الجمهوريـــة الجزائريـــة الديمقراطيـــة الشعبيـــة
People's Democratic Republic of Algeria
وزارة التعليـــم العالـــي والبحـــث العلمـــي
Ministry of Higher Education and Scientific Research
جامعـة محمـد البشـير الابراهيمـي – برج بوعريريج
University of Mohamed El Bachir El Ibrahimi - Bordj Bou Arréridj

Lecture notes on

Internet of Things course

Compiled by

SABRI Lyazid

Department of Mathematics and Information Technology

Mohamed El Bachir El Ibrahimi university

Course Objectives

The main objective of this course is to provide students with a solid understanding of the fundamental concepts, technologies, and architectures that define the Internet of Things. By the end of the course, students will be able to identify key IoT components, understand how connected systems operate, and explore various applications in domains such as healthcare, smart cities, industry, and automation. The course also aims to introduce students to communication protocols, data acquisition, and the challenges related to interoperability, and intelligent processing in IoT systems.

The rapid growth of the IoT is made possible by advances in low-power electronics, wireless communications, and cloud computing. These technologies enable the design of innovative environments where objects can interact autonomously and contextually, allowing for seamless integration. Mastering these technological building blocks is the first step toward implementing effective and scalable IoT applications.

Level of study

This course targets Master 1 students specializing in Networks and Multimedia.

Assessment procedures

Students will be assessed based on practical work and workshop activities. A final exam will be held at the end of the module.

Content

1 Introduction

The evolution of technologies has oriented the world of research and innovation towards new perspectives. Indeed, academics and industrialists are investing in what is now called the Ecosystem (e.g., home automation, building automation, transportation, health, military, etc.), where electronic components interact with the user in a transparent manner. These new applications represent another generation that owes its existence to the miniaturisation of electronic components and innovative technologies allowing the exchange of information anywhere and at any time. The complexity of modern application ecosystems explains the vast number of available components, software libraries, tools, and communication standards that facilitate communication between processes. All these sensors, which can be installed on various items (such as doors, machines, windows, desks, faucets, etc.) or worn by individuals, will play a crucial role in enhancing the quality of life for older people or those with reduced mobility, thereby increasing their autonomy. These advances and progress in artificial intelligence and robotics make it possible to enhance the intelligence of the Internet of Things (IoT) and make it more intuitive. Indeed, the IoT allows complex distributed systems to interconnect individuals and IoT devices. The technologies associated with the IoT must promote and simplify the ability to perceive and adapt to users' situations by providing personalised and tailored services, thereby enhancing user experience. For example, thanks to IoT devices, doctors, patients, environmental protection services, and citizens should be able to personalise messages based on individual health issues, personal safety, etc.

IoT systems are designed to support a wide variety of applications and operate in diverse environments. Some offer specific features such as quality of service (QoS), information tagging, and/or information routing. Most existing messaging protocols are available as open-source programming libraries. Many of them are integrated into IoT platforms, which are ready-to-use solutions that encapsulate key components of an IoT system. The problem is that there are many protocols, and choosing the right one for a given solution is not a trivial task. Examples include MQTT, CoAP, STOMP, XMPP, WAMP, AMQP, LwM2M, Weave, and HomeKit.

In the digital era, sensors with distributed intelligence are transforming healthcare. They collect a significant amount of data, posing a challenge for information gathering, detecting hidden information, and making intelligent decisions. Imagine sensors installed in the human body and connected via the internet. In a medical emergency, any situation that arises within the patient can

be immediately transmitted to the attending physician for life-saving action. This real-time processing, a challenge for IoT, is made efficient and effective through the combination of IoT and AI. Ensuring learners have a solid foundation in this field is crucial, hence the updating of all chapters of the subject.

By taking a course in the Internet of Things (IoT), the student will develop the following skills:

• Technical skills: Building an IoT system from sensor to platform.

• Analytical skills: Evaluating technical choices based on real-world constraints.

• Interdisciplinary skills: Coordinating hardware, software, network, and application aspects.

• Critical skills: Identifying the social, ethical, and regulatory issues of the IoT.

2 Chapter 1: Fundamental Concepts of the Internet of Things (IoT)

This chapter aims to introduce the conceptual foundations of the Internet of Things (IoT) and its major application areas. It provides an understanding of how physical objects, equipped with sensors and communication units, can be connected to networks to collect, process, and exchange data. The chapter also addresses the evolution of M2M (Machine-to-Machine) towards the IoT, typical architectures, software platforms, and the challenges related to standardisation. Examples of applications in the fields of health, industry, and ambient intelligence illustrate the scope and diversity of this emerging technology.

2.1 Foundations of the Internet of Things (IoT)

The study of the fundamentals of IoT aims to understand the architecture, technologies, protocols, and communication models required to design and deploy interconnected smart systems. It provides the essential foundations for understanding the IoT ecosystem, a technology that is becoming increasingly relevant and important across various fields, including health, energy, logistics, and agriculture. Consequently, the Internet of Things (IoT) refers to a network of interconnected physical devices—such as sensors, actuators, and appliances—capable of collecting, exchanging, and processing data via the Internet. These 'smart' objects can perceive their environment, interact with each other, and make simple or complex decisions, both locally and in networks or the cloud. The IoT has emerged as an essential technology, with applications in many fields. The IoT has its roots in several earlier technologies: ubiquitous information systems, sensor networks, and embedded computing. The term "IoT system" more accurately describes the use of this technology than the Internet of Things. Most IoT devices are connected to form specific systems; they are less commonly used as general access devices on a global network.

The surge of interest in the Internet of Things can be largely attributed to the proliferation of microelectromechanical sensors (MEMS). These sensors, including accelerometers, gyroscopes, and chemical sensors, have opened up a world of new applications. Their low cost and power consumption have extended the reach of IoT systems far beyond the capabilities of traditional laboratory or industrial measurement equipment. The key driver of this shift towards IoT development is the increasing use of sensor applications, all made possible by the affordability of digital and analogue electronics.

2.1.1 IoT definition

The concept of ubiquitous computing (UbiComp) was introduced in 1990 by Mark Weiser (who died on April 27, 1999), a researcher at Xerox PARC. Weiser's vision was of a world where information technologies would be seamlessly integrated into everyday environments, making them invisible yet ubiquitous. UbiComp thus refers to a set of autonomous, transparent, interconnected, and cooperative computing devices that harness the power of ICT to serve the user, often implicitly, offering a promising future. This concept prefigures what we now refer to as the Internet of Things (IoT). According to Alcañiz (2005), UbiComp involves integrating microprocessors into everyday objects, such as clothing, food, and furniture. This interdisciplinary vision also aligns with the concepts of the Internet of Everything (IoE) and the Internet of Robotic Things (IoRT), which further broaden the scope of IoT.

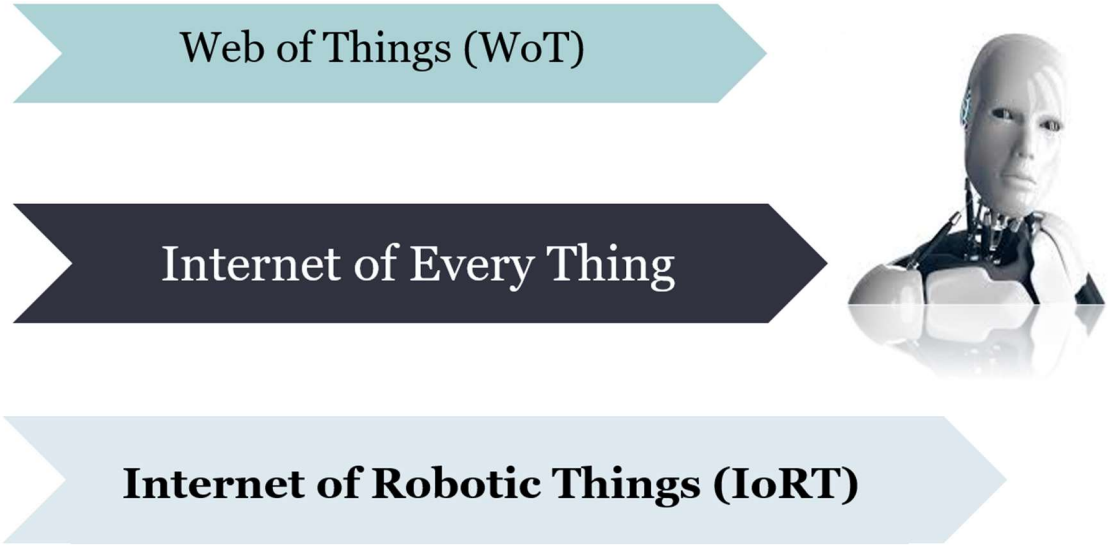


Figure 1 IoT-related paradigms

However, with the diversity of approaches and developments, it is becoming increasingly hard to propose a standardised definition of the term IoT, despite numerous attempts in the scientific and industrial literature.Given the diversity of IoT technology advocates, no single, precise definition of the term has emerged. The IoT goes beyond ubiquitous computing and information systems, which focused on data. Sensor network research has covered a diverse range of configurations. Many were designed for very low-speed data collection. The collected data was then sent to servers for processing. Traditional sensor network research did not emphasise network processing. Embedded computing focuses on either standalone devices or tightly coupled networks, such as those used in vehicles. Consumer electronics and cyber-physical systems were

two major application areas for embedded computing; both emphasised technical systems with well-defined objectives.

2.2 Key Components to Master in IoT Foundations

1. Sensors and actuators: to perceive and act on the physical environment.
2. Communication: protocols (MQTT, CoAP, HTTP, LoRa, etc.) and networks (Wi-Fi, ZigBee, NB-IoT, 5G, etc.).
3. Processing and storage: cloud computing, edge computing, IoT platforms.
4. Interoperability and standardisation: data models (RDF, JSON-LD) and standards (IETF, IEEE, ETSI).
5. Security and confidentiality: access management, encryption, reliability.

2.2.1 Connected physical objects (sensors and actuators)

Objects (devices, machines, sensors) are equipped with electronic components that allow them to collect data (temperature, movement, humidity, etc.) or to act on the environment (open a valve, turn on a light). However, system and application bottlenecks occur when a process can't keep up with the requests being made to it. A classic example of a system bottleneck is a poorly tuned database where applications and processes wait for database connections to become available or for database locks to be released. This leads to a degradation in response time, a critical issue, and eventually, requests time out.

When running on batteries, the short lifetime of sensors is a significant issue in running WSNs for environmental monitoring applications. In most of these applications, it may be difficult to change the node's battery frequently, and when the sensors exhaust their battery, this node can be considered dead. In recent decades, authors have focused on developing energy-saving techniques to minimise the energy usage of sensors at the medium access (MAC), routing, and physical (PHY) layers. To address the power constraint of battery-driven sensors, a new WSN platform that assists in harvesting energy from the surrounding environment has been proposed. Furthermore, the renewable energy system is the most promising route to address the EE problem of EHWSNs located in remote and rural areas. In the terrain profile, it may be difficult to replace batteries due to geographical limitations, which make access to this site difficult. The harvested energy is converted into an electrical signal that is stored or used directly for future purposes. For example, using solar panels to charge a rechargeable battery during the day.

The emergence of energy-harvesting circuits has sparked significant interest in various autonomous applications, including wearable IoT nodes and wireless sensor networks. Today, electronic devices that perform complex tasks consume small amounts of energy, and their uninterrupted operation can be supported by harnessing ambient energy from low-power sources, such as ambient radio frequencies (RF), heat, and indoor or outdoor light. This innovative approach is reshaping the possibilities of energy management in the field of wireless sensor networks.

The potential of energy harvesting in IoT systems is vast. These systems can contain subnetworks of different types of devices, including constrained devices with limited computing and memory resources due to price or energy limitations. Energy harvesting, also known as energy recovery or ambient power, is the process by which energy is derived from external sources (e.g., solar energy, thermal energy, wind energy, salinity gradients, and kinetic energy), captured, and stored for small, autonomous wireless devices, such as those used in wearable electronics and wireless sensor networks. This potential opens up new avenues for energy management in IoT systems.

Energy harvesters provide a minimal amount of energy for low-power electronics. Unlike some large-scale production methods that require resources (such as oil, coal, etc.), the energy source for energy harvesters is present as ambient background noise and is free. For example, temperature gradients exist due to the operation of a combustion engine. In urban areas, a significant amount of electromagnetic energy is present in the environment due to radio and television broadcasting. This abundance and accessibility of ambient energy sources provide a reassuring foundation for the future of energy harvesting.

Energy can also be harvested to power small, autonomous sensors such as those developed using MEMS technology. These systems are often very small and low-power, but their applications are limited by their reliance on battery power. Harvesting energy from ambient vibrations, wind, heat, or light could allow smart sensors to operate indefinitely. Several academic and commercial groups have been involved in the analysis and development of vibration-powered energy harvesting technology, including ARVENI, AdaptivEnergy, and MIT Boston. Typically, energy can be stored in a capacitor, supercapacitor, or battery. Capacitors are used when an application requires providing large bursts of energy. Batteries lose less energy and are therefore used when the device requires a constant flow of energy.

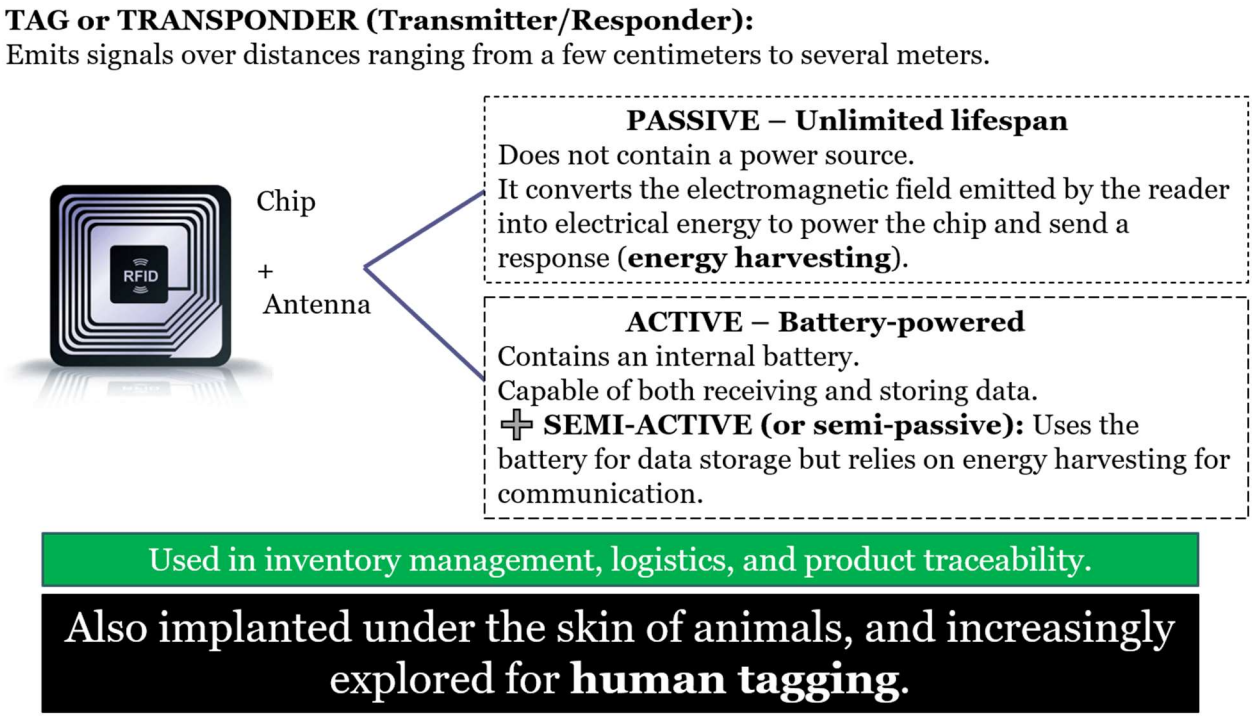


Figure 2 RFID: Basic Principle

Another way to harvest energy is by oxidising blood sugars. These energy harvesters are called biobatteries. They could be used to power implanted electronic devices (e.g., pacemakers, wearable biosensors for diabetics, and active RFID devices, Figure 2). Currently, the Minteer group at Saint Louis University has created enzymes that could be used to generate energy from blood sugars. However, the enzymes would still need to be replaced after a few years. In 2012, a pacemaker was powered by implantable biofuel cells at Clarkson University under the direction of Dr. Evgeny Katz. Biomechanical energy harvesters are also being created. A current model is Max Donelan's Biomechanical Energy Harvester, which straps around the knee. Devices like this can generate 2.5 watts of power per knee. This is enough to power about 5 cell phones. The Socket can generate and store 6 watts. Bionic Power, based in Canada, also develop a knee brace. Body energy can also be extracted, as described for wristwatches (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3146093/), from blood for pacemakers. For more information, see the following paper: https://link.springer.com/article/10.1007/s00542-014-2103-1

2.2.2 Connectivity and Communication

To share data, devices must be connected to a network. This connectivity allows physical devices (sensors, actuators, smart devices) to transmit or receive information in real time. The choice of network depends on the required range, power consumption, and throughput.

Examples of technologies:

- Wi-Fi: high-speed, for home or business use.
- Bluetooth: short-range, useful for personal devices.
- Zigbee / Z-Wave: low-power, suitable for smart homes.
- LoRa / Sigfox: long-range, low-throughput, for remote devices.
- 5G / LTE-M / NB-IoT: cellular networks suitable for industrial and mobile IoT.
    - Communication Protocols
    - Devices use specific protocols to communicate efficiently.

These protocols are designed to operate with limited resources (bandwidth, battery) and in constrained environments. Main IoT protocols:

- MQTT (Message Queuing Telemetry Transport): A lightweight protocol based on the publish/subscribe model, ideal for sensor applications.
- CoAP (Constrained Application Protocol): REST protocol for low-resource objects.
- HTTP/HTTPS: Primarily used for web interfaces, less optimised for connected devices.
- AMQP, XMPP: For specific cases requiring reliability or presence.

2.3 Data Acquisition and Processing

Raw data collected by sensors is transmitted to processing platforms (server, cloud, edge computing). Data Acquisition is the first step in the IoT data lifecycle. It involves collecting information from physical sources (sensors, RFID, cameras, GPS, etc.). The key Features consist of:

- Data types: temperature, humidity, position, image, movement, etc.
- Sampling frequency: as needed (e.g., every second, every minute, etc.).
- Reliability: Sensor accuracy and calibration are important.
- Security: data can be vulnerable from the moment it is collected (need for encryption at the source).

Technologies Used:

- Analogue or digital sensors.
- RFID modules, GPS modules.
- Microcontrollers (e.g., Arduino, ESP32) to capture data.

2.3.1 Data Processing

Once the data is collected, it must be cleaned, transformed, and analysed to make it worthwhile. This processing can be performed at different levels:

Local processing (Edge Computing)

1. Performed on the object or nearby (e.g., Raspberry Pi).
2. Advantages: Reduced latency, bandwidth savings, real-time decisions.

Cloud processing

1. CentralisedCentralised, robust, suitable for big data analysis.
2. Enables training of AI models, historical analysis, dashboards, etc.
   2.1. Typical processing steps:
3. Filtering: Removal of noise or corrupted data.
4. Aggregation: Averages, totals, max/min values, etc.
5. Analysis: Detection of events, anomalies, trends.
6. Preprocessing: Scaling, normalisation, encoding for AI.

Why is this step critical?

1. It ensures the quality of the data sent to higher layers.
2. It reduces the volume of data to be transmitted.
3. It allows automated actions to be triggered: e.g., alert if temperature exceeds threshold.

Integration into an IoT architecture

In a typical layered diagram, this phase occurs:

1. Between the perception layer (sensors) and the network layer (transmission).
2. Or in an Edge module between perception and the cloud.

2.4 IoT Platforms and Software

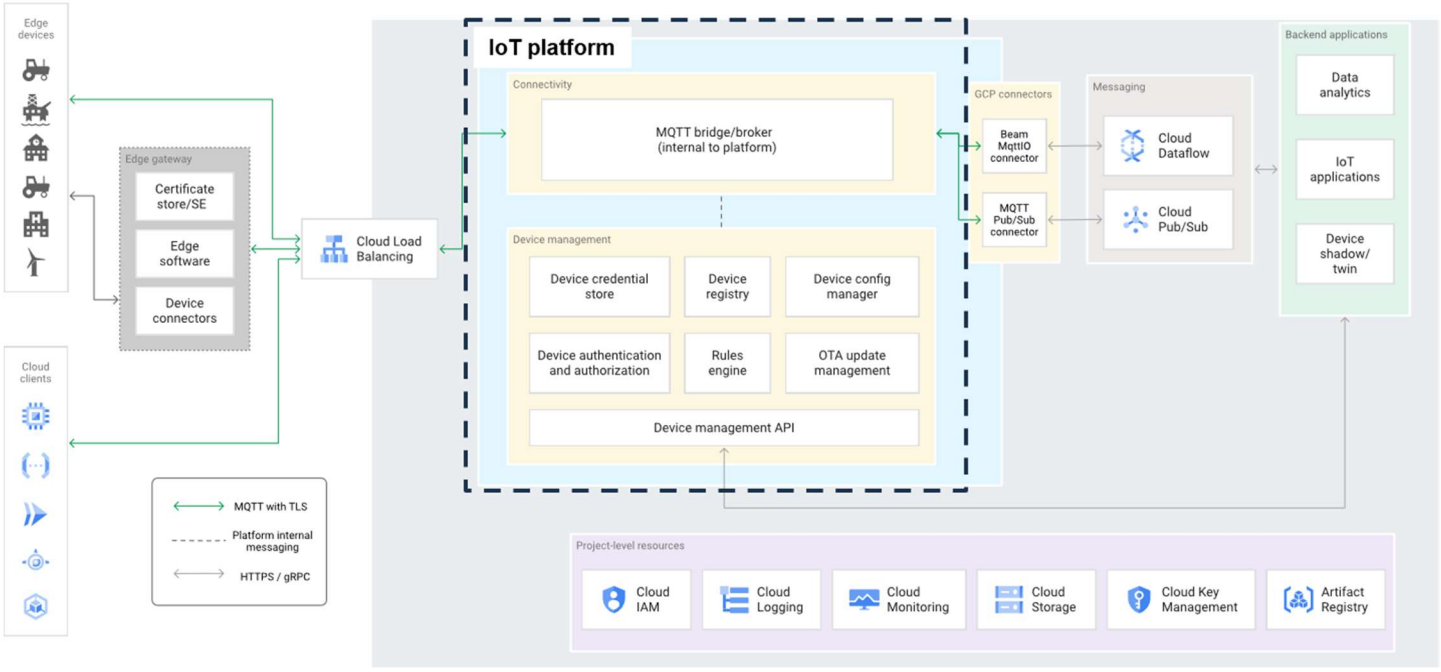


Figure 3 The IoT platform provides a set of device management services. https://cloud.google.com/architecture/connected-devices/iot-platform-product-architecture?hl=fr

Objects are connected to software platforms that enable monitoring, visualisation, remote control, and automation. Examples: ThingsBoard[1], Kaa[2], Blynk[3], Node-RED[4], AWS IoT Core[5], Azure IoT Hub[6]. IoT platform (Figure 3) products typically provide basic MQTT and HTTPS data connectivity. They also allow you to provision devices and provide functionality for authentication and management, telemetry storage and visualisation, data processing, and alerting. Enterprises often use IoT platforms when a standalone MQTT agent is insufficient for a use case and a more comprehensive IoT platform product is required. An IoT platform provides a unified interface for managing a heterogeneous collection of devices. This interface is essential for many connected device applications, and it's a key difference between an IoT platform and a standalone MQTT agent.

[1] https://thingsboard.io/

[2] https://www.kaaiot.com/

[3] https://blynk.io/

[4] https://nodered.org/

[5] https://aws.amazon.com/fr/iot-core/

[6] https://azure.microsoft.com/fr-fr/products/iot-hub

The data acquisition and processing phase is essential in any IoT architecture. It begins with the collection of data from sensors or physical devices (temperature, motion, pressure, etc.), followed by local or remote preprocessing.

This processing can include filtering, aggregation, anomaly detection, or data preparation for more advanced analysis (AI, dashboards, automated decisions). Depending on the case, processing can be performed at the edge (edge computing) for real-time response, or in the cloud for complex, large-scale analyses. This step plays a critical role in the reliability, security, and overall efficiency of the IoT system.

A complete IoT system needs	**IoT platforms help to**
1. Hardware (sensors or devices that collect environmental data or perform actions such as watering crops). 2. Connectivity: Transmitting all this data to the cloud (information or action). 3. Software hosted in the cloud is responsible for analysing the data it collects from the sensors and making decisions. 4. A user interface for interaction (e.g., a web application to manually activate or deactivate systems (e.g., irrigation).	1. Connect hardware, such as sensors and devices, 2. Manage various hardware and software communication protocols 3. Ensure security and authentication of devices and users 4. Collect, visualize, and analyze data collected by sensors and devices Integrate the above with existing enterprise systems and other web services.

2.5 System Architecture (IoT Layers)

The IoT is often based on a layered architecture, (Figure 4):

1. Perception layer: sensors and physical objects
2. Network layer: data transmission
3. Application layer: services offered to the end user

This data can be analysed in real time or stored for further processing. Using messaging as part of an overall enterprise architecture solution enables greater architectural flexibility and agility. These qualities are achieved through the use of abstraction and decoupling. With messaging, subsystems, components, and even services can be abstracted to the point where they can be replaced with little or no knowledge by client components. Architectural agility is the ability to respond quickly to a constantly changing environment.

By using messaging to abstract and decouple components, one can quickly respond to changes in software, hardware, and even business operations. The ability to replace one system with another, change technology platforms, or even change vendors without affecting client applications can be achieved through abstraction using messaging. With messaging, the message producer or client component doesn't know what programming language or platform the receiving component is written in, where the component or service is located, what the implementation name of the component or service is, or even the protocol used to access that component or service. It is through these levels of abstraction that we can replace components and subsystems, thereby increasing architectural agility more easily.

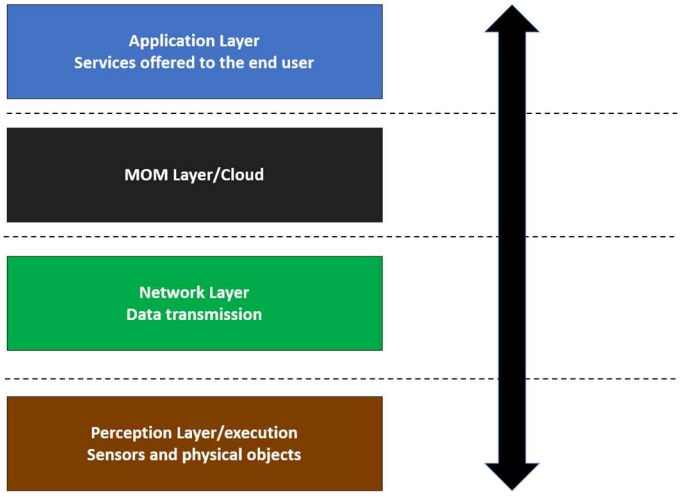


Figure 4 IoT layered architecture

2.5.1 IoT Messaging

Enterprise messaging (Figure 5), a concept with a rich history, has been supported by robust products such as IBM Web-Sphere MQ, SonicMQ, Microsoft Message Queuing (MSMQ), and TIBCO Rendezvous (https://www.tibco.com/products/tibco-rendezvous) for many years. The recent emergence of open-source messaging products like ActiveMQ, which are now being used

in enterprise production environments, further underscores the enduring relevance of this field. Additionally, the advent of service-oriented architecture (SOA) has led to the development of a new type of messaging product, the Enterprise Service Bus (ESB).A key concept in enterprise messaging is that messages are transmitted asynchronously from one system to another over a network. Asynchronous message delivery means that the sender is not required to wait for the message to be received or processed by the receiver; they are free to send the message and continue processing. Asynchronous messages are treated as self-contained units; each message contains all the data and state necessary for the business logic that processes it. This data and state could include information about the sender, the intended recipient, the type of message, and any additional context or metadata.In the world of asynchronous messaging, applications employ a simple API to construct a message, which is then passed to message-oriented middleware for delivery to one or more recipients. A message, a set of business data sent from one application to another over the network, must be self-describing. This means that it contains all the necessary context for the recipients to independently perform their tasks. It's crucial to clarify the term' client'. Messaging systems are composed of messaging clients and a messaging middleware server. Clients send messages to the messaging server, which plays a pivotal role in distributing these messages to other clients, thereby maintaining a sense of order and structure in the messaging system.

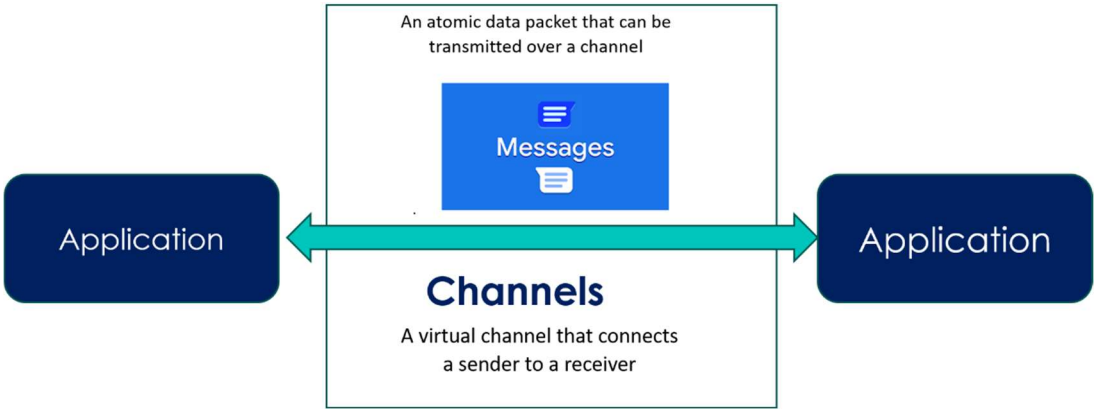


Figure 5 Some concepts

Synopsis

- Messaging makes applications loosely coupled by allowing them to communicate asynchronously.
- Two applications don't necessarily have to run simultaneously.

- Messaging enables the messaging system to transfer data from one application to another.
- Applications focus on the data to be shared without worrying too much about how to share it.

2.5.2 Point-to-Point

The point-to-point messaging model enables JMS clients to send and receive messages synchronously and asynchronously via virtual channels, known as queues. In the point-to-point model, message producers are referred to as senders, and message consumers are referred to as receivers. The point-to-point messaging model is traditionally a pull-based or polling-based model, where messages are requested from the queue rather than being automatically pushed to the client. One of the distinguishing features of point-to-point messaging is that a single receiver receives messages sent to a queue. However, multiple receivers may be listening on a queue for the same message, Figure 6.

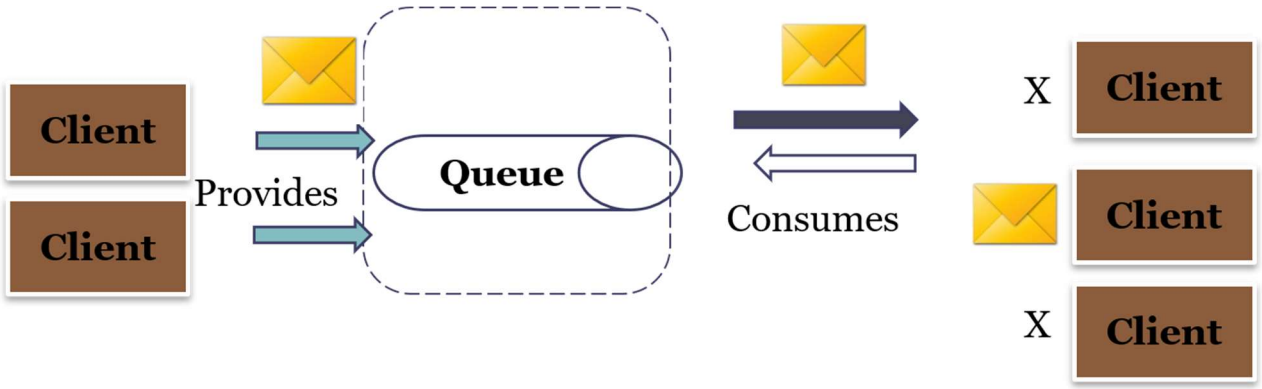


Figure 6 Point-to-point messaging model

Point-to-point messaging supports both asynchronous "fire-and-forget" messaging and synchronous request/response messaging. Point-to-point messaging tends to be more tightly coupled than the publish-and-subscribe model, as the sender generally knows how the message will be used and who will receive it.

The point-to-point model supports load balancing, which allows multiple receivers to listen on the same queue, thus distributing the load. The JMS provider handles queue management, ensuring that each message is consumed only once by the next available receiver in the group. The JMS specification does not dictate the rules for message distribution among multiple receivers, although some JMS providers have chosen to implement it as a load-balancing capability.

Point-to-point also offers other features, such as a queue browser that allows a client to view the contents of a queue before consuming its messages. This browser concept is not available in the publish-and-subscribe model.

2.5.3 Publish-and-Subscribe

In the publish-subscribe model, messages are published to a virtual channel, known as a topic. Message producers are referred to as publishers, while message consumers are referred to as subscribers. Unlike the peer-to-peer model, messages published to a topic using the publish-subscribe model can be received by multiple subscribers. This technique is sometimes referred to as broadcasting a message. Each subscriber gets a copy of each message. The publish-subscribe messaging model is a push-based model, where messages are automatically broadcast to consumers without them having to request or poll the topic for new messages, Figure 7.

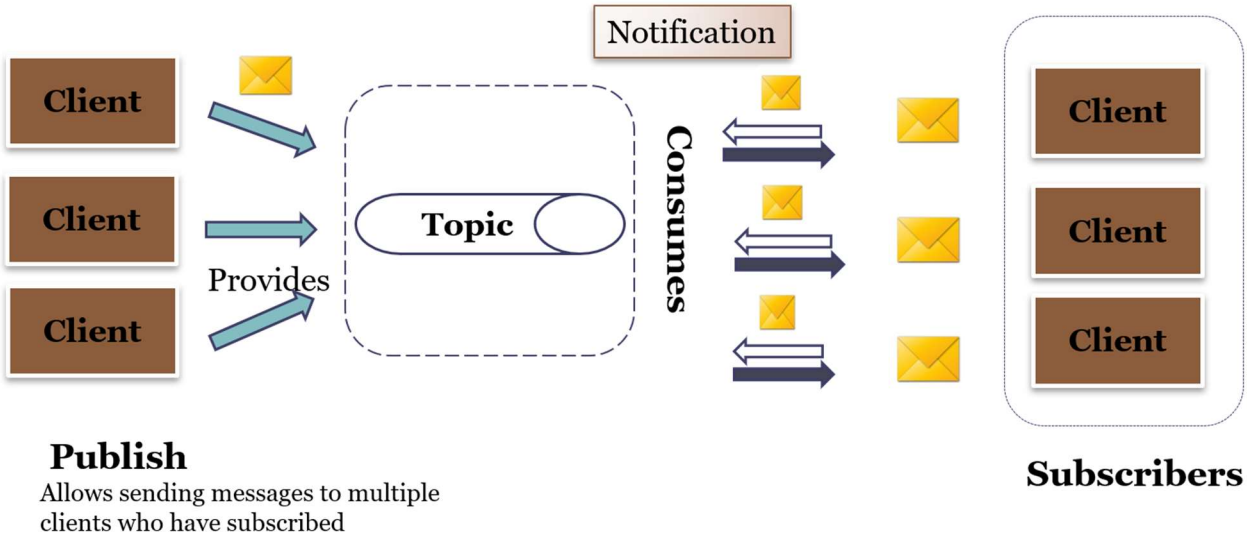


Figure 7 Publish-subscribe model

The pub/sub model tends to be more decoupled than the peer-to-peer model because the message publisher typically doesn't know how many subscribers there are or what those subscribers do with the message. For example, suppose a message is published to a topic whenever an exception occurs in a Java application. The publisher's responsibility is to broadcast that an exception has occurred. The publisher generally doesn't know or care how this message will be used.

There are various types of subscribers in the publish/subscribe messaging model. Non-durable subscribers are temporary subscriptions that receive messages only while they are actively

listening to the topic. Durable subscribers, on the other hand, will receive a copy of every published message, even if they are offline at the time the message is published. There is also the notion of dynamic durable subscribers and managed durable subscribers.

Among the general interfaces, ConnectionFactory and Destination must be obtained from the provider using JNDI (in accordance with the JMS specification; see also the labs). The other interfaces are created via factory methods in the various API interfaces. For example, once you have a ConnectionFactory, you can create a Connection. Once you have a Connection, you can create a Session. Once you have a session, you can make a message, a message producer, and a message receiver.

2.6 Context, ambient intelligence

To promote complex IoT systems with greater flexibility and user comfort, we aim to ensure dynamic reconfiguration. To do this, we must be aware of the properties and characteristics of these systems. IoT is inherently linked to ambient intelligence and autonomous control, i.e., the system's self-control capability. An IoT system is a context-aware system, meaning changes and variations strongly influence it in its context. Each connected object is composed of one or more sensors that allow it to acquire context data and be aware of the environment to which it belongs. According to Ferry et al. (2008), context can be classified into four categories to define the conditions that can trigger adaptation: user context, machine context, temporal context, and environmental context. In contrast, Bettini et al. (2010) found that rich context models, which integrate sensed, static, user-provided, and derived information, are the most useful. Other sources of contextual information can be related to the system's characteristics, or in other words, knowledge of its structure, architecture, sudden addition or deletion of existing nodes/entities (one of the characteristics of IoT).

Referring to the context-aware system structure, the detected raw data must be processed and filtered. Here, a standard data format is necessary to unify the large amount of data from various sources. Indeed, the evolution of technology has given hope to a category of people to stay at home and live in their usual environment. This has led to the need to build systems capable of monitoring and assisting them in their environment. These habitats, now known as smart homes, serve a dual purpose: providing comfort and security for individuals and saving governments money (i.e., keeping these people at home reduces the burden on hospitals and lowers the cost of healthcare and social assistance). It is therefore not surprising to see the emergence of another

class of applications, "Ambient Assisted Living (AAL)," whose goal is to formally specify the needs to assist the elderly and people with reduced mobility. This class of application aims to help people accomplish tasks, simplify the performance of daily activities (e.g., reminding them to take medication), prevent dangers by minimising risks, and take appropriate action in a situation or context by triggering an alarm if the conditions are met. The complexity of understanding situations/contexts depends on the objective for which a system is built. Indeed, it is not easy to design a system to supervise and control a category of people whose intellectual capacity is impaired, tasks are not coordinated, and gestures are often inappropriate to human reasoning. However, detecting abnormal behaviours, although challenging to achieve, remains possible compared to the latter category (e.g., to prevent Alzheimer's disease) or building context-aware systems for AAL environments in general.

Furthermore, the successful application of robots in industrial settings has prompted researchers to integrate them into human life. The creation of the "Robot Companion" concept demonstrates the desire to provide medical assistance and safety to humans. These robots are defined by ([Kim et al., 2008a], [Kim et al., 2008b]) as Ubiquitous Robots offering various services anywhere and anytime. The OMG defines a robotic system as any system that provides an intelligent service in a ubiquitous environment through the use of multiple actuators and communication interfaces with human beings. Thus, they can be augmented by a system, in this case, we speak of a "System of Systems": Where a system is embedded on the robot, allowing the latter to have a "closed" knowledge of the environment in which it operates. In contrast, a second system can be dedicated to piloting the entire environment, including the robot. Indeed, these robots are equipped with sensors that enable them to perceive their environment: a camera, a light detector, GPS, a heat detector, and a touch screen to communicate with humans, among others.

Moreover, the robot must be able to adapt to this dynamic environment (i.e., process, analyse, correlate, and understand complex events in which humans are the main actors). In this regard, [Minsky, 1955] said: “[an intelligent machine] would tend to build up within itself an abstract model of the environment in which it is placed. If it were given a problem, it could first explore solutions within the internal abstract model of the environment and then attempt external experiments.” From that date to this day, knowledge modelling and reasoning based on the symbolic representation of the real world have taken over. The system must therefore be able to collect, analyse, process, and interpret physical information. Complexity can be divided into two main parts: the physical component (e.g., a fundamental aspect at the interface level with these information sources) and the abstract representation (e.g., the system uses a symbolic abstraction

of entities and the events they generate). The emergence of middleware has made it possible to conceal the heterogeneity of these components. Indeed, they consist of an interface residing between the system and the information sources, providing an interoperable communication medium to simplify the integration of these components. The authors also emphasised the importance of having a history of all the activities performed by these individuals. They identified numerous challenges to be addressed in building a smart home to assist people with Alzheimer's disease, for example.

However, the data, knowledge, and heterogeneity of components restrict the deployment of components in this type of environment, hindering their integration, transparent exchange, and knowledge sharing. The need for the most expressive abstract model possible, capable of providing accurate answers to queries, is therefore essential. The work carried out by academics demonstrates that ontologies are a promising technique for solving the heterogeneity problem and facilitating knowledge sharing. Ontologies provide a means to define modelling primitives, which are abstractions of entities and their relationships (i.e., explaining concepts, individuals, properties, formal axioms, and common terminology). All researchers believe that a well-defined model is the key to building a context-aware system. The authors studied and presented a state-of-the-art of the approaches used for context recognition. The approaches were evaluated based on their relevance to ubiquitous computing.

2.6.1 Context-Aware Systems

First, what does a smart environment consist of?

It represents a new vision of future life, where heterogeneous components communicate using new technologies in ways invisible to humans. Thus, the definition of "ubiquitous computing" as the integration of microprocessors into everyday objects (such as clothing and food) is very appropriate for what we experience today. For (Zelkha & Epstein 1998), the characteristics of ubiquitous computing include:

1. embedded: A set of networked components within the environment

2. context-aware: Can recognise the individual and the context in which they are located

3. personalised: Can be adapted to the user's needs

4. adaptive: They can change in response to the user

5. Anticipatory: They can anticipate user intentions and desires without conscious mediation

Recently, research has shifted towards the Ambient Intelligence (AmI) paradigm. The latter aims to create intelligent environments that can react to changes in human needs. The term AmI was coined by the European Commission, whose first project was the launch of the Programme Advisory Groups, which initiated the AmI challenge. The figure above illustrates the technological evolution up to the era of the Ambient Intelligence paradigm, which is considered an extension of Ubiquitous Computing.

Many researchers believe that AmI is closely related to the intelligent service system, whose technologies are capable of automating a platform, integrating the devices required to offer personalised, adaptive, and anticipatory services. The authors have identified the main technologies that an ambient system needs to provide an acceptable level of intelligence. Thus, AmI brings together the results of research related to ubiquitous computing and context-aware systems. The need to design context-aware systems has led to the specification of another class of applications whose construction of this type of system is more challenging. This is the supervision and control of elderly or mobility-impaired people living alone, a consequence of demographic change. For these individuals, technologies are often not readily accepted; therefore, transparency and ease of use are crucial factors in the design of these systems. Thus, Ambient Assisted Living (AAL) aims to be the solution to this problem, allowing these individuals and the economy to benefit from technological advances. An AAL system should help prevent chronic and mental illnesses, assist people in their daily lives (such as finding an object, restoring comfort, reminding individuals to take medication, alerting the hospital or relatives in the event of an accident, etc.).

2.7 Security and Privacy

Security and privacy are critical pillars of the IoT. The complexity of the ecosystem, the heterogeneity of devices, and their constant exposure demand constant vigilance. An unsecured IoT system poses a risk not only to the user, but also to critical infrastructure. This risk is particularly concerning when it concerns vulnerable populations, such as children, the elderly, or people with disabilities, whose personal data, lifestyle habits, or location information can be exploited for malicious purposes. It is therefore essential to integrate enhanced protection mechanisms, adapted to these groups, into any IoT device. Connected devices are vulnerable to attacks; therefore, the main security objectives must be guaranteed:

- Confidentiality: Prevent unauthorised access to data;
- Integrity: Ensure data is not tampered with;
- Authentication: Ensure that entities are who they claim to be;
- Availability: Keep services accessible (resistance to DoS attacks);

Common Threats:

- Data interception: sniffing or network eavesdropping (mainly if the data is transmitted unencrypted);
- Identity theft: An attacker impersonates a sensor or gateway;
- Malware and IoT botnets: e.g., Mirai, which turns vulnerable connected devices into bots;
- Physical attacks: physical access to sensors, which may be poorly protected. Privacy breach: sensitive data collected without explicit consent (e.g., location, health, videos);

Before presenting some types of attacks and the proposed solutions that target IoT systems, it is essential to understand why these environments are particularly exposed. Connected objects are often deployed in large numbers, in a wide variety of contexts (home, industry, healthcare, smart cities), and have limited resources (battery, memory, computing capacity). In addition, they collect and transmit sensitive data, sometimes continuously. The absence of universal standards, irregular updates, and a lack of cybersecurity awareness exacerbate this vulnerability. These factors make the IoT a prime target for cyberattacks, which are all the more concerning when they affect children, the elderly, or other vulnerable populations.

To address common attacks on IoT systems, several security measures can be implemented: Communication encryption: Use secure protocols such as TLS/DTLS or lightweight algorithms (e.g., AES, ECC) to protect exchanged data;

- Strong authentication: Implement robust device and user identification mechanisms (certificates, tokens, complex passwords);
- Regular firmware updates: Ensure that connected devices receive security patches as soon as a vulnerability is detected;
- Network segmentation and filtering: Isolate critical devices, limit unnecessary communications, and use appropriate firewalls;

- Privacy protection: Apply strict privacy policies, collect user consent, and anonymise sensitive data;
- Protection of vulnerable groups: Strengthen security settings for devices used by children, the elderly, or dependents (e.g., parental controls, limiting accessible functions, contextual monitoring);
- Security by design: integrate security mechanisms from the development of IoT objects and services, taking into account resource constraints;

In addition to traditional security mechanisms, several advanced technologies are currently being explored to strengthen the protection of IoT systems. Blockchain guarantees the integrity and traceability of data exchanged between connected objects, eliminating the need for a trusted third party. It is particularly useful in distributed contexts (logistics, healthcare, smart cities). Combined with smart contracts, it enables the automation of security or access control rules, while ensuring their transparent and tamper-proof execution. Furthermore, role-based access control (RBAC) models assign permissions based on the functions of a user or device (e.g., a doctor can access a patient's medical data, but not their private location). In contrast, attribute-based access control (ABAC) models authorize or deny access based on a set of dynamic conditions (time, location, device security level, etc.). These approaches are particularly suited to heterogeneous and evolving IoT environments, especially when it comes to protecting the sensitive data of vulnerable groups.

The integration of machine learning (ML) and deep learning (DL) into IoT security opens new perspectives for proactive threat detection. These techniques enable the analysis of large amounts of data generated by connected devices to identify abnormal behaviour, network anomalies, or intrusion attempts. For example, a supervised model can learn to distinguish normal sensor behaviour from suspicious activity. At the same time, deep neural networks (DL) can detect complex attacks, even those unknowns in advance, thanks to their generalisation capabilities. Furthermore, approaches such as federated learning allow models to be trained locally on devices, without centralising sensitive data, thus strengthening confidentiality while maintaining intelligent surveillance. These solutions are beneficial in sensitive contexts, such as health, education or home automation, where the protection of children and vulnerable people is a priority.

2.8 Intelligence and Automation

• The IoT becomes more powerful when combined with artificial intelligence (AI), machine learning, or smart rule systems.

• This enables autonomous responses (e.g., alerts when gas is detected) or intelligent optimisations (energy management, predictive maintenance, etc.).

2.8.1 Internet of Things and Big Data

Figure 8 Big Data: The 4 V's Principles

The Internet of Things (IoT) and Big Data are closely linked and interdependent. Connected objects continuously generate vast volumes of real-time data, including sensor readings, GPS signals, user interactions, environmental changes, and more. This massive flow fuels the Big Data ecosystem, which plays a crucial role in making IoT data useful by providing the tools necessary to collect, store, process, and analyse data at scale. In turn, Big Data analytics can extract valuable insights from IoT data, facilitating predictive maintenance, anomaly detection, intelligent automation, and real-time decision-making across diverse fields, including healthcare, transportation, agriculture, and smart cities. Without Big Data technologies, the potential of IoT data would remain largely untapped.

The concept of Big Data is commonly defined by four fundamental dimensions, known as the 4 V's, Table 1.

Table 1 Big Data: The 4 V's Principles

	Definition
Volume	Refers to the massive amount of data generated every second by sensors, social media, transactions, and connected devices.
Velocity	Describes the speed at which data is generated, transmitted, and processed in real time or near-real time.
Variety	Represents the diversity of data types: structured (e.g., databases), semi-structured (e.g., XML, JSON), and unstructured (e.g., videos, text).
Veracity	Refers to the trustworthiness, quality, and accuracy of data, which may be affected by noise, inconsistencies, or ambiguity.

2.8.2 IoT and Big Data: Tools for Extracting Value from Connected Objects

The Internet of Things (IoT) generates massive amounts of data from sensors, smart devices, and embedded systems. To fully exploit this data, it is essential to leverage powerful Big Data tools, Figure 9.

1. **Hadoop** provides the backbone of distributed storage, capable of handling the growing volume of IoT data. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than relying on hardware to deliver high availability, the library itself is designed to detect and handle failures at the application layer, thereby delivering a highly available service on top of a cluster of computers, each of which may be prone to failure.
2. **Cassandra**, a highly scalable NoSQL database, is particularly well-suited for the fast and distributed storage of sensor data. Apache Cassandra is an open-source NoSQL distributed database trusted by thousands of companies for its scalability, high availability, and performance. To ensure reliability and stability, Cassandra is tested on clusters as large as 1,000 nodes and with hundreds of real-world use cases and schemas tested with replay, fuzz, property-based, fault-injection, and performance tests.
3. **Apache Spark** enables fast in-memory processing, ideal for predictive analytics or real-time applications based on IoT data. Apache Spark is a unified analytics engine for processing large-scale data. It provides high-level APIs in Java, Scala, Python and R,

and an optimised engine that supports general execution graphs. It also supports a rich set of higher-level tools, including Spark SQL for SQL and structured data processing, the Pandas API on Spark for Pandas workloads, MLlib for machine learning, GraphX for graph processing, and Structured Streaming for incremental computation and stream processing.

4. **Apache Storm** complements Spark for streaming processing, analysing data as it arrives, which is crucial in critical environments (e.g., healthcare, transportation). Apache Storm is a free and open source distributed real-time computation system. Apache Storm makes it easy to reliably process unbounded streams of data, doing for real-time processing what Hadoop did for batch processing. Apache Storm is simple, can be used with any programming language, and is a lot of fun to use. Apache Storm has numerous use cases, including real-time analytics, online machine learning, continuous computation, distributed RPC, ETL, and more. Apache Storm is fast: a benchmark clocked it at over a million tuples processed per second per node. It is scalable, fault-tolerant, guarantees your data will be processed, and is easy to set up and operate.. Apache Storm integrates with the queueing and database technologies you already use. An Apache Storm topology consumes streams of data and processes those streams in arbitrarily complex ways, repartitioning the streams between each stage of the computation however needed. Read more in the tutorial.
5. **Altair RapidMiner** offers an accessible platform for advanced analytics, machine learning, and visualisation, facilitating the exploitation of collected data. It plays a crucial role in the IoT data management process by empowering organisations to unlock data insights and harness data analytics and advanced AI automation for scalable, future-ready solutions.
6. **OpenRefine** enables the cleaning and structuring of raw data, a crucial step before any analysis, especially when sensors generate noisy or poorly formatted data. It is a powerful, free, open-source tool that can handle various types of 'messy data', such as data with missing values, inconsistent formatting, or duplicate entries. OpenRefine can clean, transform, and extend such data from one format into another, and even integrate it with web services and external data.
7. Finally, **MapR**, as a unified platform, integrates several of these components (Hadoop, Spark, Kafka, etc.) to provide a robust, real-time infrastructure for industrial IoT. This infrastructure is designed to handle the high volume and velocity of data generated by

industrial IoT applications, ensuring that data is processed and analysed in real time. MapR offers a Big Data platform that combines Apache Hadoop and Spark components, a real-time database, and storage. It is an ideal solution for companies looking to deploy a Big Data strategy without disruption, cost-effectively, and securely. MapR is a software company founded in 2009 and located in San Jose, California. It is behind several of the leading open source Hadoop projects, including Apache HBase, Apache Hive, Apache Zookeeper, and Apache Pig.

Figure 9 IoT and Big Data: tools for extracting value from connected objects

2.9 IoT Application Areas

IoT applications are numerous, including intelligent energy management, autonomous vehicles, smart cities, connected homes, e-health, precision agriculture, and industrial monitoring,

Figure 10 Examples of IoT application fields

2.9.1 medical sector

The Internet of Things (IoT) is not just a buzzword, but a powerful tool that is revolutionizing the medical sector and digital health (e-health). By connecting smart medical devices, wearable sensors, and digital platforms, the IoT is enabling a new era of continuous, real-time health data collection. This paves the way for more personalised, preventative, and responsive medicine, promising a brighter future for patient care. Connected devices, such as heart monitoring bracelets, blood glucose sensors, and remote respiratory assistance devices, are not just gadgets. They are powerful tools that enable professionals to monitor patients' health remotely, reducing unnecessary hospitalisations and enhancing home care. These devices automatically transmit data to secure cloud platforms, where it is analysed using cutting-edge artificial intelligence or big data algorithms.

In hospitals, the IoT contributes to better management of medical equipment, patient tracking, and the optimisation of logistics flows, while enhancing the safety of care. It also facilitates the real-time integration of electronic medical records (EMR) with monitoring devices. However, this revolution also raises major challenges, particularly regarding health data security, confidentiality, device standardisation, and interoperability between systems. Therefore, the medical IoT is a strategic tool for enhancing the quality of care. However, its successful implementation requires robust governance and adherence to stringent technical standards. These measures are crucial to

ensure the reliability, security, and trustworthiness of the IoT solutions deployed in the healthcare sector.

2.9.2 The Role of IoT in Robotics

The Internet of Things (IoT) plays a pivotal role in the evolution of modern robotics, enabling robots to interact in real-time with their environment, other connected devices, and humans. Integrating IoT sensors into robotic systems significantly improves their autonomy, intelligence, and adaptability. Thanks to the IoT, robots can continuously collect and exchange data via the cloud or local networks, allowing them to be monitored remotely, collaborate with other robots (collaborative robotics), or adjust their behaviour based on environmental conditions. For example, in Industry 4.0, connected robots optimise production processes using real-time data from the production line. In sectors such as logistics, agriculture, healthcare, and home automation, the IoT allows robots to perform tasks with precision while coordinating with other smart devices (drones, autonomous vehicles, temperature sensors, etc.).

This combination of IoT and robotics forms what is known as the Internet of Robotic Things (IoRT), an evolution in which robots are no longer simply autonomous but become connected, intelligent, and interoperable agents capable of interacting with an entire digital ecosystem.

2.9.3 The Role of IoT in Smart Vehicles

The Internet of Things (IoT) is profoundly transforming the automotive sector by enabling the emergence of connected vehicles. Thanks to onboard sensors, communication systems, and cloud platforms, vehicles can collect, exchange, and analyse data in real time. The IoT enables many key functions:

• Remote monitoring of vehicle health (engine, tire pressure, fuel consumption, etc.)

• Predictive maintenance, by detecting breakdowns before they occur

• Smart navigation, with real-time traffic information

• Advanced Driver Assistance Systems (ADAS), such as automated emergency braking or lane keeping

• Vehicle-to-Everything communication, where vehicles interact with other cars, road infrastructure, or pedestrians

• Personalisation of the user experience (seat position, multimedia preferences, intelligent climate control, etc.)

In public transportation and logistics, the IoT is also crucial for optimising routes, tracking fleets in real-time, and enhancing safety and energy efficiency. Ultimately, the IoT is a fundamental pillar for autonomous vehicles, as it enables real-time decision-making based on rich and constantly updated data.

2.9.4 The Role of IoT in Industry 4.0 and 5.0

The Internet of Things (IoT) is a fundamental pillar of the industrial transformations known as Industry 4.0 and Industry 5.0. As part of Industry 4.0, the IoT connects machines, sensors, production systems, and products to create smart factories. These connected objects collect real-time data on production, quality, energy, and maintenance. Companies can thus:

• Optimise manufacturing processes

• Anticipate breakdowns through predictive maintenance

• Accurately track inventory and supply chains

• Improve product traceability

The more recent Industry 5.0 goes beyond automation alone. It emphasises collaboration between humans and intelligent machines, sustainability, and human-centred production. In this context, the IoT plays a vital role in:

• Ensuring seamless human-machine interaction

• Collecting environmental data for eco-responsible production

• Offering personalised products through real-time analysis of customer preferences

In short, the IoT makes industry more agile, smarter, and more human by connecting field data to strategic decision-making systems.

2.9.5 The Role of IoT in Smart Cities

The Internet of Things (IoT) is a key technological driver in the creation of smart cities. It connects thousands of urban devices—sensors, cameras, traffic lights, meters, and vehicles—to collect, analyse, and leverage data in real-time. Thanks to the IoT, cities can:

• Optimise energy management with smart grids

• Improve traffic flow through traffic analysis and dynamic traffic light control

• Improve public transportation through real-time tracking of buses, trams, and bike sharing

• Monitor air and water quality and trigger pollution alerts

• Intelligently manage public lighting by adjusting brightness based on brightness or occupancy

• Improve security through smart video surveillance systems and automatic alerts

• Encourage citizen participation with mobile applications connected to public services

By connecting infrastructure, services, and citizens, the IoT helps build more sustainable, safer, and more responsive cities, where technology serves quality of life.

2.9.6 IoT & Artificial Intelligence for Predicting Animal and Human Behaviour

The combination of the Internet of Things (IoT) and Artificial Intelligence (AI) opens up powerful possibilities for predicting behaviour, both in the animal and human worlds. Using connected sensors (bracelets, smart collars, cameras, motion sensors, GPS, etc.), IoT enables the real-time collection of behavioural, physiological, and environmental data. This raw data is then analysed by AI algorithms capable of detecting repetitive patterns, identifying anomalies, or anticipating future events.

Main applications:

In animals (livestock, wildlife):

- Early detection of disease, stress, or behavioural disorders
- Monitoring of movements, feeding, or reproductive habits

- Alerts in the event of abnormal behaviour or escape from an enclosure

In humans (health, safety, well-being):

- Prediction of falls in the elderly using wearable sensors
- Monitoring of sleep patterns, physical activity, or cognitive disorders
- Detection of risky behaviours or social isolation
- Anticipation of care or intervention needs in real time

Combining IoT with AI not only improves quality of life and safety but also optismises human interventions, whether by caregivers, veterinarians, or environmental managers. It constitutes a key lever for building intelligent, reactive and proactive systems in the fields of health, agriculture, environmental monitoring and security.

2.10 Standardization, Industry Alliances

Technical standards play a crucial role in the digital economy by enhancing efficiency, reducing costs, and ensuring compatibility among products. They are essential for ensuring the interoperability of ICT systems, preventing vendor lock-in, and strengthening data security and confidentiality. The European Commission emphasises its importance for industrial competitiveness within the framework of the Digital Single Market. Regulation 1025/2012 provides a legal framework for standardisation in Europe. Standards support all economic actors (researchers, manufacturers, governments, etc.) and strengthen digital trust through common protocols and consistent security systems.

Technical standardisation is a key element in the development of the Internet of Things (IoT) due to the growing complexity of the devices, services, interfaces, and protocols involved. Without global standards, interoperability, security, quality of service, and management become increasingly difficult. Today, most IoT providers design proprietary solutions, resulting in technical heterogeneity and inconsistency between services. A standardised model (such as an IoT reference architecture) is necessary to ensure compatibility, portability, and common management of services, particularly in the cloud and backends.

Standardisation also addresses issues such as interconnectivity, security, and protocol divergence. It helps reduce costs, time to market, and challenges related to component design and

integration. By providing clear guidelines, standards facilitate the adoption of scalable, sustainable, and interoperable solutions in the IoT ecosystem.

2.10.1 Standards organizations

The following link presents a relevant document concerning the standardization of IoT: https://portail-qualite.public.lu/dam-assets/publications/normalisation/2020/national-technical-standardization-report-iot-june-2020.pdf

1. Internet Society (ISOC)
2. Internet Architecture Board (IAB)
3. IETF : Internet Engineering Task Force
4. IEEE : Institute of Electrical and Electronics Engineers
5. International Organization for Standards (ISO)
6. IUT International Telecommunication Union
7. OASIS Organization for the Advancement of Structured Information Standards
8. EIA :The Electronic Industries Alliance standards
9. IIC Industrial Internet Consortium
10. W3C (World Wide Web Consortium) : compatibilité des technologies du web
11. Electrical Industries Association (EIA)
12. Telecommunications Industry Association (TIA)
13. International Telecommunications Union – Telecommunications Standardization Sector (ITU-T)
14. ICANN (Internet Corporation for Assigned Names and Number)
15. Internet Assigned Numbers Authority (IANA)

2.11 The AIoT Paradigm

Our world is now home to billions of interconnected devices, all thanks to the Internet of Things (IoT) paradigm. The integration of intelligent systems and robots relies on the processing of sensor data and its interpretation at a high level of abstraction. To achieve this, these systems must employ artificial intelligence methods to extract and merge diverse information from various

sensor networks, medical records, social networks, the web, images, videos, and other sources. This results in systems that, once implemented, effectively meet the needs and requirements of manufacturers across various sectors. The potential applications are vast, including smart cars, human-machine interfaces, people detection and tracking systems, and signal analysis and interpretation systems (physiological, audio, etc.). However, it's crucial to address potential vulnerabilities that may arise when handling private data, such as confidentiality, integrity, and monitoring of physical and virtual access to devices in the context of the Internet of Everything. This also includes protection against the execution of unauthorized actions, ensuring availability, and maintaining traceability.

The rise of artificial intelligence, machine learning, and deep learning is increasingly prevalent in many fields (robotics (Internet of Robotic Things, IoRT), natural language processing, security, privacy protection, pattern detection, etc.). The primary goal of these paradigms is to optimize diagnosis and data processing. As a result, numerous applications are emerging in the health field, significantly impacting research and offering new perspectives in disease detection. However, machine learning presents several challenges, including the validation of the algorithms created, the transfer of algorithms from one population to another to ensure their performance, and the manipulation of personal data that can impede technological progress. In addition, the field of automatic natural language processing essentially enables a machine to read and understand unstructured textual, vocal, or visual (handwritten) data originating from a natural language. Therefore, modeling the world from theoretical/technical descriptions or explained by linguistic formulas requires approaches based on cognitive, symbolic and ontological psychology. It is not surprising, therefore, that the design of models in computer science coincides with models in cognitive psychology. The latter considers that a model is the set of data required by any system in order to interact with the environment. Complex systems must therefore process all the knowledge observed or inferred in an environment for decision-making. Internet and computer network technologies facilitate the manipulation, access, and use of virtual machines, resource sharing, and information storage in various databases. In addition, they provide services such as e-commerce, education, and entertainment, among others. Therefore, by intertwining the potential of AI and the Internet of Things (IoT) and IoRT, a spectrum of applications will be opened up to benefit human well-being. Thanks to AI, all devices and infrastructures are becoming smarter. Like humans, machines should use perception, knowledge representation, and reasoning techniques. The design of knowledge-based systems relies heavily on knowledge engineering. The latter is a field of artificial intelligence that focuses on the psychological/empirical modeling of an expert's

knowledge to solve a given problem. Confidentiality is a major challenge, particularly in the medical and genetic fields. Indeed, these data carry sensitive and confidential information. Therefore, it is necessary that a legal framework govern the private respect of individuals.

Thanks to the combination of IoT and AI, a new paradigm has been born: Artificial Intelligence of Things (AIoT). This new paradigm will enable the combination of deep and machine learning algorithms with IoT capabilities, allowing them to enhance their data processing, security, and other skills. These algorithms will enable the manipulation of a large amount of data generated by IoT devices in record time, compared to humans. This will allow the detection of anomalies in an IoT system, the prediction of system failures, and ultimately, the optimisation of decisions. Depending on its use, AI can leverage the speed of data analysis and prevent cybercriminal attacks on the network. Facial recognition will undoubtedly enhance security at sensitive sites, and traffic management in cities will benefit from the use of drones to regulate road traffic and maintain smooth traffic flow, particularly in emergencies. Thus, AIoT will play a significant role in the implementation of autonomous vehicles. AI will be able to identify anomalies and therefore avoid accidents, ultimately making driverless vehicles acceptable for circulation alongside other human-controlled vehicles.

2.11.1 The AIoT & Robotic

AI is at the heart of solutions to the challenges facing humans in various fields, including health, safety, ecology, chemistry, biology, and others. Innovations and advances in artificial intelligence, connected objects, and robotics will continue to transform people's daily lives. Technological advances and the emergence of ubiquitous computing and ambient intelligence paradigms have led to the creation of a new generation of service robots, known as ubiquitous robots. These robots have the cognitive ability to support vulnerable and isolated people. They can communicate with them as mobile and intelligent communicating objects, based on the vision of the Web of Things. Thus, they facilitate the creation of intelligent ecosystems to improve users' living environments. The underlying idea is to integrate these robots into small or large-scale smart spaces (e.g., smart homes, buildings, urban spaces) to offer accessible and transparent services anytime and anywhere, such as cognitive assistance, security, comfort, and entertainment. This robotic autonomy relies on the potential of artificial intelligence, particularly machine learning and neural networks, or simply using Reinforcement Learning to enhance their interactions with humans by learning from experience and imitating the cognitive abilities of human beings. Combining the Internet of Things with robots enables the creation of robots

capable of collecting information using cameras and LiDAR sensors, analysing this data, and performing reasoning to understand their environment and act accordingly. Combining AI, natural language processing, deep learning, and machine learning algorithms, along with knowledge representation, will certainly revolutionise the world of robotics.

Be AWARE

Choosing a tool with artificial intelligence is at the user's own risk. Therefore, we should all be aware of what we want from this AI and all these technological advances. I am convinced that the costs of implementing AI are a secondary issue compared to the dangers of irresponsible AI that threaten every country. Consequently, challenges follow us like our shadows. The adoption of AI by our country, Algeria, should prompt our leaders, researchers, and students to ask themselves the right question: Are we going to utilise market tools offered by third parties, or should we develop our own tools and algorithms? The answer, in my opinion, is clear: any tool you haven't created yourself is certainly a vehicle for intelligence that will control you and harm your nation. Some systems include hidden mechanisms that collect data without the user's consent or awareness, and WhatsApp and Facebook, which are part of our daily lives, constantly transmit our data.

2.12 M2M (Machine-to-Machine) & IoT

Machine-to-Machine (M2M) is an essential component of modern ICT, enabling machines to interact directly, often in industrial, medical, or urban contexts, while raising questions about the role of humans and the autonomy of systems. M2M refers therefore to direct communication between electronic devices without human intervention. In the field of Information and Communication Technologies (ICT), this paradigm plays a central role in the transition to autonomous, responsive, and intelligent systems.M2M (Machine-to-Machine) refers to an automatic communication method between two or more electronic devices without direct human intervention. These machines exchange data, trigger actions, or coordinate processes via wired or wireless networks (e.g., Wi-Fi, 4G/5G, LoRa, Zigbee, etc.).

2.12.1 Key points to remember:

1. Association: This involves the automatic pairing or interconnection of machines (sensors, actuators, PLCs, etc.) to exchange data or coordinate actions.

2. Without human intervention: M2M relies on functional autonomy, with devices making decisions or reacting to events in real time, without the need for a constant human interface.

3. Machine to serve humans: Even if M2M operates without direct human intervention, its ultimate goal often remains the improvement of human service (optimisation, security, comfort, time savings, etc.). However, this question raises an ethical issue: to what extent can or should we automate interactions between machines, and what is the role of humans in this loop?

4. Machine with intelligence: M2M is evolving towards systems integrating local analysis and decision-making capabilities, sometimes coupled with artificial intelligence (AI) techniques. This brings M2M closer to the Internet of Things (IoT) and cyber-physical systems (CPS).

Conventional M2M[7] protocols have been gradually replaced by more advanced technologies, such as LPWA (Low-Power Wide Area) and NB-IoT, designed to connect objects over long distances with low power consumption efficiently. The 5G technology is gaining ground in sectors that require extensive coverage, ultra-low latency, high reliability, and immediate availability, surpassing the capabilities of 3G and 4G. Modern IoT is moving toward integrated global solutions, rather than single-point systems, leveraging AI and machine learning to generate intelligent analytics for businesses.

In this context, the M2MSAT (Lightweight Application and Transport Protocols for Future M2M Applications) project, funded by ESA (European Space Agency) under the ARTES program, aimed to optimise IoT application protocols for M2M/IoT satellite networks. The consortium included SES TechCom, the University of Luxembourg (SnT), JOANNEUM RESEARCH, and LIST. The Luxembourg Institute of Science and Technology, LIST (Luxembourg) joined the consortium as well, as a Uni.lu subcontractor, providing support for the standardization-related activities.

1. The project focused on two widely used protocols:
2. MQTT (TCP-based, relia b le but more complex) and
3. CoAP (UDP-based, lightweight but with optional reliability via "confirmable" messages).

[7] https://portail-qualite.public.lu/dam-assets/publications/normalisation/2020/national-technical-standardization-report-iot-june-2020.pdf

4. The consortium proposed optimized configurations of these protocols for their efficient use in geostationary satellite (GEO) networks, thus improving the performance of satellite-based IoT.

2.12.2 Real examples of M2M in healthcare (non-IoT):

1. Interconnected biomedical equipment: In an operating room, anesthesia machines, ventilators, and vital signs monitors communicate with each other locally to adjust parameters. If oxygen saturation drops, the ventilator can automatically recalibrate based on the monitor reading;
2. Automated alert systems play a crucial role in patient rooms, ensuring immediate response to any emergency. These systems are designed to detect and transmit critical information, such as a fall or lack of movement, directly to the nurse call, bypassing the need for an IP network or cloud. Automated alert systems in patient rooms are designed for efficiency. For instance, a sensor detecting a fall or lack of movement can instantly transmit the alert directly to the nurse call, bypassing the need for an IP network or cloud, and ensuring a swift response to the patient's needs. This is a closed chain, often wired or using a proprietary short-range protocol (e.g., Zigbee, KNX);
3. Autonomous medical carts in a hospital: The carts move independently to deliver equipment. They communicate with each other (e.g., over a proprietary local network) to avoid collisions or traffic jams, without cloud access. For instance, 'Cart A reports its position to Cart B, allowing the latter to wait for its passage.' This communication process, which occurs over a proprietary local network, enables the carts to coordinate their movements and avoid collisions or traffic jams, all without the need for cloud access;

2.12.3 What is the difference between IoT and M2M?

The IoT (Internet of Things) and M2M (Machine-to-Machine) are two concepts related to the connectivity of electronic devices, but they have significant differences:

- M2M is a direct communication technology that enables automated data transmission between machines or electronic devices, eliminating the need for human intervention. M2M devices are generally designed to operate in specific environments and have minimal functionality;

- The IoT is an evolution of M2M, which integrates internet connectivity to enable objects to communicate with each other and with computer systems. IoT-connected objects are equipped with sensors, software, and wireless communication technologies to collect and exchange data. They can be M2M devices or more advanced devices with additional features.

2.13 SCADA vs. IIoT in Industrial Monitoring

Since the 1960s, SCADA (Supervisory Control and Data Acquisition) systems have been pillars of industrial monitoring. They enable the monitoring8, control, and automation of industrial processes in real time via a centralised architecture and closed networks using proprietary industrial protocols (Modbus, Profibus, OPC, etc.). Their robustness and longevity (15–20 years) make them reliable but inflexible solutions, costly to expand, and challenging to integrate with other modern technologies.

Beginning in the 2010s, the Industrial Internet of Things (IIoT) introduced a more open and connected approach to monitoring, relying on intelligent sensors, the cloud, and modern protocols such as MQTT or CoAP. Unlike SCADA systems, the IIoT is based on a decentralised architecture and enables massive data collection, advanced automation, predictive maintenance, and intelligent decision-making through artificial intelligence and machine learning.

Two data processing models are used in the IIoT:

- Cloud Computing: for large-scale analysis, centralised management, remote access, and process optimisation via AI.
- Edge Computing: to reduce latency, increase reliability, and enable rapid decision-making close to the equipment.
- A hybrid architecture combining cloud and edge computing is often preferred in demanding industrial environments, allowing for a combination of analytical power and local responsiveness.

However, despite its modernity, the IIoT has limitations:

[8] https://motilde.com/en/industrial-supervision-role-scada/

- Heavy reliance on connectivity (risk of latency or network outages),
- Increased cybersecurity risks due to open networks,
- Lack of certification for critical processes in specific sensitive sectors.

Conversely, SCADA systems, although old, remain essential for critical operations. They guarantee real-time control, an efficient human-machine interface, advanced alarm management, built-in redundancy, and enhanced cybersecurity thanks to isolated networks.

No technology completely replaces the other. SCADA and IIoT are not in competition, but rather complementary. Each has its own strengths and areas of application, and by understanding and leveraging these, you can create a more robust and efficient industrial monitoring system. SCADA remains preferred for critical, real-time operations. However, IIoT excels in data collection, predictive analysis, interoperability, and multi-site management. Thus, a hybrid industrial infrastructure, combining SCADA, IIoT, Edge, and Cloud Computing, is the optimal strategy to benefit from both the reliability of traditional systems and the flexibility of modern technologies.

3 chapter 2: Communication Networks for IoT

Communication networks play a fundamental role in the Internet of Things (IoT). They ensure the transmission of data between connected objects, processing platforms, and users. Depending on needs (range, throughput, power consumption, reliability), different types of networks are used, ranging from short-range technologies like Bluetooth to long-distance networks like LoRaWAN, 5G, or traditional cellular networks. The choice of network is crucial to ensure the performance, security, and scalability of an IoT system.

We will not go into detail about network layers here, as they are covered more thoroughly in other modules of the master's program. Instead, we will focus on the most important components relevant to this course.

According to the IBM[9], the IoT paradigm the Internet of Things (IoT) offers numerous benefits but also poses significant risks. The most notable are:

Security and Privacy: IoT devices are often vulnerable to cyberattacks. They collect large amounts of sensitive data, raising data protection and privacy concerns.

Interoperability: However, the lack of common standards between manufacturers makes communication between IoT devices difficult, resulting in data silos that are difficult to integrate and analyze. This underscores the need for collaboration and standardization in the industry.

Data Overload: The enormous volume of data generated can exceed the processing capabilities of unprepared companies, making analysis complex. This highlights the urgent need for appropriate data management and analysis tools.

Cost and Complexity: Deploying an IoT system requires significant investments. These include costs for hardware such as sensors and gateways, software for data processing and analysis, and infrastructure for data storage and communication. Additionally, ongoing costs for technical expertise for management and maintenance are also significant. Understanding these costs is crucial for businesses considering IoT deployment.

[9] https://www.ibm.com/fr-fr/topics/internet-of-things

Regulatory and Legal Challenges: Companies must navigate a complex and variable regulatory framework (data protection, cybersecurity), which often differs from country to country.

Furthermore, IBM advises businesses to follow the following guidelines for approaching IoT :

Managing IoT devices can be a complex and challenging task; however, there are several best practices that businesses can follow to ensure their IoT devices are secure, reliable, and optimised for optimal performance. Here are some tips for managing IoT devices:

Plan your IoT strategy: Before deploying IoT devices, businesses must have a clear understanding of their goals, use cases, and desired outcomes. This can help them select the right devices, IoT platforms, and technologies, ensuring their IoT strategy aligns with their business objectives.

Choose secure IoT products: Security is a critical issue when it comes to IoT solutions, as they can be vulnerable to cyberattacks. Businesses must select security-first devices and implement effective security systems, including encryption, authentication, and access control. Device monitoring and maintenance: IoT devices must be regularly monitored and maintained to ensure they are operating optimally and are not vulnerable to security threats. This may involve monitoring device health and performance, updating firmware and software, conducting regular security audits, and implementing predictive maintenance.

Effective data management: IoT devices generate vast amounts of real-world data, which can be challenging to manage and analyse. Companies must have a clear data management strategy, including storage, analysis, and visualisation, to extract meaningful insights from the data generated by their IoT devices.

Ecosystem building: IoT devices are often part of a larger ecosystem that includes other devices, platforms, and technologies. Companies must have a thorough understanding of this ecosystem and ensure that their IoT devices can integrate seamlessly with other systems and technologies.

3.1 Protocoles IP et non-IP

Most modern networks can be divided into two broad categories. On the one hand, there are IP-based networks, which use IP addresses for communication. This includes the most current

infrastructure, such as Ethernet, Wi-Fi, cable, DSL, and VPN connections, as well as modems. These networks form the basis of the Internet and modern business networks.

On the other hand, there are non-IP networks, which are often used in specialised or industrial environments. These networks do not use IP for communication and rely on proprietary or purpose-specific protocols.

Some devices can be fixed or mobile. Some mobile devices do not transfer data while in motion, while others do. Device mobility means that the IP address can change and cannot serve as a device identifier. A consequence of mobility can be a time lag in communication. Mobility can be easily achieved by utilising wide-area network services, such as LTE, Sigfox, and LoRa, which are offered on a pay-per-use basis. Additionally, some devices enter sleep mode to conserve energy or to maintain their expected functionality, such as the need for sensor data on a regular schedule. A proxy or broker is introduced in such cases to provide data when IoT devices are unavailable.

We can create a device-level IoT platform, i.e., middleware to be installed on a group of IoT devices, that aids in the rapid development of new systems. A selected messaging protocol or set of protocols is part of the IoT platform at the device level. Such middleware must be optimised for the specific communication needs of a given type of IoT device. Four types of such devices can be distinguished, namely:

1. Constrained devices.

2. Unconstrained devices connected to the internet with significant limits on the volume of data transferred.

3. Unconstrained devices are not always online.

4. Unconstrained online devices accessible at any time by applications or services.

An IoT system typically includes numerous processes and services running on cloud servers. Communication requirements may vary depending on the functionalities implemented. A few basic types of communication purposes can be distinguished:

1. Device discovery and configuration

2. Data acquisition – the IoT device sends data repeatedly.

3. Data polling: The IoT device only sends data upon request.

4. Notification and alarm analysis – The IoT device rarely sends prioritised data.

5. Action dispatching: The IoT device receives actions. 6. Process control—messages are sent in response to detected data, or some feedback is expected after control commands.

7. Opportunistic peer-to-peer data exchange—IoT devices directly share certain information or services.

ARPAnet was the first attempt by the U.S. Department of DefenceDefence (DoD) to design a decentralized network that was more resistant to attack, while still allowing for complete interconnection of different systems. ARPAnet was created in the 1970s, but it was in 1983 that an entirely new protocol stack, TCP/IP, was introduced. The first widely used network protocol version was IPv4 (Internet Protocol version 4), which paved the way for the development of the civilian Internet. Initially, only research centres and universities were connected, supported by the NSF (National Science Foundation), and commercial applications were not allowed. However, when the network began to grow exponentially, the NSF decided to transfer its operation and funding to private operators, lifting restrictions on commercial companies. To accommodate more addresses, you require a longer IP address space (i.e., more bits to specify the address), which necessitates a new architecture, resulting in changes to most routing and networking software. After reviewing several proposals, the IETF settled on IPv6, described in the January 1995 RFC (Request for Comment, the official name for IETF documentation) 1752, sometimes also called Next Generation Internet Protocol or IPng. The IETF updated the IPv6 standard in 1998, with the current definition covered by RFC 2460. By 2004, IPv6 was widely available in the industry and supported by most new networking equipment. Today, IPv6 coexists with IPv4 on the Internet, and the amount of IPv6 traffic is growing rapidly as more ISPs and content providers begin to support IPv6.

3.2 IPv6: Concepts

IPv6 is located at Layer 3, called the network layer. The data handled by Layer 3 is called packets. Devices connected to the Internet can be either hosts or routers. A host can be a PC, a laptop, or a sensor card, sending and/or receiving data packets. Hosts are either the source or the destination of the packets. Routers are responsible for forwarding packets and determining the next router to forward them to their final destination. The Internet is composed of a large number

of interconnected routers, which receive data packets at one interface and then forward them as quickly as possible through another interface to another router for further forwarding.

Additionally, IPv6 features include global addressing and automatic host address configuration. Since IPv6 provides as many addresses as we could need for hundreds of years, we can put a global unicast IPv6 address on almost anything we can think of. This reverts to the original Internet paradigm, where every IP device can communicate with every other IP device. This end-to-end communication enables two-way communication anywhere on the Internet and between any IP device, potentially leading to collaborative applications and new ways of storing, sending, and accessing information.

3.3 Transmission Control Protocol (TCP)

The Transmission Control Protocol (TCP) is a core protocol of the Internet Protocol (IP). TCP is a reliable stream delivery service that guarantees that all received bytes will be in the correct order. It uses a technique known as positive acknowledgement with retransmission to ensure reliable packet transfer. TCP manages received fragments and reorders the data. Applications that do not require a reliable stream service can use the User Datagram Protocol (UDP), which provides a connectionless datagram service that emphasises low latency over reliability. TCP is commonly used by services such as HTTP, FTP, and email, as well as any connection-oriented service.

The Internet Protocol (henceforth, IP) was conceived as a solution to the need for interconnecting different data networks and has become the de facto standard for all kinds of digital communications. Today, IP is present in most devices capable of sending and receiving digital information, not just on the Internet. IP is standardised by the IETF (Internet Engineering Task Force), the body responsible for all Internet standards, ensuring interoperability between software from different vendors. All standard operating systems and network libraries support IP for sending and receiving data. As part of this "everything-connected-to-the-Internet" framework,

If a given protocol uses connection-oriented transport, such as TCP, the party that listens is referred to as a server, and the party that initiates the connection is called a client. This is due to the TCP convention, regardless of the role played by the application process. As a result, the terminology commonly used for communication paradigms is somewhat inconsistent. The term "client-server" refers to a model with listening and connecting parties, and can also suggest an architectural model with multiple client nodes requesting specific services from a server node.

More complex communication models are applied to large systems with nodes that may appear dynamically, be available, or not.

With a message broker, the broker interconnects multiple publishers and subscribers. It can temporarily store messages and control access rights. A broker's implementation can be centralised or distributed. With an RPC reseller, the reseller interconnects multiple RPC callers and callees. It can control access rights and manage boundaries between end parties. A dealer's implementation can be centralised or distributed.

With a message router, the router selects the destination or finds a path to the destination if multiple routers are part of the system. The router can process annotations attached to the transferred message. This paradigm allows for high scalability of the system. With a message server, the server can perform some of the message processing, for example, aggregating them. Additionally, the server can serve as a message broker or router.

3.4 Bluetooth technology vs. Wi-Fi

Wi-Fi operates in the 2.4 GHz, 5 GHz, and 6 GHz frequency ranges, providing a wide spectrum for data transfer. On the other hand, Bluetooth operates only in the 2.4 GHz band, offering a more focused range for its applications.

Wi-Fi technologies and supporting infrastructure are designed to efficiently manage data transfer between large numbers of devices, making it ideal for network connections. In contrast, Bluetooth focuses on host-to-device data transfers over shorter distances, simplifying the data management process. Unlike Bluetooth, which does not require supporting infrastructure such as routers or access points to function, Wi-Fi devices operate on a fixed channel and offer a simple setup process. Connecting to a Wi-Fi network for the first time does not require a dedicated pairing mode; users can simply enter the necessary network credentials and establish a connection, making it an easy-to-use technology.

Wireless technologies used in IoT, Figure 11**:**

• Wi-Fi: High-speed, medium-range, used in home or business environments. Good compatibility but high power consumption.

• Bluetooth: Short-range, low-power, ideal for personal connected devices (healthcare, audio, wearables).

• ZigBee: Low-power, medium-range, used in home automation and low-speed sensors. Works in mesh networks.

• LoRa (Long Range): Very low power, long range (up to several kilometres), reduced data rate. Ideal for sensors in agriculture or smart cities.

• NB-IoT (NarrowBand-IoT): Low-power, long-range cellular technology, suitable for fixed sensors sending small amounts of data.

• 5G: Very high-speed, low-latency, enables massive and critical communications (Industry 4.0, autonomous vehicles, telemedicine).

Technologyc	Range	Data Rate	Power Consumption	Topology	Typical Use Cases
Wi-Fi	~50–100 m	Up to 1 Gbps	High	Star	Smart homes, video, industrial IoT
Bluetooth 2.0 + EDR	~10 m	Up to 3 Mbps	Medium	Piconet	Audio, mobile data transfer
Bluetooth 5.0	~40–240 m (LE Long Range)	2 Mbps (or 125–500 kbps LE)	Very low	Star/Mesh (LE Mesh)	Wearables, sensors, smart homes
ZigBee	~10–100 m	20–250 kbps	Low	Mesh	Home automation, lighting, industrial
LoRa	Up to 15–20 km	Up to 50 kbps	Very low	Star (LoRaWAN)	Agriculture, asset tracking, smart cities
Sigfox	Up to 50 km (rural)	Up to 100 bps	Very low	Star (proprietary)	Remote sensors, metering, low data needs
NB-IoT	Up to 10 km	Up to 250 kbps	Low	Cellular	Smart meters, building monitoring
5G	100 m to several km	Up to 10 Gbps	Medium to high	Cellular	Autonomous vehicles, remote surgery, AR/VR

Figure 11 Key communication protocols

Strengths of Bluetooth 5.0:

- 4x the range of the previous version.
- 2x the data rate (in high-speed mode).
- Introduces Bluetooth Low Energy (BLE) Mesh for better network coverage.

- Widely used in home and medical IoT.

Key points about LTE Cat M1:

- Part of 4G LTE but optimized for low-power IoT.
- Supports voice (VoLTE) and international roaming.
- Lower latency than NB-IoT.
- Ideal for mobile applications (moving objects, e.g., connected vehicles).

Advantages of LTE Cat 1:

- High speed (up to 10 Mbps), sufficient for light video, payment terminals, surveillance cameras, etc.
- Compatible with existing 4G networks, so no need for dedicated infrastructure.
- Supports full mobility, like a mobile phone.
- Supports voice (VoLTE).
- Consumes more power than Cat M1 or NB-IoT, making it less suitable for ultra low-power devices.

3.5 LoRaWAN vs Sigfox

LoRaWAN[10] is a Low Power Wide Area (LPWA) network specification designed to wirelessly connect battery-powered devices to the Internet over large-scale networks (from regional to global). It meets key IoT requirements, including bi-directional communication, end-to-end security, mobility, and location services. Operating in unlicensed frequency bands, LoRaWAN offers long-range coverage (up to 15 km in rural areas), low power consumption, and the ability to connect a large number of devices to a single network. Its architecture includes end devices, gateways, network servers, and application servers, making it highly scalable and flexible for a wide range of IoT applications.

[10] https://lora-alliance.org/

LoRa and LoRaWAN are related but serve different functions within wireless communication networks. Here are the key differences:

- **LoRa** (Long Range): Refers to the physical (PHY) layer, which includes the modulation technique used for long-range, low-power communication. LoRa is the radio signal that transmits the data over the air using spread spectrum technology.
- **LoRaWAN** (Long Range Wide Area Network): Refers to the network standard and system architecture for managing communication between LoRaWAN devices and the network server. It defines the communication standard and system architecture at the MAC (Medium Access Control) layer.

Function:

- **LoRa**: Ensures long-distance communication and robust data transmission using Chirp Spread Spectrum (CSS) modulation. It is responsible for the low-level radio transmission.
- **LoRaWAN**: Manages device communication, data rates, and battery life, overseeing how data is sent and received, including security, data integrity, and the overall network structure.

2. **Scope**:
   - **LoRa**: Deals with the physical transmission of data over the air. It is concerned with the signal characteristics and the method of transmitting data.
   - **LoRaWAN**: Encompasses the entire network stack, including network management, communication protocols, and application interfaces, ensuring data is properly routed and devices are managed efficiently.
3. **Components**:
   - **LoRa**: Focuses on LoRa chipsets and their ability to transmit data to the cloud over long distances using minimal power.
   - **LoRaWAN**: Includes end-devices, gateways, network servers, and application servers, coordinating how data is relayed, managed, and utilized across the network.

4. **Security**:

    - **LoRa**: Does not include security mechanisms; it is purely a physical layer technology.

    - **LoRaWAN**: Implements security measures like encryption and device authentication to protect data integrity and privacy.

Sigfox[11]0G technology is an Low-Power Wide-Area (LPWA) networking protocol owned by UnaBiz[12]. It is designed to connect sensors and devices securely at low-cost in the most energy efficient way to enable Massive IoT. Since its inception in 2010, the 0G technology has been adopted by 70+ national 0G IoT Solution Providers globally. They own and commercialise the global 0G network in the respective countries they operate in.

3.5.1 LoRaWAN vs Sigfox: Which Technology Should You Choose for Your IoT Project?

The performance and reliability of your IoT network largely depend on the technology you choose. You may be wondering: "Sigfox or LoRaWAN – which one is right for me?" These two major players each offer unique features tailored to different needs in terms of architecture, range, and cost. This comparison will help you identify the most suitable solution for your project.

Both Sigfox and LoRaWAN offer notable advantages for connecting your devices to the digital world, , Table 2:

- **Sigfox** stands out for its simplicity and extended coverage through a global network ;
- **LoRaWAN** impresses with its flexibility, bidirectional communication, and adaptability to custom deployments involving a large number of devices.

[11] https://sigfox.com/

[12] UnaBiz is a global Massive IoT service provider & integrator that specialises in sensor product design, manufacturing, and data platform services across a hybrid of low-power wide area (LPWA) technologies such as Sigfox, LTE-M, NB-IoT and LoRaWAN, to power business growth. The company focuses on Utilities, Supply Chain & Logistics, Security and Facilities & Building Management

Table 2 LoRaWAN vs SigFox, https://iotindustriel.com/autres/normes-protocoles/lorawan-vs-sigfox-quelles-differences/

Feature	Sigfox	LoRaWAN
Network Structure	Proprietary	Open and standardized
Topology	Star (device to base station)	Star or mesh (device to gateway, then to the network)
Frequency Band	Specific, dedicated to Sigfox	ISM bands (868 MHz in Europe)
Communication	Mostly unidirectional (device → cloud)	Bidirectional, suitable for complex data exchanges
Data Rate	100 bps (EU), 600 bps (US)	More flexible – up to 50 kbps
Coverage	Global single network	Depends on deployment (public or private)
Power Consumption	Very low	Low
Flexibility	Limited	Highly configurable
Cost	Generally more affordable	Variable depending on infrastructure
Range	3–10 km (urban), up to 50 km (rural)	2–5 km (urban), up to 15 km (rural)

The choice of protocols for an IoT project depends on several key criteria related to the specific requirements of the intended application. The necessary network range must be considered: some technologies, such as LoRaWAN or Sigfox, are suitable for long distances, while Wi-Fi or Bluetooth are better suited to confined environments. However, energy consumption is a crucial criterion, especially for battery-powered devices. Protocols like NB-IoT or Bluetooth Low Energy (BLE) are specifically designed to minimize this consumption, highlighting the significance of this factor. The required data rate also plays a crucial role: video surveillance applications, for example, require high-speed protocols such as LTE Cat 1 or Wi-Fi. Other criteria must also be considered:

3.5.1.1 1. Communication Range

The choice of protocol depends primarily on the distance between the connected objects and the data collection point. For local applications (home automation, wearables), technologies like Bluetooth or ZigBee are sufficient. However, agricultural, industrial, or urban applications require a more extended range, which only LPWAN (Low-Power Wide-Area Network) technologies like LoRaWAN, Sigfox, or NB-IoT can offer, ranging from several kilometres to dozens of kilometres in rural areas. In extreme contexts (remote areas, maritime areas), satellite connectivity becomes necessary.

3.5.1.2 2. Energy Consumption

Most IoT sensors are designed to operate on batteries for several years. It is therefore crucial to choose an energy-optimised protocol. Protocols such as Sigfox, LoRaWAN, or NB-IoT are designed for sporadic and short-term communications, thereby minimising power consumption. Conversely, protocols like Wi-Fi or LTE-M offer better performance but require more power, making them more suitable for continuously powered devices (video surveillance, industrial equipment).

3.5.1.3 3. Required Data Rate

The volume of data exchanged strongly influences the protocol to adopt. For low-frequency or event-driven applications (temperature sensors, tank level), a few bytes are sufficient, and low-bandwidth protocols like Sigfox or LoRaWAN are ideal. However, applications for video surveillance, advanced maintenance, or remote control require high-speed protocols like Wi-Fi, LTE-M, or 5G.

3.5.1.4 4. Unidirectionality vs. Bidirectionality

Some protocols, like Sigfox, operate primarily in unidirectional mode (sending data from the sensor to the cloud), which limits response or control options. Others, such as LoRaWAN, MQTT, or NB-IoT, support bidirectional communication, enabling more complex applications (remote activation, OTA updates, conditional responses).

3.5.1.5 5. Latency Sensitivity

The required response time is a key criterion. For critical or real-time applications (robot control, security alarms), low latency is essential. Technologies such as 5G, Wi-Fi, or LTE-M are preferred. Conversely, applications that tolerate delays (environmental surveys or metering) can use LoRaWAN or Sigfox, which offer greater battery life but increased latency.

3.5.1.6 6. Mobility of Connected Objects

When objects are mobile (such as vehicles, luggage, or people), the chosen protocol must allow for seamless handover between antennas or stations. This is particularly true for LTE-M, NB-IoT, and 5G, which enable continuous monitoring. Protocols such as LoRaWAN or Sigfox are more suited to semi-mobile or static devices.

3.5.1.7 7. Infrastructure and Subscription Costs

Some technologies require little infrastructure (e.g., Sigfox via an operator network), while others require private deployment (e.g., LoRaWAN gateway). Costs also vary depending on the business model: monthly subscriptions (NB-IoT, LTE-M), unit cost (Sigfox), or initial investment (private LoRaWAN). It is crucial to weigh the cost against the project's lifespan.

3.5.1.8 8. Security and Confidentiality

Sensitive applications (healthcare, personal data) require a protocol with strong authentication, encryption, and data integrity. Modern protocols such as MQTT with TLS, LoRaWAN with AES-128, NB-IoT, or HTTPS offer a high level of security. Physical protection of devices is also an additional issue.

3.5.1.9 9. Interoperability and Open Standards

To avoid vendor lock-in, it is advisable to favour standardised or widely supported industry protocols, such as LoRaWAN, MQTT, CoAP, and ZigBee. This facilitates integration into cloud platforms, project scalability, and inter-vendor compatibility.

3.5.1.10 10. Scalability and Deployment Density

Large-scale projects (smart cities, connected agriculture) require protocols capable of managing a large number of nodes simultaneously, with anti-collision mechanisms, frame management, etc. LPWAN networks, although slow, are very effective for this purpose. NB-IoT and LoRaWAN are often the best choices.

3.6 Classification of IoT sensors and their application areas

Sensors are essential devices in the IoT, enabling the measurement and conversion of physical or chemical quantities (temperature, pressure, movement, brightness, gases, etc.) into

digitally usable signals. Several types of sensors are distinguished depending on the nature of the data collected:

1. Environmental sensors (temperature, humidity, pressure) used in precision agriculture or smart buildings;
2. Motion sensors (accelerometer, gyroscope) are found in wearable connected devices and drones;
3. Biometric sensors (heart rate, fingerprints) integrated into medical and security devices;
4. Optical and light sensors (cameras, LDRs) for video surveillance or smart lighting;
5. Position sensors (GPS) for vehicle or fleet tracking;

Current or vibration sensors for predictive maintenance in industry. A diversified category encompasses more specific sensors, such as RFID/NFC sensors (contactless identification), sound sensors (acoustic monitoring), or level sensors (for tanks and silos), thereby expanding the fields of application in various sectors, including logistics, healthcare, transportation, and smart cities. The choice of sensor depends on specific needs in terms of accuracy, acquisition frequency, autonomy and environmental conditions, Figure 12 :

3.6.1 1. Environmental Sensors

- **Temperature sensors** (e.g., LM35, DHT11): measure ambient temperature.
- **Humidity sensors** (e.g., DHT22): detect moisture in the air.
- **Air quality sensors** (e.g., MQ135): monitor pollutants like CO2, VOCs.
- **Gas sensors** (e.g., MQ2, MQ9): detect combustible gases, smoke.
- **Light sensors** (e.g., LDR, photodiode): measure light intensity.
- **Sound sensors** (e.g., microphones): detect noise levels.

3.6.2 2. Motion & Position Sensors

- **Accelerometers** (e.g., ADXL345): detect acceleration or movement.
- **Gyroscopes** (e.g., MPU6050): measure orientation and angular velocity.
- **Magnetometers** (e.g., HMC5883L): act like digital compasses.
- **PIR sensors** (Passive Infrared): detect human movement.
- **Ultrasonic sensors** (e.g., HC-SR04): measure distance via sound waves.
- **Proximity sensors** (e.g., IR sensors): detect nearby objects.

3.6.3 3. Mechanical & Physical Sensors

- **Pressure sensors** (e.g., BMP180): measure atmospheric pressure.
- **Force sensors** (e.g., FSR): respond to physical pressure or force.
- **Vibration sensors**: detect oscillations or movements in structures.

3.6.4 4. Health & Biometric Sensors

- **Heart rate sensors** (e.g., MAX30100): monitor pulse.
- **Temperature sensors (medical)**: track body temperature.
- **ECG sensors**: record heart's electrical activity.
- **SpO2 sensors**: measure oxygen saturation in blood.
- **EEG/EMG sensors**: read brain/muscle activity.

3.6.5 5. Optical Sensors

- **Camera modules**: capture visual data.
- **Barcode/QR scanners**: read encoded information.
- **IR sensors**: used for presence detection and remote control.

3.6.6 6. Industrial Sensors

- **Flow sensors**: measure the rate of fluid/gas flow.
- **Level sensors**: detect levels of liquids/solids in containers.
- **pH sensors**: measure acidity or alkalinity of solutions.
- **Turbidity sensors**: monitor water quality.

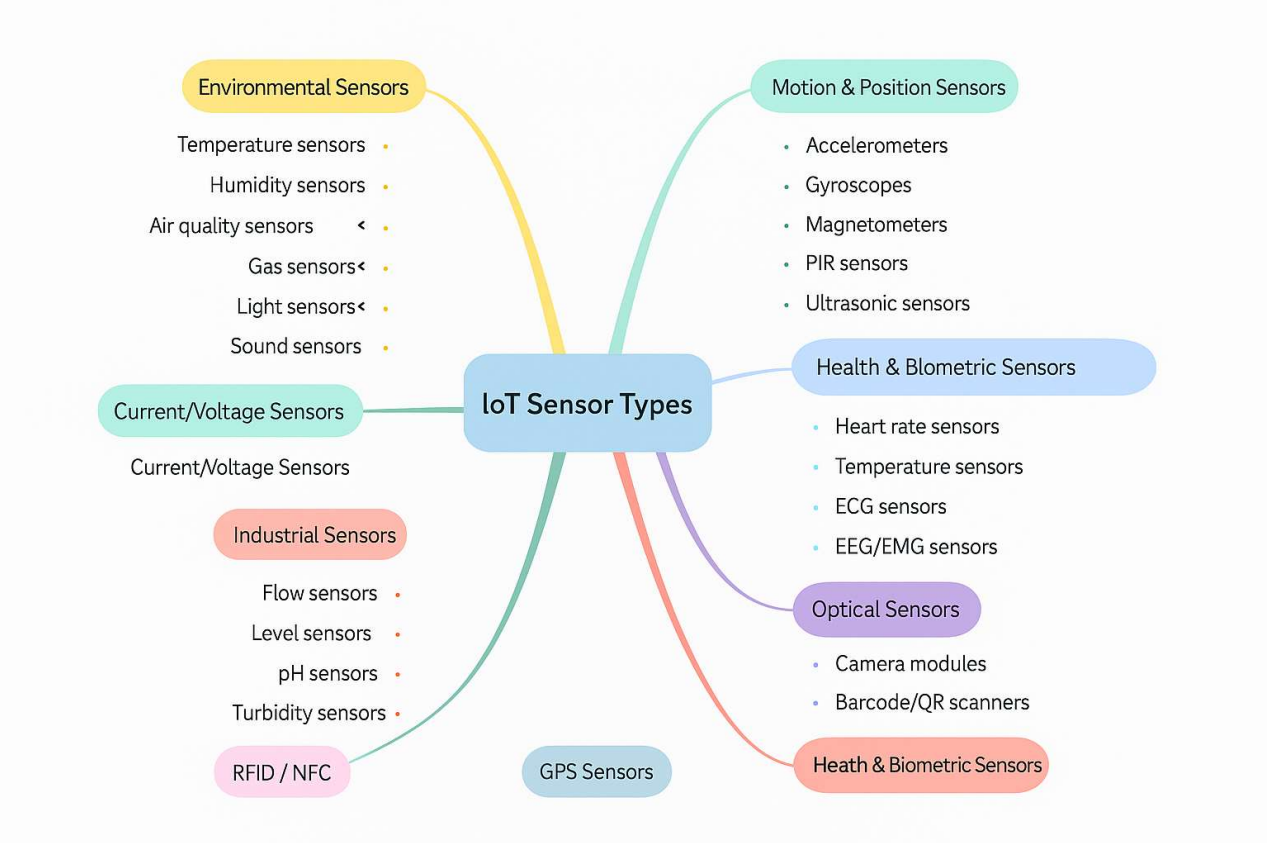


Figure 12 Overview of IoT Sensor Categories

Central Node: IoT Sensor Types

Temperature Sensors

- Function: Measure ambient or object heat
- Examples: Smart thermostat, vaccine storage, HVAC systems

Humidity Sensors

- Function: Detect air or material humidity
- Examples: Precision agriculture, smart greenhouses, air conditioning

Pressure Sensors

- Function: Measure gas or liquid pressure
- Examples: Pipeline monitoring, smart tires, weather stations

Motion Sensors

- Subtypes:
    - Accelerometer (detects acceleration/movement)

    - Gyroscope (measures orientation and rotation)
    - PIR (Passive Infrared) (detects presence/motion)
- Examples: Fitness bands, drones, alarm systems

Light Sensors (LDR)

- Function: Measure light intensity
- Examples: Smart lighting, energy-efficient buildings

Gas Sensors

- Function: Detect specific gases (CO_2, methane, etc.)
- Examples: Air quality monitoring, gas leak detection, mining safety

Proximity Sensors

- Function: Detect nearby objects without contact
- Examples: Smart parking, hands-free devices

Level Sensors

- Function: Measure liquid or solid levels in containers
- Examples: Water tank management, grain silos, fuel monitoring

Sound Sensors

- Function: Detect noise or sound levels
- Examples: Industrial monitoring, voice assistants

Current/Voltage Sensors

- Function: Measure electrical parameters (current, voltage, power)
- Examples: Smart meters, solar energy systems

Biometric Sensors

- Function: Measure biological data
- Examples: Heart rate, fingerprints, health wearables

GPS Sensors

- Function: Provide geographic location
- Examples: Fleet tracking, vehicle geolocation, logistics

Camera/Image Sensors

- Function: Capture images or video

- Examples: Surveillance, facial recognition

RFID / NFC

- Function: Object identification via radio signals
- Examples: Logistics tracking, access control, smart tags

3.7 Operating systems for IoT

An OS provides a link to the underlying hardware while facilitating application development by avoiding the need to redevelop the features it offers as standard. Most connected devices, such as computers and smartphones, now require an OS to function. Of course, the simplest devices, particularly many screenless devices, have limited memory and power resources and offer limited functionality. They do not require an OS and draw on simple libraries. At the heart of the OS is the kernel, a crucial component that oversees other programs, manages their access rights, and their interactions with the hardware. This stress on the role of the kernel in managing programs and hardware is vital for developers to understand. It is the foundation upon which applications connect. This kernel can offer real-time functionality if necessary. In the current context, cybersecurity is essential: the operating system must offer standard features such as message encryption, certificate management, system integrity verification (secure boot), and support secure protocols (TLS or DTLS). Operating systems must provide developers with an environment adapted to 'constrained' hardware, with few resources and minimal power, and sometimes blending into their surroundings (such as sensors cast in concrete or integrated into clothing). To be frugal, the operating system must nevertheless be able to support a large number of architectures (x86, MIPS, ARM, etc.) and communication protocols (Bluetooth, 6LoWPAN, IPv6). Objects must ensure, with all the necessary security, communications over the Internet or to the Internet using protocols developed for this purpose—communication protocols such as ZigBee, Thread, Bluetooth, 6LowPAN, SigFox, LoraWAN, and MQTT, Figure 13. Objects often have graphical interfaces and screens. They can support advanced features such as facial recognition or speech recognition. In the Internet of Things, many operating systems rely on a monolithic approach. If one of the drivers becomes corrupted, the entire system can be disabled and even allow data stored on the hard drive to be hacked. To secure them, several approaches are proposed: certification of controllers, software platforms, and secured for end-to-end security.

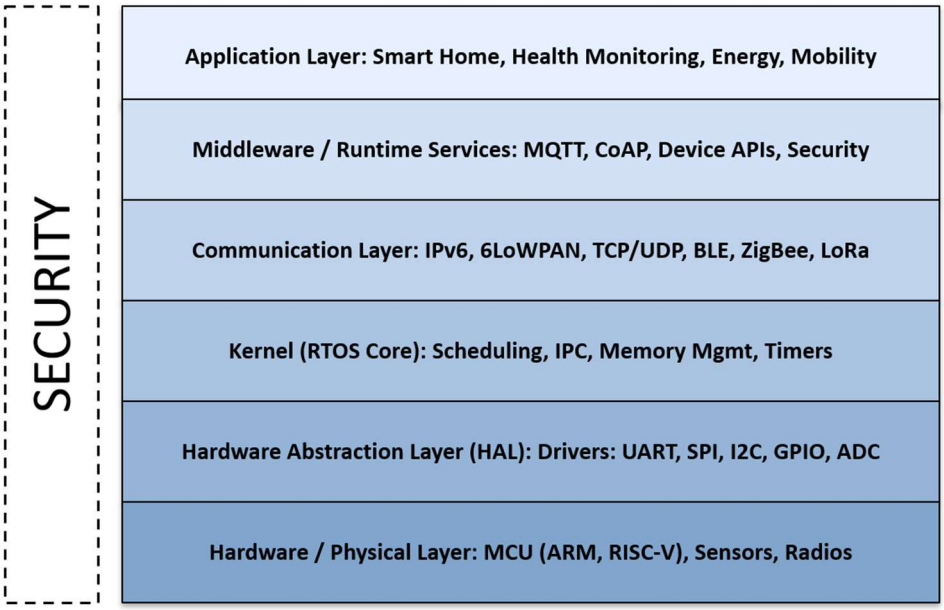


Figure 13 Simplified Architecture of an IoT Operating System.

3.7.1 Different Types of Information Systems

In the very dense niche of small devices, such as motion detectors or environmental sensors, many manufacturers have designed their own customised and relatively rudimentary operating systems. But the emergence of major standards and security requirements suggests that this approach will reach its limits. Examples include open source operating systems such as FreeRTOS, adapted to real-time applications; Contiki for environmental sensors; TinyOS for miniature systems with very little memory; RIOT, which presents itself as the Linux for connected objects based on a microkernel architecture; VxWorks for objects with a visual interface; and Ubuntu and its variant, Ubuntu Core. Furthermore, processor and microcontroller designers such as Intel, ARM, and Texas Instruments are also developing their own solutions, which they are striving to have recognised as standards.

Meanwhile, IT giants are focusing on devices with screens: Google with Android Wear, Microsoft with Windows 10, BlackBerry with QNX, and Samsung with Tizen. Everyone wants to leverage their smartphone expertise to conquer their "cousins": watches, bracelets, wearable technology accessories, cars, and smart TVs. For devices with screens, it would be absurd to start from scratch when highly effective solutions already exist. One of the advantages of open-source OSes is their ability to support a large number of hardware platforms. Another selection criterion is adherence to POSIX, the standard that defines the interfaces common to Unix-like systems.

3.7.2 Some examples of OS

- Adapted RTOS :– FreeRTOS[13] – Lepton[14] (POSIX layer on a kernel) – eCOS[15]
- Dedicated OS :– TinyOS – Contiki[16] – RIOT[17] – Zephyr (Linux Foundation)
- Adapted GPOS – GNU/Linux[18] – Android & Co[19]

IoT operating systems are designed for resource-constrained devices (low CPU, low memory, low power) while ensuring connectivity, security and reliability, Table 3.

Table 3 Commonly Used IoT OS Platforms

OS	Type	Langage	Particularités
RIOT OS	RTOS	C/C++	Open source, multitasking, IPv6, 6LoWPAN, CoAP, MQTT-SN. Compatible with ARM, AVR, MSP430…
Contiki-NG	RTOS léger	C	Very compact, supports 6LoWPAN, RPL, uIP. Cooja simulator. Ideal for sensors.
TinyOS	RTOS événementiel	NesC (variant de C)	Ultra-lightweight, designed for sensor networks, very low level.
FreeRTOS	RTOS	C	very popular, very modular, used with AWS IoT, supports many microcontrollers.
Zephyr	RTOS modulaire	C	Maintained by the Linux Foundation, BLE compatible, 802.15.4, very active.
Mbed OS (ARM)	RTOS	C++	Developed by ARM for Cortex-M, cloud support and TLS security.
LiteOS (Huawei)	RTOS	C	Lightweight, real-time, for industrial connected objects.

We have already covered the IoT platform part in the first chapter. However, more details will be outlined in this part of this course. Here's a clear and structured overview of IoT platforms and tools, with a focus on ThingsBoard, Kaa, and other essential alternatives. It's worth noting that

[13] https://www.freertos.org/

[14] https://lepton-eda.github.io/lepton-manual.html/Supported-operating-systems.html

[15] https://ecos.sourceware.org/

[16] http://contiki-os.blogspot.fr/2014/02/thingsquares-contiki-iot-workshop.html

[17] https://www.riot-os.org/

[18] https://www.gnu.org/gnu/linux-and-gnu.en.html

[19] https://source.android.com/?hl=fr

the growing diversity of Internet of Things (IoT) applications has led to the emergence of numerous platforms supporting device management, data processing, communication protocols, and real-time analytics. Choosing the ideal platform depends on several factors, such as supported protocols, scalability, openness, integration capabilities, and cloud/edge deployment options. The table below provides a comparative overview of some of the most widely used IoT platforms, highlighting their key features and supported communication protocols, Table 4. First, remember that IoT platforms are particularly effective in:

- Managing connected devices centrally and efficiently
- Collecting, processing, and visualizing sensor data seamlessly
- Defining automation rules, generating alerts, and triggering actions
- Integrating effortlessly with external systems such as cloud services, AI modules, and databases

And concerning the core Functionalities of IoT Platforms, it includes:

- Device Provisioning & Lifecycle Management
- Protocol and Network Management (e.g., MQTT, CoAP, HTTP)
- Data Visualization through interactive dashboards
- Real-Time Data Analytics and Processing
- API/REST Integration & Big Data Storage
- Security Features (e.g., authentication, authorization, data encryption)

Table 4 Overview of Leading IoT Platforms: Features, Protocols, and Technologies

Plateform	Language/Stack	Protocols	Features
ThingsBoard https://thingsboard.io/	Java (Spring)	It enables device connectivity via industry standard IoT protocols - MQTT, CoAP and HTTP and supports both cloud and on-premises deployments	Device management, data collection, processing and visualization for your IoT solution
Kaa IoT https://www.kaaiot.com/	Java/Scala	The Kaa platform supports popular lightweight IoT protocols for device connection, such as MQTT and http. In case your device does not have an IP connectivity or already implements some communication capabilities, Kaa employs a gateway architecture where a gateway talks to the device over a local network protocol or a proximity protocol	Device management, data collection, visualisation, microservices, extensible
Mainflux https://mainflux.com/	Go/C	Multi-protocol support, hardware agnostic design, and connectivity across any device or application, using a PUB/SUB multiprotocol messaging bridge (HTTP, MQTT, WebSocket, CoAP) powered by the NATS broker.	is a company offering an end-to-end, open-source IoT platform, edge computing gateway and consulting services covering the software and hardware layers of the internet of things technology
DeviceHiv https://devicehive.com/	Java/C#/Node.js	Connect any device via REST API, WebSockets or MQTT. The DeviceHive team supports libraries written in various programming languages, including Android and iOS libraries which make the platform device-agnostic	Has various deployment options, and that is why it fits ideal for every company either it is a mature enterprise or a small start-up. With the docker compose and kubernetes deployment options, you can go with private, public or hybrid cloud and scale from a single virtual machine, to enterprise grade cluster.
OpenRemote https://openremote.io/	Java	There are several protocol Agents you can use, eg. HTTP REST, a KNX Gateway, a Velbus System, Z-wave, or a Controller 2.5. Agents are a special type of asset which link external services/devices with your OpenRemote system via protocols; agents can be put into the following categories: • Specialised agents (Velbus, Z-Wave, KNX, etc.) • Generic agents (HTTP, TCP, UDP, WS, MQTT, etc.)	Create automated behaviour of the system using rules. Generate alerts and directly control your devices based on live data. Create the rules with easy drag and drop interfaces, or if needed with advanced scripting tools.

3.8 Data Flow in ThingsBoard

ThingsBoard, an open-source IoT platform, is meticulously designed to efficiently manage the flow of data from connected devices to dashboards, external systems, and decision-making tools. Its data flow involves several structured stages, ensuring a seamless and reliable process. ThingsBoard's data flow architecture ensures secure, modular, and scalable processing of IoT data from edge to cloud. The platform's flexible tools for data ingestion, processing, and visualization empower developers, giving them the control to build intelligent IoT solutions, Figure 14.

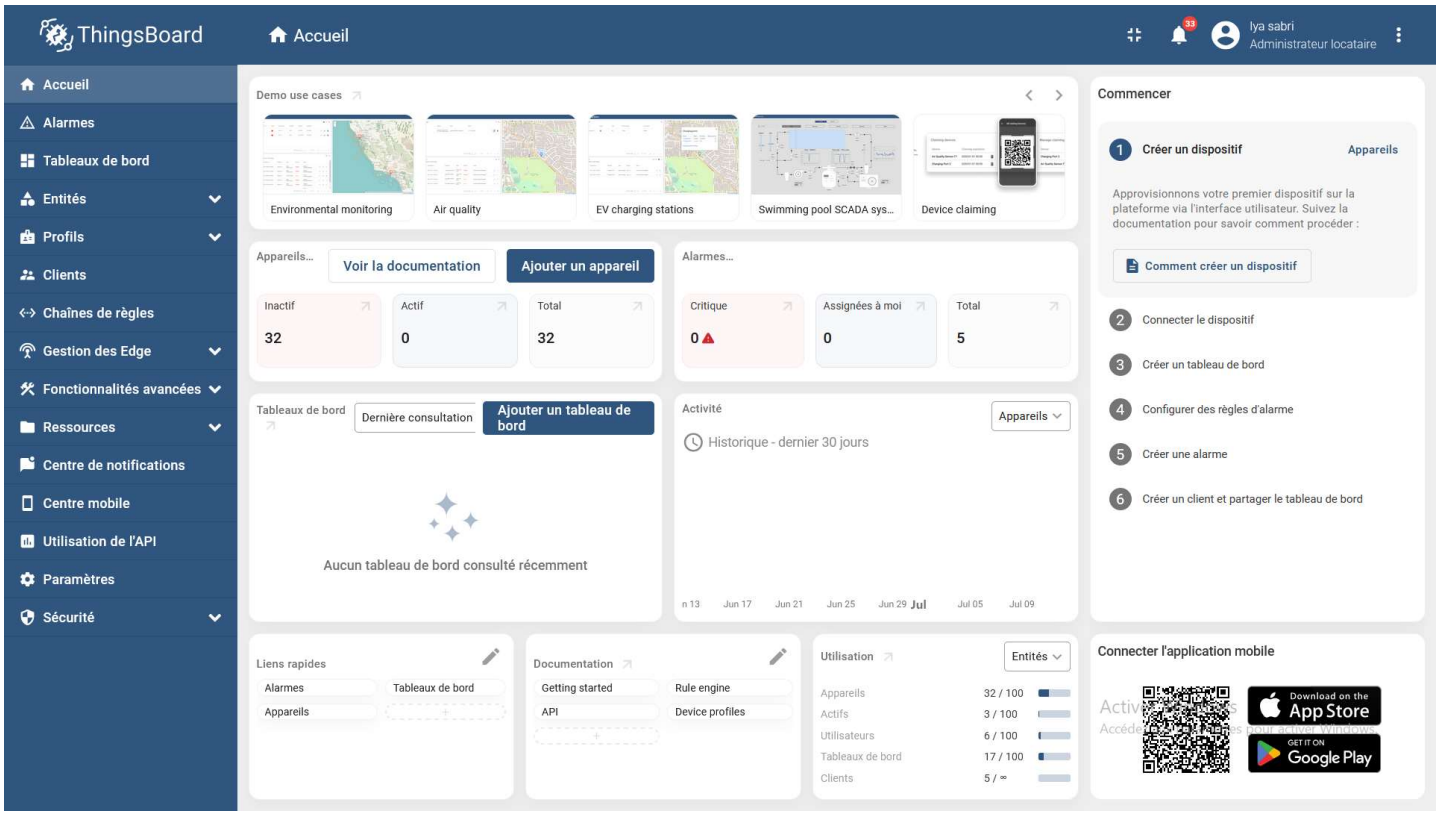


Figure 14 ThingsBoard platform

3.8.1 Data Collection from Devices or Gateways

IoT devices (e.g., sensors, controllers) send telemetry data, attributes, or Remote Procedure Calls (RPCs) using standard protocols such as MQTT, HTTP, or CoAP.

In some cases, a gateway, which acts as an intermediary between the IoT devices and the ThingsBoard platform, is used to collect data from multiple end devices and forward it to the platform. This is particularly useful in scenarios where the devices use different communication protocols or when the devices are located in areas with poor network connectivity.

The built-in transport protocol implementations are applicable for devices that communicate over those protocols and are able to connect directly to ThingsBoard.

- MQTT API reference
- MQTT Sparkplug API reference
- CoAP API reference
- HTTP API reference
- LwM2M API reference
- SNMP API reference

Most of the protocols above support JSON, Protobuf or own data format. This is the best option for new devices when you have control over the firmware

ThingsBoard provides a lot of device connectivity options. The diagram below is designed to provide a visual overview of existing options and help you to choose the correct option for your devices, Figure 15.

ThingsBoard IoT Gateway helps to connect devices that are located in the local network and do not have access to the internet or use specific non-IP protocols. IoT Gateway supports MQTT, OPC-UA, Modbus, BLE, HTTP, CAN, BACnet, ODBC, SNMP and other protocols. The gateway converts the data from devices to internal ThingsBoard format and upload it over MQTT to the platform.

More details will be given in Practical Sessions and the you can find more in following link: https://thingsboard.io/docs/user-guide/rule-engine-2-0/tutorials/validate-incoming-telemetry/

And the following youtube channel : https://www.youtube.com/channel/UCDb9fsV-YR4JmnipAMGsVAQ/videos and https://thingsboard.io/docs/guides/

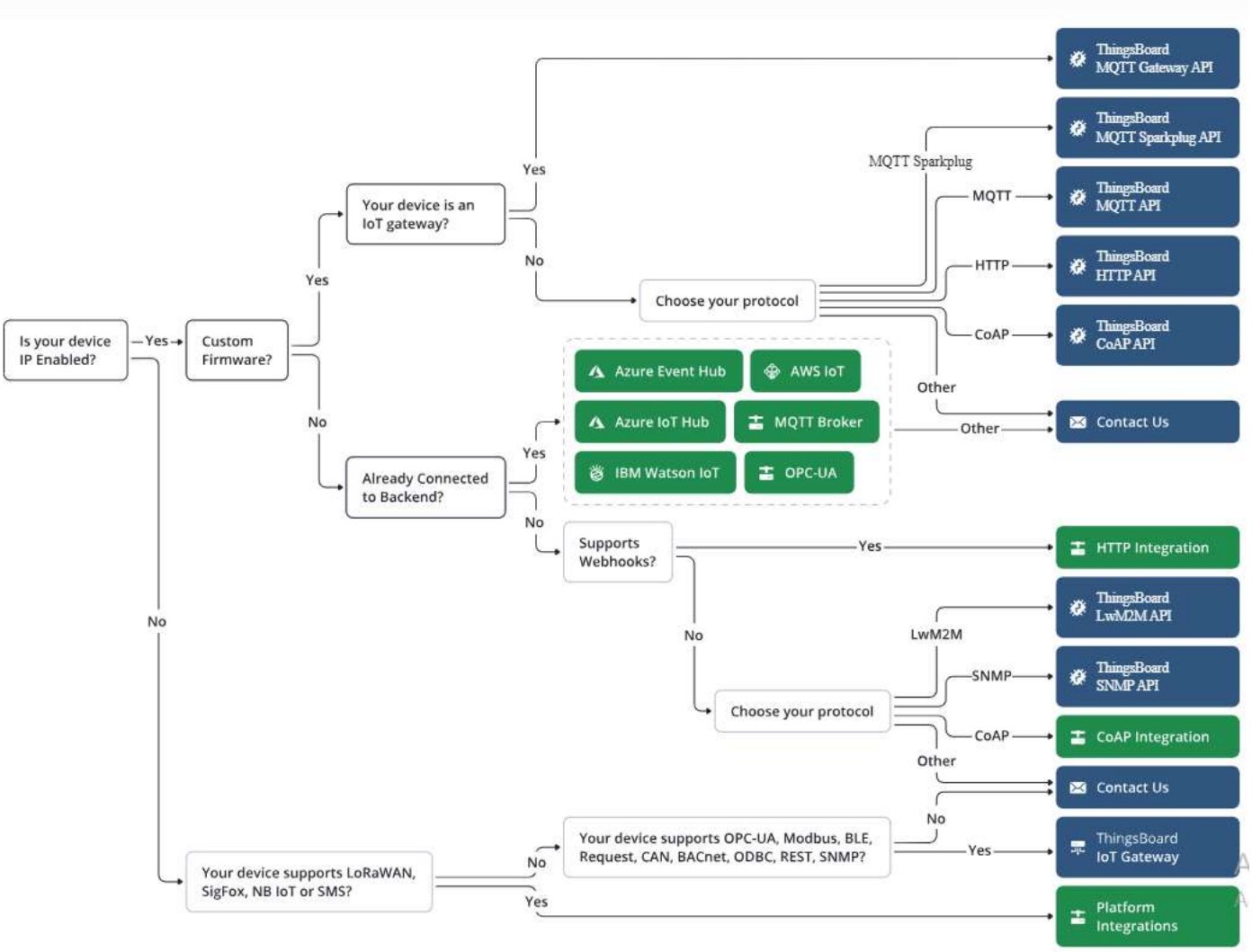


Figure 15 Connectivity diagram https://thingsboard.io/docs/getting-started-guides/connectivity/

3.8.2 Transport Layer Reception and Decoding

Incoming messages are handled by the Transport Layer, which:

- Authenticates the connected device,
- Decodes the message according to its protocol,
- Converts it into a standard **Platform Message Format** for internal processing.

3.8.3 Core Service Routing and Storage

The **Core Service** of ThingsBoard manages message routing:

- **Telemetry data** is stored in a **Time Series database** (e.g., TimescaleDB),
- **Attributes** are saved in a relational database (e.g., PostgreSQL),

- Relevant data is passed to the **Rule Engine** for further automation or logic processing.

3.8.4 Rule Engine Processing

The Rule Engine in ThingsBoard is a user-friendly framework designed to build event-driven workflows. It is composed of three main components:

1. **Message**

   Represents any type of incoming event — such as data from devices, REST API calls, RPC requests, or device life-cycle events.

2. **Rule Node**

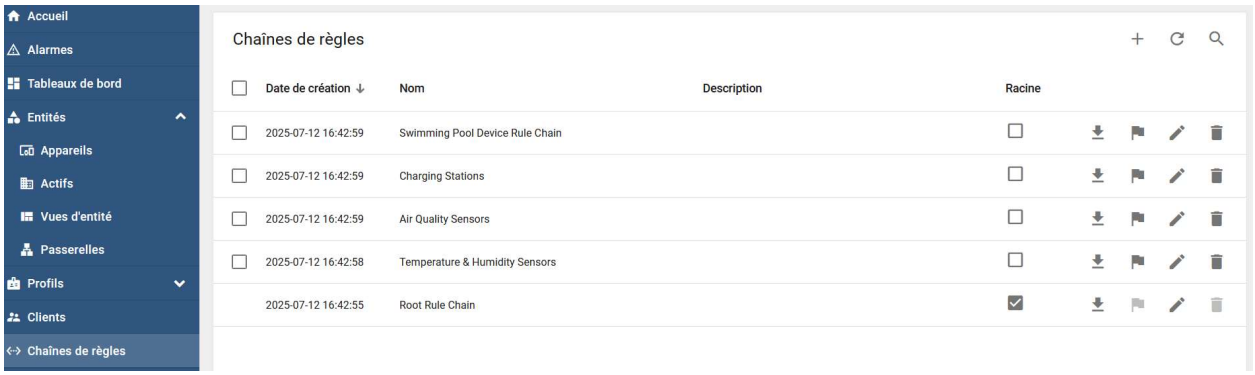
   A functional unit that processes the incoming message. Different node types are available to filter, transform, or trigger actions based on the message content.

3. **Rule Chain**

   A sequence of interconnected rule nodes. The output of one node is sent to the next connected nodes, forming a logical flow of operations.

The Rule Engine enables the execution of user-defined logic through visual rule chains:

- Filtering and transforming incoming data,
- Enriching it with metadata or external inputs,
- Triggering actions such as sending an email, making an HTTP call, or pushing data to external systems like Kafka.



3.8.5 Visualization on Dashboards

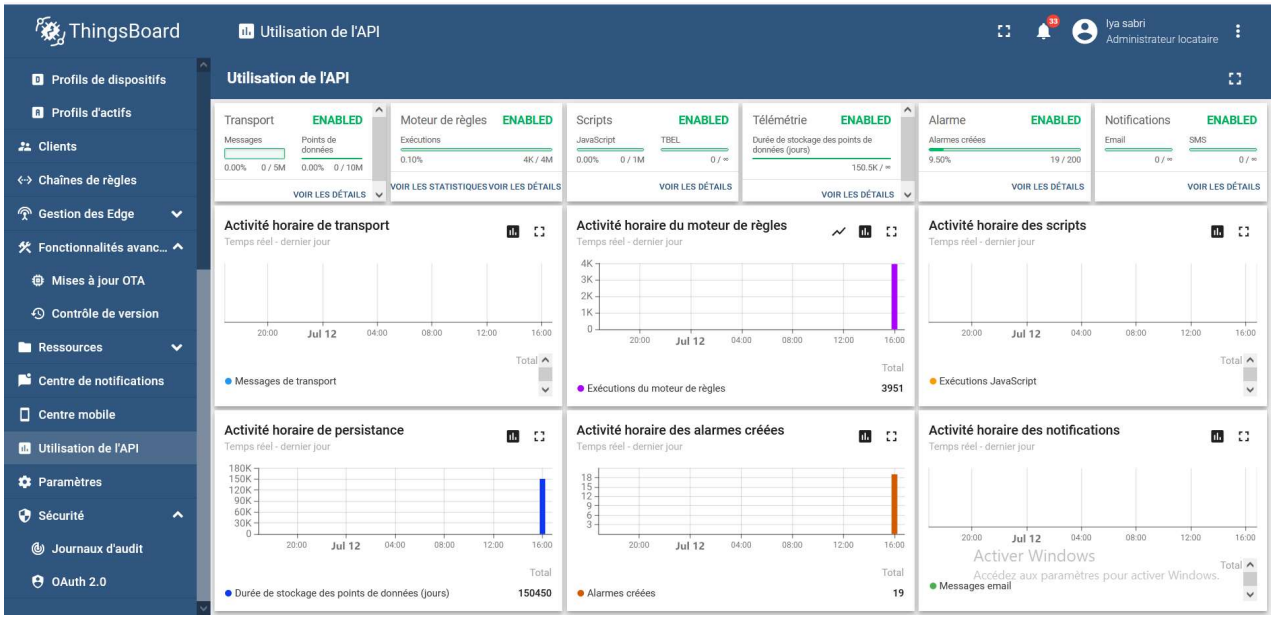
Users can visualize real-time and historical data via customizable **dashboards**:

- Dashboards display data using widgets such as graphs, gauges, alarms, and maps,
- Interactive components allow for direct device control and monitoring.

3.8.6 Integration with External Systems

ThingsBoard can connect with external platforms and services through:

- **REST API**, **WebSocket**, or streaming protocols,
- Cloud integration with providers like **AWS**, **Azure**, or **Google Cloud**,
- Exporting data to Big Data systems or business intelligence tools.



3.9 IoT platform Applications Across Sectors

The Internet of Things (IoT) drives major improvements in efficiency, safety, traceability, and sustainability. By harnessing real-time data collection and analysis, each industry can address its unique challenges and goals, enabling smarter and more responsive decision-making. IoT transforms industries by enabling real-time monitoring, predictive maintenance, and automation. otherwise, in manufacturing, it boosts productivity and quality control. Agriculture benefits from precision farming and livestock monitoring. In energy, smart grids and renewable monitoring enhance efficiency. Transportation uses IoT for fleet tracking and smart warehouses. Healthcare

relies on remote device monitoring and standards compliance. And in the mining sector, IoT plays a crucial role in improving safety and efficiency through automation and environmental sensing.

IoT is a key player in driving sustainability in Smart Cities, Smart Buildings, and Smart Industry (Industry 4.0). By optimising resources and services, it enhances Smart Transport, Farming, Energy, and Water Management, making operations more sustainable and efficient. Fleet Management is streamlined through telematics and connected solutions, Figure 16.

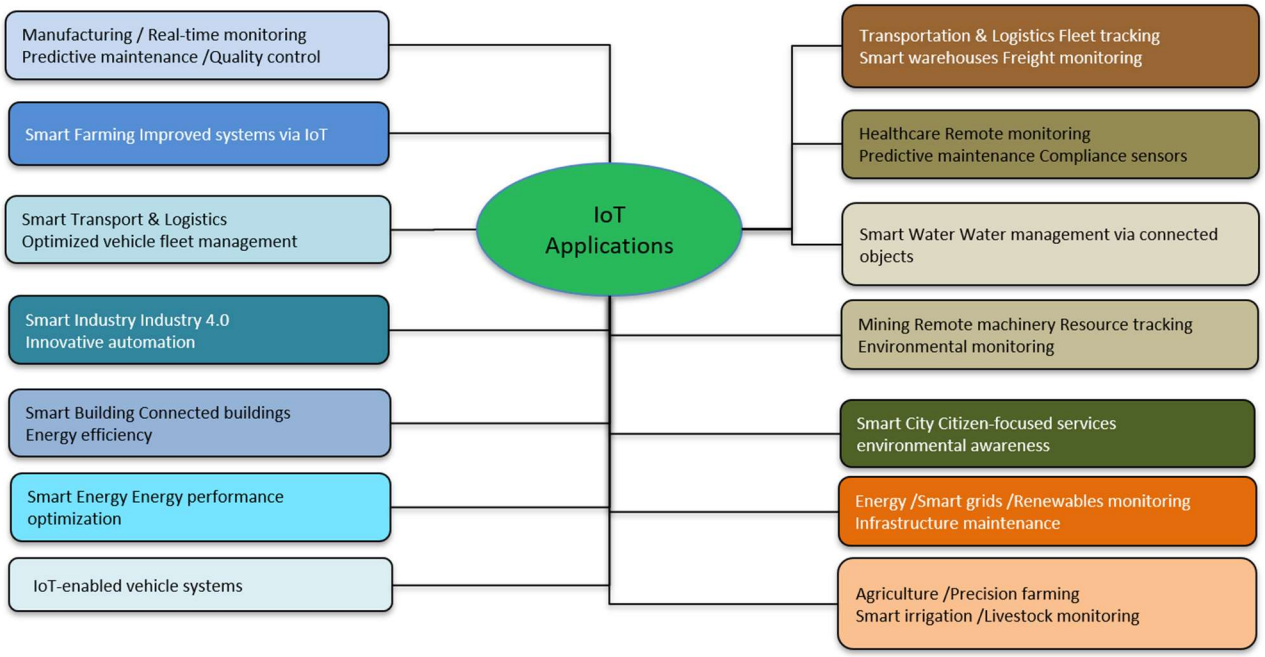


Figure 16 Main Pplatform Applications Across Sectors

.

4 Chapter 3: Communication networks and messaging protocols for Internet of Things

Data communication protocols are essential in IoT systems to ensure the efficient and secure exchange of information between sensors, gateways, and cloud applications. MQTT (Message Queuing Telemetry Transport) is one of the most popular protocols, designed for unreliable networks with low bandwidth consumption. CoAP (Constrained Application Protocol), a lightweight and easy-to-implement protocol, is suitable for constrained devices, utilising UDP. HTTP, although heavier, is still used for some IoT web applications. AMQP (Advanced Message Queuing Protocol) is a reliability- and security-oriented protocol, often used in critical industrial applications. DDS (Data Distribution Service) is a real-time, distributed protocol, particularly suited for embedded systems and robotics. Finally, LwM2M (Lightweight Machine-to-Machine) is optimised for the remote management of low-power IoT devices. The choice of protocol depends on the application constraints: resources, latency, security, and scalability needs.

4.1 HTTP: What's the problem?

Some traditional protocols are unsuitable for the IoT paradigm due to their verbose nature, excessive resource consumption, and half-duplex operation, such as HTTP. HTTP is a well-proven and effective protocol in traditional web architectures. However, it is not suitable for implementing the IoT paradigm, as it does not meet the specific constraints of this domain, which only allows communication in one direction at a time: the client sends a request, and then the server responds. This model is inefficient in IoT environments where responsiveness and lightweight performance are essential. Furthermore, the pull mode, in which the client periodically polls the server for new information, results in unnecessary bandwidth and energy consumption, especially when the server has nothing new to transmit. Conversely, the push mode, more suited to the IoT, allows the server to send data only when an event occurs, making the system more efficient and less energy-intensive.

4.1.1 Overcoming HTTP Limitations: Simulating Server Push and Real-Time Updates

Although HTTP is designed for a pull-based communication model, several techniques have been developed to simulate server-side push behaviour to approximate real-time

performance. The first method, polling, involves sending periodic, synchronous HTTP requests even when there are no new messages. This generates significant network and server load, especially if messages are rare. An improvement on this technique is long polling, which keeps a request open until a response is available or a timeout occurs. This reduces the number of requests but does not eliminate the overhead of connection management. Finally, the streaming method relies on continuously opening a connection between client and server, without closing it, even when there is no data. However, this approach, while closer to true push, remains limited to one-way communication and poses compatibility issues with some proxies and firewalls, which tend to block long connections. These solutions illustrate the limitations of HTTP in real-time environments, such as IoT, but also point to a promising future with the emergence of lighter and more suitable alternative protocols, such as MQTT or CoAP.

In the following, we will focus on a more theoretical aspect, delving into the concepts related to communication protocols, particularly MQTT. However, for a better understanding of the principles covered, practical exercises are provided. These allow us to illustrate the mechanisms discussed concretely. In addition, we will also introduce other vital protocols such as JMS and CoAP, broadening your perspective and providing a comprehensive understanding of messaging solutions in IoT.

4.2 MQTT (Message Queuing Telemetry Transport)

MQTT, developed by IBM in 1999 for embedded device telemetry, is a lightweight protocol based on the publish/subscribe model, ideal for the IoT, Figure 17. It allows a message to be broadcast to multiple clients over a single connection, without them knowing each other. Each client can publish or subscribe to notifications. MQTT is adaptable to different network types, operating on top of TCP/IP, but also having a version suitable for non-TCP/IP networks (MQTT-S). Independent of the message content, it is particularly ideal for wireless networks with limited bandwidth. MQTT is an open standard maintained by OASIS. It follows a publisher-subscriber paradigm with a broker. Publishers and subscribers act as clients, while the broker serves as the server. The broker acts as an intermediary (node) that relays messages based on their topics. Topics are organized in a hierarchical structure. The broker may discard a message sent to all subscribers of a given topic; it may also discard a message if there are no subscribers.

An IoT device, network service, or process can include a publisher, a subscriber, or both. The broker can be deployed on a cloud or edge server, or on an IoT gateway that separates a

network of devices from the internet. A server that supports the broker can act both as a publisher and a subscriber to another broker, enabling hierarchical and scalable deployments.

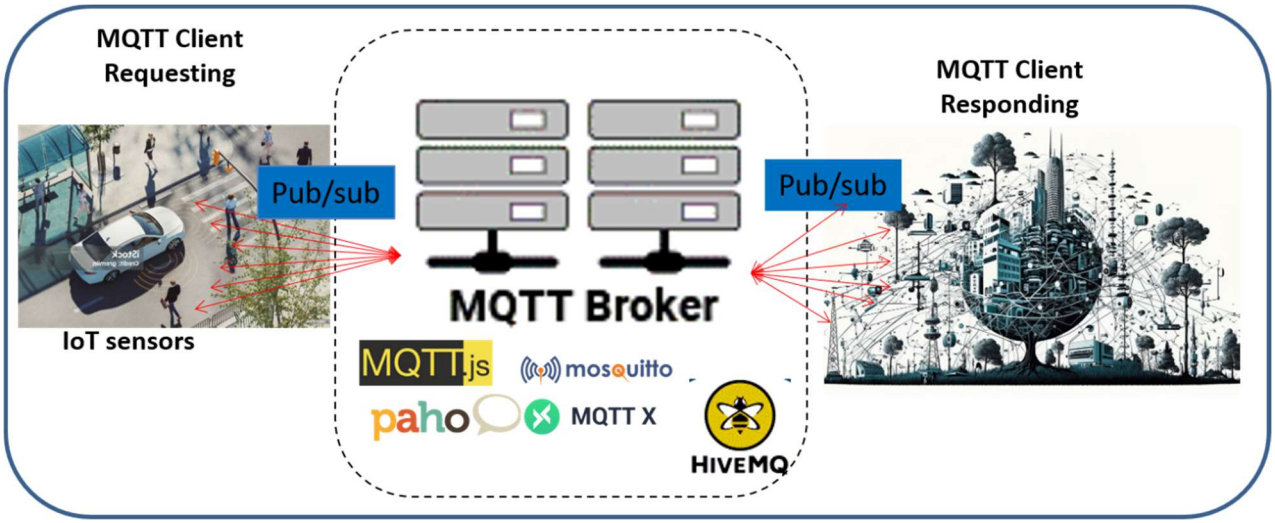


Figure 17 MQTT Publish/Subscribe Architecture

MQTT, a lightweight and efficient protocol, is a versatile solution for IoT environments. It not only enables bidirectional communication, but also offers high scalability (up to millions of devices) and ensures reliable message delivery through Quality of Service (QoS) levels. It supports session resumption and message buffering in case of disconnection, while integrating security mechanisms such as TLS encryption and client authentication. This bidirectional communication feature underscores MQTT's flexibility and adaptability, making it a powerful tool for IoT developers and network engineers.

MQTT 5[20] is a significant but non-disruptive evolution of version 3.1.1. It retains the strengths that made the protocol so successful—lightweight, push-based communication, simplicity, scalability, and adaptability to mobile networks. While remaining true to the original spirit of MQTT, this new version introduces technical improvements and tweaks that strengthen its position as the leading IoT protocol.

4.2.1 Concrete Applications of MQTT

It is widely used in IoT, IIoT, and M2M communications and adopted by large companies such as BMW, Air France-KLM, Mercedes-Benz, Hytera, Awair, etc. Sectors concerned: Automotive, telecoms, energy, public safety, connected objects.

Application Examples:

[20] https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html

Smart Homes: Connecting thermostats, cameras, light bulbs, etc., for remote control via mobile app.

Industrial Automation: Communication between machines and sensors for real-time monitoring and reduced production downtime.

Agriculture: Monitoring soil moisture, weather, and crop growth for optimised irrigation (precision farming).

Mobile robotics: seamless communication between embedded modules for navigation, obstacle detection, and coordination.

Social messaging and real-time chat: using MQTT for instant notifications and lightweight message transmission in mobile applications.

Transportation and connected vehicles: real-time tracking, remote diagnostics, and route optimisation.

Healthcare: Connecting medical devices (glucose monitors, heart rate monitors) for remote patient monitoring.

4.2.2 MQTT Basics

- MQTT operates on a publish/subscribe model involving two main components : MQTT clients and an MQTT broker.
- The broker acts as a central intermediary, receiving messages from clients (publishers) and forwarding them to other clients (subscribers) based on topics.
- Clients publish messages to topics and subscribe to topics of interest to receive relevant data.
- The broker manages subscriptions and ensures that each message is delivered to the correct recipients.
- MQTT supports message buffering for clients that are temporarily disconnected, improving reliability.
- To handle different reliability needs, MQTT offers three levels of Quality of Service (QoS) for message delivery.

There are three levels of QoS (Quality of Service) for data delivery[21]:

- ✓ QoS 0 – at most once: best-effort delivery. This level ensures "at most once" delivery, where messages are sent without confirmation and may be lost. This is the lowest QoS level, typically used in situations where message loss is acceptable or when the message is not critical.

- ✓ QoS 1 – at least once: duplicates may occur. The publisher sends the message to the broker and waits for confirmation before proceeding. If the broker does not respond within a specified time, the publisher resends the message. This QoS level is typically used in situations where message loss is unacceptable, but duplication is tolerated. For example, a QoS of 1 might be appropriate for sending command messages to devices, where a missed command can lead to system failure, whereas duplicate commands are not a concern.

- ✓ QoS 2 – exactly once: reliable delivery. Thid level is the highest level of QoS and is typically used in situations where message loss or duplication is entirely unacceptable. The publisher and broker engage in a two-step confirmation process, where the broker stores the message until it has been received and acknowledged by the subscriber. This level of QoS is typically used for critical messages such as emergency alerts, Figure 18.

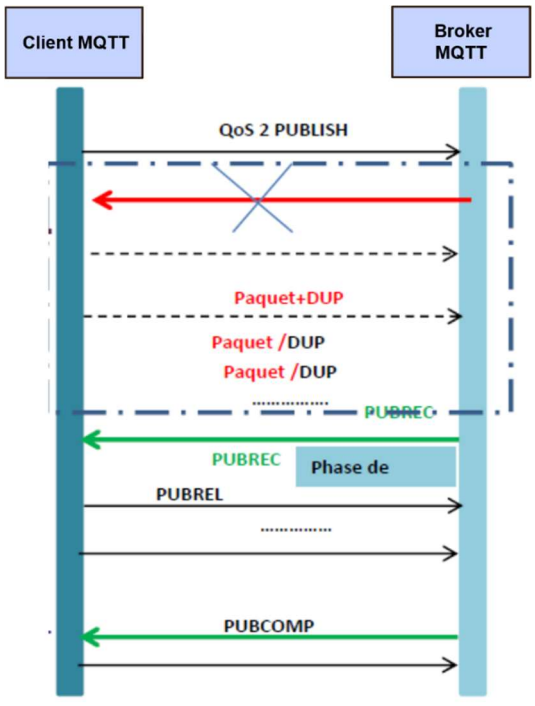


Figure 18 QoS 2 Message Transmission

[21] https://www.hivemq.com/mqtt/

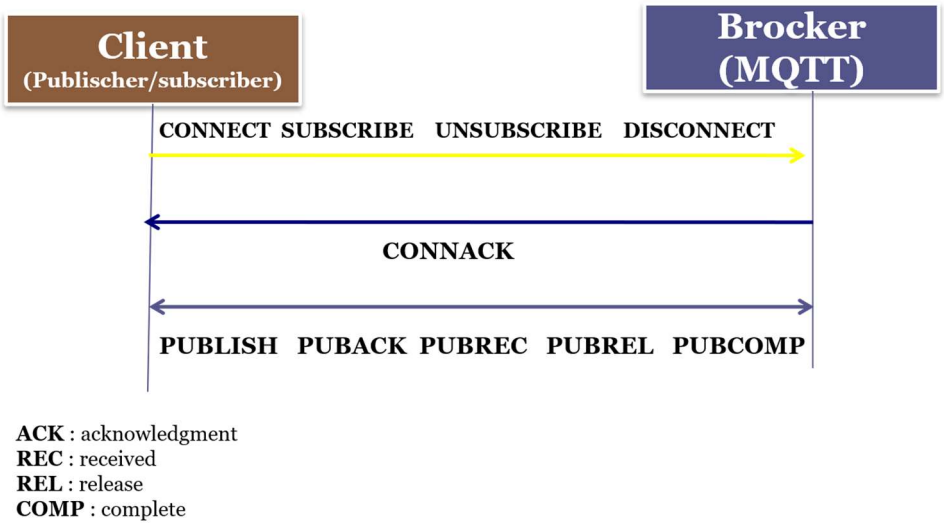


Figure 19 MQTT: Communication

4.2.3 MQTT Clients & brokers

There are many open-source MQTT client libraries available to developers across different programming languages, making it easier to implement MQTT in IoT systems, Figure 19.

- ✓ HiveMQ offers high-performance MQTT client libraries focused on security, usability, and reliability, supporting languages such as Java, C#, C++, Python, and WebSockets.
- ✓ Eclipse Paho provides widely used client libraries for C, C++, Java, Python, and others, and is known for its strong community support;
- ✓ Mosquitto, while primarily known as a lightweight MQTT broker, also includes command-line client tools like mosquitto_pub and mosquitto_sub, allowing users to publish and subscribe to MQTT topcs easily for testing and scripting purposes;
- ✓ A comprehensive list of MQTT clients and libraries is maintained on mqtt.org, providing developers with a broad choice depending on their programming environment;
- ✓ ActiveMQ is an open-source messaging system developed by the Apache Foundation. It implements the Java Message Service (JMS) protocol, allowing asynchronous message exchange between different distributed applications or services. Thanks to its compatibility with several protocols (including MQTT, AMQP, STOMP), ActiveMQ is widely used in message-oriented architectures, particularly in Java environments. It offers advanced features, including message persistence, fault tolerance, and queue and topic management;

Unlike request-response approaches that create a bottleneck that slows performance, the Pub/Sub mechanism decouples the message sender from the subscribers. The subscriber and publisher are each unaware of the other's existence. The broker manages the connection between them. This decoupling enables faster and more efficient communication, eliminating the need for the exchange of IP addresses and ports. It also ensures decoupling, allowing the two components to communicate continuously and without interruption during publication or reception. The pub/sub model ensures efficiency through three decoupling types: **Space** (no mutual knowledge about location), **time** (asynchronous operation), and **synchronisation** (independent during processing). Furthermore, The pub/sub model ensures high scalability through event-driven processing, enabling the efficient handling of multiple clients in parallel. Features like message caching and intelligent routing further enhance scalability by reducing processing time and network traffic.

However, the Pub/Sub architecture does generate some drawbacks, which we can summarise as follows:

1. In some cases, the subscriber is unavailable to receive a message for a particular topic, resulting in message loss. To address this, MQTT provides Quality of Service (QoS) levels.

2. Guarantee that each subscriber only receives the messages they are interested in (this is called filtering). As we will see, MQTT offers filtering by topic, content, and type. Each option has advantages and disadvantages, and the choice of filtering method depends on the use case.

3. Security is a crucial aspect of any messaging system. MQTT provides several security options, including user authentication, access control, and message encryption, to safeguard the system against unauthorised access and data breaches.

4. To ensure scalability, MQTT offers features such as multiple brokers, clustering, and load balancing to ensure the system's ability to handle large numbers of subscribers and messages.

5. Because messages are sent asynchronously, it becomes challenging to ensure that subscribers receive messages in the correct order. However, MQTT offers Quality of Service (QoS) levels that ensure the proper transmission of messages from the client to the broker or from the broker to the client. Because MQTT operates asynchronously, tasks are not blocked while

waiting for or publishing a message. Most client libraries rely on callbacks or a similar model, resulting in a generally asynchronous message flow.

6. Real-time constraints are critical, and the pub/sub architecture may not be the best choice.

Unlike text-based protocols like HTTP or SMTP, MQTT relies on a compact binary message format for communication between clients and brokers, ensuring more efficient data exchange, Figure 20.

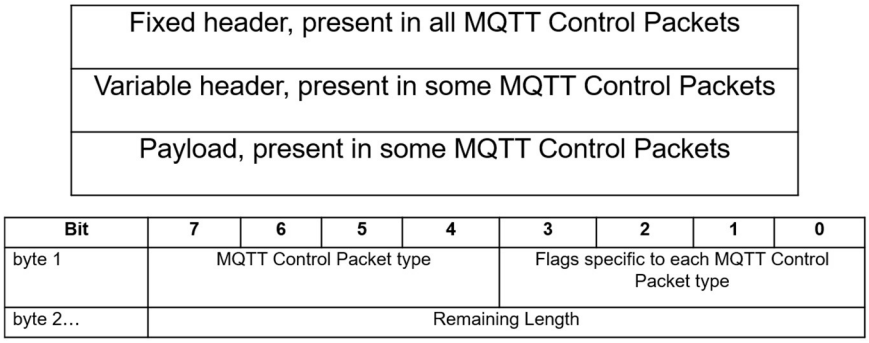


Figure 20 MQTT Fixed Header format example. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html

In the following, we will consider the use of ActiveMQ as a broker. In principle, the methodology remains broadly similar to that of other brokers, with differences mainly being technical or syntactic, and without a significant impact on the fundamental concepts.

4.2.4 How to Set Up MQTT Client-Broker Communication

The figure Figure 21 illustrates the various stages of the connection process between an MQTT client and a broker, from initialisation to connection maintenance. It highlights the logical sequence of packet exchanges and the role of each component in establishing communication.

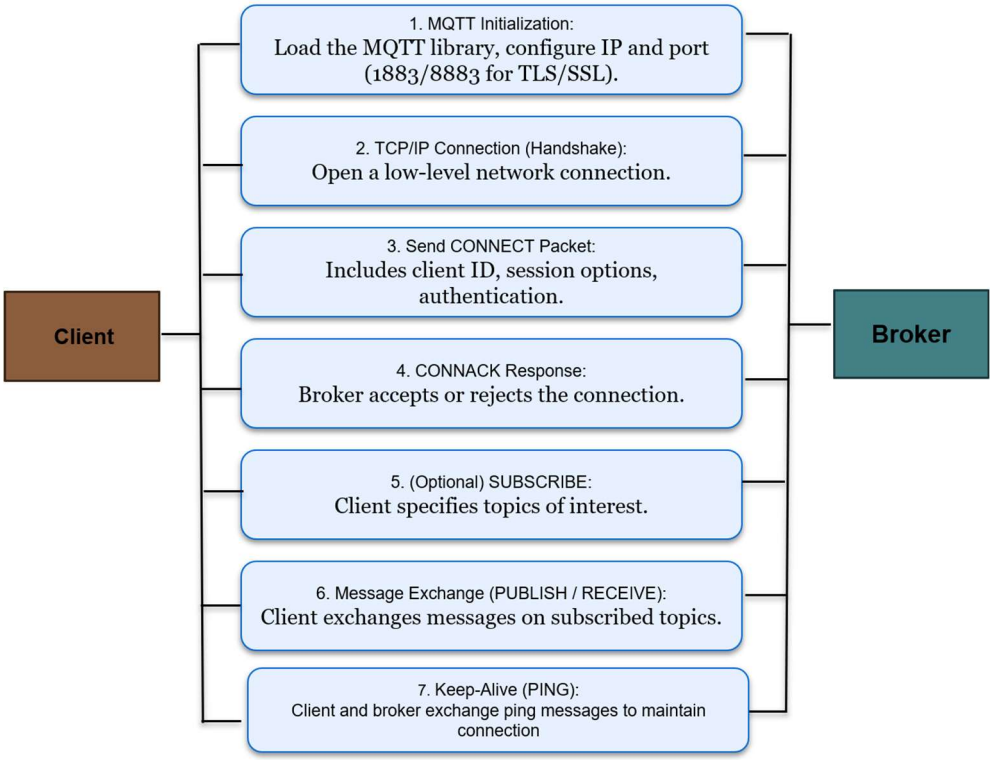


Figure 21 Connection Flow Between an MQTT Client and a Broker

The latest version of MQTT, version 5.0, was released in 2019. Since 2017, work has been ongoing to define a TLS profile for MQTT. This profile authorizes secure connections between clients and brokers and protects resources identified by topic names. An authorization server provides tokens to clients, which prove their right to access the broker to publish or subscribe to a given topic.

Several security solutions for MQTT have been proposed in the past. For example:

- In 2014, Neisse et al. integrated their *Model-based Security Toolkit* with MQTT to address security and privacy requirements.
- In 2015, Singh et al. proposed a security extension for MQTT and MQTT-SN called Secure MQTT (SMQTT). This extension relies on lightweight elliptic curve cryptography and allows encrypted messages to be broadcast to multiple nodes.

However, these solutions were not included in the latest 2019 MQTT version, which introduced an enhanced authentication method. This method typically carries the SASL (Simple Authentication and Security Layer) mechanism but is not limited to SASL—others like Kerberos can also be applied. The MQTT specification strongly recommends using TLS to secure message exchange with the broker.

MQTT operates over TCP, TLS, or WebSocket. The standard defines 14 types of PDUs. The PDU length can vary from 2 bytes to 256 MB, and the header length ranges from 2 to 5 bytes. The published payload content is application-specific. The topic is encoded as a UTF-8 string, and the payload can be a byte or a UTF-8 string.

The MQTT standard defines various protocol fields in fixed and variable headers, as well as many properties specific to each PDU type. The application developer can set the values of these protocol elements; only a few are fixed by the standard. There are 99 elements in total, which illustrates the protocol's complexity.

MQTT was designed to minimize client implementation complexity. However, broker implementation must be robust. Redundancy may be required to eliminate the risk of a single point of failure. When high scalability is needed, a distributed broker implementation is necessary. MQTT is supported by many programming libraries, frameworks, and learning tools. See https://mqtt.org.

4.2.5 MQTT-SN (MQTT for Sensor Networks)

MQTT-SN is a variant of MQTT developed by IBM and accepted by OASIS. It is designed for constrained devices. It operates efficiently over wireless radio links, where low bandwidth and a high transmission failure rate are expected, and messages must be short. MQTT-SN provides the same features as standard MQTT. The main difference is that it can operate over non-TCP/IP networks. It was initially developed for ZigBee, but Bluetooth and UDP are also used. Other differences between MQTT and MQTT-SN can be seen as extensions.

4.2.6 MQTT Topic Name

An **MQTT topic** is a hierarchical string, structured with forward slashes (/), used to label messages. Clients subscribe to specific topics to receive only relevant data.

In MQTT, topics are organized in a hierarchical tree, where each level is separated by a slash /. This structure allows for great flexibility in message subscriptions through the use of wildcards. The + symbol allows targeting a single level in the hierarchy, while # allows covering multiple levels from a given point. The $ character is used to exclude specific topics, often reserved for internal broker use. Topics beginning with $ are reserved and cannot be used for publishing. These mechanisms perform both precise and powerful message filtering, facilitating

communication management in complex IoT environments.“+” matches a single level (e.g., house/+/livingroom)

This structure enables efficient and targeted message filtering. To maintain performance, it's best to use specific topic filters instead of broad wildcards. Shared subscriptions can distribute the load across multiple clients, reducing broker overhead. Regularly monitoring and tuning the broker helps ensure optimal efficiency.

4.2.7 MQTT Retain Flag, Payload, MQTT LWT, Keep Alive, and DUP Flag:

- ✓ **Retain Flag**: When activated, the broker preserves the most recent message for a topic and transmits it to new subscribers, a crucial feature that guarantees they receive the most up-to-date data. However, an excessive number of retained messages or frequent updates can negatively impact broker performance.
- ✓ **Payload**: This is the actual data carried by the message. While MQTT supports all data types, it's essential to be mindful of the potential impact of large payloads on memory and network efficiency. Keeping payloads small is recommended, especially with high-frequency messages, to ensure efficient data management.
- ✓ **DUP Flag**: Indicates a **duplicate message** resent due to missing acknowledgement (used with Quality of Service (QoS) levels higher than 0). While MQTT handles duplicates automatically, they can increase network traffic and affect performance ;
- ✓ **Handling Disconnections with MQTT LWT**: The Last Will and Testament (LWT) is a mechanism that allows a client to define a message to be sent by the broker in case of an unexpected disconnection. This message, which includes a topic, payload, QoS level, and optional retention, is set during the initial connection to the broker. If the client goes offline without sending a proper disconnect signal, the broker publishes the LWT message to alert other clients. This ensures that devices or services depending on that client are aware of its unavailability. LWT is particularly useful in IoT environments, where network instability or power loss is common, and helps maintain communication integrity across the system;
- ✓ The **Keep Alive** mechanism in MQTT ensures that a client maintains an active connection with the broker by sending regular **PINGREQ** packets within a negotiated time interval. The broker replies with **PINGRESP**, confirming the connection is still alive. This feature is essential for **monitoring connection status**, detecting failures, and optimising resource usage, especially in **mobile or unstable networks**.

4.2.7.1 MQTT Topic Naming and Structure: What to Know

MQTT topics are essential for enabling efficient communication, filtering, and routing between clients and brokers. They must contain at least one character and can include spaces for clarity, however, avoid using spaces and uncommon characters in MQTT topics, as they can complicate readability and debugging. For clarity and efficiency, topics should remain short, concise, and free of unnecessary formatting. It's crucial to remember that topics are case-sensitive, and even a single slash / is considered a valid topic (serve as a wildcard for subscribing to multiple topics simultaneously). Proper topic structure is key to ensuring reliable and scalable data exchange in MQTT systems.

In MQTT, wildcards allow clients to subscribe to multiple topics at once, making message filtering more flexible. They can only be used in subscriptions, not in publishing, and help match either specific or broader topic patterns.

4.3 CoAP (Constrained Application Protocol)

CoAP is an IETF standard. It can be considered a specific version of the HTTP protocol designed to communicate with resource-constrained devices. It suits the RESTful programming approach and follows the client-server paradigm. The server provides resources identified by URIs. The client communicates with a server at a known and predefined address, although it is possible to use dynamic server discovery mechanisms within local networks. A given device may act as both server and client.

An extension defined by RFC 7641 allows the client to subscribe to a server resource that is updated over time, offering a publisher-subscriber communication model. CoAP supports several communication architectures, such as peer-to-peer and client-server.

Any IoT platform for constrained or non-constrained devices can use CoAP. However, in the case of constrained devices, particular attention must be paid to how the payload is encoded to minimize its size.

- An IoT device acting only as a client can be used for data acquisition, notification/alarm analysis, as well as device discovery and configuration.
- An IoT device acting only as a server can be used for executing commands, querying data, and remote control processes.

- An IoT device acting as both client and server can be used in any communication scheme, including peer-to-peer communication between neighboring devices.

CoAP can be used in any Web-of-Things application. It is as flexible as HTTP but better suited for device-to-device communication. It is efficient for communication with constrained devices provided the system is carefully designed and programmed.

4.4 STOMP (Simple/Streaming Text Oriented Messaging Protocol)

STOMP is developed and maintained by the programming community (http://stomp.github.io). STOMP follows the publisher-subscriber paradigm with a STOMP broker (server). A client can act as a publisher, subscriber, or both. Through the server, STOMP can provide one-to-many communication. The server is an intermediate node that relays messages according to their destination value. The STOMP destination plays the same role as the MQTT topic.

Multiple servers can be deployed in a chain linking two clients. The server can modify and add headers to the transmitted messages. The protocol supports a client-message_server-client communication model. The message server can perform part of the message processing, which goes beyond a basic broker functionality.

STOMP messages are transported over TCP. It is also possible to use WebSocket to avoid issues with firewalls that block most TCP ports.

Depending on application requirements, the features of a STOMP server can be reduced or extended; thus, the software may be quite simple or very complex. Regardless of the software simplicity, it is difficult to imagine an application where an IoT device would act as a message broker.

An IoT device acting as a STOMP client can be used in any communication scheme except for peer-to-peer communication between neighboring devices. However, the STOMP server can be part of an application server that manages a group of IoT devices as STOMP clients.

4.5 WAMP (Web Application Messaging Protocol)

WAMP is an open protocol maintained by Crossbar.io (https://wamp-proto.org). It is a WebSocket subprotocol that enables the transfer of Remote Procedure Calls (RPC) and Publisher-

Subscriber messages. The protocol supports client–message router–client and client–router communication models.

A client communicating with a WAMP router (server) can register procedures to be executed and subscribe to message topics. Another client can invoke registered procedures and publish messages to specific topics. The router thus acts as a message broker, a remote procedure broker, or both.

There are no restrictions preventing an application from playing multiple roles among the following: callee, caller, publisher, subscriber, and broker. The WAMP protocol defines the communication between the client and the router. Using TLS is recommended to secure communications. Depending on the application's needs, the features of a WAMP router can vary, but it must be a robust and reliable communication node.

An IoT device acting as a WAMP client can be used in any communication scheme, except for direct device-to-device communication.

WAMP can be used in any web application that uses microservices or offers user collaboration, such as video games. It can also be used in certain IoT systems, namely those that support WebSocket transport. Many IoT devices simultaneously offer procedures to control actuators and publish messages from sensors.

4.6 AMQP (Advanced Message Queuing Protocol)

AMQP is maintained by OASIS and standardized by ISO and IEC. A communication node in AMQP can act as a producer, consumer, or message queue. A single process can include multiple such nodes, allowing for the construction of very complex network components. A process can act as a client, server, broker, or router. The router may forward a message immediately or store it, depending on the application's needs.

The connected end processes form a peer-to-peer network. The main communication model of the protocol is client–message_router–client. However, an implementation may provide only basic broker functionality instead of a full message router.

A noteworthy feature offered by AMQP is the ability to handle transactions, i.e., organizing groups of messages for joint processing. Transactions can be defined, rolled back, and

executed. A separate link is created to control transactions, independently of the links used for message routing.

The rich syntax of AMQP messages and message fields makes the protocol universal and flexible. Most of its features are optional, which allows tailoring the size and complexity of the implementation to a given application's needs. During connection and session establishment, the peers signal the features they expect and support.

The origins of this protocol are tied to distributed financial applications. It is primarily used in business applications.

4.7 JMS (Java Message Service)

Over the years, systems have significantly increased in terms of complexity and sophistication. The need for more reliable, scalable, and flexible systems than in the past has led to the emergence of more complex and sophisticated architectures.

In response to this growing demand for better and faster systems, architects, designers, and developers have leveraged messaging as a way to address these complex challenges.

Although the Java Message Service (JMS) API hasn't changed much since its introduction in 1999, messaging is widely used to solve reliability and scalability issues. It is also used to tackle other problems encountered in many business and non-business applications. Heterogeneous integration is a primary area where messaging plays a key role. Messaging also enables asynchronous request handling, offering architects and developers solutions to reduce or eliminate system bottlenecks, increase end-user productivity, and improve overall system scalability.

Since messaging provides a high degree of decoupling between components, systems using messaging also offer a high degree of architectural flexibility and agility.

Whether due to mergers, acquisitions, business requirements, or simply a shift in technological direction, more and more companies face the challenge of integrating heterogeneous systems and applications within and across the enterprise. It is not uncommon to find a mix of technologies and platforms within a single organization or even a single division — including Java EE, Microsoft .NET, etc.

JMS is a vendor-independent Java API that can be used with many different enterprise messaging providers. JMS is analogous to JDBC in the sense that application developers reuse the same API to access different systems. If a vendor provides a JMS-compliant service, the JMS API can be used to send and receive messages through that vendor's system (e.g., SonicMQ, IBM WebSphere MQ).

A subscription can be durable or non-durable. A durable subscription requests the broker to store messages for a given client while it is offline, such as during sleep mode. Additionally, a publisher can ask the broker to retain messages even if no subscribers are currently registered for the topic. In such cases, the next message will overwrite the previous one, and any newly connected subscriber will receive the most recent value for the topic.

4.8 Heterogeneous Integration

Communicating and integrating heterogeneous platforms is perhaps the most classic use case for messaging. Using messaging, you can invoke services from applications and systems implemented on completely different platforms. Both open-source and commercial messaging systems provide seamless connectivity between Java and other languages/platforms by leveraging a built-in message bridge that converts a message sent via JMS into a common internal message format.

Examples of such messaging systems include:

- ActiveMQ (open-source)
- IBM WebSphere MQ (commercial)

Both of these systems support JMS, but they also expose native APIs for use by non-Java or non-JMS messaging clients (such as those in C or C++). The key point is that, depending on the provider, it is possible to use JMS to communicate with non-Java or non-JMS clients.

4.9 JMS Messaging Model

JMS supports two types of messaging models:

- Point-to-point
- Publish/Subscribe

These messaging models are sometimes referred to as messaging domains. Point-to-point and publish/subscribe messaging are often abbreviated as p2p and pub/sub, respectively.

At its simplest:

- Publish/Subscribe is used for one-to-many message distribution.
- Point-to-point is used for one-to-one message delivery.

From a JMS perspective:

- Messaging clients are called JMS clients
- The messaging system is called the JMS provider.

A JMS application is a business system composed of multiple JMS clients and typically one JMS provider.

- A JMS client that produces a message is called a message producer.
- A JMS client that receives a message is called a message consumer.

A JMS client can be both a producer and a consumer of messages. When using the term consumer or producer, we refer to a JMS client that consumes or produces messages, respectively.

4.10 Some Key Definitions

In client-server communication systems, several modes of data exchange exist, each adapted to specific needs in terms of responsiveness, network load or resource management, see also Figure 22:

- **QueueBrowser**
  A QueueBrowser is a specialized object that allows you to preview messages pending in a queue without consuming them.
- **Server Push**
  Server Push is a client-server communication mode in which the dialogue is initiated by the server;

- **Pull and push** represent two opposing approaches to information exchange. In pull mode, the client actively requests or queries data from a server or source. This mode allows precise control over when and what information is retrieved. Conversely, in push mode, the client is passive: it automatically receives data as soon as it becomes available, without having to request it. This approach is often used for real-time notifications, but it requires the server to decide when to send the information.
- **Polling**
    - Periodic requests sent by the client, synchronously;
    - Even if there are no new messages, requests are sent (empty requests);
    - Generates a large number of connections and unnecessary traffic;
    - Only suitable if messages are regular and frequent;
    - Inefficient in terms of resources and bandwidth;
- **Long Polling**
    - An improved version of polling: the client sends a request and waits for a response before sending another;
    - The request remains open until a response is available or until a timeout;
    - If no messages arrive, the connection is closed anyway (unnecessary keep-alive cost);
    - Reduces the number of requests but remains based on a request pattern;
    - More efficient than traditional polling, but not optimal at large scale;
- **Streaming**
    - The client opens a persistent connection with the server;
    - Data is pushed as soon as it becomes available, without any new requests;
    - No connection closure even if data is missing;
    - Unidirectional (no communication from client to server in some cases);

- May cause problems with some proxies or firewalls due to the connection time;
- Ideal for continuous feeds or real-time updates.

Mode	Direction of Communication	Simultaneous	Example
Simplex	One-way	No	Television, unidirectional sensor
Half-duplex	Two-way, alternate	No	Talkie-Walkie, IoT serial port
Full-duplex	Two-way simultaneous	Yes	Telephone, WebSocket

Figure 22 Communication Modes Comparison

What Is Shadow IT?

Shadow IT refers to IT systems deployed by individuals or departments outside the central IT department of an organization. These unofficial systems are often set up due to perceived or actual gaps in centrally managed information systems.

4.11 Comet and WebSocket

In a Comet-based web application, the HTTP request generated by the browser receives data from the server without needing additional requests. The Comet application replaces traditional polling and page-by-page rendering with a real-time persistent HTTP connection.

According to the HTTP 1.1 specification, the browser must not maintain more than two simultaneous HTTP connections to the web server. Therefore, keeping one connection open for real-time server events can lead to issues. For example, the browser may remain idle and unable to generate a new request until the previous one is fulfilled — this has a negative impact on browser usability.

4.11.1 WebSocket

In streaming applications, Comet opens a single persistent connection for all comet events. For each new comet event on the server side, the client handles it incrementally without closing the connection. However, streaming negatively impacts most modern browsers. The advantage of

Comet is that there is always a link between the client and the server; the server pushes data to the client as soon as it becomes available. In contrast, WebSocket are:

- ✓ More efficient in real time: Messages are transmitted immediately as soon as they are available, without waiting or repeated requests.
- ✓ Bandwidth savings: No overhead associated with repetitive requests (as with polling). Only one connection is maintained.
- ✓ Low latency: The connection is established only once, then data flows as soon as it is ready.
- ✓ Bidirectional communication (full-duplex): The client and server can send and receive data simultaneously over the same connection.
- ✓ Single connection (a single socket): Reduces network and memory footprint on both the client and server sides.
- ✓ URL-based, but uses TCP/IP in the background: Combines the simplicity of the web (URL) with the robustness of TCP for data transmission.
- ✓ The WebSocket protocol is built on two distinct URI schemes: ws:// and wss://. The ws ://scheme, similar to HTTP for WebSocket, is used for insecure connections. In contrast, the wss (WebSocket Secure) scheme, which operates via TLS, is designed to provide a secure WebSocket connection, much like HTTPS for HTTP. This setup ensures the confidentiality and integrity of data exchanges.
- ✓ WebSocket was created to allow web browser applications to access TCP-based socket APIs (hence the name WebSocket).
- ✓ MQTT and WebSocket are complementary.
- ✓ WebSockets behave similarly to TCP sockets, so any protocol that works over TCP can also work over WebSockets.

4.11.2 The WebSocket Handshake

The expected data the server receives from a valid client must include several HTTP headers such as, Figure 23:

- ✓ Communication between client and server is bidirectional.
- ✓ WebSocket is ideal for real-time systems (e.g., chat, gaming, live updates).
- ✓ Enables full-duplex communication over a single TCP connection.
- ✓ The client always initiates the connection via a handshake.
- ✓ Connection termination can be initiated by either the client or the server.

✓ The header Sec-WebSocket-Accept indicates that the server has accepted the WebSocket connection.

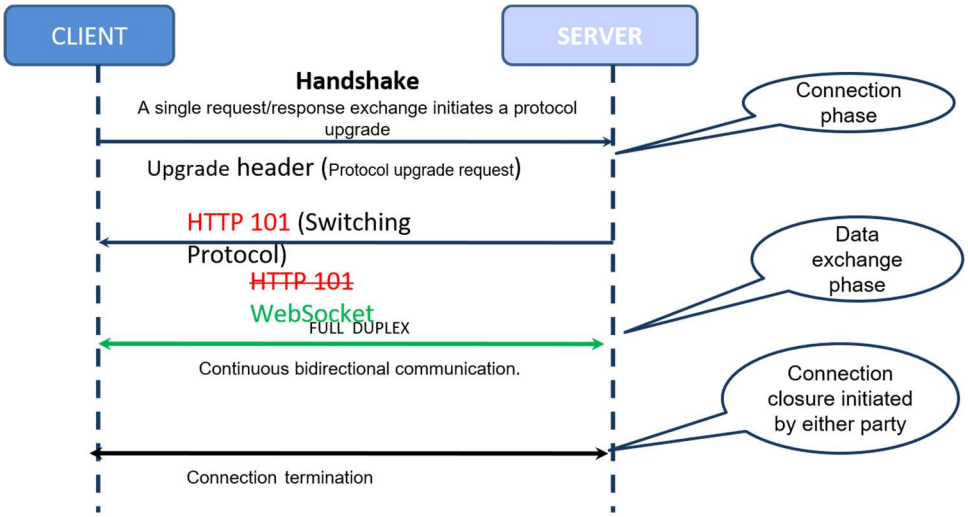


Figure 23 WebSocket: Connection Process

4.11.3 MQTT over WebSocket?

MQTT, known for its lightweight design and reliability in demanding environments, is now fully accessible via a web browser thanks to WebSocket, a real-time, bidirectional communication protocol. Integrating MQTT over WebSocket greatly facilitates interactions with connected objects by making the protocol directly accessible from a web browser, eliminating the need for specialised software. This enables simplified, universal, and real-time communication directly within web applications with IoT devices. It's a powerful solution for modern systems requiring responsiveness, efficiency, and web interoperability. Here are the main advantages[22]:

• **Real-time communication**: WebSocket maintains a persistent connection with low latency, ideal for instantaneous IoT data transmission. Direct interaction via browser: No need for dedicated MQTT clients.

• **Efficient use of resources**: MQTT remains lightweight, even via WebSocket, making it suitable for mobile devices and limited networks.

• **Bidirectionality**: WebSocket's full-duplex communication, combined with MQTT's pub/sub model, enables dynamic interactive exchanges.

[22] https://thingsboard.io/docs/mqtt-broker/user-guide/mqtt-over-ws/

• **Web compatibility**: WebSocket makes MQTT compatible with modern browsers, without plugins or complex configuration.

• **MQTT features retained:** LWT, persisted messages, clean sessions, etc. All MQTT functions are maintained. Efficiency: MQTT is lightweight, and WebSocket is widely supported in JavaScript. Enhanced visualisation: Browsers become a powerful interface for leveraging MQTT streams.

4.11.4 WebSocket Events

The WebSocket API is event-based. This section describes the four events your application code can listen for. We'll describe each event, explain how to handle real-world scenarios, and build examples based on what you learn.

WebSocket triggers four events, available through the JavaScript API and defined by the W3C:

- **open**

  When the WebSocket server accepts the connection request and the handshake is complete, the **open** event is triggered, and the connection is established. Once this occurs, the server is ready to send and receive messages from your client application:

  javascript

  CopierModifier

  ```
  // WebSocket connection established
  ws.onopen = function(e) {
     console.log("Connection established");
     ws.send(……);
  };
  ```

  From this handler, you can send messages to the server, display connection status on screen, and initiate **bidirectional communication**.

- **message**

  Once the connection to the WebSocket server is established, it can be used to **send and**

receive messages. The WebSocket API prepares the complete messages to be processed inside the **onmessage** handler:

javascript

CopierModifier

```
// WebSocket message handler
ws.onmessage = function(e) {
   var stocksData = JSON.parse(e.data);
   // Handle data
};
```

- **error**
  When a failure occurs for any reason, the handler attached to the **error** event is triggered. In many cases, it's assumed that the WebSocket connection will **close shortly after an error**, and a **close event** will also be fired.

Attributes like the error code and reason may help diagnose what went wrong. Here's an example of how to handle errors, and optionally attempt to reconnect to the WebSocket server:

javascript

CopierModifier

```
ws.onerror = function(e) {
   console.log("WebSocket failure, error", e);
   handleErrors(e);
};
```

- **close**
  (This event was listed but not described in the original text — would you like a full explanation of the close event and reconnection logic?)

4.11.5 The close Event in WebSocket

The close event is triggered when the WebSocket connection is closed, and the onerror callback will also be executed. You can manually trigger the onclose event by calling the close() method on a WebSocket object, which will terminate the connection with the server. Once the connection is closed, communication between the client and server will no longer continue.

In JavaScript, you listen for these events using either the on<eventname> handler or the addEventListener() method. Your code will provide a callback function that is executed whenever that event is triggered.

With WebSockets, there are only two methods: send and close. Once your connection is established, you're ready to begin sending (and receiving) messages to/from the WebSocket server. The client application can specify the type of data being transmitted, and several formats are supported — including string, binary, or JSON values.

Of course, sending data arbitrarily is not always appropriate. As discussed, WebSocket is event-driven, so you must ensure the connection is open and ready to send messages. You can do this in two main ways. One option is to check the readyState attribute to ensure that the WebSocket object is ready to receive messages:

```
javascript

CopierModifier

function processEvent(e) {

  if (ws.readyState === WebSocket.OPEN) {

    // Socket open, send!

    ws.send(e);

  } else {

    // Show an error, queue it for sending later, etc.

  }

}
```

For more technical details, refer to lab exercises and lecture notes.

4.12 HTTP 0.9

The birth of the World Wide Web gave rise to the earliest versions of the HyperText Transfer Protocol (HTTP). The first version of HTTP was conceived by Tim Berners-Lee alongside HTML (HyperText Markup Language).

HTTP 0.9 was incredibly simple. A client requested content using only the **GET** method:

http

CopierModifier

GET /index.html

The simplicity of HTTP 0.9 meant you could only request two things: plain text or HTML. This initial version had no headers, which meant there was no support for media types. Essentially, as a client, you would request a resource from the server over TCP, and once the server finished sending it, the connection was closed.

4.12.1 HTTP 1.0 and 1.1

The simplicity of version 0.9 didn't last long. In the next version of HTTP, the complexity of a request/response pair increased. Later versions added the ability to send HTTP headers with each request. With this growing set of headers, features such as POST requests (forms), media types, caching, and authentication were introduced in HTTP 1.0.

In HTTP 1.1, support for virtual hosting using the Host header, content negotiation, persistent connections, and chunked responses was added — features still used in modern production servers. As HTTP became more complex, the size of headers also grew.

4.12.1.1 WebSocket and HTTP Upgrade

One of the many benefits of the WebSocket protocol is that it starts its connection with a simple HTTP request. Browsers and clients that support WebSocket send a request to the server with specific headers asking to upgrade the connection to use WebSocket.

The Connection: Upgrade header was introduced in HTTP/1.1 to allow the client to notify the server of alternative communication methods. It is now primarily used to upgrade HTTP to WebSocket, but it can also be used to upgrade to HTTP/2.

According to the WebSocket specification, the only indication that a WebSocket connection has been accepted is the Sec-WebSocket-Accept header field. Its value is a hash of a predefined GUID and the client's Sec-WebSocket-Key HTTP header.

5 Chapter 4. Requirements and Challenges of IoT

In an increasingly connected world, the Internet of Things (IoT) is profoundly transforming the way we collect, analyse, and exploit data. However, for IoT systems to deliver their full potential, several technical and conceptual challenges must be addressed. Among these, the integration of data from heterogeneous sources, privacy protection, and semantic interoperability between devices are fundamental conditions for their effectiveness. At the same time, there is hope and optimism as artificial intelligence (AI) plays a central role in overcoming these challenges, particularly through machine and deep learning algorithms adapted to the constraints of IoT, such as TinyML for low-power objects. This synergy between AI and IoT paves the way for concrete and innovative use cases in many sectors. The following sections provide a detailed examination of these essential aspects.

5.1 Data Integration: Definition and Challenges

Data integration aims to gather, harmonise, and unify data from multiple and heterogeneous sources (databases, IoT sensors, files, cloud services, APIs, etc.). This process yields a coherent and actionable view for analysis, operational optimisation, and informed decision-making. In a complex digital environment, data is often dispersed, redundant, or of variable quality. Data integration helps break down silos, correct inconsistencies, and transform raw information into a strategic resource. Data integration is not limited to data ingestion. It covers the entire processing cycle, from analysis to visualisation and Business Intelligence (BI) workflows. It is therefore a central pillar of modern data engineering.

5.1.1 Main Objectives

- Centralise data dispersed in different formats.
- Improve the quality, consistency, and reliability of information.
- Facilitate analysis, visualisation, BI, and AI model training.
- Eliminate redundancies and resolve data conflicts. Data Source Types
- Structured: Relational databases (MySQL, Oracle, etc.)
- Semi-structured: Formats such as XML, JSON, CSV
- Unstructured: Text, images, videos, logs
- Real-time: Data from sensors, IoT streams, REST APIs

5.1.2 Data Sources from IoT Devices: A Strategic Lever for Integration.

Connected devices in the Internet of Things (IoT) continuously generate large volumes of data, often comprising several thousand data points per day. This data comes from a wide variety of devices: Medical sensors that monitor patients' health in real time, Smart home devices (thermostats, cameras, voice assistants) that optimise comfort and security, or industrial sensors (IIoT) deployed in factories or urban infrastructure to monitor production, maintenance, or energy management in smart cities. Most of this data is transmitted to the cloud or remote processing platforms, where it can be analysed, cross-referenced with other sources, and used for decision-making, predictive, or automated purposes. Regardless of the data format, its frequency of transmission, or its technological origin, the main challenge is to identify the most relevant sources for business objectives, then to define an effective integration strategy that takes into account both the technical constraints and the analytical opportunities they offer.

5.1.3 Key Steps in the Data Integration Process

According to IBM[23], data integration involves a series of successive steps that transform heterogeneous data into a coherent, unified, and usable resource. These steps combine technical processes, specialised tools, and governance strategies, which play a crucial role in maintaining control and management. This process ensures reliable and sustainable integration, which is essential for leveraging data in complex environments, such as IoT or Business Intelligence. It allows the transformation from dispersed raw data to strategic, ready-to-use information:

1. Source Identification: Identification of the various sources to be integrated: databases, spreadsheets, cloud services, APIs, legacy systems, sensors, etc.
2. Data Extraction: Retrieval of data from the identified sources, via queries, APIs, or file transfers.
3. Mapping and Alignment: Establishment of mappings between heterogeneous data schemas or structures to ensure content harmonisation.
4. Validation and Quality Checking the consistency, accuracy, and integrity of data to eliminate errors and duplicates.
5. Data Transformation: Cleaning, enrichment, standardisation, and conversion to a standard format suitable for analysis.
6. Data Loading: Importing the transformed data into a target platform (data warehouse, data lake, etc.), in batch or real-time mode.
7. Continuous Synchronisation: Regular or real-time updates of integrated data to ensure its freshness.
8. Governance and Security Enforcement of confidentiality rules, regulatory compliance (e.g., GDPR), and securing data flows and storage.

[23] https://www.ibm.com/fr-fr/topics/data-integration

9. Metadata Management: Data documentation (source, structure, meaning) to facilitate understanding, searchability, and use.
10. Access and Analysis: Providing data for visualisation, reporting, BI, or training artificial intelligence models.

There are several approaches to data integration, each with its advantages and limitations. Choosing the most appropriate method depends on various factors, including business objectives, existing technology infrastructure, performance expectations, and the organisation's budgetary constraints. The choice of integration method depends on the type of data, the urgency of processing, the complexity of the systems involved and the end use (analysis, decision-making, automation). A hybrid approach is often necessary in modern architectures.

1. ETL (Extract, Transform, Load)

   • Data transformation before storage.

   • Suitable for cases where quality, cleansing, and consistency are priorities.

   • Often uses an external staging area.

   • More rigorous, but may be slower than other methods.

2. ELT (Extract, Load, Transform)

   • Data loaded before transformation, directly into a modern warehouse.

   • Recommended for Big Data or real-time processing projects.

   • Takes advantage of the computing power of modern storage systems for better performance.

3. Real-Time Integration

   • Capture and process data as soon as it appears in source systems.

   • Ideal for fraud detection, monitoring, or continuous analytics. CDC (Change Data Capture): A variant of real-time integration that applies only changes detected in source systems to target systems.

4. Integration via API (Applications)

   • Ensures seamless data sharing between interconnected software applications

• Useful for ensuring interoperability (e.g., consistency between HR and financial systems).

5. Data Virtualisation

• Creation of a virtual layer that unifies access to data from multiple sources.

• Data is not moved: it is accessible on demand.

• Promotes agility and real-time access without duplication.

6. Federated Integration

• Data remains in its original systems.

• Queries are executed directly on the sources, without transfer.

• Reduces duplication, but may suffer from latency or performance reduction depending on the complexity of the sources.

5.1.4 Main Challenges of Data Integration

Data integration presents several technical and semantic challenges, including:

• The diversity of data formats and structures, from various sources (SQL databases, JSON, real-time streams, etc.).

• Resolving semantic ambiguities, such as the correspondence between equivalent but differently named fields (e.g., "Name" vs. "LastName").

• Managing real-time synchronisation, essential in dynamic or critical systems.

• Managing large volumes and rapid data flows, particularly in IoT or Big Data environments.

5.1.5 Examples of Concrete Applications

• IoT Platforms: Integration and correlation of data from sensors, surveillance cameras, and alert systems for centralised monitoring.

• Smart Cities: Cross-referencing data flows related to traffic, weather conditions, and pollution levels to optimise urban services.

• Healthcare: Aggregation of data from electronic medical records, physiological sensors, and clinical histories for personalised patient monitoring.

• E-commerce: Unification of data from CRM, purchase history, and social media interactions to refine segmentation and recommendations.

5.2 Autonomous and Intelligent Operation of IoT Devices

IoT devices often interact with software located elsewhere on the network and operate autonomously, without requiring human intervention. When combined with data analytics and machine learning, these devices can adopt proactive behaviour, detect relevant patterns, and provide users with recommendations to improve their quality of life (health, environment, finances, etc.). With billions of connected objects worldwide, securing the IoT is no longer an option, but a necessity. This requires a systematic, automated, and standardised approach, based on collaboration among manufacturers, service providers, and end-users. Without this, the risks of privacy breaches, network compromise, and destabilisation of critical systems will continue to grow.

2. Specific Risks and Vulnerabilities

Despite their advantages, IoT devices introduce a multitude of cybersecurity and privacy risks:

a. Device Diversity and Lack of Standards

• A multiplicity of operating systems, protocols, and environments makes the application of universal security standards difficult.

• The lack of a unified global framework leaves the door open to vulnerabilities that can be exploited on a large scale.

b. Weak Hardware Capabilities

• Many devices cannot support robust security features (such as encryption, updates, and authentication).

c. Communication Vulnerabilities

• Many devices exchange data in plain text, without encryption or authentication.

• Insecure updates can be intercepted or tampered with.

• Communications between devices on the same home network are often not isolated.

d. Persistence of Vulnerabilities

• Infrequent or non-existent software updates, either due to manufacturer negligence, consumer ignorance or inaction.

• Security issues can persist throughout the device's lifecycle.

e. Data Leaks

• Cloud leaks in the event of an attack or misconfiguration.

• Device-to-device leaks (e.g., unintentional monitoring of personal data on a shared home network).

f. Compromise and Malicious Use

• Compromised devices can be used to conduct attacks (botnets, spam, DDoS).

• These attacks affect network performance, shared services, and overall system stability.

5.2.1 Recommended Protection

Strategies Faced with these challenges, several measures should be implemented to ensure adequate cybersecurity for IoT systems:

a. Authentication and Access Control

• Multi-factor authentication, digital certificates, identity management (IAM/CIAM).

• Role-based access control (RBAC).

b. Systematic Data Encryption

• Encryption in transit and at rest, secure protocols, end-to-end encryption.

c. Regular Software Updates

• Centralised deployment via MDM (Mobile Device Management) or UEM (Unified Endpoint Management) solutions.

• Automated updates to limit human dependencies. d. Proactive Monitoring

• Use of EDR/XDR, IDS, and AI-based behavioural analysis tools.

e. Network Segmentation

• Isolation of IoT devices from critical networks via VLANs or subnets. f. Continuous Risk Assessment

• Penetration testing, regular security audits, and attack simulation to identify vulnerabilities before they are exploited.

g. Synergy between MDM and EDR

• Harmonisation of security policies, integrated incident detection, rapid response, and automated patching.

5.3 Sémantique interopérabilité pour l'IoT (RDF, OWL)

Semantic interoperability refers to the ability of heterogeneous systems, services, or devices to understand, interpret, and exchange data consistently, despite differences in format, vocabulary, or structure. In the context of the Internet of Things (IoT), this implies that sensors, platforms, objects, or applications—often designed by different vendors—can share a common understanding of the data being exchanged. To address this challenge, the use of ontologies plays a central role. An ontology is a formal representation of knowledge in the form of concepts, relationships, properties, and logical rules. It serves as a common semantic framework allowing systems to align their understanding of data, thus facilitating automated processing and interoperability. Objectives of semantic interoperability with ontologies: Giving meaning to IoT data: Going beyond the simple syntactic format to interpret the contextual meaning of data exchanged between connected objects. Structuring knowledge using ontologies: Defining semantic models to consistently describe objects, events, locations, or units of measurement. Automating data integration: Using ontologies to facilitate automatic reasoning, data fusion, and alignment between heterogeneous vocabularies. Dynamic service discovery: Using ontological

descriptions, enabling systems to dynamically discover, compose, and orchestrate IoT services based on context or environment. Reducing semantic ambiguity: Harmonising meanings (e.g., “ambient temperature” vs. “internal temperature”) to avoid misunderstandings or conflicts when interpreting data.

In the Internet of Things (IoT), semantic interoperability poses a significant challenge, as it enables heterogeneous devices to communicate effectively. Ontologies play a central role in overcoming the obstacles related to sensor diversity, data formats, and usage contexts. For example, in a smart home, temperature, motion, or energy consumption sensors may come from different manufacturers and generate data with disparate structures. The use of standardised ontologies such as SSN[24] (Semantic Sensor Network) or SOSA[25] (Sensor, Observation, Sample, and Actuator) provides a common representation of these sensors and their observations. In healthcare, connected medical devices must transmit vital data (heart rate, blood pressure, etc.) that various hospital systems can interpret. Thanks to semantic annotations in RDF[26]/OWL[27], data can be enriched with meaning, thus facilitating its automatic processing. OWL-based reasoning also makes it possible to automate the discovery of relevant services, such as triggering a medical alert or optimising a care procedure. Finally, when multiple systems or applications use different ontologies, alignment mechanisms ensure consistent knowledge fusion. Thus, ontologies are a crucial building block for creating an IoT ecosystem that is smart, interoperable, and user-oriented.

5.3.1 The Role of Semantic Technologies in the IoT

The Internet of Things (IoT) relies on a multitude of interconnected sensors and devices. Semantic technologies are essential for these elements to exchange data in a way that is understandable and usable by all systems. They provide a common foundation for structuring, linking, interpreting, and querying data, ensuring semantic interoperability between different connected objects. RDF (Resource Description Framework): Structuring Knowledge. RDF is a fundamental technology of the Semantic Web. It allows data to be represented as triples: subject – predicate – object. This format is simple yet extremely powerful for modelling facts and

[24] https://www.w3.org/TR/vocab-ssn/

[25] https://www.w3.org/TR/vocab-ssn/#SOSAObservableProperty

[26] https://www.w3.org/TR/rdf11-concepts/

[27] https://www.w3.org/TR/owl-features/

relationships between entities in a formal and interoperable manner. For example, in a smart home:

Sensor1 rdf:type :Temperature .

Sensor1 :measurementValue "22.4" .

These two triplets indicate that the Sensor1 object is a temperature sensor and that it measured a value of 22.4°C. RDF makes the data not only machine-readable but also compatible with other sources, facilitating the merging of information. OWL (Web Ontology Language): Enriching and Reasoning on Data Based on RDF. OWL allows us to go further by defining rich ontologies, which are semantic models that describe classes, relationships, properties, hierarchies, and more. OWL provides the ability to perform automatic reasoning, that is, to deduce new information from existing knowledge. Example in the healthcare sector: If a sensor measures a vital parameter (such as heart rate) and the value exceeds a certain critical threshold, OWL reasoning can automatically deduce that an emergency exists, triggering a medical alert without requiring human intervention.

5.3.2 SPARQL: Querying RDF Graphs

SPARQL is the standard query language for RDF. It allows for precise searches on semantic graphs, extracting only relevant information. This is essential in data-rich environments such as smart cities, connected healthcare, or residential energy management. For example, a SPARQL query can retrieve all temperature sensors that have detected excessive heat:

```
SELECT ?sensor ?value
WHERE {
        ?sensor rdf:type:Temperature .
       ?sensor:measurementValue ?value .
FILTER (?value > 30) }
```

Such a query could, for example, detect risks of overheating in a building or medical equipment. These three building blocks—RDF for structuring, OWL for knowledge formalisation and inference, and SPARQL for intelligent querying—constitute the basic infrastructure of

semantic IoT systems. They go beyond simple data exchange to provide automatic, dynamic, and contextualised understanding of information, essential in critical environments such as healthcare or smart homes.

Although SPARQL was originally designed as a query language for RDF, it is also widely used to query OWL ontologies, as OWL is built upon RDF. SPARQL, therefore, allows you to exploit not only OWL data but also OWL inferences if a reasoning engine is used (such as Apache Jena, OWLIM, Protegé + Pellet, etc.). Example of a SPARQL query on an OWL ontology. Consider the following example in a healthcare context: OWL ontology:

• Sensor is a class.

• Temperature is a subclass of :Sensor.

• measureValue is a property.

• CriticalSensor is a class defined as any sensor measuring a vital parameter and whose value exceeds a threshold.

If the inference engine has already classified some sensors as :CriticalSensor, they can be queried like this:

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX: <http://example.org/ontology#>

SELECT ?sensor ?value

WHERE { ?sensor rdf:type:CriticalSensor . ?sensor:ValueMeasurement ?value . }

What this query does:

• It retrieves all objects (?sensor) that have been inferred as belonging to the class : CriticalSensor using the OWL reasoner.

• It also extracts the value measured by each of them.

Note that, this query will only work if the OWL ontology has been previously enriched by inference (e.g., Pellet[28], HermiT[29]) to infer that certain sensors are critical automatically.

[28] https://github.com/stardog-union/pellet

SPARQL doesn't do this reasoning on its own; it queries what the OWL engine has already inferred.

5.3.3 Description Logic

Another DL (Description Logic) query that queries an OWL ontology in a declarative formalism, often used in tools like Protégé (DL Query tab). In Protégé, "DL Query" menu, they allow you to check whether the axioms of the OWL ontology allow the expected inferences, using an engine like HermiT or Pellet.

Ontology Setup (OWL):

Sensor ⊑ ∃measuresValue.Data

CriticalSensor ≡ Sensor ⊓ ∃measuresParameter.VitalParameter ⊓

∃measuresValue.xsd:float[≥ 39.0]

(Or, using a simpler syntax without numerical value constraints, we assume that individuals have already been classified.)

DL query 1

CriticalSensor

This will return all individuals inferred to belong to the class CriticalSensor

DL query 2

Sensor ⊓ ∃measuresParameter.VitalParameter

All sensors that have measured temperature (assuming Temperature ⊑ Parameter):

Sensor ⊓ ∃measuresParameter.Temperature

Despite the undeniable advantages of semantic technologies in the Internet of Things (IoT), several significant challenges persist. First, the complexity of OWL ontologies, with their rich and expressive structures, makes their design, maintenance, and use challenging for

[29] http://www.hermit-reasoner.com/

non-experts. Second, there is still no universal standardisation covering all types of IoT objects and sensors, which limits interoperability between heterogeneous systems. Furthermore, semantic annotation of real-time data streams remains a costly task, both in terms of human resources and automatic processing. Finally, the performance of reasoning engines is often tested when faced with massive volumes of continuously generated data, which poses concrete limits to large-scale semantic inference. Overcoming these challenges is crucial to fully realise the potential of semantic IoT in critical domains, such as healthcare, smart homes, and manufacturing.

5.3.4 Practical Examples of AI & IoT Implementation

1. **Smart Home – Activity Recognition**

   Connected devices (bracelets, blood pressure monitors, thermometers) send health data in real time. AI analyzes this data to detect anomalies (e.g., persistent fever, irregular heart rate) and automatically alert medical staff or relatives.

   → Example: *Embedded early diagnosis systems for elderly patients..*

2. **Smart Agriculture – Plant Disease Detection**

   IoT sensors collect data on soil moisture, temperature, and leaf condition (via cameras). Computer vision algorithms identify diseases or deficiencies from images.

   → Example: *Embedded systems in agricultural drones.*

3. **Industry 4.0 – Predictive Maintenance**

   Industrial machines are equipped with IoT sensors measuring vibrations, noise, temperature, and pressure. AI analyzes variations to predict failures before they occur, thus avoiding production downtime.

   → Example: *Early detection of engine failures in factories.*

4. **Energy Management – Optimization via Machine Learning**

   Sensors measure energy consumption room by room in a building. AI learns usage patterns and proposes reduction strategies (automatic shutdown of equipment, forecasting consumption peaks).

   → Example: *Smart grids with embedded AI (TinyML) for local control.*

5. **Energy Management – Optimization via Machine Learning**

   Sensors measure energy consumption room by room in a building. AI learns usage patterns and proposes reduction strategies (automatic shutdown of devices, forecasting

consumption peaks).

→ Example: *Smart grids with embedded AI (TinyML) for local control.*

Bibliography

1. Pethuru Raj, Anupama C. Raman, The Internet of Things: Enabling Technologies, Platforms, and Use Cases, CRC Press, 2017.
2. Seifeddine Messaoud, Abbas Bradai, Syed Hashim Raza Bukhari, Pham Tran Anh Quang, Olfa Ben Ahmed, Mohamed Atri: A survey on machine learning in Internet of Things: Algorithms, strategies, and applications. Internet Things 12: 100314 (2020)
3. Elias Dritsas, Maria Trigka: A Survey on Cybersecurity in IoT. Future Internet 17(1): 30 (2025)
4. Enabling the Internet of Things: Fundamentals, Design and Applications Éditeur : IEEE Press / Wiley, Année : 2021 ISBN : 978-1-119-67158-4
5. Song, Glenn A. Fink, Sabina Jeschke Security and Privacy in Cyber-Physical Systems: Foundations and Applications, Éditeur : Wiley-IEEE Press Année : 2017, ISBN : 978-1-119-22060-7
6. Houbing Song, Ravi Srinivasan, Tamim Sookoor, Sabina Jeschke, Smart Cities: Foundations, Principles and Applications, : Éditeur : Wiley Année : 2017 ISBN : 978-1-119-22039-3
7. Olivier Hersent , L'Internet des objets – Les principaux protocoles M2M et leur évolution
8. David Hanes, Gonzalo Salgueiro, Patrick Grossetete, Robert Barton, Jerome Henry IoT Fundamentals: Networking Technologies, Protocols, and Use Cases, Éditeur : Cisco Press, 2017
9. A Cirani, S., A Ferrari, G., A Picone, M., A Veltri, L T Internet of Things: Architectures, Protocols and Standards, 2018.
10. Ripal Ranpara: A semantic and ontology-based framework for enhancing interoperability and automation in IoT systems. Discov. Internet Things 5(1): 22 (2025)
11. Lynda Djakhdjakha, Brahim Farou, Hamid Seridi, Hamadoun Cisse: Towards a semantic structure for classifying IoT agriculture sensor datasets : An approach based on machine learning and web semantic technologies. J. King Saud Univ. Comput. Inf. Sci. 35(8): 101700 (2023)