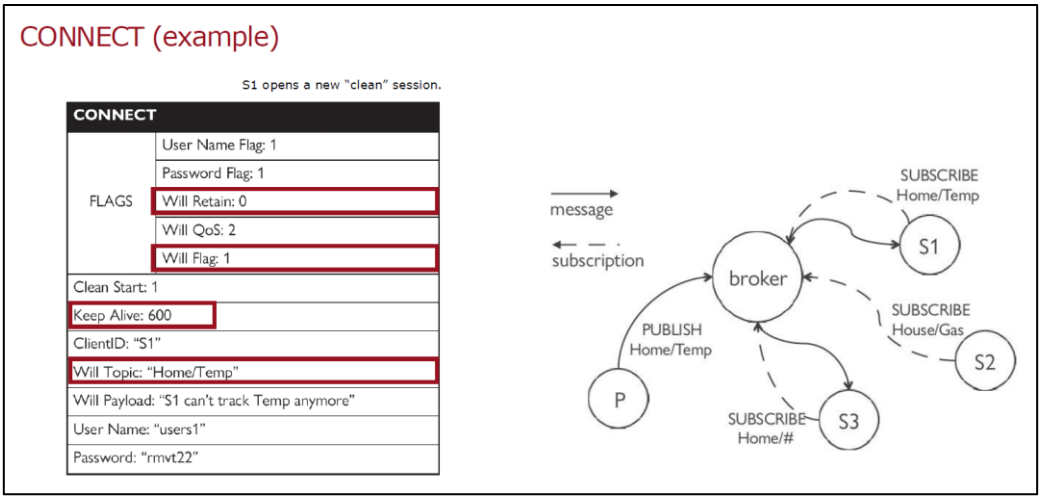
Keep Alive

MQTT protocol is based on TCP, which can have **half-connection** problems. Half-connection means that connection is disabled on one side, and active on the other side. The half-connected party may send data, which never reach the other side.

Keep Alive allows the client and MQTT server to determine whether there is a half-connection problem and close the corresponding connection.

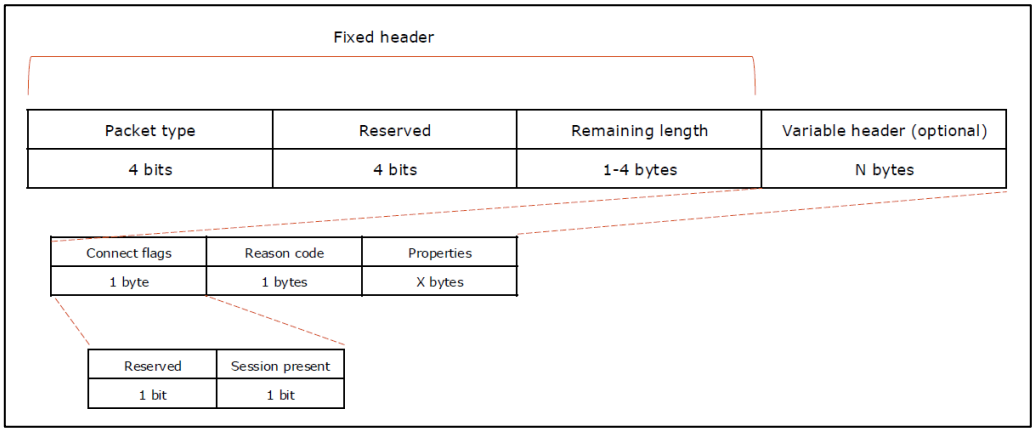
- Integer from 0 to 65535 (2 bytes).
- It represents the **maximum time** (in seconds) **allowed to elapse between MQTT pkts**.
- The client needs to ensure that the interval between any two MQTT protocol packets it sends does not exceed the Keep Alive value.
- Keep Alive is typically used in conjunction with Will Message: if the server does not receive any packets within the Keep Alive timer, it will send a **Will Message**.

Retained Messages, Will Messages and Keep Alive configuration are in fact used to improve IoT networks performance.



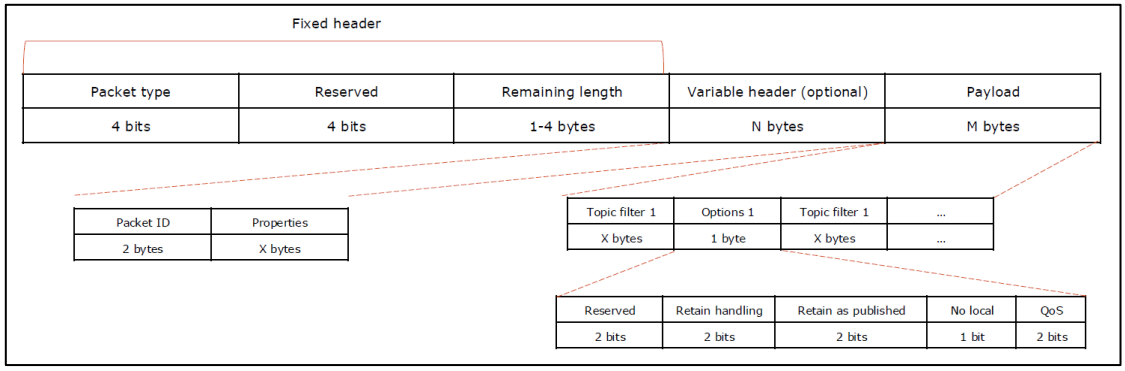
CONNACK

- Aim: sent by the server after CONNECT, to inform of the connection result.
- CONNACK has no payload.
- **Reason Code**: used to indicate the result of the connection.



SUBSCRIBE

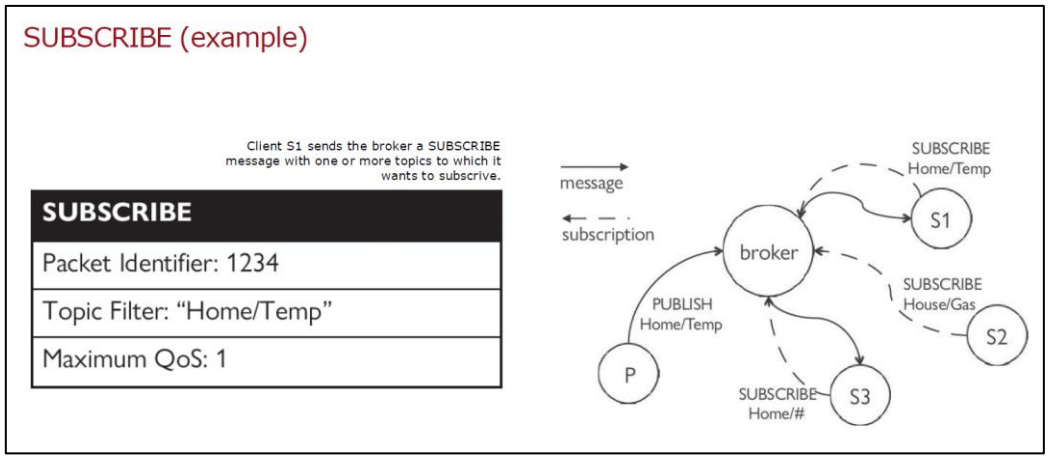
- Aim: used to initiate a subscription request for the topic(s) a client wants to subscribe to, each with a specified **QoS level**. The request is forwarded to the broker.
- Followed by **SUBACK** (and **UNSUBSCRIBE + UNSUBACK**).



- **Packet ID**: used to uniquely identify the subscription request. PUBLISH, SUBSCRIBE, and UNSUBSCRIBE packets use a set of packet identifiers, which means they cannot use the same packet identifier simultaneously.
- **Retain Handling**: used to indicate whether the server needs to send Retained Messages to this subscription when the subscription is established.
- **Retain As Published**: used to indicate if the server needs to keep the Retain flag in the message when forwarding the application message to this subscription.
- **No Local**: used to indicate whether the server can forward the application message to the publisher of the message.
- **QoS**: determines the maximum QoS level that the server (broker) can use when forwarding messages to this subscription. There can be some messages more important

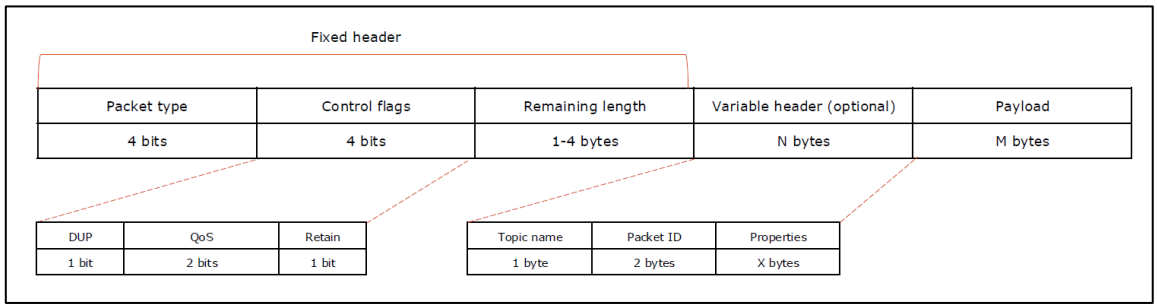
than others and a higher QoS is used to increase transmission performance for those messages.

Note: MQTT is implemented via a star topology, having the broker at the center.

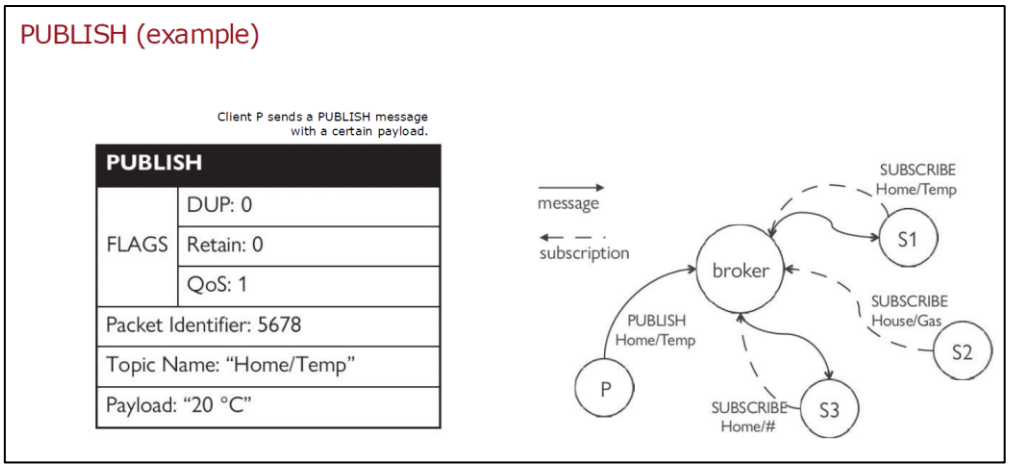


PUBLISH

- Aim: used for message distribution: the client publishing messages to the server, or the server forwarding messages to the subscriber.
- Followed by: **PUBACK**, **PUBREC**, **PUBREL**, and **PUBCOMP**.



- **DUP**: = 1 if it is a retransmitted packet (a duplicate packet). The number of packets with DUP set to 1 can reveal the quality of the current communication link.
- **QoS**: determines the QoS level of the message.
- **Retain**: = 1 if the current message is a Retained Message, meaning that it should be stored by the broker and delivered to any new subscriber. It allows an immediate update of new subscribers.
- **Topic Name**: used to indicate which channel the message should be published to.
- Packet Identifier: used to uniquely identify the message currently being transmitted. It only appears in the PUBLISH packet when the QoS level is 1 or 2.
- **Payload**: the content of the application message we send.

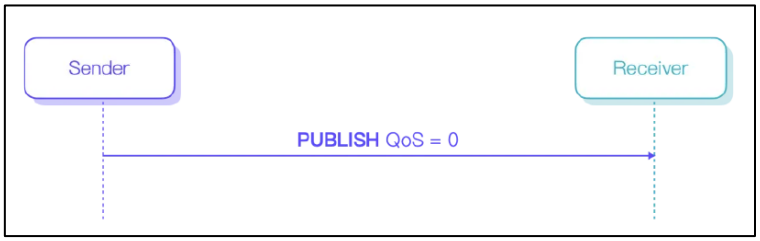


QoS levels

In unstable network environments, MQTT devices may struggle to ensure reliable communication using only the TCP transport protocol (think about real-time applications where reliability must be sacrificed to reach real-time performances). MQTT includes a Quality of Service (QoS) mechanism to provide the message with different levels of service, catering to the user's specific requirements (MQTT5.0: QoS should not be applied if the transport layer is working properly, as TCP should already provide the required level of reliability). QoS level is specified in the subscription request.

QoS 0: At Most Once (aka "fire and forget").

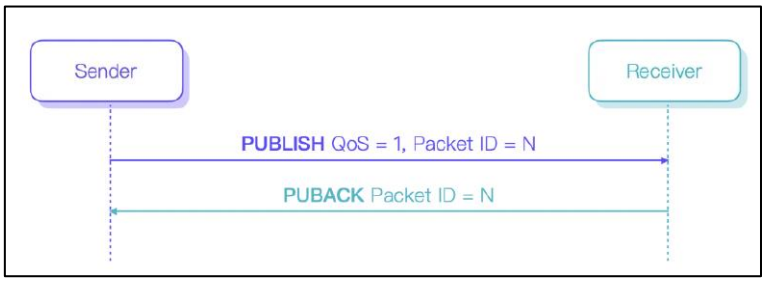
- The sender does not wait for acknowledgement or store and retransmit the message, so the receiver does not need to worry about receiving duplicate messages since the message is not retransmitted.
- In this case the transmission is simple and fast, having so a low latency and low reliability level.



QoS 1: At Least Once.

- When the sender receives a PUBACK packet from the receiver, it considers the message delivered successfully. Until then, PUBLISH packet is stored for potential retransmission.
- Packet ID is used to match the PUBLISH packet with the corresponding PUBACK packet.
- In this case the transmission time is longer but, on the other hand, we are sure about the message delivery.
- The sender can trigger multiple transmissions for the same message but here the duplicated messages are not controlled. The sender only wants to ensure that at least

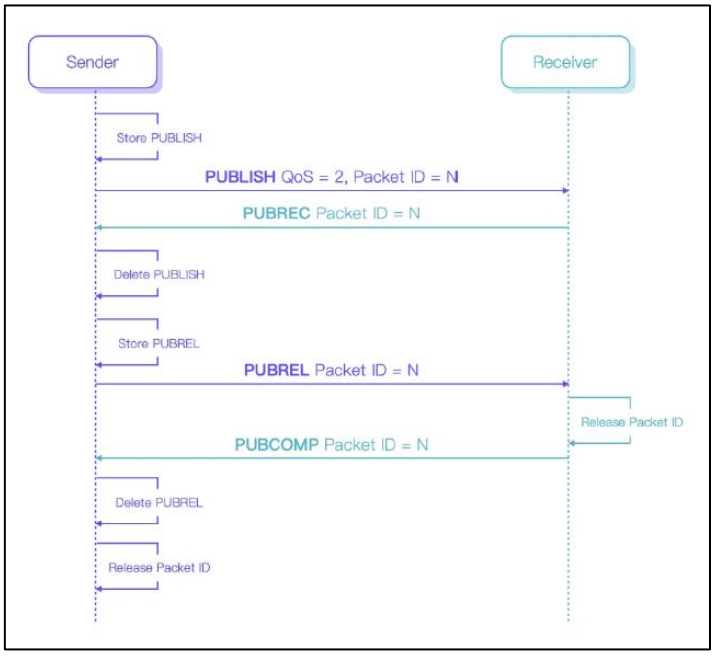
one copy of the message is delivered to the broker. So, in terms of efficiency and resources consumption, this approach is not so efficient.



QoS 2: Exactly Once. Ensures that messages are not lost or duplicated, unlike in QoS 0/1.

1. The sender stores and sends a PUBLISH packet with QoS 2 and then waits for a PUBREC response packet from the receiver. This process is similar to QoS 1, with the exception that the response packet is PUBREC instead of PUBACK.
2. The sender can delete its locally stored copy.
3. The sender sends a PUBREL packet to release the Packet ID. It is stored for potential retransmission.
4. The receiver responds with a PUBCOMP packet. This is the main difference with QoS 1. Here the sender cannot generate multiple copies of the same message because the PUBCOMP reception is needed before retransmitting the packet.

- Anyway QoS 2 is more complex to implement and handle. The good part is that we can choose the QoS level depending on our system constraints and requirements.



MQTT final considerations

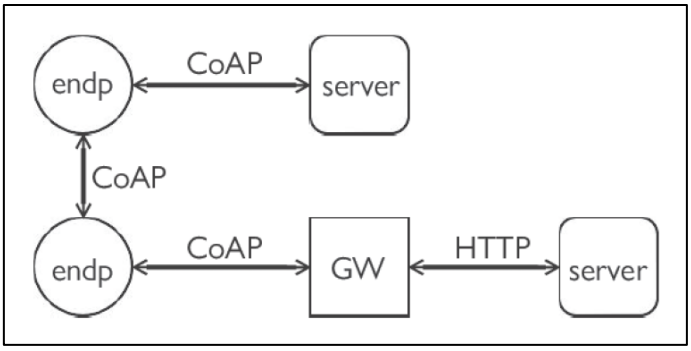
- It is a very good choice for IoT systems with constrained endpoints.
  - Lightweight protocol.
  - Can be implemented on many embedded platforms (Arduino, Raspberry Pi, …).
  - Several open-source implementations.
  - Retrain mechanism, to implement the seamless addition of new endpoints.
  - LTW, to ensure good management in case of connectivity outages.
- Main drawback: centralized broker can create scalability and reliability issues. The broker may not be able to satisfy all the requests requiring more time in forwarding messages (or even worse).

4.3 CoAP

Constrained Application Protocol (CoAP) is another example of an IoT-specific application protocol. It was designed to allow for small, low-power devices with limited computational resources, scarce bandwidth, and long sleep times, to use the wider Internet.
It is normally layered on top of UDP (vs. TCP for MQTT). This makes CoAP **unreliable by design**. To provide security, it uses IPsec (at the network layer) or DTLS.
CoAP is implemented based on the **client-server** paradigm. Messages are a sequence of requests (aka methods) and responses, like in HTTP. It is not equivalent to HTTP, but internetworking across the two protocols is easy. Unlike HTTP, CoAP devices can «**observe**» the state of resources, similar to the publish-subscribe paradigm.

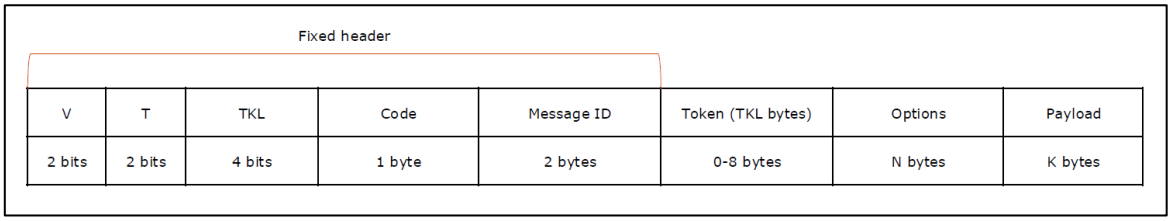
Architecture

Endpoints (clients) may talk to a CoAP server or an HTTP server through a gateway. They can also communicate directly in **peer-to-peer**. The "lightweight" characteristics of the protocol make the configuration of a CoAP server feasible even on resource-constrained endpoints. The CoAP server may act as a client too.
This architecture **avoids the MQTT single point of failure**. In fact, MQTT uses only one server (broker) while CoAP allows to have different servers that manage different requests. Another difference with MQTT is that here end nodes can communicate each other, acting both as publishers and subscribers.



Frame structure

- Given the use of UDP, it requires dedicated **error recovery mechanisms**.
- The format (and header) of messages is **more compact than HTTP**.
- The format (and header) of messages may be slightly **more complicated than MQTT.**

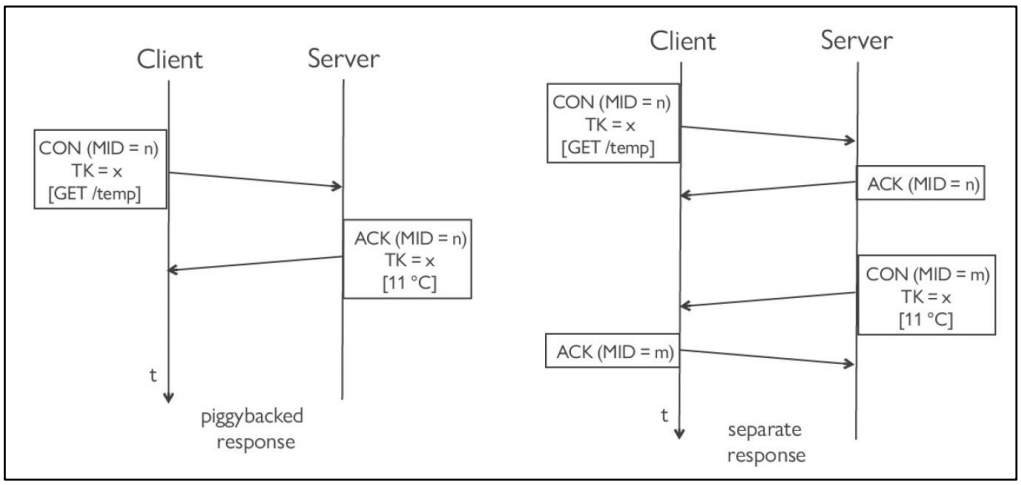


- **Version (V)**
- **Type (T)**: **Request** (Confirmable, Non-confirmable), **Response** (ACK, Reset).
  - Confirmable (0): this message expects a corresponding acknowledgement message.
  - Non-confirmable (1): this message does not expect a confirmation message.
  - Acknowledgement (2): a response that acknowledges a confirmable message
  - Reset (3): indicates that the receiver cannot (or does not want to) process a request.
- **Token Length (TKL)**: length of the Token field in the header (may be 0).
- **Code**: it is the code (ID) of the method/request or response. It consists of a **class** (3 bits) and a **code** (5 bits).
- **Message ID**: used to detect duplicates and/or retransmissions.
- **Token**: used to match a response with its request (may be 0 if no ambiguity).

Token vs Message ID

- **Message IDs** are unique values that are used to identify duplicate messages and match Confirmable messages to Acknowledgement messages.
  - Messages are paired to their ACKs by the Message ID.
- **Tokens** are unique values used to match Requests to Responses.
  - Requests are paired to their responses by the Token.
  - Matching requests and responses is not done with the Message ID because a response may be sent in a different message than the ACK (which uses the Message ID).
  - Responses may be sent as separate response or piggybacked.
    - **Separate response**: useful for low latency, if the ACK is generated after some time → hard real-time scenarios. **Same Token, different ID**. However, it requires an ACK for confirmation, having so two messages (separate responses) sent by the server.
    - **Piggyback**: response is sent in the ACK relative to the (Confirmable) request. **Same Token and ID**. It does not require an ACK (if client in Confirmable mode already expects an ACK).

- Token may not be used in non-confirmable mode, or if there is a single pending request.



Requests (or methods)

- Some example of the most popular CoAP requests (aka methods).

Method/request (0.xx)	Code	Action
GET	0.1	Asks for (the state of) a resource
POST	0.2	Requests the processing of the message content, possibly resulting in the creation or update of a resource.
PUT	0.3	Substitutes the resource specified in the URI or, if non-existing, creates a new one
DELETE	0.4	Removes the resource

Responses

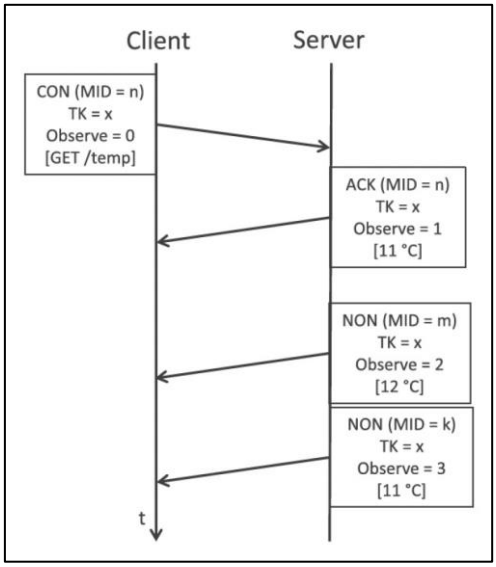
- Responses are characterized by some **codes**.
- Some example of the most popular CoAP responses.

Responses	Code	Action
SUCCESS (2.xx)		
Created	2.1	Resource has been created
Deleted	2.2	Resource has been delated
Content	2.5	Carries the content of the resource
CLIENT ERROR (4.xx)		
Unauthorized	4.1	The client is not authorized to perform the requested action.
Request Entity Too Large	4.13	The size of a client's request exceeds the server's limit.
SERVER ERROR (5.xx)		
Not implemented	5.1	The server cannot fulfill the client's request (e.g., the server does not recognize a certain method).

Resource observation allows the possibility of a device to be asynchronously updated on the state of a resource. Useful in IoT since sensory measurements may be unsolicited.

- GET message (with "Observer" option set to 0): the "observer" will receive a sequence of responses with the same Token, all carrying information/measurements about the state of the resource.
- Sequence number is increased at each response for the observer to do reordering.
- Observation ends (1) after a Reset, or (2) after a GET message with "Observer" option 1.

Resource observation allows for the server to send unsolicited messages. The rate of update depends on the nature of the resource (e.g., resolution of measures). For example, the resource representing a temperature sensor may be defined with a 1 °C resolution: in principle, the CoAP server will generate one-tenth of the updates of a 0.1 °C resource defined for the same sensor. Furthermore, rate-limiting mechanisms may be in place to reduce flooding of messages.

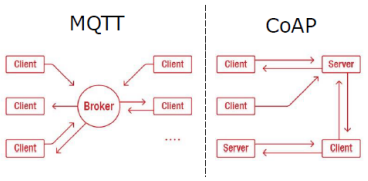


CoAP final considerations

- It is a competitor of MQTT given some nice features:
  - Simple design (good for IoT networks with many resource-constrained end nodes).
  - Support for asynchronous events with no strict real-time requirements.
  - Alignment and interoperability with HTTP.
  - REST model (easy for Web designers to build/implement CoAP applications).
- It comes with some drawbacks: even though it has an "Observer" method, it is still a client-server application. In order to implement a kind of publish-subscribe paradigm, it is more convenient to use MQTT.

CoAP vs. MQTT

Comparison



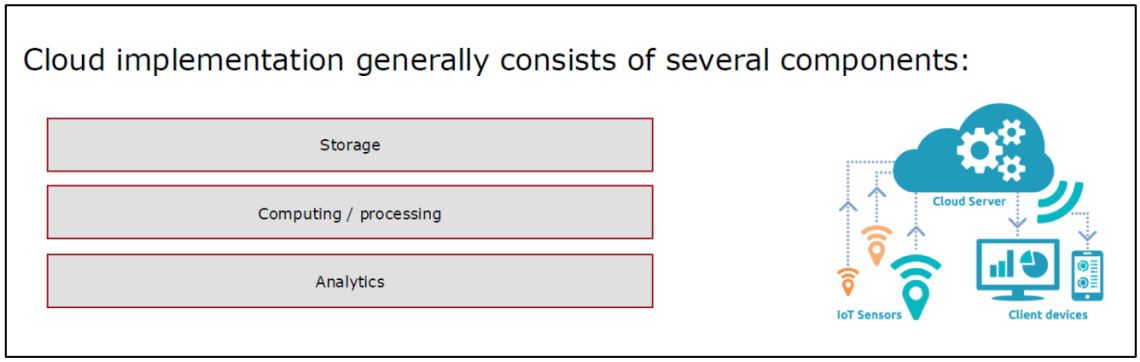
Feature	MQTT	CoAP
Purpose	Messaging and communication in IoT	Designed for resource-constrained devices in IoT
Transport Protocol	TCP (Transmission Control Protocol)	UDP (User Datagram Protocol)
Communication Style	Publish/Subscribe model	Request/Response (Client-Server-like) model
Header Size	2 bytes fixed header	4 bytes fixed header
Payload Format	Supports binary and text payloads	Supports binary and text payloads
QoS (Quality of Service)	Levels 0, 1, and 2 for message delivery	Reliability through "confirmable" and "non-confirmable" messages
Message Types	Publish, Subscribe, Connect, Disconnect, etc.	GET, POST, PUT, DELETE, etc.
Resource Discovery	Not inherent, requires additional mechanisms	Built-in resource discovery through CoRE Link Format

5. IoT Cloud

"Back-end" processing is depicted by a generic **cloud**. Data from a variety of diverse sources are aggregated and processed for optimization and discovery of global trends and relations. Depending on nature and real-time requirements, sensor data may be processed "in-flight" as streams, stored for post-processing and archival purposes, or both. It may also contain some common services such as large-scale storage, analytics-processing engines, data visualization and graphing, as well as management functions such as security and provisioning. Machine learning (ML) and artificial intelligence (AI) algorithms are usually operated in the cloud where they can work with large aggregations of data.

The cloud is the point of high-level **aggregation** and **processing** of data in an IoT system. Aggregation and processing may be located in different "places":

- At the **lower levels** (e.g., at the end nodes): limited data sets, limited network and computational resources, more energy constraints, local data, etc.
- At the **cloud level**: can span multiple IoT domains to provide **system-level insights**.



5.1 Cloud computing

Cloud computing allows on-demand ubiquitous access to resources such as servers, storage, network, and services. The resources are **virtualized** (allowing to have different cloud structures allocated to the same hardware, each one its "private" space) and can be **dynamically allocated** based on the received requests, forming a shared pool of configurable computing resources. **Elasticity** allows the amount of resources to be allocated to grow and shrink dynamically, adjusting to changes in load and demand. Cloud systems provide **externally visible access points** with ample bandwidth for data ingestion, enabling end users to easily upload and activate services.

Cloud services can be classified as:

- **Public**: the resources are provided by the commercial cloud so that everyone can access them through subscription (e.g., Google Drive, AWS).
- **Private**: the resources are managed within an enterprise. This means more control and privacy. They can generally count on a more limited set of dedicated resources, the ones

the company decides to allocate. Typically, if you want more you have to change something, such as the Cloud structure.

- **Hybrid**: the resources are split into a private and a public portion. This allows to meet security, regulatory and performance demands.

Cloud computing has several strengths compared to local computing such as:

- The computing resources in the cloud appear to be infinite.
- The computing resources can be **scaled** on-demand in accordance with load variations.
- There is no need for a big starting investment in hardware resources (good for start-ups).
- Simplified operations via **virtualization** and **multiplexing of workloads**.
- Potentially increased **reliability**, **resilience**, **security**.

Virtualization

Virtualization decouples workload by encapsulating it into a **Virtual Machine (VM)**. A VM can be defined as a "*compute resource that uses software instead of a physical computer to run programs and deploy apps. One or more virtual "guest" machines can run on a physical "host" machine, with each virtual machine running its own operating system and functioning separately from the other VMs, even when they are all running on the same host.*"

The **hardware and software can be decoupled**, and VMs can be activated when required and dimensioned at will. This creates a **scalable** computing infrastructure (easily allocating more resources when necessary), allowing response to traffic spikes when needed, and follows a **pay-per-use** approach with respect to computing resources, which is advantageous for start-ups. There is no need for an accurate estimate of the needed computing capacity.

The physical infrastructure of the cloud is composed of **multiple data centers** deployed in different places and even in different countries, this also to avoid the single point of failure. This approach has both technical and legislative implications:

- By replicating important data, **redundancy** is realized.
- If the same resources are available in different locations, **latency** can be reduced.
- It allows compliance with **different data protection regulations**.

Some disadvantages of the cloud are:

- Cloud **outages** can cause the unavailability of the hosted resources.
- **Real-time constraints** may be not satisfied due to network latency. Consider that it is not always true that on-site computing is more efficient in terms of latency, compared to the external cloud computing. This depends on the available computation power.
- A vendor **lock-in** may occur, as most cloud computing solutions are proprietary.

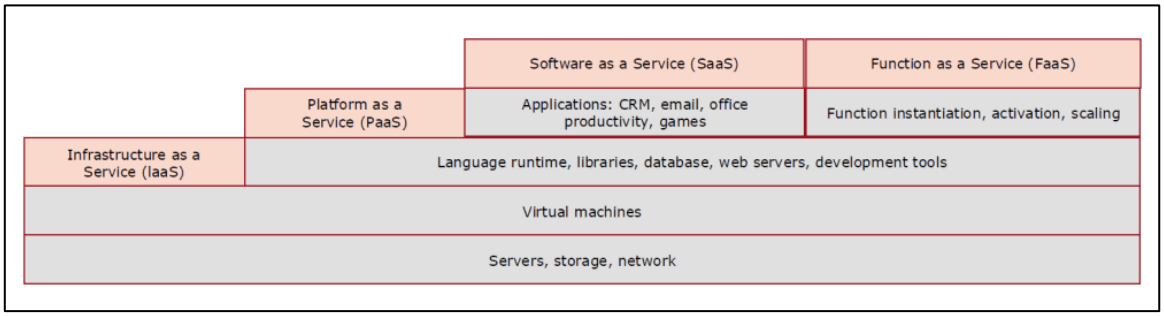
Cloud computing requirements:

- **Data distribution**: how fast/well data is distributed from the source (sensor) to the consumer (actuator, applications etc.), after (cloud) computing.
- **Scalable storage**: data may not be accessed immediately after collection. Data can be stored to process it later rather than immediately after receiving it.

- **Processing service**: temporal and spatial aggregation and correlation between multiple devices can provide useful insights.
- **Flexibility**: heterogeneous sources must be handled. Different sensors that measure different types of data should be handled, that is flexibility.
- **Reliability**: the application scenario may have strict QoS requirements.
- **Scalability**: the cloud infrastructure should provide an adequate QoS independently from the number of information sources and consumers.
- **Security**: collected data may be sensitive, so that they must be properly handled.

There are different models that shape how cloud computing services are formed:

- **Infrastructure as a Service (IaaS)**
- **Platform as a Service (PaaS)**
- **Software as a Service (SaaS)**



Infrastructure as a Service (IaaS)

The cloud vendor provides virtual machines, networking, and storage resources, with users responsible for executing VM images on their operating systems, providing runtime environments, middleware, and applications. Users must monitor and patch VMs themselves since VMs are not run and executed by the vendor side which offers only the access to its resources → local computation at the client side.
Benefits of IaaS include flexibility, automation, cost reduction, control, and scalability, while drawbacks include compatibility with legacy systems, the need for internal training (maintenance and execution of VMs), and security concerns (the need to update VMs to meet security standards).
Examples: AWS, IBM Cloud, Microsoft Azure, Cisco Metacloud, Oracle Cloud.

Platform as a Service (PaaS)

PaaS provides developers with a framework and tools to build applications tailored to their organization's needs. The vendor is responsible for updating and monitoring the runtime system.
Benefits of PaaS include cost reduction, scalability, migration ease, less coding, and development freedom, but drawbacks include data security issues, runtime challenges, and integration limitations (the created ad-hoc apps should be able to talk with VMs which have a non-proprietary implementation).
Examples: Google App Engine, OpenShift, Force.com, Apache Stratos, Magento Commerce.

Comparison: IaaS vs PaaS

- IaaS offers a great deal of control over your operating systems.
- PaaS can build apps without having to host them on-premise, so to benefit from more flexibility but with a little less control.

Feature	IaaS	PaaS
Control	Full control over infrastructure	Limited control, focus on application logic
Management	User manages OS, middleware, runtime, and apps	Provider manages OS, middleware, and runtime
Flexibility	Highly customizable	Pre-configured, less flexible
Skill Requirement	Requires expertise in managing infrastructure	Suitable for developers, minimal infrastructure skills
Use Cases	Data storage, legacy application hosting	Rapid app development, microservices

Software as a Service (SaaS)

SaaS provides complete applications managed by the vendor, which users can access through web browsers or mobile apps.
Benefits of SaaS include cost reduction, scalability, integration, and ease of use, while drawbacks include security concerns, limited customization (not allowed to implement proprietary solutions), interoperability challenges (especially with in-house applications), reduced control and wasted resources (bundle packets).
Examples: Salesforce, Dropbox, Slack, Google Apps, Microsoft Office 365.

Cloud computing models

Comparison

Managed by the user

Managed by the service provider

On-site	Iaas	Paas	Saas
Applications	Applications	Applications	Applications
Data	Data	Data	Data
Runtime	Runtime	Runtime	Runtime
Middleware	Middleware	Middleware	Middleware
O/S	O/S	O/S	O/S
Virtualization	Virtualization	Virtualization	Virtualization
Servers	Servers	Servers	Servers
Storage	Storage	Storage	Storage
Networking	Networking	Networking	Networking

Comparison: SaaS vs PaaS

- PaaS is generally used to build new products on top of already existing network.
- SaaS products are entirely managed by the vendor and ready to use by the team.

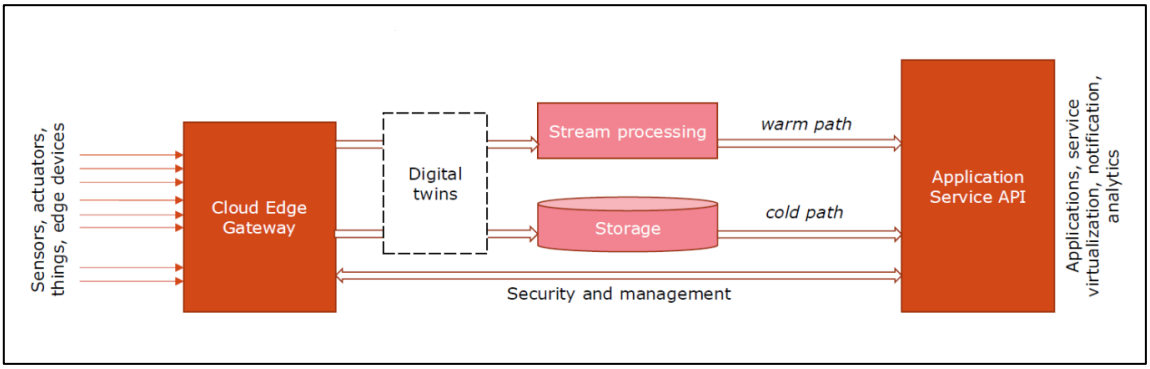
Feature	SaaS	PaaS
Purpose	Provides fully operational software	Provides a platform for developing applications
Target Users	End-users	Developers and IT professionals
Customization	Minimal customization, mostly configuration	High customization for app development
Management	Provider manages everything	Users manage apps and data, while the provider manages the platform
Skill Requirement	None (general use)	Development and deployment skills needed

In terms of market share, IaaS has been around the longest, offering cloud services, such as databases, event processors, notification systems, AI toolkits, and visualization. There are some nascent offerings of IoT PaaS and SaaS that are somewhat experimental in terms of features and business models.

5.2 Cloud components

In order to support IoT operations, the cloud infrastructure typically includes:

- An edge interface for data injection.
- Time-series data stream processing.
- Data aggregation and storage.
- Security and management.



Cloud Edge Gateway

The cloud edge gateway represents the boundary between the edge and the cloud, similar to the role of IoT gateways but within the cloud's security perimeter. Data processed here may include **sensor readings** sampled at regular or irregular intervals, **notifications** triggered by alarms, **timestamps**, and **metadata** like origin information and contextual details. Different IoT environments may have varying security policies, and this gateway serves as the **secure entry point to the cloud**, capable of performing authentication, access control (which data is accepted and which not), and filtering functions.

Incoming data can arrive at the cloud edge gateway either directly from IoT devices or through IoT gateways. This data may follow the "**warm path**" for stream processing or the "**cold path**" for storage. The gateway routes data to the correct destination based on type, topic, and content, handling high and diverse volumes.

The data volume to handle may be very high and diverse → **data translation** may be necessary for format differences (also to allow the routing algorithm to easily read data), and scalability techniques balance loads. Outgoing data, directed towards the edge, includes actuation commands and configuration information.

Digital twins

Digital twins are defined as "*synchronized cyber representations that mirror the states and behaviours of physical things*" (a sort of simulator). They **predict physical behaviour without affecting the actual infrastructure**. They can reduce latency for commands compared to direct communication, and act as always-on test platforms. Digital twins reduce bandwidth and power usage by providing access to virtual models instead of directly accessing devices.

Developing a digital twin for an actual physical object involves a process:

- Creating a virtual model of the physical product (e.g., CAD) → the most critical part.
- Adding sensors or other devices that can collect relevant performance data.
- Feed real-time data into the software where the virtual model is hosted.
- Run simulations of a product's performance in different environments and develop or test new iterations.
- Develop a new physical product which can be fitted with monitoring devices to create new product digital twins.

Note: looking at the initial figure that represents all the components that build a cloud system, it is possible to note that digital twins set is dashed because it is something not necessarily mandatory in a cloud system. Anyway, note that real sensor data are provided as input for the digital twins to refine time by time the virtual model, allowing to represent always more realistic scenarios.

Stream processing

Stream processing in the cloud occurs in **real-time** for data flowing directly from source to consumer (via the warm path). It handles tasks like **detecting anomalies**, **transforming data**, and **aggregating input** from multiple domains. During normal operations, data within bounds undergoes simple processing for visualization, whereas critical events trigger complex processing and further transformations.

Storage

IoT data and events ingested from edge sources are stored for batch processing and archival purposes, enabling long-term comparisons and audits. Computationally intensive tasks, such as machine learning and AI, utilize these data archives for creating and refining inferencing models. Data storage strategies include **short-term storage** for recent access and **long-term storage** for historical analysis and machine learning model training.
Short-term storage retains data for predefined periods, allowing fast access to recent history, typically lasting hours or days. Long-term storage aggregates and down samples high-precision data for trend analysis and archival purposes. It also supports training and testing of machine learning and deep learning models.

Storage: type of data

IoT data share the basic properties of big data:

- **Volume**: the amount of data is huge (even though of smaller size).
- **Velocity**: data production is fast, and its processing and management need to be fast.
- **Variety**: data may come from different types of sources and have different formats.
- **Veracity**: data may contain errors, missing samples.

IoT data have unique features:

- **Time-series data**: data are usually sampled at regular or irregular intervals in time. Each sample is usually associated to a timestamp.
- **Semi-structured** and **unstructured** data.
- **Data management**: different applications can have different policies for data storage.
- **Writing and accessing data**: data may be accessed sequentially or randomly.

Storage: databases

Cloud implementations of IoT storage services typically consist of:

- Input stage: receive incoming data through posting and subscriptions to the topics to be stored and write them in the database.
- Output stage: the storage service implements queries and APIs for data retrieval.

The central and most critical part of storage process is the **database**, where data is actually stored. Databases are generally distributed, having thousands of servers in multiple geographic regions. Capacity, scalability, and fault tolerance by means of partitioning and replication. The selection of the type of database for IoT systems is primarily based on its fitness for representation and querying of IoT-style data and metadata which tend to be variable-length collections of time-stamped semi-structured and unstructured data.

Storage: flat file

In early database implementations, data complexity resided in the program, with data stored as simple records in flat files like CSV or binary formats. Flat files lack explicit relationships between data records and require manual management.
Flat files are simple and clear, but they present disadvantages for IoT networks. Updating data structures is complex since changes require updates to all dependent programs. Performance issues arise with large files, data protection is absent, and accessing data can be time-consuming.

Storage: Relational Database Management System (RDBMS)

RDBMS organize data in two-dimensional tables of rows (records) and columns (attributes), where each cell contains an atomic value. Logical requests to the database are translated by the DBMS into commands for data operations. The fixed and well-defined structure of RDBMS is beneficial for logical data management, but they are not always suited for IoT's semi-structured data.

Example on financial transactions: a transaction may need to debit one account and credit another with the same amount in a consistent manner (both operations are successful, or neither is in the case of failures).

Legacy RDBMS are generally **not a good fit for the storage of IoT data**.

- Impose a rigid schema on data (vs. variable semi-structured IoT data).
- Any changes of format or addition of new types of sensors may require redesign of the schema to accommodate their data models and reconfiguration of the database.
- Slow and expensive process.
- Lower throughput and scalability due to the transactional overhead.
- High licensing and maintenance costs.

Storage: NoSQL Databases

It is NOT a Structured Query Language (SQL). This is to distinguish NoSQL from the traditional commercial databases designed for structured data and commonly queried using the variants of SQL. NoSQL says what those systems are not, but it does not indicate what they are.

NoSQL databases are designed to satisfy the following properties:

- Easily expansion as data grow.
- Guarantee low latency even when the number of requests increases.
- Do not support transactional processing and guarantees (so lower overhead).
- Scalability: designed for big-data and web applications with high data volume.
- Support multi-node and clustered implementations (geographic distribution and replication).

Key-value stores:

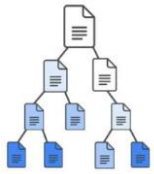
- Data are stored as a collection of key-value pairs.
- No need for a specified structure since each key-value pair is stored as a single record.
- Keys are unique identifiers for values. They can be generic numbers, or specific descriptions of the values they are associated with.
- Values can be as complex as required (numbers, sentences, and even key-value pairs).
- Example: dictionary (word: key; word meaning: value).
- Use case: record sessions in applications that require logins. Each session can be marked with identifiers and all the corresponding information is associated to it (profile info, targeted offers, payment details…).

Key-value stores **advantages**:

- Simplicity: they are simple to use and to define.
- Speed: simplicity implies that the requests can be quickly satisfied.
- Scalability: they can scale both vertically and horizontally.

Key-value stores **disadvantages**:

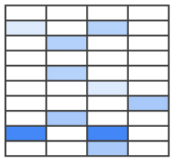
- Simplicity: depending on the application, they could be too simple.
- No filter possibility: values are seen as a single element, therefore the whole value is always returned thus preventing the possibility of extracting a specific information.
- No query language.



Document-oriented databases:

- Similar to key-value stores (keys are used as unique identifiers).
- However, **keys are associated to a document**.
- The content of documents is classified using metadata (so the database "understands" what class of information it holds).
- They have a query language, which allows querying documents based on the metadata.
- Documents can be of variable length (additional benefit in IoT systems as new types of sensors with different message formats, and lengths are introduced).

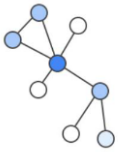
Document-oriented databases offer the advantage of flexibility, as documents do not require consistency, making them suitable for diverse and dynamic data structures. However, they have a significant disadvantage: while it is possible to create links between documents, this process can add complexity to the database, potentially impairing its overall performance.



Column-oriented databases:

- Data is organized as **rows and columns**.
- At first sight, they may look schema, relational databases. However, the key difference is that column-oriented databases store data by columns, rather than by rows.
- They have a dynamic schema, and each column is stored separately.
- Designed for petabytes of data spread across thousands of servers in multiple data centers.
- Use case: search applications.

Column-oriented databases provide notable advantages, including scalability through massive parallel processing and responsiveness with reduced load and query times. However, they also come with disadvantages, such as inefficiency in updating single column fields and slower performance for row-specific queries, which can limit their utility in certain applications.



Graph databases:

- Use graph structures for semantic queries of data.
- They consist of nodes and edges. **Nodes** are the entities (i.e., the agents and objects) of the relationship. **Edges** represent relationships between entities.
- Graph-based structure can help in terms of **data visualization**.
- Use case: representing complex structures and relationships in IoT systems (e.g., layout and connections among the components in an HVAC system of a large building).

Graph databases offer the advantage of a relationship-oriented representation, making it easier to identify trends and recognize the most influential elements within a dataset. However, their primary disadvantage lies in scalability, as it can be challenging to distribute and manage graph databases effectively across multiple servers.

Cloud-based IoT has the following limitations:

- **High or unpredictable latency**: for some applications (such as industrial automation) the low latency is a key requirement. However, the physical distance between the cloud infrastructure and the devices often defines the minimum latency that can be achieved.
- **High bandwidth requirement**: if the gateways do not have enough bandwidth capacity, they cannot access the cloud-based services.
- **No in-network filtering or aggregation**: in cloud-based IoT systems, all raw data from devices is sent directly to the cloud for processing. This can be inefficient for applications where only summarized or aggregated data (e.g., averages or totals) is needed → some application collect data from different geographical locations, and the relevant information may be an aggregated measure instead of the reading of the single devices. Without local processing to filter or combine data at the network's edge, bandwidth usage increases, and the cloud must handle unnecessary or redundant data.

5.3 Cloud analytics

In a single environment multiple IoT devices may be deployed, each of which can report data on a regular basis and due to that the amount of data grows very quickly. IoT data can be analysed in order to:

- Gain insights about the system state and behaviour.
- Perform descriptive analysis: *"what happened?"*
- Predict future behaviours: *"what will happen next?"*
- Define a set of actions to achieve a desired behaviour: *"how to get there?"*

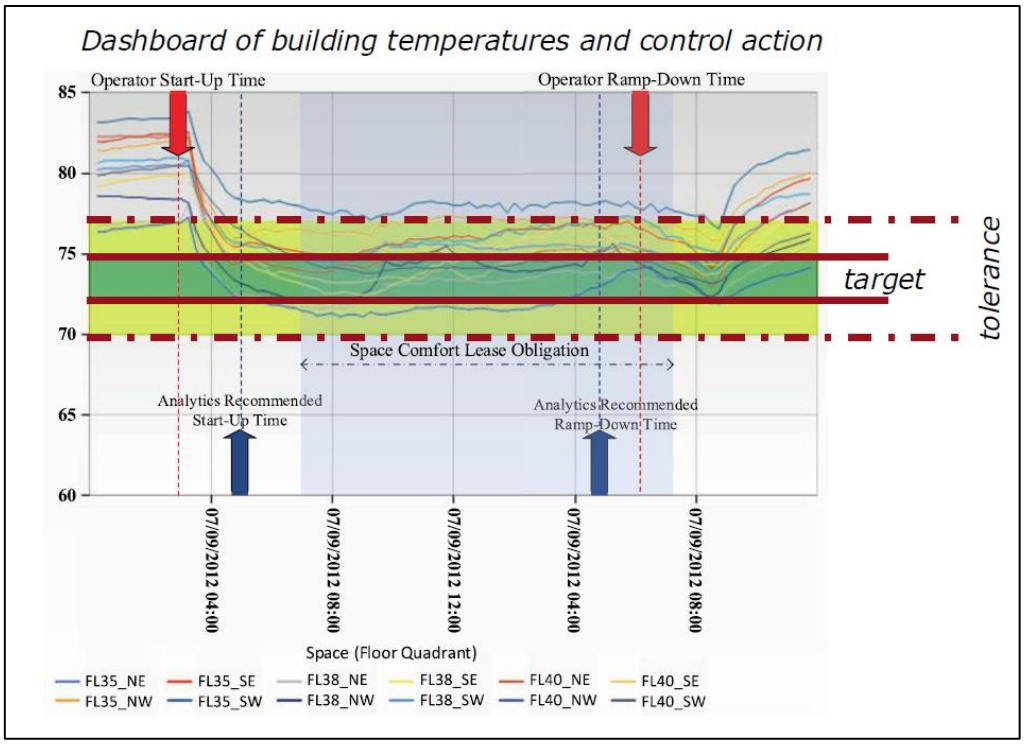
Visualization

The output of the analysis often needs to be shown to a person which will take decisions based on it. The user interface that shows the system state and allows the human operator to perform control actions is usually referred to as system **dashboard**.
Data visualization can often allow to gain some insights. For this purpose, however, the charts need to be as informative as possible:

- Always indicate the label for the chart axes.
- Avoid using abbreviations and acronyms.
- Highlight the important chart contents.
- Do not make the chart more complex than needed.

An example of IoT analytics

- Consider a building management system which shows the temperature measurements of 3 different floors (35, 38 and 40).
- The temperatures are reported in Fahrenheit and the interval 72-75°F represents the **target** temperature. It is shown in green in the dashboard.
- The corresponding **tolerance** interval is shown in yellow.



Objective: to control costs, the intensity of cooling is reduced, and temperatures may be allowed to drift out of the range comfortable to humans when the building is not occupied, i.e., before arrival and after departure of the occupants.

- **Bring-up (aka start-up time)**: the time when the more intense cooling starts to in order to normalize the building ambient temperature in time for occupant arrival. Fixed for the season. Determined by the system operators a day in advance, based on weather forecast, expected occupancy for a particular day, and prior experience with the building behaviour.
- **Ramp-down (aka bring-down)**: reduce the cooling before occupant departure. Initiated in advance because the building ambient conditioning has a degree of inertia that causes some lag between the change of settings and their measurable effect.

Constraints: **Space Comfort Lease Obligation,** i.e., period of time when the building owner is obligated to maintain the agreed-upon space comfort conditions by the lease agreement. Failure to do so is subject to financial penalties.

- The IoT control system is subject to this constraint when optimizing.
- There might be some other constraints.

- **Normal operations** (autopilot): monitor temperature thresholds and use control scripts to adjust the cooling as and where necessary.
- When **something unusual** happens or a fault is detected, the BMS alerts the operators using the display indicators, notifications, or alerts commensurate with the severity of the condition.
- (Additional) **fine adjustments** to depend on the building's changing conditions:

- Reduce the intensity of cooling when the weather turns cloudy.
- Reduce the intensity of cooling when building occupancy drops (e.g., holidays). Building occupancy can be dynamically tracked by sensors (e.g., security gate counters).

This example is based on a real deployment.

- **Analytics system**: predict building behaviour and suggest control actions to reduce energy consumption while maintaining occupant comfort.
- Based on a set of thermodynamic models, machine learning, weather data, and historical data on building's operation recorded for the past several years.
- Predictions 24h in advance + short-term 2h predictions for fine tuning.
- Results of building's operation with and without analytics:
  - Optimizations for $505,000 during a measured winter season (~$5M in total).
  - Average energy savings of 12% per building and on the order of $10 per square meter.

MACHINE LEARNING

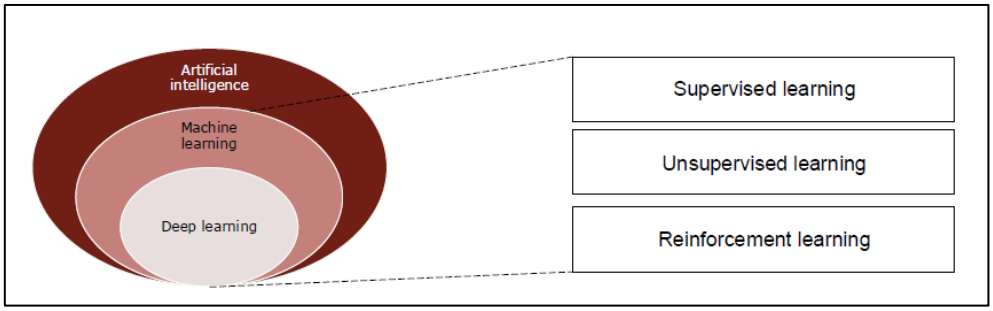
Machine learning consists in training an algorithm based on a set of available data (i.e., the training set) so that it like perform a pre-defined task on a set of samples which it has never seen (i.e., the test set).

Different types of machine learning algorithms:

- **Continuous estimation and optimization** (predictions or actions).
- **Classification** (assess to which category a given input belongs to).
- **Clustering** (identify groups sharing some features, which may not be evident at first).
- **Anomaly detection** (detect events different from the nominal system behaviour).
- Recommendation system (make predictions and suggestions based on past patterns).
- Transcription (converting an unstructured input into a sequence of characters).
- **Data generation** (generating new samples like those which are available).

Based on the type of training, machine learning algorithms can be classified as:

- **Supervised**: training set is labelled, so that the output is known for each input.
- **Unsupervised**: training set is not labelled.
- **Semi-supervised**: training set is partially labelled.

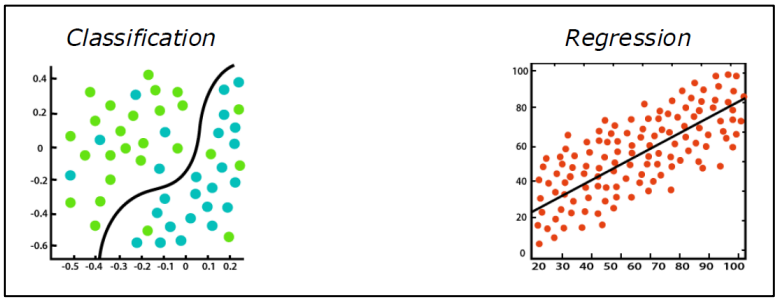


Machine learning: Supervised

- Definition: "Training data has explicit examples of correct outputs based on inputs."
- Example: hand-written digit recognition.
    - INPUT: hand-written images representing digits.
    - OUTPUT: digits corresponding to the hand-written images.
- A (human) supervisor must take the trouble to look at each input and manually determine the correct output.
- The result is a labeled input dataset, where labels are the outputs associated to each of the inputs.
- As input data is fed into the model, it adjusts its weights until the model has been fitted appropriately, which occurs in the cross-validation process.

Supervised learning can be separated into two types of problem:

- **Classification**: accurately assign test data into specific categories (e.g., cats/dogs).
- **Regression**: understand the relationship between variables (e.g., speed of a sprinter).



Some examples of supervised learning algorithms include:

- Neural networks
- Naïve Bayes
- Linear regression
- Logistic regression
- Support vector machine
- K-nearest neighbor
- Random forest

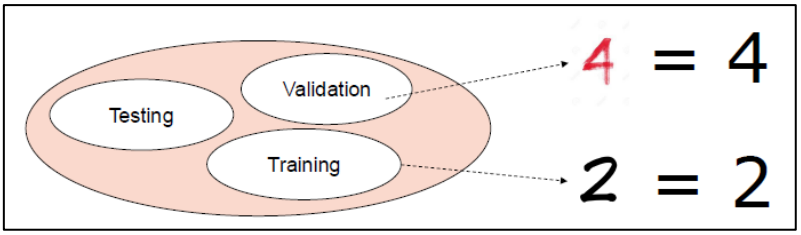
Data should be broken down in three distinguished and separate datasets:

- **Training dataset** is used to train the machine learning model, i.e., find the unknown target function *g* that best approximates *f*. The training dataset contains only labeled data (we need to know the input-output relationship to train the machine).
- **Validation dataset** is used to validate the machine learning model during training. Used to adjust the weights of the model, if needed. Ensure that the model is not overfitting to the data in the training set. Even here the set contains only labeled data (we need to know the input-output relationship to validate the model's results).

- **Testing data** to evaluate the machine learning method. Unlabeled data → that is indeed the final objective of our machine learning model: predict the output based on the training. Must be unseen data with respect to the training and validation datasets.

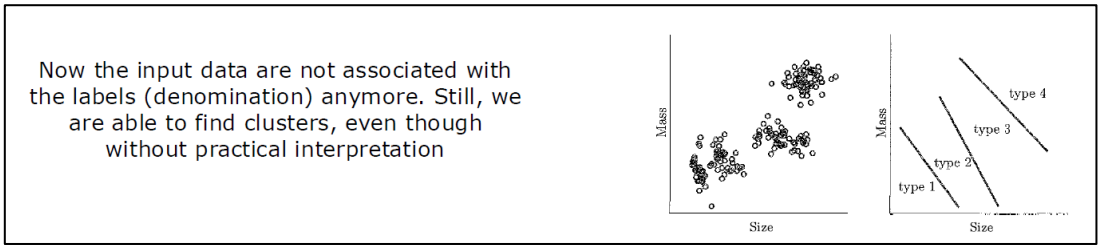
The validation dataset does not consist of samples that the model was already familiar with from training → ensure that the model is not overfitting.

- **Overfitting**: the model becomes very good at predicting the data in the training set but is unable to generalize on data that was not trained on.
- Validation data to check whether the model works well on different data.



Machine learning: Unsupervised

- Definition: "The training dataset does not contain any output information."
- Example: coin classification.
  - INPUT: coins
  - OUTPUT: none
- Unsupervised learning discovers hidden patterns or data clusters from the (unlabeled) inputs, without the need for human intervention.



Unsupervised learning is a practical method for different use cases:

- **Clustering**: group unlabeled data based on their similarities or differences
- **Association rules**: find relationships between variables in a dataset
- **Dimensionality reduction**: reduce the number of data inputs to a manageable size while also preserving the integrity of the dataset →preprocessing data stage
- **Precursor to supervised learning**: develop a better representation of the input data *(example: Spanish language)*.

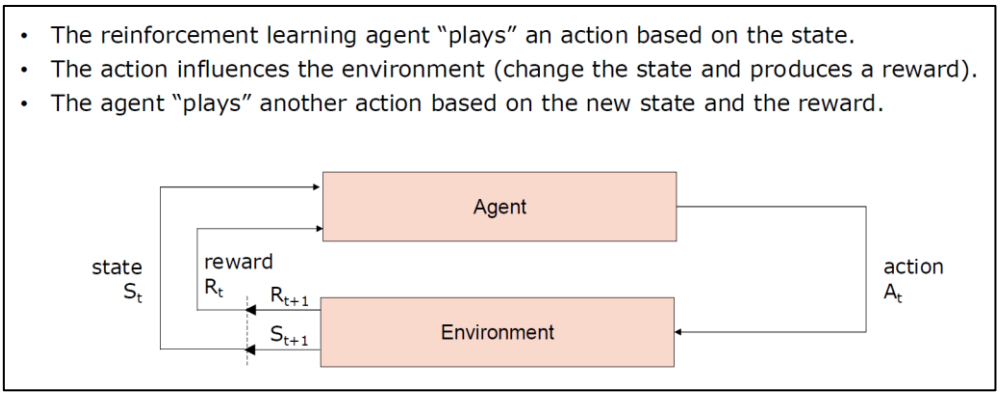
Some examples of unsupervised learning algorithms include:

- K-means clustering
- Gaussian Mixture Models

- A-priori algorithms
- Principal component analysis
- Singular value decomposition
- Autoencoders

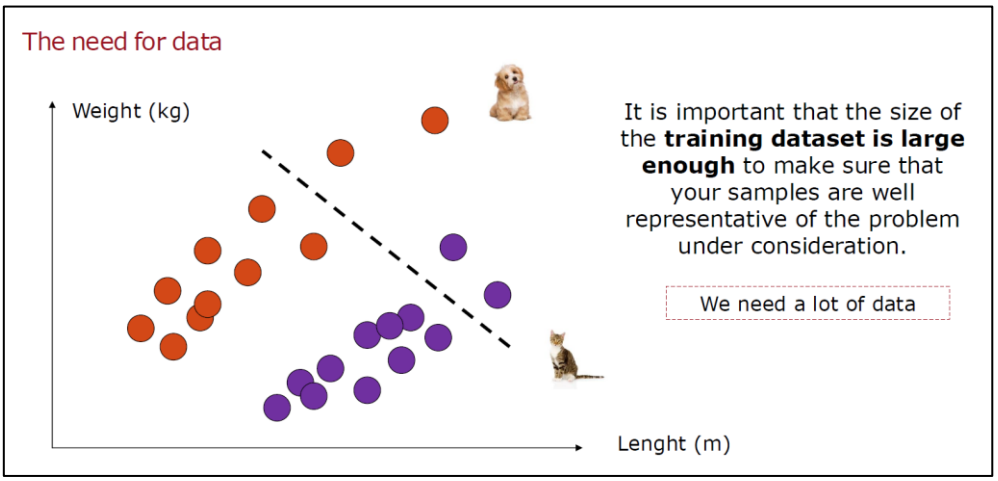
Machine learning: Reinforcement

- Definition: "Training data doesn't contain the correct output for each input."
- Example: Toddler learning not to touch a hot cup of tea.
  - INPUT: actions of the toddler touching or not touching the cup of tea
  - OUTPUT: the results of the touching action (pain or not pain)
- The training dataset does not contain the target output, but some possible outputs, together with a measure of how good that output is: **reward**.
- Based on the reward, the algorithm is reinforced to the better actions, i.e., those that maximize the reward on the testing data.
- There is no need to have labeled data.



Some examples of reinforcement learning algorithms include:

- Q-learning
- Deep Q-learning
- Markov Decision Process



6. IoT Security

The investments in IoT security are following a rapid increase with an estimation of 168.56$ billions for 2033. More than 70% of IoT devices are highly vulnerable to attacks, so additional efforts are required to guarantee that IoT systems are safe and secure.

IoT systems come with **inherent security concerns**:

- IoT systems are distributed systems with many geographically dispersed heterogeneous nodes with different capabilities, connected via separate physical networks.
- IoT systems can interact directly with the physical world and can thus impact the health and safety of people.
- IoT systems are constrained devices (energy, capacity, etc.).
- IoT systems are deployed in unprotected and unattended environments.
- IoT systems often deal with confidential data, so privacy becomes a primary concern.

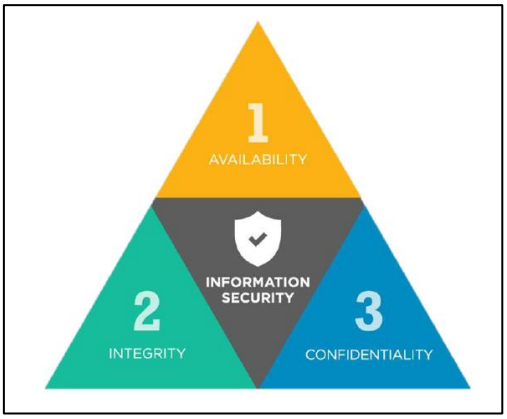
Anyway, there are **some good features**:

- IoT Edge and many other components are fixed or narrow-function devices, not general-purpose computing systems ("IoT sensors are not smartphones").
- IoT endpoints can be directed to communicate only with a limited population of known authenticated entities that may be identified and with whom trust can be established.
- Additional precautionary measures may be taken to reduce exposure:
    - Not allowing the downloading of unsolicited or unauthenticated software.
    - Closing of ports not used for IoT communication.
    - Elimination of OS features that allow remote login, shell access, or support unsafe protocols.

Security threats

IoT security can be defined as "protecting it from unauthorized access or changes." Security objectives can be represented as a pyramid (**CIA**):

- **Confidentiality**: ensure that information (data) is available only to authorized parties.
- **Integrity**: ensure that information (data) is accurate and unchanged (untampered).
- **Availability**: ensure that information is available for access whenever an authorized user requires it.



SECURITY ATTACKS

Every node can communicate with gateways, remember there is nothing that blocks it. **Physical attacks** consist in manually attacking the target system:

- Node substitution/cloning.
- Physical damage.
- Creation of faulty measurement (e.g., using a lighter for a thermal sensor).
- Hinder a measurement (e.g., using a piece of paper in front of a security camera).

Software attacks can cause different altered behaviours (e.g., modify the way an actuator interacts with the physical world):

- Viruses and worms.
- Trojans.
- Code injection: commanding the actuator to perform unsafe actions.
- Phishing emails.
- Configuration of nodes with the default common user and password combinations.

Network attacks occur on the communication channel:

- Sniffing or eavesdropping (on wired or wireless channels).
- Spoofing.
- Man-in-the-middle.
- Denial of Service (e.g., flooding a wired channel, jamming a wireless channel).
- Sinkhole attack (routing): a node declares itself as having exceptional resources and power, thus causing its neighbours to choose it as a routing waypoint.
- Wormhole attack (routing): two malevolent nodes claim that there is only a single hop between them and thus divert a lot of routed traffic to themselves.

Wireless networks can provide additional exposure because they can be eavesdropped without requiring physical taps on the communication medium.

SECURITY PRINCIPLES

The requirements for IoT systems are:

- **Safety**: components need to be certified, and continuous monitoring is needed.
- **Reliability**: the system needs to operate continuously (redundancy can help).
- **Resilience**: the system has to absorb and limit the effects of small issues and resume operation rapidly after major accidents.

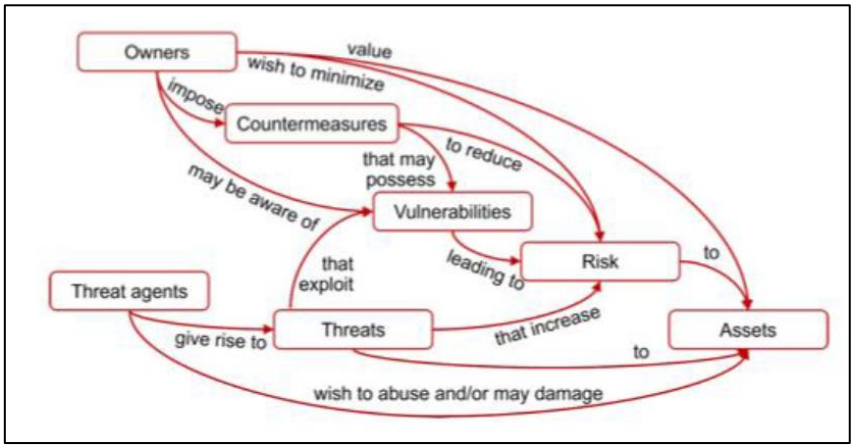
Saltzerand Schroeder outlined general design principles for protection:

- *Least privilege*: every user should use the least privileges necessary for a task.
- *Separation of privilege*: satisfy more than one condition (e.g., 2 keys for a vault).
- *Least common mechanism*: minimize the number of mechanisms shared among users.
- *Economy of mechanism*: keep the design simple.
- *Complete mediation*: every access should be checked for authorization.
- *Fail-safe default*: the default condition should be lack of access.

- *Open design*: the design should not be secret (do not rely on the ignorance of attackers).
- *User acceptability*: the mechanism needs to be user-friendly.
- *Work factor*: stronger security measures should make attackers work harder.
- *Compromise recording*: system keeps attack records to discover unauthorized use.

6.1 Security planning and analysis

Security should be considered in an IoT system from the initial design, and it is a continuous process.



Some useful definitions for risk analysis:

- **Risk** can be defined as *"the possibility that a threat agent will exploit a vulnerability to damage an asset."*
- A **vulnerability** can be defined as *"a software/hardware bug or misconfiguration that a malicious individual can exploit."*
  - Simply put, it is a weakness in a system (or countermeasures for protecting it).
  - The vulnerability meaning is related to the threat concept.
- A **threat** can be defined as *"any activity or event that can cause an unwanted outcome (e.g., damage, disruption or loss of an asset)."*
  - The relevance of a threat depends on its impact.
- The amount of risk the vulnerability presents depends on:
  - The number of systems affected by the vulnerability.
  - The **criticality** of the affected systems. It represents the measure of how valuable the asset is to the organization, if compromised.
  - The **exposure** those systems present to the organization.
- Therefore, the risk can be computed as:

  Risk = Vulnerability × Attacks × Threat × Exposure
- Risk cannot be eliminated, but a proper analysis of threats can help in risk mitigation. Risk analysis allows to define the proper level of security for satisfying the operational objectives at a cost that provides an acceptable return on investment.

- **Risk avoidance** seeks to eliminate the risk by eliminating exposure to the specific threats (e.g., eliminate nonessential features).
- **Risk acceptance** consists in deciding not to invest in preventive measures if the attack has a very low probability, or its impact can be managed.
- **Risk transference** consists in transferring a risk to a third party (e.g., insurance company) upon payment. Valid for unanticipated (low probability) incidents.

Example

Risk analysis is important not only to protect the assets which are clearly critical, but also all the other assets connected to them. Consider a castle containing a treasure that is being attacked by an army.

- **Exposure** represents how exposed the castle is to the attack (e.g., walls, moat?).
- **Periphery** represents the extent of the walls/moat, and the number of openings.
- **Threat** is a measure of the enemy armies who are performing the attack.
- **Attack** is represented by the actual arrows and breach attempts on the walls.
- **Vulnerability** is a measure of how easy is for the treasure to be accessed.
- **Criticality** represents a measure of the value of the treasure.

SECURITY THREAT MODELING

Threat modeling: identify how an attacker can attempt to compromise the system. The basic steps in the threat modelling process are:

- **Model the system**, i.e., create an architecture diagram (components and flows).
- **Enumerate threats**.
- **Mitigate threats**.
- **Validate mitigations**.

Model the system is usually performed by analysing the system model. It should identify all key components (sensors and actuators), their connections, data and control flows between specific components and the overall system. The model is evolved by determining the requirements that specify what the system needs to do (e.g., monitor and control the operation of a process).

Enumerate threats

- Known threat types are summarized in the **STRIDE** model.
- Attacks may combine several of these modes.

Threat	Violated property	Description
Spoofing	Authentication	Impersonating someone/something else
Tampering	Integrity	Altering hardware, data, code...
Repudiation	Non repudiation	Denial of being involved
Information disclosure	Confidentiality	Exposing the asset to unauthorized parties
Denial of Service	Availability	Denial of offered services
Elevation of privilege	Authorization	Gain capabilities without authorization

The **DREAD** model has been developed to rating the threats.

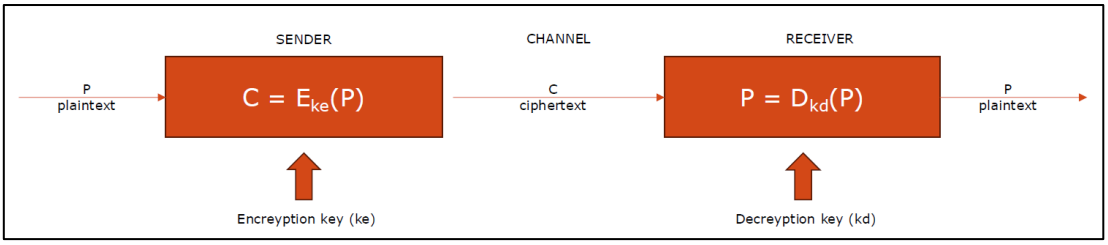
- **Damage potential**: the potential damage that an exploit can cause.
- **Reproducibility**: how easy it is to reproduce an exploit.
- **Exploitability**: what is required to make the exploit successful.
- **Affected users**: the number of users affected by the exploit.
- **Discoverability**: the ease with which the threat can be uncovered.

Mitigate and validate threats

- Designing the appropriate defence mechanisms.
- Validate that the chosen mitigations address all identified vulnerabilities.
- When analysing threats, the general advice is to **think like an attacker** and to examine ways in which they may attempt to penetrate the system: reconnaissance, scanning, access, maintain access, cover the tracks.

6.2 Cryptography

Cryptography is the **enciphering** and **deciphering** of messages in secret code (note: cryptography is not used to protect the system from possible breaches, its scope is to protect data when sent between a sender and a receiver). Cryptography is the principal solution for providing confidentiality, integrity and authentication. It allows to convert a message (**plaintext**) to a secret code (**ciphertext**). The transformation is performed using a **key**, so without the correct key it is not possible to perform the inverse conversion.



Threats

Attacks targeting cryptographic systems can be:

- **Ciphertext only**: the attacker sniffs the traffic and has to analyse the encrypted data through cryptanalysis techniques.
- **Known plaintext**: the attacker has access to the plaintext and the ciphertext.
- **Chosen plaintext**: the attacker is able to get the source system to insert into the system a message chosen by him/her.

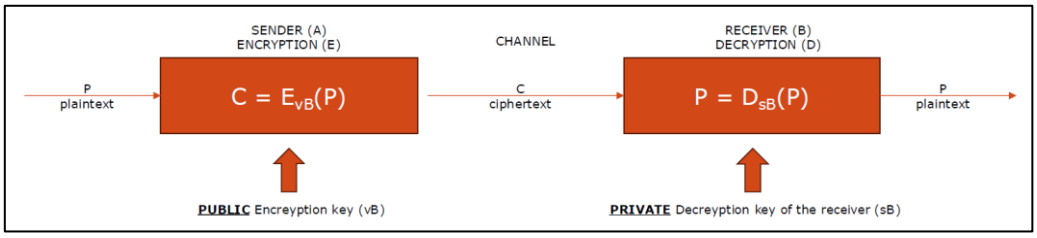
Cryptographic systems are subject to **brute-force attacks** where the ciphertext can be computer-analysed by rapidly trying many, and in some cases all, different combinations or keys. The process can be aided by exploiting known weaknesses (e.g., language patterns). While increasing the key length may be a good approach for traditional IT systems, for IoT networks this does not represent an effective solution since we are talking about resource-constrained devices → a secret should be shared to avoid brute-force attacks.

Types of cryptography

- **Symmetric**: use the same key for both encryption and decryption.
  - The (private) key should be secret.
  - An additional mechanism for secure key distribution is required.
  - This may be accomplished by using separate channels or key exchange mechanisms.
- **Asymmetric** (or **public**): different keys are used for encryption and decryption.
  - Encryption and decryption algorithms are easy to compute.
  - It is computationally easy to generate a key pair.
  - It is computationally infeasible to derive a private key from the corresponding public one.
  - Given the ciphertext and the public key, it is infeasible to reconstruct the plaintext.

ASYMMETRIC CRYPTOGRAPHY (OR PUBLIC-KEY CRIPTOGRAPHY)

- Asymmetric system.
- **Use different keys for encryption and decryption** (publicly visible and private).
  - Public keys may be visible to all.
  - Private keys are kept secret by each node and stored in hardware-secured registers.
- Each entity uses a method to compute a matching pair of public and private keys.

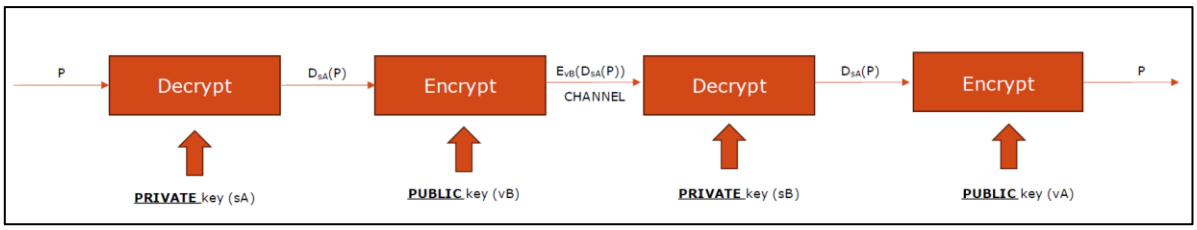


- The ciphertext is sent to the recipient over insecure communication links.
- **Confidentiality: ensured**, since **only the recipient B knows its private key** with which the message can be decrypted. An eavesdropper can obtain the ciphertext and access the public keys but finds computationally infeasible to reconstruct the original message.
- What about **integrity** and **authentication**?

Double transformation

- Used to simultaneously achieve confidentiality, integrity, and authentication.
- The sender (A) performs a decrypt transformation on the message using its private key (sA). This process allows the sender to “sign” the packet with its private key, ensuring integrity.
- The sender (A) then encrypts the result using the recipient’s public key (vB) and sends the message to the receiver (B).

- The receiver (B) also performs a double transformation, starting by decrypting the received ciphertext to obtain D_{sA} (P), the signed message.
- This ciphertext is encrypted using A's public key (vA) to verify the signature (integrity) and obtain the plaintext P.



Digital certificates

- Public keys need to me somehow distributed, and **the link between the key owner and the key must be certified.**
- Digital certificates can be employed for this purpose.
- They are managed by a trusted third party: **Certificate Authority (CA).**
- To obtain a digital certificate, an entity provides its identity and public key to the CA which validates the identity and issues the certificate.
- The certificate includes:
  - Name of the entity that owns it and its public key.
  - Certificate validity/expiration.
  - The CA which issued it (and many more data).
  - It is signed with the CA's private key (anyone with its public key can verify the certificate).

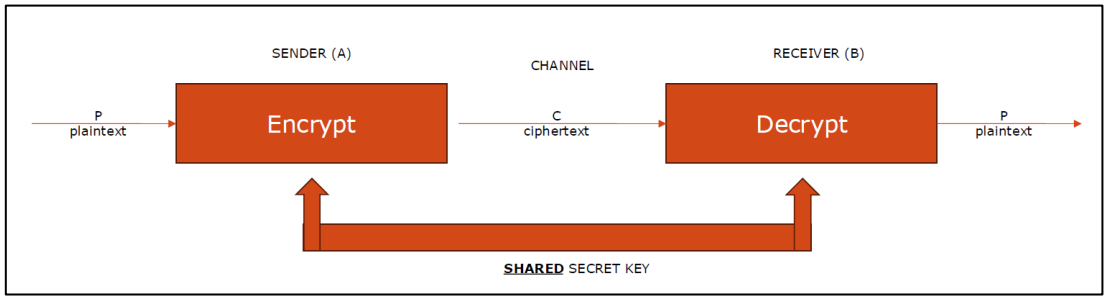
The standard cryptographic algorithms **may not be directly applied to IoT systems** since they are formed by constrained devices. To increase security, use of longer keys is recommended, but it also increases the computational burden on node.

Some recommendations for IoT systems:

- Use **Lightweight cryptography (LWC)** (which does not mean "weak").
- Coupled with the addition of **dedicated hardware assists**.
- Use symmetric cryptography due to the smaller computational cost.
- Use asymmetric cryptography only to exchange the symmetric keys.

SYMMETRIC CRYPTOGRAPHY

- **Use the same key for encryption and decryption**.
- Used to ensure confidentiality, integrity, and authentication of messages.
- Communicating nodes must share a secret key.



Symmetric-key algorithms are public and thus follow the principle of open design.

- The sending node (A) uses a mutually agreed-upon encryption algorithm to encrypt the message using its copy of the shared key.
- The receiving node (B) decrypts the message using the matching algorithm and its copy of the shared key.
- The cryptographic algorithm defines encryption and decryption as inverses of each other.
- Any other recipient (or interceptor) does not have the shared key and cannot recover the original plaintext message sent by A.

Key exchange

- Nodes using symmetric cryptography need to **securely** establish secret keys.
- This can be done through a message exchange using **public-key cryptography**. However, this approach imposes the burden of implementing a public-key cryptographic system which may be too much overhead if it is only needed for the key exchange.
- Another option: jointly construct the key (**Diffie and Helman**).
    - Execute incrementally a specified algorithm with the exchange of intermediate results and parameters over an insecure channel.
    - By knowing each other's intermediate outputs, both nodes compute the same secret key.
    - It is infeasible for a third party to compute the same secret key by knowing the chosen algorithm, parameters, and partial outputs exchanged by the two nodes.
    - The identical secret key is computed without having to actually exchange it.
    - Susceptible to a man in the middle attack.

Comparison

Aspect	Symmetric cryptography	Asymmetric cryptography
Key usage	Single shared key for encryption and decryption.	Uses a pair of public and private keys.
Speed	Faster because of simpler algorithms and shorter key lengths.	Slower due to more complex algorithms and longer key lengths.
Key management	Requires secure distribution of the shared key to all parties.	No need to share private key; public key can be freely distributed.
Security	Less secure if the shared key is intercepted or leaked.	More secure as private key is never shared.
Computational overhead	Low computational overhead, efficient for large-scale encryption.	High computational overhead, more suited for initial secure handshakes.

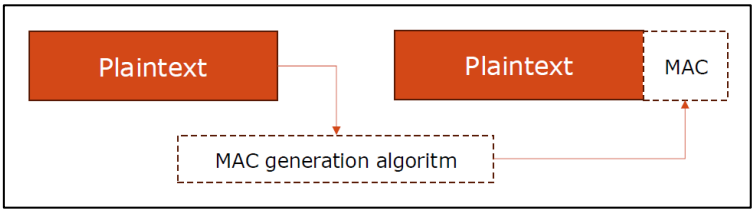
Message authentication

Introduction

- Encryption schemes provide various combinations of message confidentiality, integrity, authentication, and nonrepudiation.
- Assurance that the message has not been altered (intentionally or by chance).

Encryption scheme	Confidentiality	Integrity	Authentication	Digital signature
Asymmetric $E_{VB}(P)$	YES	YES	NO	NO
Asymmetric $E_{SA}(P)$	NO	YES	Partial (YES with MAC)	YES
Double transformation	YES	YES	YES	YES
Symmetric, shared key	YES	YES	Partial (YES with MAC)	NO

- Realized by computing a **Message Authentication Code (MAC)**.
- It implies the usage of a **shared secret key** (possibly different from encryption key).
- Computed by the sender node using a known function and the secret key to produce a relatively short fixed-length value that serves as the authenticator (aka **tag**).
- Once computed, the MAC is appended to the message.
- MAC-generation algorithms have the property that it is not possible to alter the message without affecting the authentication tag.



- Upon receipt of the message, B uses the shared secret authentication key to compute the MAC on the payload part of the message and compares it with the MAC that was received.
- If the two match, the receiver knows that the message was not tampered with, and A is the only other party that has the secret key used for MAC generation.
- For security of authentication, it should be computationally infeasible to compute a valid authenticator (tag) of the given message without knowledge of the key.
- To be of practical value, computation of the MAC itself should be relatively simple when the encryption key is known.
- Hashing is a commonly used way of computing MACs. Hashing functions produce a fixed length signature that may be used for file identification. With the addition of a secret key to compute the hash, hashing can be made secure and computable only by the communicating parties who have the secret key.
- The MAC allows to authenticate the message, but it does not provide confidentiality. Okay for some types of IoT applications where data integrity may be more of a concern than confidentiality, such as reports of the non-sensitive sensor data.
- If confidentiality is needed, the message should be encrypted (a/symmetric). Higher security means higher overhead with increased resource requirements and latency.

6.3 Endpoint security

Basic objectives of IoT endpoint security include:

- Node identification and authentication.
- Secure state (bringing up and maintaining).
- Secure communications.
- Security monitoring and attestation.

There is a wide range of techniques for implementation of endpoint security:

- **Hardware** endpoint security.
- **Software** endpoint security.

Hardware Security Modules (HSM)

Hardware-based dedicated security component (a physical device with its own separate processor and storage) designed to provide high levels of security.
They typically provide a safe isolated execution environment for implementation of security functions, usually coupled with the capability to generate and safeguard cryptographic keys that can be used for authentication, platform integrity, and secure communication.
HSM can assist in bringing up of a node in a known good starting state. Several modalities:

- Trusted Platform Module (TPM).
- Secure Element (SE).
- Trusted Execution Environments (TEEs).

Trusted Platform Module (TPM)

A chip embedded in devices to provide a **secure processor** for storing cryptographic keys, performing cryptographic operations, and ensuring the integrity. Some of the key components and capabilities of a TPM include:

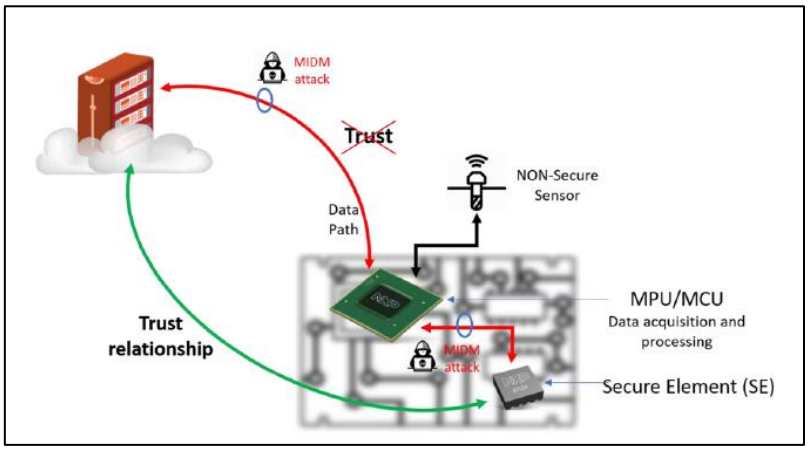
- **Secure storage**: cryptographic keys, passwords, certificates.
- **Cryptographic functions**: key generation, encryption, decryption, hashing, digital signing.
- **Integrity measurement**: stores the integrity of the system's software (e.g., BIOS, firmware, bootloader) to detect unauthorized modifications.
- **Attestation**: provides evidence to a remote or local entity that the system's configuration has not been tampered with.

Several TPM forms (in decreasing order of security):

- Discrete TPM: a separate dedicated hardware component.
- Integrated TPM: into other processing elements.
- Firmware TPM: in a separate protected execution environment.
- Software TM: an emulator.

Secure Element (SE)

Tamper-resistant hardware component capable of securely hosting code and confidential data (coupled with a dedicated processor for execution of secure code). Examples: authentication, identification, signatures, and PIN management.
It acts as a **vault**, protecting what is inside the SE (applications and data) from typical malware attacks in the host (i.e., the device operating system).

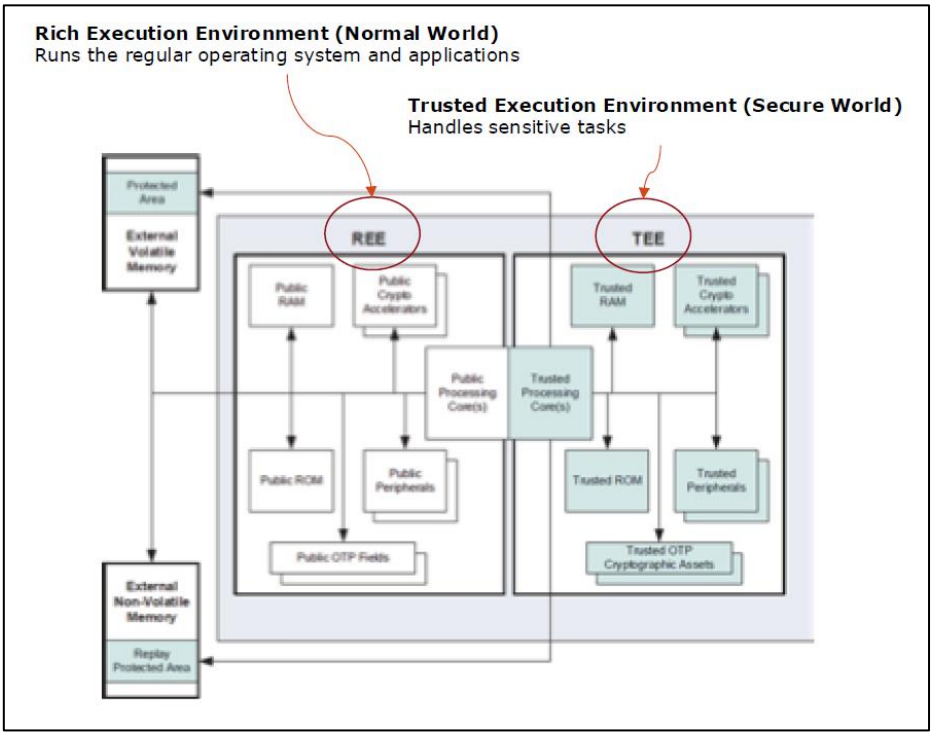


Trusted Execution Environment (TEE)

It is a **secure area of a main processor** that isolates sensitive data and computations from the rest of the system. It may be used to:

- Storage of cryptographic keys.
- Dedicated storage and execution of security code.
- Allow complete separation of trusted and untrusted portions of the system.

The main processor is divided into the Normal World (REE) and the Secure World (TEE).



Comparison

Feature	TPM	SE	TEE
Definition	A hardware chip for cryptographic operations and platform integrity.	A dedicated hardware component for secure data storage and processing.	A secure area in the main processor for running trusted code.
Primary purpose	Platform security and integrity (e.g., attestation, key storage).	Store sensitive data (e.g., payment info, credentials) securely.	Isolate sensitive operations and data from the main OS.
Location	Separate hardware chip or integrated module.	A standalone hardware module (sometimes embedded in the SoC).	A hardware-isolated region within the main processor.
Key use cases	Secure boot, remote attestation, encryption, and signing.	Payment systems (e.g., EMV), SIM cards, and NFC transactions.	Mobile app security, DRM, and secure key handling.
Performance	Dedicated, so highly optimized for security tasks.	Typically slower but highly secure due to dedicated hardware.	Shares resources with the main processor, so faster than SE but less secure.
Hardware isolation	Full isolation as a discrete module.	Full isolation due to dedicated hardware.	Partial isolation (rely on main processor architecture).

6.4 Network security

IoT networks are implemented using a layered design. Therefore, communications can be secured at different layers. It is believed that IP will be the base common network protocol for the IoT. However, there are (and will be) also devices organized in networks implementing application-specific communication protocols. For example, at the application level, CoAP does not provide security functionalities.

	Internet	IoT
Application	HTTP	CoAP
Transport	TCP	UDP
Network	IP	IPv6/6LowPAN
Link	MAC	MAC
Physical	PHY	PHY

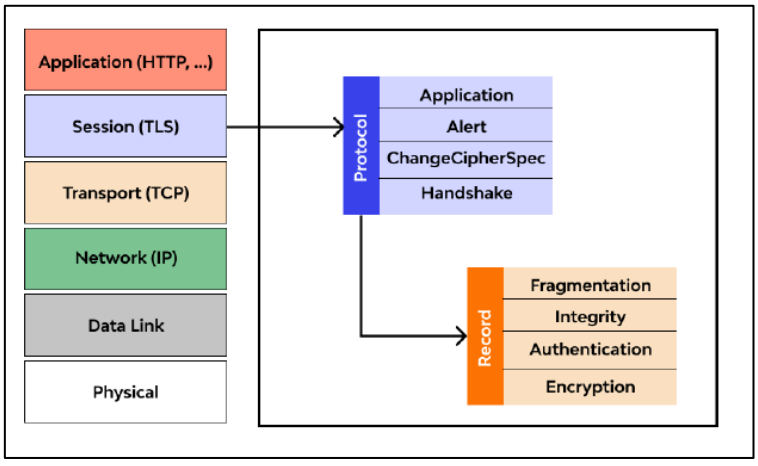
Security functionalities are implemented by the **application** (protecting the data before the CoAP encapsulation) or at one of the **underlying layers**.

- **Application-level security**:
  - End-to-end protection can be guaranteed.
  - It simplifies the requirements for underlying layers and reduces the cost in terms of packet size and data processing: per data and not per-packet overhead is introduced.
- **Transport/network-level security**: the same security mechanism can be shared by multiple applications.

Transport Layer Security (TLS)

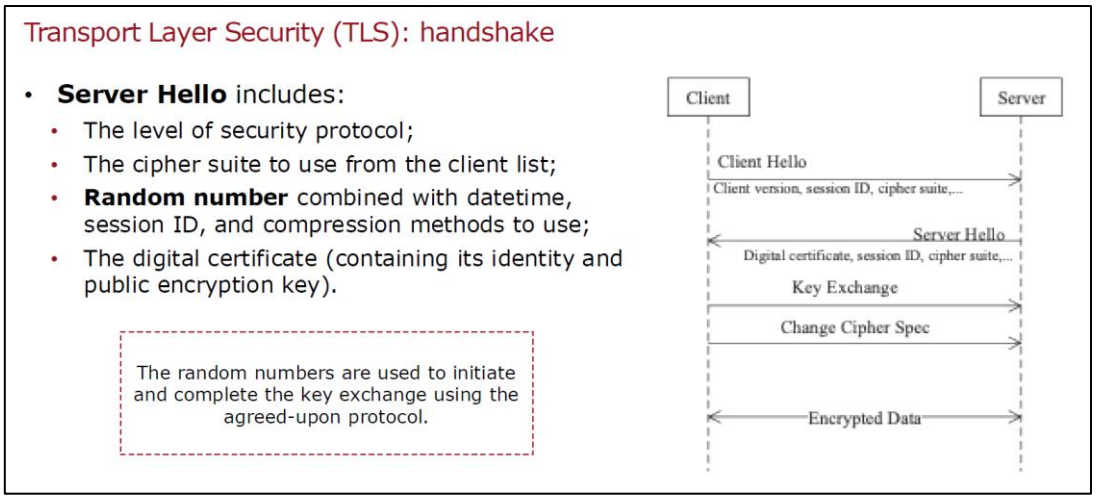
Designed to work with TCP. The main functionalities are:

- Authenticating the endpoints and define the set of cryptographic keys.
- Exchanging confidential data through symmetric encryption.
- Authenticating messages through secure hashing.



Two primary components:

- **Handshake protocol**: used to negotiate cryptographic modes and parameters, authenticate communicating parties, and establish the shared key material.
- **Record protocol**: uses parameters established by the handshake to protect traffic between endpoints and divides traffic into a series of records, each of which is independently protected using traffic keys.



Datagram Transport Layer Security (DTLS)

- Designed to work with UDP.
- Problem: overhead caused by DTLS (the underlying protocols, such as IEEE 802.15.4, have limited packet size). Packet optimization and compression mechanisms.
- DTLS creates a point-to-point secure association (through a handshake process similar to TLS handshake): not compatible with multicast IP communications.
- DTLS provides three security modes:
  - PreSharedKey: the devices store symmetric pre-shared keys.
  - RawPublicKey: the devices own a private-public key pair without certificate.
  - Certificate: the devices store an X.509 certificate.

Internet Protocol Security (IPsec)

It provides confidentiality, integrity, data-origin authentication and protection against replay attacks. The IPsec protocol suite includes two principal protocols:

- **Authentication Header (AH)**: it provides source authentication and data integrity.
- **Encapsulation Security Payload (ESP)**: it also provides confidentiality (attacker cannot access to the plaintext if they do not know the secret key).

IPsec has two different packet forms: Tunnel mode and Transport mode.

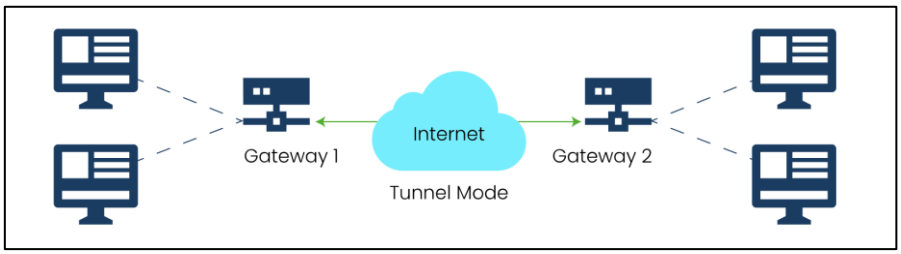
When we talk about network layer security, we are referring to protect IP addresses in a way to protect the device privacy. Note that network layer aims to provide the path between two parts and so it manages IP addresses by definition. In particular, we are protecting the header part (containing the device IP address) of any IP packet transmitted in the network → **IPsec packet**.

IPsec packet format: Tunnel mode

- It protects the whole IP packet.
- The entire original IP packet is encapsulated to become the payload of a new IP packet.
- A **new IP header** is added on top of the original IP packet.
- Useful for **protecting traffic between different networks**: the packet is sent with a new IP header, and intermediate routers cannot see anything about the original packet.

- It simplifies the key exchange procedure.
- Final packet:

IPSec header	IPSec payload

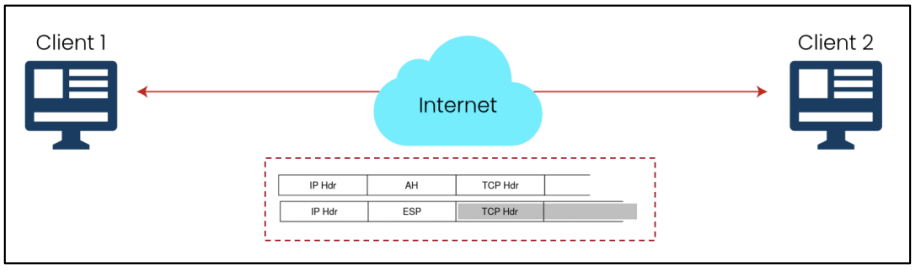


IPsec packet format: Transport mode

- It retains the original IP header.
- It is employed for end-to-end communication between two hosts that previously established a secure IPsec tunnel.
- Since a new IP header is not created, the process is less complex than tunnelling.
- Drawback: the IP header is not protected, thus allowing traffic analysis.
- Final packet:

IP header	IPSec header	IPSec payload*

*containing the payload of the original IP packet



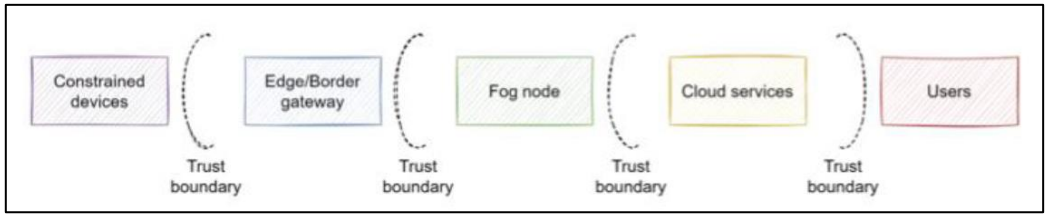
Further considerations on IPsec:

- Before sending IPsec datagrams, the source and destination must create a logical connection called **Security Association (SA)**.
- It is a one-way connection. If IPsec traffic has to flow in both directions, so two SAs are needed.
- The state information of a SA include:
  - A Security Parameter Index (SPI) which identifies the SA.
  - The origin and destination interface of the SA (the IP addresses).
  - The security protocol identifier which indicates if it is an AH or ESP security association.
  - The type of encryption to be used.
  - The type of integrity check.
- The keys can be dynamically exchanged using the **IPsec INternetKey Exchange (IKE) protocol**.
- Problem: it uses asymmetric cryptography and is computationally heavy for constrained devices. IKE extensions using lighter algorithms should be considered.

- Problem: strong data overhead due to the additional headers. Header compression techniques are needed.
- Problem: no end-to-end security guarantees if data flows through proxies and application-level gateways. A security gateway is needed.

Network segmentation

We have seen transport and network layer security but there is another approach to enhance security: different portions of the IoT network may have **different security requirements**.
A first classification can be made between the data plane and the control plane. Since the actuation commands have direct impact on the physical world, they may be subject to stronger protection.
It is a good design practice to divide networks into segments (**trust zones**). Each zone has access control, authentication rules, security policies and protocols. The main drawback resides in the implementation complexity.



Denial of Service (DoS)

- Example for IoT: **denial of sleep**.
- It consists in keeping IoT devices always active (receiving or processing data) thus preventing the switch to energy-saving sleep mode.
    - The DTLS handshake poses a relevant issue for DoS attacks towards the server.
    - An adversary may continuously transmit ClientHello messages to the DTLS server and start a large amount of DTLS handshakes.
    - When the handshake starts, the server allocates resources for the connection: waste.
    - If the attacker spoofs the IP of a victim node and starts a DTLS handshake with a server, the server will send reply messages to the innocent node: amplification effect.
- Possible countermeasures:
    - *Frequency hopping*: more difficult for an eavesdropper to know the tx frequency, anyway difficult to implement the coordination between server and client,
    - *Randomize the wake-up time*: more difficult to know when the device transmits.

Secure data aggregation

- Data aggregation requires to perform some processing in intermediate nodes to minimize the traffic so that only relevant information is passed in the network.
- Concatenating the payloads: inefficient.

- Semantic concatenation: process data being aware of the final goal.
- Security issues if aggregators are not special nodes:
  - Aggregators perform decryption and encryption procedures in addition to aggregation processing: computational and energy impact.
  - Aggregators must keep secure associations (i.e., symmetric keys) with every node to which it communicates: memory issue.
  - Aggregators access the data they receive: privacy concerns.
- Solution: homomorphic encryption techniques.

Authorization

- The process of granting approval to a client to access a resource.
- It allows to answer to the following questions:
  - Which users can access some given data?
  - How should the information be presented to a given user?
  - Which operations is a user allowed to perform?
- There are three main authorization approaches:
  - **Discretionary access control (DAC)**: restrict access based on the subject identity.
  - **Role-based access control (RBAC)**: map permissions to roles assigned to subjects.
  - **Attribute-based access control (ABAC)**: map permissions to attributes for the subjects.
- The **Open Authorization (OAuth) protocol** has been introduced for allowing authorized third parties to access personal user data.
- It has to be adapted to the IoT scenario to be compatible with constrained devices.
- Header compression minimizing the amount of sent information.
- It defines different roles:
  - **Resource owner**: the entity that grants access to a protected resource.
  - **Resource server** (Service Provider – SP): a server hosting user-related information.
  - **Client** (Service Consumer – SC): third parting which needs access to user data.
  - **Authorization Server** (AS): server which issues access tokens to the client after obtaining authorization from the resource owner. The tokens are authentication credentials containing information about the SC identity and the user identity.

Authorization: IoT-OAS

- An example of an architecture based on OAuth for IoT is **IoT-OAS**.
- It defines an external service which can be invoked so that:
  - The smart object keeps its simplicity, since it only needs to invoke an external service
  - The access control policies at the SP can be dynamically and remotely configured, whereas it may be difficult to act directly on the smart object.

Security monitoring and management

- Security monitoring can follow different strategies:
  - **Real-time**: keep track of current system state, raise alarm when an anomaly is detected.
  - **Predictive**: identify patterns identifying when an attacker has breached a portion of the system or indicating a future attack.
  - **Forensic**: record data concerning past attacks to gain insights and, if possible, enhance the system security in the future.
- Security analytics can be:
  - **Rule/signature-based**: rely on pre-defined rules or signatures to detect anomalies.
  - **Behavioural analysis**: nominal behaviour patterns are learnt, and an anomaly is declared when the current system behaviour deviates from the modelled one.
- After a security breach, an incident response has to be applied:
  - Identify the affected parts.
  - Isolate the affected parts (e.g., disconnection).
  - Start the recovery action to restore normal operation conditions.
  - Perform forensic analysis to determine the root cause and enhance system for the future.

6.5 Privacy

Manufacturing or production-centric organizations will be more focused on the risk scenarios and their impact on business. Organizations whose IoT systems are customer-centric (i.e., they sell IoT services) will be more concerned with privacy and legal issues.
Privacy is one of the main issues with IoT systems, due to the use of IoT devices → example: home devices; wearable devices; in-vehicle devices.
In addition, data is recorded regularly and in a continuous fashion. This provides information about habits and trends, leading to **profiling issues**.

User's data should be collected only with the user **explicit consent**. Moreover, the user should be aware of the following information:

- Which personal data are collected.
- Who has access to the data.
- If data are secured during transfer and storage.
- How long the data are kept.
- What are the notification procedures when data breaches are detected.

The main issue with IoT is related to the difficulty of interacting with the customers. In fact, many IoT devices do not have screens and those who have screens may have small ones. Some IoT products are not consumer facing (e.g., a sensor for electricity monitoring).

IoT developers should distribute their products in a way that guarantees that the **user receives all the relevant information** concerning the data collection.

- Providing consumers with **opt-in choices** at the time of purchase.
- Offering video tutorials.
- Affixing QR codes on devices.
- Incorporating notices and providing choice as part of set-up wizards.
- Providing a management portal for privacy settings that consumers can configure and revisit, possibly from a separate device such as a smartphone.
- Use of icons.
- Communicating with the customer outside the device, for example, through email.

A key principle which needs to be followed is **data minimization** (→ minimize the privacy problem). It consists in limiting the amount of data collected and delete them when they are no longer needed. There are many data policies such as:

- Not to collect data.
- To collect only the type of data necessary or the functionality of the product being offered.
- To collect less sensitive data.
- To deidentify the data collected.
- To seek consumers' consent for collecting additional, unexpected categories of data.

Deidentification allows to reduce the risk concerning the users' privacy while preserving the value of the collected information. It reduces the risk of a privacy breach if the data is lost, stolen or accessed by unauthorized people. However, deidentification is not flawless, in some cases it has been demonstrated that deidentified data can be re-identified.

Cybersecurity regulations in the EU

1. Cybersecurity is regarded as a highly complex issue which requires the active involvement of a range of stakeholders, including the legislator.
2. This is a significant change since initially it was perceived as a **purely technical matter** related to measures ensuring the availability, integrity and confidentiality of information and information systems.
3. The first European Union Cybersecurity Strategy (2013) identifies cybersecurity as a new EU policy area: *"an excellent example of an area in which the different policy fields need to be combined (a requirement for horizontal consistency), and where measures need to be taken at the level of both the EU and Member States (calling for vertical consistency)"*.

Five strategic EU cybersecurity priorities have been identified:

1. **Achieving cyber resilience** by establishing minimum requirements for the functioning, cooperation and coordination of national competent authorities for network systems.
2. **Reducing cybercrime** by:
    - Ensuring a swift transposition of the cybercrime related EU Directives;
    - Encouraging ratification of the Council of Europe's Budapest Convention on Cybercrime;
    - Funding programmes for the deployment of operational tools.
3. **Developing cyberdefence policy and capabilities related to the Common Security and Defence Policy (CSDP)** by:
    - Assessing operational EU cyberdefence requirements;
    - Developing the EU cyberdefence policy framework;
    - Promoting dialogue and coordination between civilian and military actors in the EU;
    - Facilitating a dialogue with international partners.
4. **Developing the industrial and technological resources for cybersecurity** by:
    - Establishing a public-private platform on Network and Information Security (NIS) solutions;
    - Providing technical guidelines for the adoption of NIS standards and good practices;
    - Encouraging the development of security standards for technology "with stronger, embedded and user-friendly security features."
5. **Establishing a coherent international cyberspace policy for the EU and promoting core EU values** by mainstreaming cyberspace issues into EU external relations and Common Foreign and Security Policy (CFSP), and by supporting capacity building on cybersecurity and resilient information infrastructures in third countries.

The main EU legal acts guiding the maintenance of information systems are:

- **General Data Protection Regulation (GDPR)**: it aims at strengthening and clarifying rights concerning personal data of natural persons. It affects organizations globally if they manage data belonging to users in the European Union.
- **Directive on Security of Network and Information Systems (NIS)**: it will likely be adopted differently by member states, although it defines what can be considered a minimum level of security responsibilities for information systems. While GDPR is oriented to end customers, NIS regards organizations (the ones who provide services to end customers) privacy protection.
    - **Preventing risks**: use measures that are appropriate and proportionate to the risk.
    - **Ensuring IT security**: ensure a level of security of the information appropriate to the risks.
    - **Handling incidents**: minimize the impact of incidents on the systems used for the services.

Cybersecurity regulations in the EU for IoT

- Data quality from data generators, such as sensors, must, in light of the GDPR, be accurate and reliable.
- Meta information concerning for instance data source and access rights, should be recorded along with the data when it becomes associated with a natural person.
- During feature engineering, efforts must be made such that descriptive statistics do not introduce new features that can be considered sensitive.
- The need for transparent processing and transparent data transfers / manipulations / removals may be best met by an immutable data storage solution, such as a ledger-based technology like blockchain.

7. IoT Cloud platforms

An **IoT platform** is a collection of components providing some functionalities which are common to different IoT systems. An IoT system designer can focus on their specific product features and rely on the platform for the common functionalities. This allows to reduce complexity, cost, and the time needed for development.

Requirements

Producer side	Platform side
Miniaturized sensor	Data processing
Component integration	User data aggregation (cross-population trends)
Battery design	Mobile app
Wireless connectivity	Security features
Firmware	Database management
Data fusion	

An IoT platform needs to be reliable, customizable, scalable and resilient. It includes the following functionalities:

- Data acquisition from the edge.
- Data routing.
- Integration with cloud services (processing, storage, analytics, visualization).
- Security management.
- Device management.
- Development tools and kits (SDKs).

Usually, there is a **cloud based IoT core** (or **hub**) which establishes connectivity and security between edge devices and cloud services. It is a gateway implementing bi-directional messaging, which filters/routes the messages. The core (access to the cloud) is accessed through standard transport and application layer protocols such as TCP, HTTP and MQTT. IoT core and cloud services are offered as “managed services”: the user pays for it, on a usage basis.
Due to the variety of IoT objects, no support for raw sensor connectivity. Users are responsible for creating a software that interfaces to their specific sensors and makes them compatible with the IoT platforms.

There are many variants of IoT platforms that are available from commercial vendors and the open-source community. Commercial cloud providers tend to provide rather comprehensive and robust IoT platforms that build upon their existing infrastructure and technology.
The two most popular IoT platforms, in terms of breadth of coverage, relevance and market presence, are **Amazon AWS IoT** and **Microsoft Azure IoT.**

7.1 Amazon Web Services (AWS)

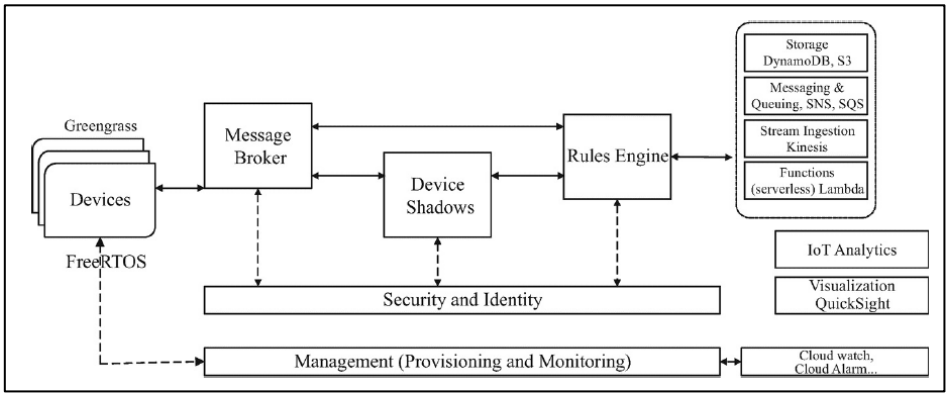
The Amazon IoT platform is known as **AWS IoT**. It is an example of **Infrastructure as a Service (IaaS)** → customers can build their own cloud environment using the functionalities provided by AWS. AWS IoT is composed of:

- An **IoT core** which offers cloud-hosted functionalities.
- An **edge gateway** for external execution (**Greengrass**). Greengrass provides a subset of IoT core functionalities, and resides in the user premises, outside the AWS cloud perimeter.
- A real-time **operating system** for microcontroller-based things (**FreeRTOS**). FreeRTOS allows the construction of IoT things and constrained devices that can be connected to the AWS IoT cloud.

The **IoT core** is composed of:

- **Data plane**: drives sensor data into the cloud for processing and storage.
  - Message broker.
  - Rules engine.
  - Device shadows (Digital twins).
- **Control plane**: security and management components.

AWS IoT



Data plane is enabled by the following components:

- **Message Broker**: supports publish/subscribe via MQTT.
- **Rules Engine**: processes the payloads to route them to other services.
- **Device Shadows**: shadows are persistent digital representations of the devices containing the last reported state and, if applicable, the desired state.

The use of shadows allows for the following functionalities:

- Cloud services access **shadows** to obtain the last known reading and to initiate actions by modifying the desired state to what an output should be.

- If there is a discrepancy between a reported and a desired state, the shadow instructs the endpoint to initiate the necessary action.
- Completion of the action is confirmed when the endpoint reports a new state that matches the desired one.
- Shadows allows applications and services to operate on the last known data even when devices are **disconnected**, thus minimizing disruptions caused by the temporary outages.
- Cloud services can operate with a simple output abstraction of the desired state, with the edge software handling device-specific steps and protocols.

Talking instead about Control plane, it contains all the components that implement security and management functionalities.

The **security** component allows to:

- Secure the connection and the payload exchange.
- Handle authentication of devices.
- Manage authorization. Authorization is expressed as a policy attached to each entity before it accesses the system.

The **management** component allows to:

- Register devices (through an ID, custom attributes and security certificates).
- Perform remote management (software update, security patches, reboot, ...).

AWS additionally provide **supporting services** such as:

- Database (e.g., Amazon Simple Storage (S3) and DynamoDB).
- Messaging: the **Simple Notification Service (SNS)**sends and receives notifications through SMS, email and social media.
- Queuing: the **Simple Queue Service (SQS)** stores data in queues retrieved by apps.
- IoT analytics: interoperates with Amazon general analytics providing tools for data cleansing, forecasting, image, video, and text analysis. It can be integrated with user-supplied functions (data filtering, interpolation, outlier elimination...).
- General visualization service (**QuickSight**).

AWS IoT Edge: FreeRTOS

- Real-time OS (**RTOS**) designed for microcontrollers and small microprocessors.
- Can support the deterministic execution of real-time tasks, memory allocation, static and intertask coordination primitives (e.g., notifications, message queues, multiple types of semaphores, and stream and message buffers).
- Extensions for AWS IoT connectivity, security, and over-the-air updates.
- Primarily intended to enable IoT things and constrained devices to be connected to AWS IoT cloud directly or via a supporting gateway (e.g., Greengrass).

AWS IoT Edge: Greengrass

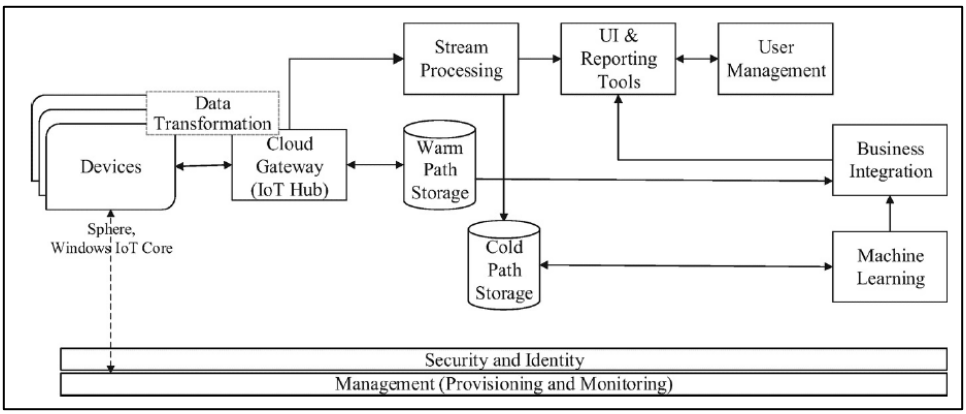
- Software acting as an **edge gateway** to provide secure connection and exchange of data and commands even when disconnected from the cloud.
- It operates at the edge (e.g., on premises and outside of AWS trusted perimeter).
- It easily connects to larger AWS IoT installations through the cloud and provides a local execution environment for cloud-developed services.
- **Data plane** functions: local message broker, device shadows, Lambda runtime, and support for local execution of ML inference engines.
- **Control plane** functions: security and authentication management, support for local secrets manager, and hardware security modules including trusted platform module (TPM).

7.2 Microsoft Azure IoT

Microsoft Azure IoT is another example of **Infrastructure as a Service (IaaS)**. The main component is the **IoT Hub**: It is the linchpin between the data sources and the cloud back-end processing services is a cloud hosted IoT gateway.

- It provides the first-point interface for data ingestion and ensures authenticated and secure communication between endpoints and cloud services.
- It supports brokered-based bidirectional communication and routing of messages based on their type and processing rules.
- Data routing.
- Device twins (digital twins).
- Control plane: security and management.

Microsoft Azure IoT



Data plane is enabled by the following components:

- Device connectivity modes include posting via HTTP and brokered messaging via AMQPP and MQTT in both native and web-socket variants.
- **Warm path** (towards the stream processing for low-latency processing).

- **Cold path** (towards the storage): supports more complex queries on larger aggregate data sets that may be required by analytics and ML that take longer time.

Control plane components aim to provide security and management:

- **Security**: authentication, rule-based authorization and encryption (at the transport layer using TLS or DTLS for TCP and UDP connections, respectively).
- **Management**: Device planning, provisioning (identity and security credentials), and retirement of individual devices and in bulk for groups.

Azure IoT provides additional **supporting services** such as:

- **Time Series Insights (TSI)**: an analytics storage and visualization service.
- **CosmosDB**: a NoSQL database.
- **Azure monitor**: a tool which allows the connection to a device for extracting its logs and obtain insights about the operation of the system (+ visualization functionality).
- **Azure IoT Edge**: it provides the following functionalities:
    - It runs on customer premises outside the cloud.
    - It provides a subset of IoT functions, support for local execution of applications.
    - It allows cloud connectivity.
    - When connectivity to the cloud is lost, the IoT Edge hub goes into the offline mode and buffers messages destined to go to the cloud.

Azure IoT Edge

Edge devices can connect to the cloud via an IoT hub assuming they meet security requirements and can handle the required protocols and procedures.
For less capable and legacy devices, Microsoft provides the **Azure IoT Edge** that runs on customer premises and outside of the cloud. It provides:

- Device upward cloud connectivity and performs communications gateway functions and protocol and format conversions if necessary.
- Connection multiplexing for underlying devices over a common single connection to the cloud (congestion control and isolation to support security).
- Data format and protocol translations, if needed.
- Support for local execution of applications, and limited operation when offline.
- Hardware security features, such as trusted platform module (TPM) and hardware security module (HSM), if available.

Azure IoT Edge includes two operating systems that may be used at the edge to streamline development and cloud connectivity:

- **Windows IoT Core**: a scaled down version of Windows OS targeted for embedded devices that uses tools and interfaces familiar to Windows developers.
- **Azure sphere**: an OS designed for constrained edge devices. It is a secure operating system for real-time processing that supports cloud connectivity and over-the-air software and security updates.

Azure IoT offerings also include SaaS and PaaS variants:

- **Azure IoT Central (SaaS)**: intended to simplify the development of IoT solutions that do not require much service customization.
    - It makes use of Azure IoT architecture and services (such as the IoT hub and the basic analytics), without exposing their intricacies directly to the users.
    - Dashboard and functions can be accessed from browsers on PCs and tablet.
    - It can be used to define and manage connected devices.
- **Azure IoT solutions accelerators (PaaS)**: complete IoT reference solutions for common IoT scenarios with open-source code that users can customize and deploy.
    - Focus primarily on the application part and connection with Azure IoT services.
    - Main scenarios: remote monitoring, connected factory, predictive maintenance, device simulation.

There are anyway other IoT cloud platforms such as:

- Google IoT
- IBM Watson
- Oracle IoT Cloud
- Cisco IoT Cloud Connect
- ThingWorx IIoT Platform
- Altair SmartWorks