Introducing Recent Electrical Engineering Developments into undErgraduate cuRriculum

Co-funded by the Erasmus+ Programme of the European Union

Reference No.: IREEDER-D2.2
Date: June. 15, 2021
Version: v2.0

DELIVERABLE 2.2

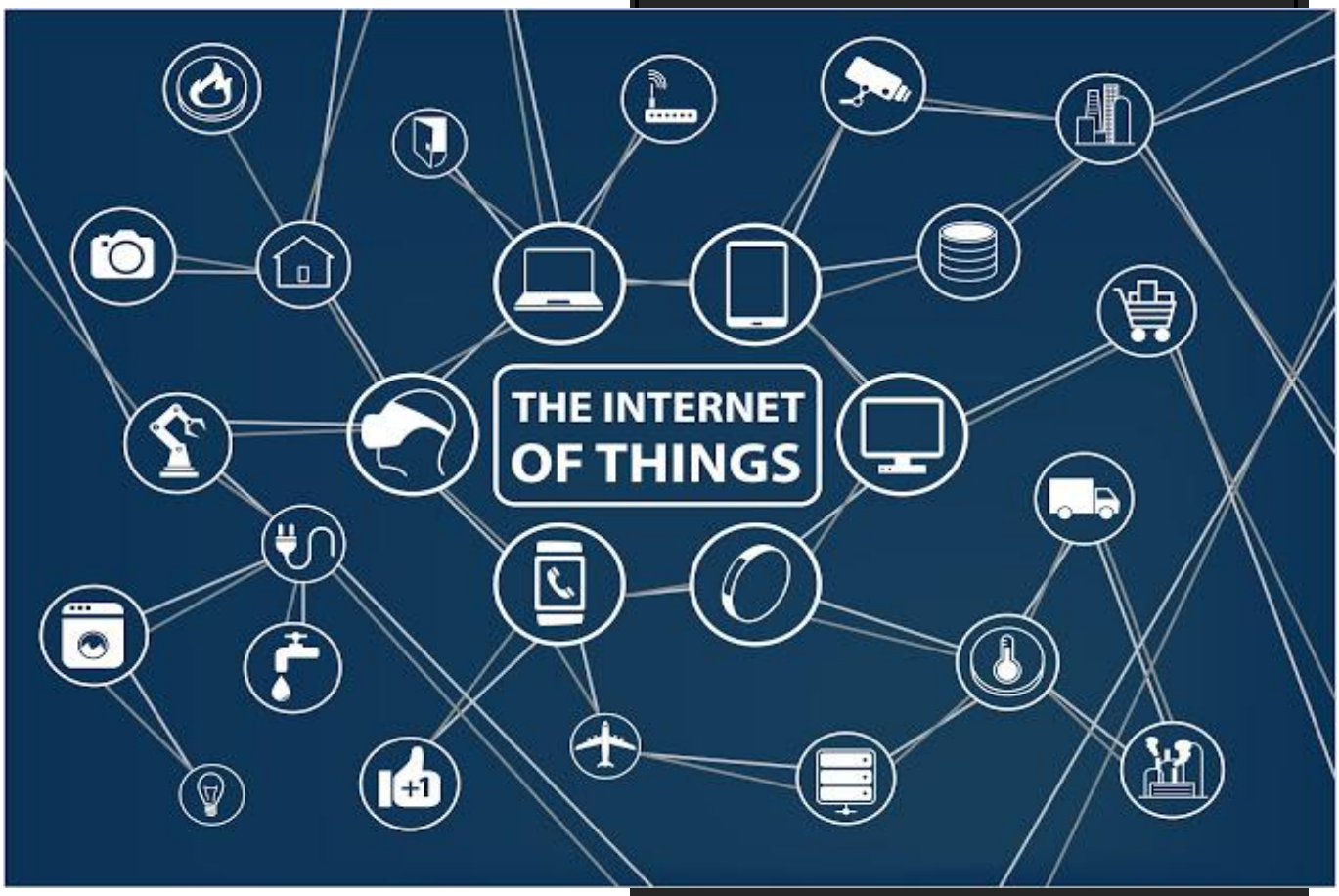
Teaching Materials - Internet of Things Lecture Notes

Written by	**Responsibility**
Marios Raspopoulos (UCLAN)	WP2 Leader
Nearchos Paspallis (UCLAN)	Member
Stelios Ioannou (UCLAN)	Member
Josephina Antoniou (UCLAN)	Member
Eliana Stavrou (UCLAN)	Member
Fabrizio Granelli (UNINT)	Member
Claudio Sacchi (UNINT)	Member
Omar R Daoud (PU)	Member
Mohammed Bani Younis (PU)	Member
Saleh Saraireh (PU)	Member
Rasha Gh. Freehat (PU)	Member
Jonathan Rodriguez (IT)	WP5 Leader
Georgios Mantas (IT)	Member
Maria Papaioannou (IT)	Member
Claudia Barbosa (IT)	Member
Felipe Gil-Castiñeira (UVIGO)	WP4 Leader
Cristina López-Bravo (UVIGO)	Member
René Lastra Cid (UVIGO)	Member
Saud Althunibat (AHU)	Project Coordinator
Moath Safasfeh (AHU)	Member
Samiha Falahat (AHU)	Member
Edited by	
Marios Raspopoulos (UCLAN)	WP2 Leader
Approved by	
Saud Althunibat (AHU)	Project Coordinator

This publication was produced with the financial support of the European Union. Its contents are the sole responsibility of the partners of IREEDER project and do not necessarily reflect the views of the European Union

Introducing Recent Electrical Engineering Developments into undErgraduate cuRriculum

Introduction to the Internet of Things



Editor: Marios Raspopoulos

UClan Cyprus

June 2021

Co-funded by the Erasmus+ Programme of the European Union

This publication was produced with the financial support of the European Union. Its contents are the sole responsibility of the partners of IREEDER project and do not necessarily reflect the views of the European Union

Abstract

This document serves as a comprehensive handbook for a course named "***Introduction on the Internet of Things***". The course has been prepared within the scope of the iREEDER project which is funded by Erasmus+ (Call: Capacity Building in the field of Higher Education, Project No. 609971-EPP-1-2019-JO-EPPKA1-CBHE-JP (2019-1975/001-001)). The course aims to present the fundamental principles and architecture of IoT, discuss, examine, and evaluate the key technological components underpinning IoT, learn how to practically Design, Code and Build IoT solutions and review the key technological applications of IoT. The specific learning outcomes are:

1. Understand the definitions, operating principles, components and use of IoT Systems.
2. Demonstrate advanced knowledge about the architecture, the key technologies and protocols/standards used in IoT Systems.
3. Analyse and effectively use available frameworks/platforms to design, program, and implement IoT systems.
4. Explore the relationship between IoT, cloud computing, and big data and be able to identify necessary security measures.
5. Appraise the applicability of IoT in various engineering/business contexts and discuss future challenges of IoT in various sectors.

Chapter 1 serves as a introductory overview of the IoT. It defines the term IoT and, reviews the history and overviews the key-enabling technologies. It summarizes the main applications of IoT and identifies the key research directions and connections. **Chapter 2** serves as a revision of Basic Programming covering topics like variables, conditional statements, looping functions, I/O and presents the IoT IDE. **Chapter 3** is about Software Development for IoT Embedded Systems. It covers embedded programming in C including flow control, function decomposition, data representation and structures. **Chapters 4 and 5** present IoT architectures and the main IoT components. There is also a reference to Cyber-Physical systems, smart devices, and basicon storage and CPU, data movement, fetch-execute, accelerators, input/output inc. SPI/I2C, peripherals. There also an overview of the Embedded device memory architecture; SRAM, DRAM, Flash etc. **Chapter 6** is about IoT

Microcontrollers, Sensors for Data Acquisition and Actuators. **Chapter 7** is about IoT Connectivity Technologies presenting the main Wireless technologies for the IoT (WiFi, Bluetooth, Zigbee, 6LowPAN, LoraWAN, etc.), Wireless sensor networks (Z-wave etc.) and mobile Technologies (4G, 5G). **Chapter 8** presents the main IoT Connectivity Protocols. These include edge connectivity and protocols, Network and Data Protocols and data transmission using IoT protocols (e.g. MQTT). **Chapter 9** is about Data Storage and Cloud Systems whereas **chapter 10** presents Data analysis and applications used in IoT. **Chapter 11** overviews the principles of IoT Security and various security measures and standards proposed. **Chapter 12** covers the ethical aspects in IoT overviewing topics related to data ownership, data protection, trust, transparency etc. Finaly, **chapter 13** overviews some key-enabling technologies and Appications in IoT like Identification, Mobility, Positioning/Localization and also topics on how power up the IoT likeEnergy Harvesting, Battery Life Optimisation etc.

Table of Contents

1 Introduction to IoT

Author(s): Fabrizio Granelli

1.1 Introduction

1.1.1 What is the Internet of Things?

Today, strong interest by researchers and companies in several fields is being captured by the so-called “Internet of Things”, as enabling technology to embed the Internet into the “real world”. Applications such as smart city, smart grid, smart home would not be possible without the Internet of Things.

From a technical viewpoint, the IoT technology or Internet of Things is a network of connected things which can communicate data without human involvement, such as sensors, smart appliances, electronics, and machines.

For a more precise definition, the ITU "Internet of Things Global Standards Initiative" defines the Internet of things (IoT) as “*a system of interrelated computing devices, mechanical and digital machines provided with unique identifiers (UIDs) and the ability to transfer data over a network without requiring human-to-human or human-to-computer interaction*”.

However, the potential of IoT goes much beyond simply connecting objects, as today the Internet of Things is not the "hype" or a "buzzword" anymore, but it has the power to change our world and lives.

Indeed, regardless of the actual number of connected devices (Gartner estimated 50B devices connected by 2020), there is a concrete indication that IoT will play an important role in our everyday life.

An interesting aspect of the Internet of Things is that, while such technology is already being deployed and partially already present in our homes (see for example, Google Nest, or Amazon Alexa), still it presents several challenges involving research and development that should be addressed in order to consolidate IoT as a permanent building block of the Society of the Future.

This course is intended to provide an introduction to the Internet of Things, and it is intended to describe the basic and applications of the IoT technology.

1.1.2 IoT History

The Internet of Things does not represent a novel paradigm of the last few year. The concept of connecting objects to the Internet goes back in time to the early days of the Internet, which can to some extent be considered as the first days of the Internet of Things, too. Indeed, in 1982, Carnegie Mellon University became the host of the first Internet-connected appliance when a modified Coca-Cola vending machine was connected to the network. In such first case, the idea was to connect the vending machine to the Internet in order to enable the machine to report its inventory as well as loaded drinks temperature through the Internet (see Figure 1-1).

Figure 1-1: First object connected to the Internet (Source: CMU).

Later on, in the '90s, a famous Mark Weiser's paper on ubiquitous computing, "The Computer of the 21st Century", as well as academic works presented at conferences such as PerCom and UbiComp, produced a contemporary vision of the IoT. Meanwhile, several companies gained interest in the IoT and started building IoT-inspired solutions. We can mention in this area Microsoft's at Work or Novell's NEST.

The IoT field gained momentum in 1999, when, during the World Economic Forum at Davos, Bill Joy presented device-to-device communication as a part of his "Six Webs" framework. In the same year, Kevin Ashton of Procter & Gamble introduced the term "Internet of things", though he declared to prefer the sentences "Internet for things".

In such period, specific attention was focused on Radio-frequency identification (RFID) technologies. Indeed, RFID was considered an essential building block for the Internet of things, as it was considered the most appropriate solution to allow computers to recognize and interact with real-world objects.

Some experts define the Internet of things as "*simply the point in time when more 'things' or 'objects' were connected to the Internet than people*". By using such definition, it is possible to estimate that IoT was "born" between 2008 and 2009, since data from Cisco Systems shows the connected things/people ratio growing from 0.08 in 2003 to 1.84 in 2010.

This trend is continuing also in the recent years, with an estimate of total connected devices to the Internet approaching 500 billions in 2022.

1.1.3 IoT Facts

In order to try to describe the IoT scenario, in the following some relevant facts about the Internet of Things are reported:

- The number of devices connected to the Internet was 7 billion in 2018, and it is expected to approach 10 billion by the end of 2020. [1] Other forecasts predict up to 500 billion connected devices by 2022.
- Other forecasts predict a larger amount of "connected things" by 2020. Indeed, following an analysis by Gartner, over 14 billion devices were connected by the end of 2019, and the forecast is over 25 billion by the end of 2021. [2]
- At the moment, it seems that the large majority of IoT devices is consistituted by smartphones. Indeed, the total number of smartphone users reached the level of 3 billion in 2018 (source: Newzoo). [3]
- Similar analysis underlined that the majority of IoT devices that are currently connected are typically located at home or at work. More than half of all IoT devices were connected to Wireless Personal Area Networks in 2018 (Bluetooth, Zigbee, etc.). [1]
- Expectations on the potential market of the Internet of Things are extremely high. It is estimated that the global market of the Internet of Things was over $150 billion in 2018 and it is expected to exceed $1.5 trillion by 2025. [1]

1.1.4 IoT Penetration

A relevant question in order to understand the relevance and dimension of the IoT World is the penetration in terms of connected devices/people worldwide. The following figure presents an analysis by IoT Analytics Research showing clearly that, while the number of non-IoT connections is expected to grow in a slow linear manner, IoT connections are expected to show a more aggressive exponential growth.

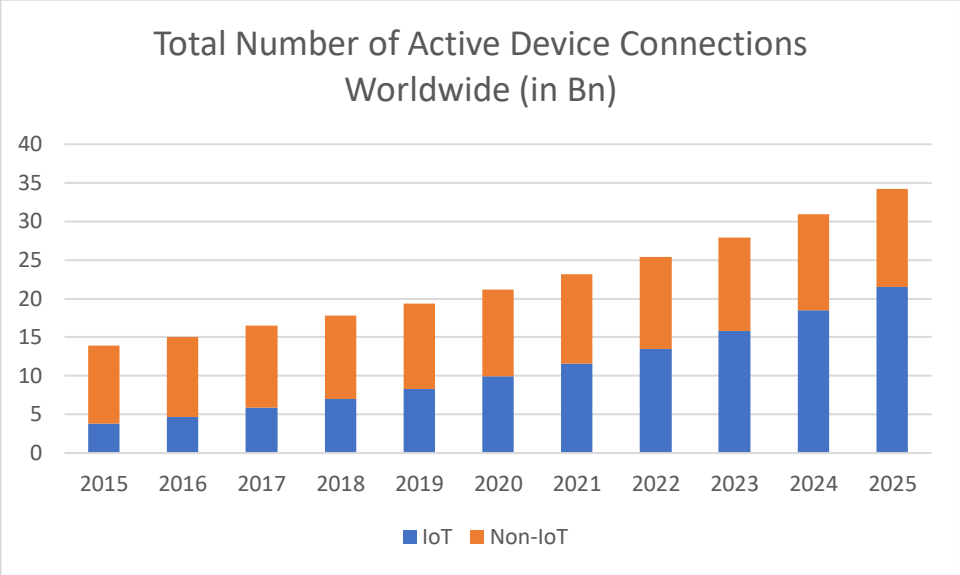


Figure 1-2: Active device connections worldwide (Source data from IoT Analytics).

1.1.5 What happens today?

Is IoT already a reality? In some cases, it is. For example, **Error! Reference source not found.** shows the percentage of connected devices in several application scenarios. The majority of patient monitoring systems are today connected, and more than half of energy meters are capable of automating the energy consumption measurements. This scenario depends of course on the geographical area and level of development of a country. As an example, Italy implemented a large-scale deployment of smart meters in the early 2000s. 36.7 million meters were deployed between 2001 and 2011, accounting for about 86% of the points of delivery.

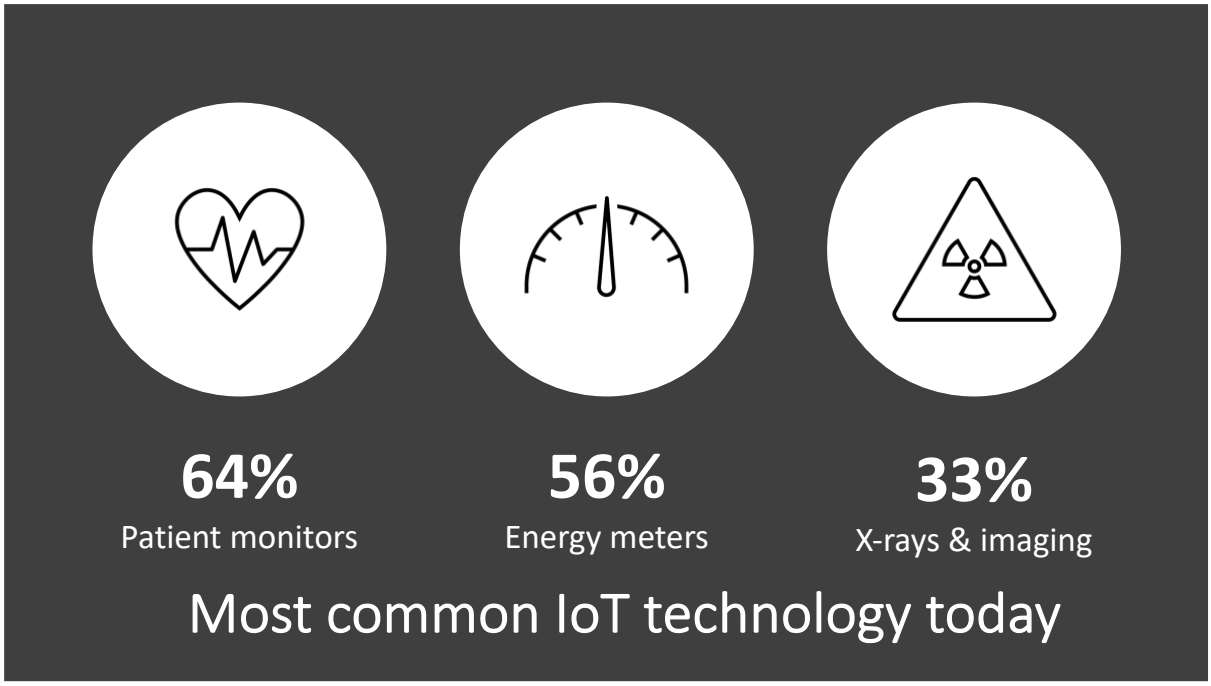


Figure 1-3: Number of connected devices in different application scenarios.

1.1.6 Applications

The Internet of Things represents one of the markets with biggest potentials nowadays, since it impacts on several vertical areas of application, ranging from manufacturing to healthcare, from transportation to agriculture. The following figures provides a breakdown of value-add by IoT to different sectors. More details on the main fields of application of IoT are illustrated in Section 1.3.

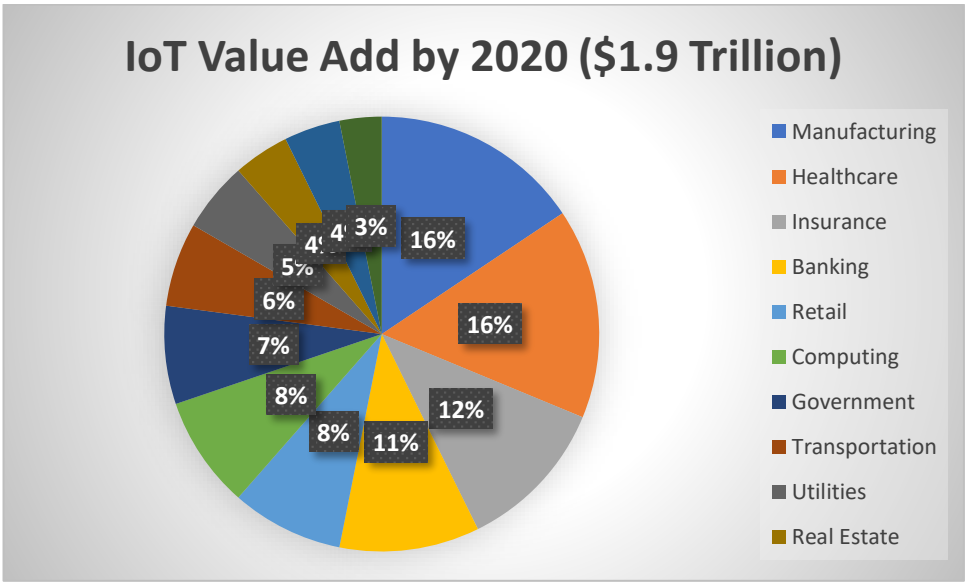


Figure 1-4: Value-add by IoT to different services in 2020 (based on data from Gartner).

1.2 Enabling technologies for IoT

The Internet of Things is in some texts defined as a system of systems. Indeed, IoT is not a single technology, but more specifically an eco-system, which is constituted by the integration many technologies. A crucial aspect of the Internet of Things is the network used to communicate between devices of an IoT installation. Several available wireless or wired technologies may fulfil such role. However, this is only one of the several technologies that are integrated by the IoT.

The next sub-sections summarize the major functionalities and technologies required by IoT.

1.2.1 Addressability

In order to identify and reach the connected objects, addressability represents the key technology required to enable the Internet of Things. The original idea was to exploit the Auto-ID Center, based on RFID-tags, and distinct identification through the Electronic Product Code.

This has evolved in modern IoT into objects having an IP address or URI (Universal Resource Identifier).

1.2.2 Application Layer

An application layer protocol and supporting framework for implementing IoT applications is required. The purpose of the application layer would be to offer a suitable interface to interface with the connected objects and devices, and to enable efficient access to information on their status and functionalities.

Examples of protocols operating on the application layer of IoT are Auto Discovery Resource Control (ADRC) and MQTT (Message Queue Telemetry Transport - ISO/IEC PRF 20922).

1.2.3 Communication Technologies

Connectivity represents a building block to build IoT services. In this section we will review some of the more interesting and diffused communication technologies for the Internet of Things.

1.2.3.1 Short-range wireless

Communication technologies enabling connectivity in short-range (1cm-100m) are those typically used for building Wireless Personal Area Networks (WPANs) or Wireless Local Area Networks (WLANs). The most diffused technologies include:

- Bluetooth – The standard provides a specification that enables to implement a mesh networking variant to Bluetooth low energy (BLE). Such technology allows to interconnect with increased number of nodes via device-to-device communication and offers a standardized application layer.
- Light-Fidelity (Li-Fi) – Li-Fi proposes the usage of visible light communication as an alternative wireless communication technology with respect to the Wi-Fi standard. Li-Fi offers increased bandwidth, but it provides a shorter range and requires line-of-sight.
- Near-field communication (NFC) – NFC offers protocols designed to implement communication between two electronic devices within a few centimeters range.
- Radio-frequency identification (RFID) – RFID is based on the usage of tags embedded in real life objects. By using electromagnetic fields RFID technology allows to read digital data stored in such tags.
- Wi-Fi – Common technology used for implementing wireless local area networking based on the IEEE 802.11 standard. Wi-Fi devices may communicate through a shared access point or directly via device-to-device communication.
- ZigBee – A wireless personal area networking solution based on the IEEE 802.15.4 standard. Zigbee is designed to provide low power consumption, low data rate, and low cost.
- Z-Wave – It represents a wireless communication protocol which is used primarily for home automation and security applications.

1.2.3.2 Medium-range wireless

Medium-range wireless connectivity is typically offered by Wide Area Networking Technologies. In the case of wireless communications, WANs are dominated by cellular networks, currently implemented by means of:

- LTE-Advanced – LTE represents the most diffused cellular technology nowadays, being the 4th Generation of cellular networks standardized by 3GPP. LTE offers broadbad access to the Internet for mobile users and devices. In particular, LTE-Advanced provides additional improvements, enabling extended coverage, higher throughput, and lower latency.
- 5G – The 5th Generation of 3GPP standards is expected to address novel requirements, including the capability to connect a large number of IoT devices, even mobile ones. 5G usage scenarios include massive Machine Type Communications (mMTC) and Ultra-Reliable Low-Latency Communications (URLLC), which might be adequate to support different IoT applications.

1.2.3.3 Long-range wireless

Larger geographical areas can be covered by using communication technologies fine-tuned to the requirements of the Internet of Things, and in particular:

- Low-power wide-area networking (LPWAN) – Low power WAN solutions represent long-range communication technologies that target low data rate and reduced power and costs. Examples of available LPWAN technologies include: LoRaWan, Sigfox, NB-IoT, Weightless, RPMA.
- Very small aperture terminal (VSAT) – VSATs and other similar solutions provide narrowband and broadband connectivity by exploiting satellite communications. This allows for greater coverage, also in case of lack of terrestrial infrastructure in the considered geographical areas.

1.2.3.4 Wired Technologies

As an alternative to wireless connections, wired connections can be used in cases where the objects are installed in pre-defined and fixed positions. Most common communication technologies include:

- Ethernet – Ethernet represents an extremely successful wired technology. Ethernet was originally designed as a Local Area Network technology operating on co-axial cables or twisted pairs, but it later extended to a wide range of applications with the introduction of optical communications, including also MANs/WANs.

- Power-line communication (PLC) – Power-line communications are based on the concept to exploit electrical wires to carry both power and data. This allows the exploitation of the power lines as a communication medium, leading to standards such as HomePlug or G.hn for potential utilization of PLC in networking IoT devices.

1.2.4 Standards and Standards Organizations

Given the complexity of the IoT architecture and services, several standards and standards organizations are involved. The following list provide the main bodies involved in IoT standards:

- Auto-ID Labs - Auto Identification Center
- EPCglobal - Electronic Product code Technology
- Government bodies, e.g. FDA - U.S. Food and Drug Administration
- GS1 - Global Standards One
- IEEE - Institute of Electrical and Electronics Engineers
- IETF - Internet Engineering Task Force
- ISO/IEC - International Organization for Standardization/ International Electrotechnical Commission (ITU-T M 3000)
- MTConnect Institute
- O-DF - Open Data Format
- O-MI - Open Messaging Interface
- OCF - Open Connectivity Foundation
- OMA - Open Mobile Alliance
- XSF - XMPP Standards Foundation

1.3 IoT vertical applications

This section provides a short review of the most relevant vertical applications of IoT. In general, several IoT devices are designed for consumer use. Those include connected vehicles, sensors/actuators for home automation, wearable items, connected health devices, and appliances with remote monitoring capabilities.

However, IoT applications go much beyond such scenarios.

As a consequence, the following is a brief classification of the main IoT vertical applications, including services for consumers, organizations, industry, infrastructure, and military applications.

1.3.1 Consumer Applications

1.3.1.1 Smart Home

In the smart home concept, IoT devices only represent a part of the larger system designed for home automation. Home automation involve lighting system, heating and air conditioning, media delivery and security systems. Home automation systems enable to achieve energy savings (by automatically ensuring lights and electronics are turned off) and remote usage of home appliances.

1.3.1.2 Elder care

One key application of a smart home or most in general of the Internet of Things is to support quality of life for elder people. Smart home services might be used to provide assistance for those with disabilities and elder people. In those scenarios, the connected home systems will use assistive technology to accommodate an owner's specific disabilities. This will include the use of different kind of sensors for movement, video, vital signs acquisition, etc.

1.3.2 Organizational Applications

1.3.2.1 Medical and healthcare

Clearly, the integration of the IoT in medical and health-related scenario represents an extremely relevant vertical application. This leads to the concept of the Internet of Medical Things (IoMT), which focuses on data collection and analysis for e-heath, and remote patient monitoring.

1.3.2.2 Transportation

The Internet of Things can be successfully applied also in the field of transportation. In this case, it can enable the integration of communications, control, and information processing across various transportation systems. Typical applications include fleet management, transportation systems monitoring and auditing.

1.3.2.3 V2X communications

Vehicular communications might benefit from the integration with the Internet of Things. In this scenario, the vertical application is typically defined as vehicle-to-everything (V2X). V2X includes vehicle to vehicle communication (V2V), vehicle to infrastructure communication (V2I) and vehicle to pedestrian communications (V2P). For example, V2X represents a relevant building block towards connected autonomous driving and smart road infrastructure.

1.3.2.4 Building and home automation

The concept of smart home can be generalized and applied to a wider range of scenarios, that belong to the building and home automation vertical applications. Those application use IoT technology to monitor and control the mechanical, electrical and electronic systems used across different types of buildings (e.g., public and private, industrial, institutions, or residential).

1.3.3 Industrial Applications

Industrial vertical applications belong to the so-called Industrial Internet of Things (IIoT). In the IIoT, IoT devices are used to regulate and monitor industrial systems. The goal is to gather and analyze data from connected devices, locations and people in order to improve industrial processes and enabling simpler co-existence of human and robots.

1.3.3.1 Manufacturing

The application of IoT to manufacturing is based on the integration of the IoT with manufacturing devices, leading to an integrated and smart cyber-physical space. Seamless integration of manufacturing devices equipped with sensing, identification, processing, communication, actuation, and networking capabilities leads to the definition of new businesses and market opportunities.

1.3.3.2 Agriculture

Precision farming is enabled by the integration of the IoT in farming processes. This scenario includes applications such as collecting data on temperature, rainfall, humidity, wind speed, pest infestation, and soil content. The objective is to automate farming techniques when possible, to enable informed decisions to improve quality and quantity of the results, minimize risk and waste, and reduce effort required to manage crops. Sample applications

include farmers being able to monitor soil temperature and moisture from afar, and to apply IoT-acquired data to precision fertilization programs.

1.3.4 Infrastructure Applications

In this framework, the term "infrastructure" is used to represent urban and rural infrastructures, such as bridges, railways tracks, wind farms, etc. The usage of IoT infrastructure can support monitoring of such critical infrastructure for detecting changes in the structural conditions, and enabling to decrease associated risks and increase safety. Usage of IoT devices is expected to provide benefits in the monitoring and operating infrastructure. As a result, incident management and emergency response coordination should be improved, as well as the quality of service, up-times and reduced costs of operation in all infrastructure related areas.

1.3.4.1 Metropolitan scale deployments / Smart City

Smart city represents one of the most relevant applications of IoT concepts. A smart city is a modern metropolitan area, planned to be wired and automated. By employing large IoT deployments, it will be possible to reduce human intervention and allow a smart city to improve the quality of life of the citizen, achieve better management and sustainability, etc. Some smart city examples are being proposed, such as Songdo, South Korea, the first of its kind fully equipped and wired smart city, with approximately 70 percent of the business district completed as of June 2018.

1.3.4.2 Energy management / Smart Grid

The Internet of Things provides significant support to energy management, not only as a provider for Internet connectivity but also in terms of tools to balance and optimize energy consumption. Indeed, several energy-consuming devices (e.g. lamps, household appliances, motors, pumps, etc.) already integrate Internet connectivity.

The IoT technologies also represent an enabler for the Smart Grid, which provides improved management of electricity by incorporating Internet technologies in a utility-side application scenario.

1.3.4.3 Environmental monitoring

Environmental monitoring represents a direct application of the Internet of Things. In environmental monitoring networked sensors are employed to assist in monitoring air or water quality, atmospheric or soil conditions, including remote areas where it might be important to monitor the movements of wildlife and their habitats.

1.3.4.4 Living Lab

A particular example of infrastructure applications includes the notion of Living Lab. The Living Lab concept is related to the integration of the research and innovation processes, with the idea to bring innovation to the people or territory. In several cases, the integration of IoT represents an enabler for this kind of activities.

1.3.5 Military Applications

IoT technologies can be integrated in the military domain for objectives related to reconnaissance, surveillance, and other battlefield objectives. This leads to the definition of the Internet of Military Things (IoMT).

IoMT involves the use of sensors, ammo, vehicles, robots, human-wearable biometrics, and other smart technology for application and deployment on the battlefield.

1.3.5.1 Internet of Battlefield Things

The Internet of Battlefield Things (IoBT) represents an instance of the Internet of Military Things. It is a project by the U.S. Army Research Laboratory (ARL) that focuses on the application of IoT to enhance the capabilities of Army soldiers.

1.3.5.2 Ocean of Things

Similarily, the Ocean of Things is project led by DARPA to design an Internet of Things solution for oceanic areas. The project is designed to establish an Internet of Things across large ocean areas for the purposes of collecting, monitoring, and analyzing environmental and vessel activity data.

1.4 Identification of key research directions and connections

1.4.1 Trends and Characteristics

This section describes the trends and characteristics of the Internet of Things. Indeed, the major trend of the IoT in the last years was represented by the exponential growth of devices connected to the Internet and controllable from the Internet.

On one side, the broad range of IoT applications described before underline a common trend in the heterogeneity of devices belonging to the IoT eco-system. Indeed, IoT devices can be extremely diverse in terms of specifics. However, most of them share some of the basic characteristics described in the following sub-sections.

1.4.1.1 Intelligence

Ongoing research is focusing on integrating the concept of IoT with the concept of autonomous control. This vision is steering towards considering the connected objects as a key aspect for the design of an autonomous IoT.

Considering the architecture of the Internet of Things, intelligence can be deployed in three different levels: IoT devices, Edge/Fog nodes, and Cloud computing. Intelligence can be concentrated in one location or distributed, with the basic principle that positioning control and decision-making at any level depends on the impact on the time sensitiveness of the corresponding IoT applications.

As an example, we can consider a camera used for real-time obstacle detection in the framework of autonomous vehicles. Considering the time constraints required for safety, it is most appropriate to perform the prediction/recognition of obstacles locally on the vehicle, rather than sending a huge amount of data to the Cloud and wait for the predictions back to the vehicle.

1.4.1.2 Architecture

We can simplify the architecture of the Internet of Things by defining three tiers:

- Tier 1: Devices/Things
- Tier 2: the Edge Gateway
- Tier 3: the Cloud.

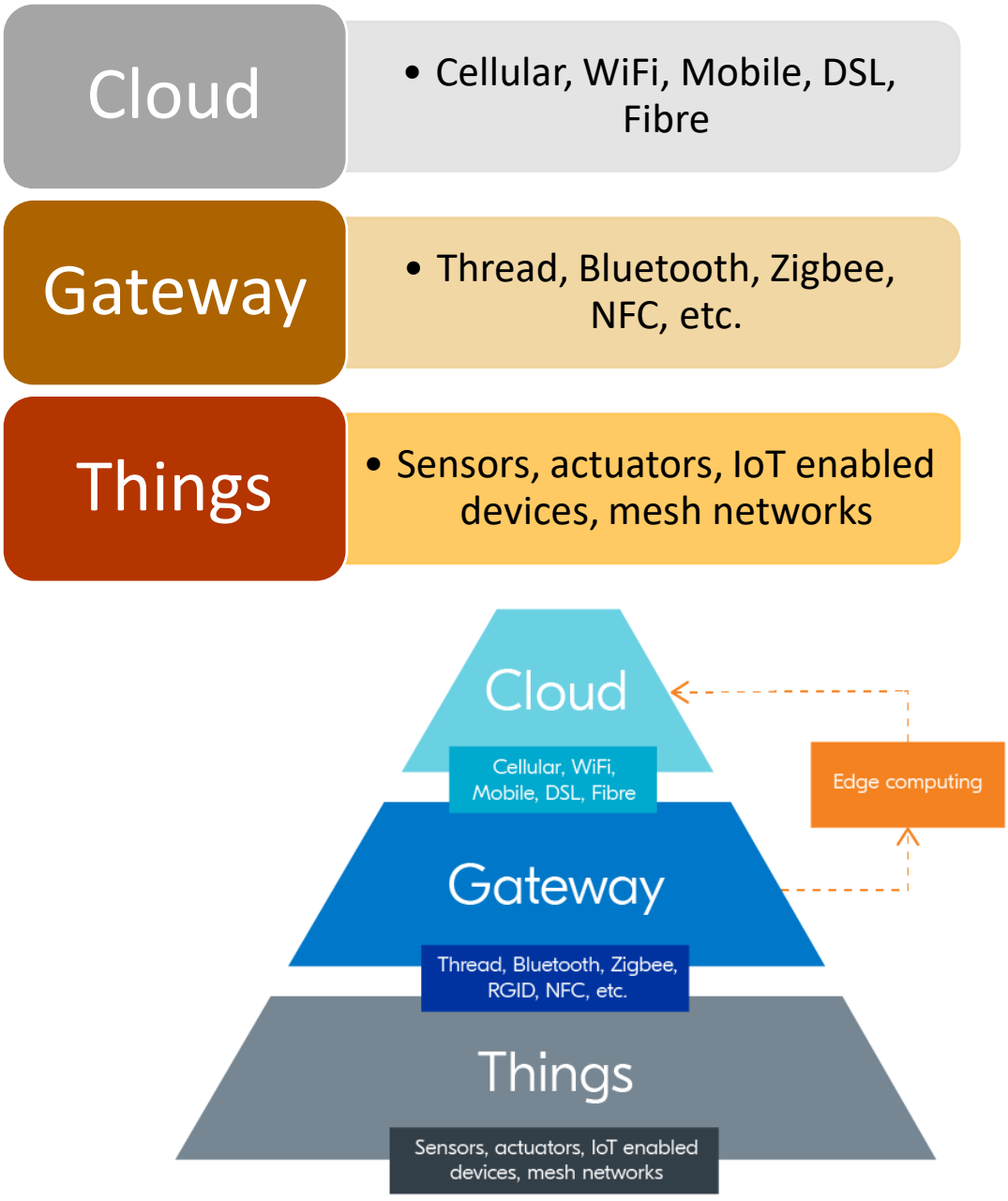


Figure 1-5: The IoT Architecture.

Tier 1 includes the "Things", or networked devices, such as the sensors, actuators and other forms of objects that might be connected.

Tier 2 consists of the Edge Gateway. The Edge Gateways are designed for aggregating sensor data and providing additional processing capabilities, such as pre-processing of the data, securing connectivity to cloud, using systems such as WebSockets, the event hub, and, even in some cases, edge analytics or fog computing support.

Finally, tier 3 includes the cloud environment. Typical IoT applications are deployed in the Cloud, exploiting the scalability of the microservices or virtualized architecture.

One of the major characteristics of the Internet of things architecture is scalablity. Indeed, the IoT should be capable of connecting a continually-increasing amount of devices. Some relevant trends (some of those will be analyzed later on during the course) are outlined in the next paragraphs:

- The overall IoT architecture should enable to interconnect billion of devices. As a consequence, the Internet address space should be extended accordingly. For this reason, IPv6 is expected to play a major role in the network layer.
- The requirement to support low power operation will require simplified networking protocols. In this framework, IETF 6LoWPAN could represent a credible solution to connect devices to IP networks.
- The presence of IoT brokers is expected in most application scenarios. For this purpose, the IoT architecture should incorporate lightweight data transport solutions, such as IETF's Constrained Application Protocol, ZeroMQ, and MQTT.
- Scability will also be required in terms of the aggregate size of data flows potentially being generated by IoT across the Internet. Local/distributed processing will require proper edge or fog computing solutions.

1.4.1.3 Size Considerations

The Internet of things will represent an unprecedented size in terms of connected devices. Some expectations predict 50 to 100 trillion connected objects, and the capability to track and serve those objects. In urban environments, human beings are expected to be surrounded by 1000 to 5000 trackable objects each.

In 2015 there were already 83 million smart devices in people's homes, expected to grow to 193 million devices by 2020. On the other side, online capable devices grew 31% from 2016 to 2017 (in one year) to reach 8.4 billion.

1.4.1.4 Space Considerations

A novel and common requirement deriving from the nature of the Internet of Things is the need for estimating the precise geographic location of an object. This derives from the fact

that most “things” belonging to the IoT are sensors, and sensor location is usually important for IoT applications.

For example, GeoWeb[1] and Digital Earth[2] are emerging IoT applications that become possible only if things can be organized and connected by location.

1.4.2 IoT Challenges

Technologies characterized by rapid development might incur in profound security challenges, as typically most efforts are related to the actual implementation of such technological advances. For this reason, security is surely one of the biggest concerns in the adoption of the Internet of things technology. However, this does not represent the only challenge in the IoT eco-system.

1.4.2.1 Security

Internet of Things provides Internet connectivity to objects capable of operating in the real world and of generating potential threats. Indeed, Internet of Things technology provides access to new areas of data, and it can often enable to control physical devices. Already in 2014 it was possible to understand that many Internet-connected appliances could "spy on people in their own homes". Those objects included smart televisions, kitchen appliances, cameras, and thermostats.

This requires the definition of proper security protocols and architectures to protect the IoT from attacks and avoid anomalous/malicious usage of its services. Some examples of ongoing efforts in this area include:

- In 2015, The Internet of Things Security Foundation (IoTSF) was launched with the mission to secure the IoT by promoting knowledge and best practices. The founding members are technology providers and telecommunications companies, that are now supported by large IT companies continually developing innovative solutions to protect the security of IoT devices.
- In 2017, Mozilla launched Project Thing. The project aims at increasing IoT security by routing IoT devices traffic through a safe Web of Things gateway.

[1] http://geowebforum.com/
[2] http://www.digitalearth-isde.org/

1.4.2.2 Regulation

Also in IoT the risk exists that government regulations might take a long time to catch up with the current state of technology. The problem is that with the rapid of IoT, providing new advancements on a daily basis, governments suffer in catching up and as a consequence businesses are often left without crucial information they need to make decisions.

The lack of strong and harmonized IoT regulations represent also a relevant reason why the IoT remains a severe security risk, and the problem is likely to get worse as (with the evolution of the IoT eco-system) the potential attack surface expands to include ever more crucial devices.

1.4.2.3 Compatibility

A plethora of IoT solutions is available on the market, however they are rarely compatible. Incompatibility might happen at several layer in the communication protocols stack, but also in representation of information, etc.

Indeed, compatibility is current the biggest problem in unleashing the full potential of the Internet of Things. Several standards are available for similar scenarios, and the market seems to require years before settling on a single universal standard or a unifying IoT architecture.

In short-range connectivity, Bluetooth has long been the reference standard. However, especially in home automation, several competitors have sprung up to challenge Bluetooth's mesh network offerings, including protocols such as Zigbee and Z-Wave.

Continued compatibility for IoT devices also depends upon users keeping their devices updated and patched, which can be pretty difficult. When IoT devices that have to talk to each other are running different software versions, all kinds of performance issues and security vulnerabilities can result.

1.4.2.4 Bandwidth

Connectivity, and as a consequence bandwidth, is a bigger challenge to the IoT than experts might expect. As the size of the IoT market continues along it exponential growth, bandwidth-intensive IoT applications such as video streaming will soon start draining resources from other services or generating unexpected congestion events. Moreover, the massive number

of communicating devices might represent a limiting factor for bandwidth, especially in the network control plane.

1.4.2.5 Energy consumption

Most IoT devices (and sometimes even gateways) are battery powered and, in several cases, applications require IoT sensors to run for months or years.

For this reason, energy management represents a clear challenge, that implies the utilization of proper design techniques for optimization in operation, idle/sleep time management, and in general energy-aware operation.

In the recent years, energy harvesting is playing a relevant role in this area.

1.4.2.6 Customers' expectations

IoT is an exciting sector with a lot of potential to change the Society. Indeed, IoT is already having a relevant impact on the way we live, work and play. This is generating high customers' expectations, which are difficult for companies to follow and satisfy.

Indeed, businesses looking to enter this competitive and innovative sector should be prepared for an ever evolving market and customers who always want a smoother and more advanced experience.

2 Revision of Basic Programming and IoT IDE

Author(s): Nearchos Paspalllis

2.1 Introduction to Programming for IoT

A core component of most IoT systems is a microcontroller board used to interface various components, such as sensors and actuators, as well as computing components such as NFC readers, Bluetooth and WiFi network interfaces, etc.

The great success of Arduino and Arduino-like boards has fuelled a trend—commonly referred to as the "Maker movement"[3]—which resulted in many powerful and affordable microcontroller-based solutions. Such boards come in different sizes and shapes and offer varying properties: computing power, memory size, pin types and numbers, battery and power consumption, additional embedded components, and of course price.

All these boards are characterized by one important feature: they are "programmable". This means that you can develop your own custom program—featuring any functionality you want—and run it on your targeted board. The program is generally used to enable your system to *read* data from the sensors (e.g., the "temperature"), and to *write* data to actuate change (e.g., via a "step motor"). Such interfacing is generally achieved via specialized input/output pins and corresponding software drivers. Additionally, most microcontrollers can implement common functionality such as *computation* and *networking*, i.e., exchange data, typically via the Internet.

Programming such a microprocessor is relatively easy if you are at least familiar with the core programming concepts. This document aims to provide a quick overview of these concepts. While it helps to get to this point with some programming experience, it is also possible to use this as your starting point in the adventurous world of programming.

2.2 Programming fundamentals

Computer programming has evolved over the decades, from 1st generation languages which were very low-level (aka "machine language") to general purpose, 3rd generation languages—such as C/C++, Java, etc.[4]

[3] Maker culture is discussed in Wikipedia: https://en.wikipedia.org/wiki/Maker_culture

[4] Even higher generation languages exist but these are more problem specific rather than general purpose. These include languages such as Python and SQL(4th generation) as well as Prolog, etc. (5th generation).

This section focuses on the C programming language, as it is the language most frequently used in the most popular microcontroller platforms, including Arduino and ESP8266.

2.2.1 Prerequisites

While you do not need to know programming to cover this section, you are expected to be familiar with a couple of concepts and skills:

- Binary numbers: The binary system is a system for representing numbers—and any other sort of information possible—using the binary digits of zero (`0`) and one (`1`).
- Logical (or Boolean) operations: These enable the formation of logical conditions. They are commonly used for the formation of complex logical statements in your programs. The most frequently used operands are the AND, OR and NOT.
- Using an *Integrated Development Environment* (IDE): Programs are expressed in high level languages (such as C in this section) using special keywords and syntax. While it is possible to write the listing of a program in any kind of text editor (including Notepad), an IDE is more specialized and offers features which assist the developer. For instance, they offer syntax highlighting, warnings and suggestions, debugging, streamlining the build and deployment process, etc.

2.2.2 Programming concepts covered

This section is aimed as a refresher—or quick introduction—to programming for IoT. The following concepts are covered:

- General overview of procedural programming.
- Variables and simple statements (including mathematic and logical expressions).
- Conditional statements.
- Loop statements.
- Function declaration and invocation.

2.3 Procedural programming

Procedural programming is one of the most fundamental programming paradigms and the basis for newer paradigms such as the Object-Oriented Programming paradigm. Some of the most popular languages using it include Fortran, COBOL, Pascal, and of course C.

In its most simple form, procedural programming consists of programming statements—i.e. the simplest possible commands which can be executed independently—executed sequentially, i.e. line by line. While the execution of programs line by line is simple and has its merits, it lacks in modularity. For this purpose, code focusing on a single purpose can be expressed as a smaller sub-program—a.k.a. a *function*—in C-terminology[5] and be invoked as needed.

Additionally, the typical line-by-line execution can be modified using conditional statements and loops statements. These are further discussed in the following sections.

2.4 Variables, Expressions and Simple Statements

Variables are the most fundamental elements used in programming. They generally represent values of various types, including numbers, text, etc. With the use of language-specified keywords, they form expressions and statements. All these terms are discussed below.

2.4.1 Data Types

Some languages like JavaScript have a more flexible, implied type while others like C/C++ and Java are so-called strictly typed languages. In the latter, all values must be declared to have a specific type. How the language keywords interact with variables depend on their declared type. For example, if two variables are numbers, then when printed they are automatically transformed to text representing their decimal value.

In C, the basic data types are the following:

- `int` – used to encode integer numbers (i.e., with no fractional part).
- `double` – used to encode real, floating point numbers (i.e., with a fractional part).
- `char` – used to encode a character in an alphanumeric value (i.e., a letter in a text).

Examples of values with such types are:

- `int x = 7;` // here 'x' is a variable of type int, initialized to 7.
- `double rate = 1.23;` // here variable 'rate' is declared as a real number and initialized to the value 1.23.

[5] While in C/C++ sub-programs are called *functions*, in modern object-oriented languages—like Java—they are called *methods*. The following terms have also been used with a similar meaning: sub-routines, sub-procedures.

- `char initial = 'N';` // a variable of type character is declared with name 'initial' and initial value the letter "N".

An important observation is that C does not specify an explicit Boolean/logical value type. Instead, other types—typically `int` values—are used in its place, where the convention is that the zero (`0`) binary value corresponds to FALSE and any other value (but typically `1`) corresponds to TRUE.

For example, the following variable can be used as a Boolean representing FALSE.

- `int flag = 0;` // the variable 'flag' is declared as int but with the intention to be used as a Boolean-it is initialized to 0 which corresponds to False.

In practice, this lack of direct support of Booleans is considered one of the weak points of C, which makes it more subtle to errors—especially to newcomer programmers. Newer programming languages—such as Java—aimed to fix this by introducing explicit Boolean data types.

Note also that C does not have direct support for *text* (e.g., words, sentences, etc.) but this is implicitly supported where the corresponding values take the form of *sequences* of `char`.

All variables have three important properties: A *name*, a *type* and a *value*.

As C is a strictly typed language, all variables must be *declared* before they can be used. In other words, the following program would fail as the variable 'x' is undeclared:

```
x = 2; // cannot succeed unless the variable 'x' is declared first
```

Listing 1: Variables must be declared first before use

A variable can be declared simply by specifying its type first, and its name next. Optionally, a variable can also be initialized at declaration using the equals character '=' and the initial value as a literal[6].

This is illustrated in the following examples:

[6] A 'literal' is a value expressed inside the source code. Numbers are typically expressed using the corresponding text (using digits 0-9) and characters are expressed as the corresponding character in single quotes (e.g. 'k').

```
char a; // a new variable is declared with type 'char' and name 'a'.
double z = 121.35; // a new variable of type 'double' and name 'z' is
declared and initialize to the value '121.35'
```

Listing 2: Variable declaration examples

2.4.2 Comments

Every programming language has a form of adding *comments* to the code. These are defined to help the programmers by serving as extra information reminding why/how code works. Comments are also important for narrating code expected to be used/maintained by others.

Good use of commenting is considered an important skill for programmers—among other qualities which relate to writing legible and structured code. In principle, always remember that clarity is very important, and can have an important impact on the success of your code.

In C, comments can be added in two main ways:

- Using the double slash characters to signify that the rest of the *line* is a comment.
- Using the double markers of slash and star to indicate the start (/*) and the end (*/) of a comment.

These two methods of comments can be better illustrated in the following examples:

```
// this is a single line comment
int x = 1; // single line comments can be added after a statement
/* Sometimes a comment
can take multiple lines.
In this case the multi-line markers
are more suitable. */
```

Listing 3: Examples of comments in C

2.4.3 Expressions

Expressions are either part of, or a full statement. Most commonly, they take the form of *mathematical* or *logical* expressions.

In terms of mathematical expressions, the typical mathematics operators for *addition*, *subtraction*, *multiplication* and *division* work as expected. These are the '+', '-', '*', '/' respectively. An additional operator called *module* is represented by the '%' symbol.

The following examples illustrate the use of operators:

```
int x = 10; // declare and initialize a variable 'x' with the value 10
int y = 1 + 2; // declare a variable 'y' and initialize with the result of adding 1+2
y = 10 * 2; // modify 'y' so its value is replaced with the result of multiplying 10*2, i.e. the new value of 'y' will be 20
```

Listing 4: Simple examples demonstrating the use of operators

Variables can also be used in the formation of expressions—besides being the assigned value. For example, a variable can be assigned the value of another variable—possibly as part of an expression.

Importantly, a variable can be defined as a function of itself. For example, the statement "x=x+1" doesn't mean that x is equal to itself plus 1, but that the *new* value of x is computed by adding 1 to the current value of x. In this case it is important to note that the semantics of the equals ('=') symbol is not *equality* but *assignment*[7]. The assignment essentially means: first compute the value of the expression to the *right* of the assignment symbol, then use that to replace the value of the variable at the *left* of the assignment symbol. Note that in pseudocode, the assignment is often denoted with a left arrow, i.e. '<-', as a means of disambiguation (e.g. 'x <- x +1') where in many programming languages including C/C++ and Java, it is denoted with the single equals symbol (i.e. 'x = x + 1').

The semantics of assignment statements are illustrated in the following examples:

```
int z = x; // declare a variable 'z' and initialize it with the value of x, i.e. 10
z = x + y; // change the value of a variable 'z'—make it equal to the result of adding x and y, i.e. 10+20, so the result will be 30
z = 10 * z; // change the value of 'z' by making it equal to the former value of 'z' times 10, i.e. 10*30=300
```

Listing 5: Examples of assignments

The *precedence* defines the priority with which operations are executed. For instance, multiplication and division take precedence—i.e., execute *before*—addition and subtraction. Keep in mind that precedence can be overridden using brackets. These are illustrated in the following examples:

[7] In programming, the *assignment* is operation of modifying the value of a variable. In many languages it is denoted with the *equals* symbol '='. In this regard, the *equality check* operator is frequently marked with a *double equals* symbol, i.e., '=='.

```
int x = 1 + 2 * 3; // 'x' gets the value 7 (not 9) because multiplication has precedence over addition
int y = 9 - 6 / 3; // 'y' gets the value 7 (not 1) because division has precedence over subtraction
```

Listing 6: Demonstrating precedence of operations

Quite frequently, in C/C++ the following special operators are used:

- '++' Increment by 1. For example, 'x++;' is equivalent to 'x = x+1;'.
- '--' Decrement by 1. For example, 'y--;' is equivalent to 'y = y-1;'.
- '+=' Increment by value. For example, 'x+=10;' is equivalent to 'x = x+10;'.
- '-=' Decrement by value. For example, 'y-=4;' is equivalent to 'y = y−4;'.
- '*=' Multiplied by value. For example, 'z*=2;' is equivalent to 'z = z*2;'.
- '/=' Divided by value. For example, 'w/=3;' is equivalent to 'w = w/3;'.

The '++' and '--' are unary operators—i.e., unlike most operations (such as addition and multiplication) which operate on two operands, unary operators apply to a single operand. For this reason, their precedence is important. There are two ways to apply them, the so-called postfix (after) and prefix (before). The postfix is applied *after* the value is used in its expression. The prefix is applied before. This is demonstrated in the following examples:

```
int x = 10; // initialize 'x' with value 10
int a = x++; // a gets the original value of x, i.e. 10, and afterwards, x is increased to 11
int b = ++x; // x is first increased further to 12, then assigned to b, which becomes also 12
x*=2; // x is doubled, i.e. becomes 24
int c = x--; // c gets the original value of x, i.e. 24, and afterwards, x is decreased by 1 to 23
int d = --x; // x is first decreased by 1 to 22, then assigned to d which becomes also 22.
```

Listing 7: Demonstrating prefix and postfix use of unary operators

Besides arithmetic operations, various logical operations can also be used for computing Boolean values. Recall that in C, Boolean values are stores as integers, where zero indicates `False` and non-zero values indicate `True`.

The following operators are commonly used for comparing arithmetic values, with a Boolean result:

- '<' Smaller than. For example, 'x<10' is true if x is smaller than 10, false otherwise.
- '<=' Smaller than or equal to. For example, 'y<=-2' is true if y is -1 or smaller, false otherwise.
- '>' Greater than. For example, 'myVal>2.13' is true if 'myVal' is greater than 1.23, false otherwise.
- '>=' Greater than or equal to. For example, 'z>=-9.99' is true when z is equal to -9.99 or greater.
- '==' Equal. For example, 'sum==0' is true if the value of 'sum' is zero, false otherwise.

At this point we should point out that while equality is rather straightforward for integers, it is less so fractional numbers. For instance, the expression 9.0-8.3-0.7 should normally result 0.0. However, there are some case where this is not exactly the case. The reason has to do with the representation of real-precision numbers in binary format. As real numbers cannot be perfectly fit in fixed-size binary values (typically 64bits are used for each 'double' value), it is possible that the result of an expression expected to be zero is some non-zero value, e.g. 0.000000001. Where this could lead to logical errors in your program, it is advised to instead check if the difference of your checked value against zero is smaller than a predefined error threshold. For example, you could check if (9.0-8.3-0.7 < 0.001) which would be true if the expression was *sufficiently small*, i.e., close to zero. In reality you would have to check both ways, as the result could be a very small positive or negative value. Consider the following example for checking if a value x is (nearly) zero:

```
double e = 0.000001; // in this example, 1 millionth is the threshold
double x = 9.0-8.3-0.7; // assign x some value which you want to check if
it is zero
int isZero = x>-e && x<e; // the 'isZero' variable is true if x within (-
e, e) - note that '&&' is the logical AND operator discussed below
```

Listing 8: Checking if a value is close enough to zero

Besides arithmetic operations, logical operations can be used for forming logical expressions. The main logical operators—in order of precedence—are the following:

- ! Logical NOT
- && Logical AND
- || Logical OR

For example, consider the following code listing:

```
int day = 4; // where Sunday is 0, Monday 1, ..., Saturday 6
double temperature = 32.7; // assume temperature in Celsius
int weekend = day == 0 || day == 6; // equality has precedence over assignment, so that value of 'weekend' is false because (day == 0) is false and (day == 6) is also false
int mustGoToBeach = weekend && temperature > 30; // the value of 'mustGoToBeach' is set to zero/false as the value of 'weekend' is false—even though the value of (temperature > 30) is true
```

Listing 9: Demonstrating logical operators in C

The precedence of the most relevant operators is listed below—from higher to lower precedence:

- ++ and -- // Postfix increment and decrement
- ++ and -- // Prefix increment and decrement
- '!' Logical NOT
- * and / and % // Multiplication, division and remainder
- + and - // Addition and subtraction
- < and <= and > and >= // arithmetic comparison operators
- == and != // Equality and inequality operators
- && // Logical AND
- || // Logical OR
- = and += and -= and *= and /= // simple assignment and assignment by addition/subtraction/multiplication/division

When an expression contains two or more operations listed above, then their execution follows the above precedence. This means that the one with the highest precedence executes first and the result replaces it in the expression, then the one with the second highest

precedence, and so on. For example, an expression containing an *assignment*, a logical AND, and a *multiplication* will be executed resolving the operations in this order: *multiplication*, then *logical AND,* then *assignment*.

Parentheses can be used to override the order, just like in mathematical formulas.

The full list of operator precedence in C can be found online at cppreference.com[8].

2.4.4 Code Blocks

Code blocks are groups of statements at the same *level*. An important characteristic of code blocks is that they also define a name space, i.e., a space where the declared variables are *recognized*.

Code blocks are defined using the curly brackets '{' and '}'. As it will be discovered later, they are also commonly used in line with defining conditionals, loops, and functions.

A variable declared in a code block is *visible* inside that block, and all nested blocks. See the following example.

```
{
  int x = 1;
  // x is visible here
  {
    // x is visible also here, as this is a block nested in the block where
x was declared
  }
}
// x is not recognizable here, as it was declared in another code block
```

Listing 10: Examples of nested code blocks and implication of variable visibility

Another important note for code blocks, also seen in the previous example, is that code blocks can assist with code comprehensibility and safety. Specifically, by properly using code blocks, you can limit variable visibility to the space needing it, thus making it harder to make mistakes[9].

[8] See https://en.cppreference.com/w/c/language/operator_precedence

[9] The idea of breaking code down to individual parts with minimum—and controlled—interaction with each other is a foundational idea of code reusability and readability, originally proposed by Parnas in his paper titled "On the Criteria to Be Used in Decomposing Systems into Modules"—see https://www.win.tue.nl/~wstomv/edu/2ip30/references/criteria_for_modularization.pdf.

Also, for better readability, the programmer is advised—but not enforced[10]—to use proper *indentation*, i.e. consistent spacing before each statement in a new line, reflecting the nesting depth of the line.

2.4.5 Statements

The *statements* are the building blocks of any program. In C/C++, statements are *terminated* by the semicolon '**;**' character. Effectively this means that any statement must end with a semicolon.

In theory multiple statements can be added on the same line, but in most cases, this is discouraged as it leads to cumbersome, hard-to-read code.

In C, there are three main types of statements: Assignments, Conditionals, and Loops.

Assignments have been discussed previously, and Conditionals and Loops are discussed in the following sections.

Statements can also comprise of function *calls*. Function calls can be used to interact with lower-level operations too, such as reading/writing from a file or printing to the console—aka the screen.

For example, a simple way to write a message in C, is by using the *printf* function. A simple 'Hello World' program[11] in C is as follows:

```
#include <stdio.h>
int main() {
   printf("Hello, World!\n");
   return 0;
}
```

Listing 11: Hello World program in C

When executed, this code produces the unsurprising output:

```
Hello World!
```

[10] Some modern programming languages—most notably Python—force programmers to be consistent with indentation as it is by definition the method with which nesting and code blocks are defined in code written for Python.

[11] In most programming languages, the 'Hello World' is the starting point of the journey of learning that language. It comprises of a bare-bones program which simply prints a greeting message on the screen, typically 'Hello World'.

Here is a line-by-line explanation of this code:

- The '`#include <stdio.h>`'statement is used to import a library. In this example, the library is a standard library containing functions like '`printf`' (see below) for printing messages to the console. We will be using various libraries when writing code for Arduino in order to interface with different hardware components.
- The starting point of a C program is the 'main' function[12]. By convention, this function is named `main`, takes no arguments[13] and returns an integer (normally zero if everything was OK). We will later see that Arduino-based programs use a different set of methods for starting up—instead of `main`. It should be noted that as main is a function it explicitly defines a code block which is marked with the standard starting/ending curly brackets.
- The `printf` is a function defined in the `stdio.h` file—imported in the first line. It allows for printing text on the console. In C, text[14] can be declared as code literal using double codes, e.g. "Hello World" is the literal text corresponding the sequence of characters 'H', 'e', etc. The '\n' is a special character meaning "*line return*" and has the semantics of moving the cursor to the next line of the console. As IoT-based platforms often come with no display, we will see that some development platforms allow printing to a connected computer for debugging purposes.
- Finally, the `return 0;` statement terminates the function (main) which effectively terminates the program. By convention, the returned value should be zero if the programme succeeded, and non-zero if it failed for any reason.

2.4.6 Conditionals

The conditionals are simple statement which enable the program to respond to a *condition* (commonly the value of a variable) by following one or another path.

There are two main forms of conditionals in C:

1. `if <condition> { one or more statements ... }`

[12] Functions are discussed in another subsection below.
[13] There are variations of the 'main' function which provide arguments and allow the programmer to read and handle input from the user—input which is specified when the program is called from command line.
[14] In programming, text values are also commonly termed *strings* or *alphanumerics*.

2. `if <condition> { one or more statements ... } else { one or more statements ... }`

In the first case, the statements enclosed in the curly brackets are executed *only if* the condition is true. In the second case, the statements enclosed in the first curly brackets pair are executed *if* the condition is true, otherwise the statements enclosed in the second pair is are executed.

The if-conditional is illustrated in the following example:

```
#include <stdio.h>
int main() {
  int day = 5; // assume Saturday=0, Sunday=1, Monday=2, ..., Friday=6
  if(day==0 || day==1) { // if 'day' equals 0 or 1 ...
    printf("Weekend! Hooray!\n");
  }
  return 0;
}
```

Listing 12: Example demonstrating the use of if conditionals

This example will produce the output[15]:

```
Weekend! Hooray!
```

The if-else-conditional is illustrated in the following example:

```
#include <stdio.h>
int main() {
  int day = 5; // assume Saturday=0, Sunday=1, Monday=2, ..., Friday=6
  if(day==0 || day==1) { // if 'day' equals 0 or 1 …
    printf("Weekend! Hooray!\n");
  } else {
    printf("Weekday... :-(\n");
  }
  return 0;
}
```

Listing 13: Example demonstrating the use of if-else conditionals

[15] Question: What would the output be if the value of day were 4 (in the Listing 12)?

This example will produce the output[16]:

```
Weekday... :-(
```

In general, the condition can be any expression which can be resolved to `true` or `false`[17]. This can combine Boolean values with logical operators (and, or, not) as well as arithmetic comparisons, etc.

Finally, note that the curly brackets can be omitted when there is just one statement. However, it is a good practice to always use them, even for enclosing just one statement, as this improves the *readability* of the code.

2.4.7 Loops

One of the main strengths of computers is that they can execute the same commands again and again, tirelessly and super-fast. From the programmer's perspective, this usually involves the use of *loops*.

There are two main forms of loops in C: using the `while` and the `for` keywords.

The former even has two variations, which are summarized as follows:

1. `while <condition> { one or more statements ... }`
2. `do { one or more statements ... } while <condition>`

In the first variation, the *condition* is checked first, and the statements are executed if the condition is `true`. This is repeated until the condition becomes `false`[18].

In the second variation, the *condition* is checked last, which means that the statements always execute at least once. This is used when the intended logic requires that: For example, if you need to write a program where you ask the user to make a choice and then decide whether to exit or not based on their input, the second variation is a better fit.

[16] Question: What would the output be if the value of day were 4 (in the Listing 12)?
[17] For conditionals where the operand is a number (e.g. integer) then it is resolved to false if it equals zero, and to true in all other cases.
[18] This adds a risk where in poorly defined code, the condition is always true. This is commonly referred to as *infinite loop*. Programmers are advised to be extra careful when they set the conditions which terminate loops.

The use of while-conditions is illustrated in examples as follows:

```
#include <stdio.h>
int main() {
  int x = 1; // define a variable 'x' and initialize its value to 1
  while(x < 24) { // repeat while x is smaller than 24
    x = x * 2; // in each loop, duplicate the value of 'x' ...
    printf("x: %d\n", x); // ...and print its value
  }
  return 0;
}
```

Listing 14: Example demonstrating the use of while loop

This example produces the output:

```
x: 2
x: 4
x: 8
x: 16
x: 32
```

The code is explained in its comments. It basically initializes a variable to 1 and then doubles its value in each iteration. As the value is doubled before printing, the first value that appears in the output is '2'.

Note that the command `printf` can be used to also format variable values appearing in the printed text. In this case, the value of 'x' is included in the text as an integer with the notation %d[19].

The for-loop is defined as follows:

```
1. for (<init>; <check>; <step>) { one or more statements ... }
```

[19] The printf command, along with the different formatters it supports are described online at https://www.tutorialspoint.com/c_standard_library/c_function_printf.htm

Again, the use of while-conditions is better illustrated in an example as follows:

```
#include <stdio.h>
int main() {
  for(int i = 1; i < 10; i++) { // repeat for values of i in the range 0..9
    printf("i: %d\n", i); // in each loop, print the value of 'x' ...
  }
  return 0;
}
```

Listing 15: Example demonstrating the use of while loop

This example produces the output:

```
i: 1
i: 2
i: 3
i: 4
i: 5
i: 6
i: 7
i: 8
i: 9
```

Effectively, the `for` statement defines three arguments: An *initializer*, e.g. '`int i = 0`' in this example, a *conditional check*, e.g. '`i < 10`', and a *step increment*, e.g. '`i++`'. These three arguments are separated by semicolons '`;`'.

The *initializer* executes first, exactly once. The *conditional check* is evaluated in every iteration *before* the execution of the statements—if true they execute, otherwise the loop terminates. Last, the *step increment* executes in every iteration *after* the execution of the statements.

Note that in the above example, the printed values are `0` to `9` (and not `10`). The reason for this is that the continuation check '`i < 10`' is evaluated before the execution of the statements. So when '`i`' eventually becomes 10 (and thus the *continuation check* becomes `false`), the loop terminates immediately, and the statements are not executed.

For loops are commonly used to iterate all the indices of arrays—see below.

Similar to what was discussed with *conditionals*, the curly brackets can be omitted when there is just one statement. However, like before, it is a good practice to always use them, even for enclosing just one statement, as this improves the *readability* of the code.

2.4.8 Arrays

A commonly used and widely useful data type is the *array*. These are sequences of values. They are commonly stored in consecutive addresses in the computer memory. They are accessed directly using an *index*—which in C is defined using square brackets '[' and ']'.

An array has a *size*, which is defined at initialization—also using square brackets. The *size* indicates the number of values it contains. By convention, the address of the first element is at index 0, the next one at 1, and the last one at *size-1*.

So for example, for an array named 'a' of integers with a size 10 (i.e. containing 10 elements), the first one can be accessed with a[0], the second one with a[1], etc. The last one is accessed at index *size-1*, i.e. with a[9] in this case.

These are illustrated in the following example:

```
#include <stdio.h>
int main() {
  int a[3]; // create an array of integers named 'a' with 4 values
  a[0] = 10; // initialize the first element to 10
  a[1] = 20; // initialize the second element to 20
  a[2] = 30; // initialize the third element to 30
  a[3] = 40; // initialize the fourth element to 40
  printf("a[0]: %d\n", a[0]); // print the value of the first element
  printf("a[3]: %d\n", a[3]); // print the value of the last element
  return 0;
}
```

Listing 16: Illustrating the use of arrays in C

The array is first declared, specifying its size. Next the four elements of the array are explicitly assigned a value. Last, the values of the first and last elements are printed.

A more complex example, showing how loops can be used to iterate all the elements of an array is listed below:

```
#include <stdio.h>
int main() {
  const int SIZE = 5; // constants are in capitals by convention
  int a[SIZE]; // create an array of integers with the given size
  for(int i = 0; i < SIZE; i++) { // repeat for i in the range 0..SIZE-1
    a[i] = i+1; // make each element equal to its index plus 1
  }
  for(int i = 0; i < SIZE; i++) {
    printf("a[%d]: %d\n", i, a[i]); // print the index and value of each
  }
  return 0;
}
```

Listing 17: Using a for loop to iterate all the elements of an array

This listing includes the following:

- First, note the use of the `const` keyword. It stands for *constant* and it is useful for defining values that you know should not change[20]. In this case the `SIZE` is set to 5[21].
- An array named 'a' is defined and initialized to have a specified *size*.
- The first *for-loop* is used to iterate all the elements of the array and set the value of each one to that of the corresponding index plus 1, i.e. '`i+1`'.
- The second *for-loop* prints the value of each element in 'a'.

When executed, this listing produces the following output:

```
a[0]: 1
a[1]: 2
a[2]: 3
a[3]: 4
a[4]: 5
```

2.4.9 Functions and Function Calls

In the previous sections we explained how built-in functions are *called*. However, functions are incredibly useful to also enable *modularity* in your code. For this, you need to be able to define and use your *own, custom functions*.

[20] This is good because it clarifies to anyone reading the code that this value is constant—i.e. cannot change. If the value is modified—e.g. by mistake—in a later version, the problem is easily spotted as the compiler will generate an appropriate error message.

[21] Feel free to modify the value of SIZE and see how it affects the output. Make sure to try out some corner cases, like zero.

In C, functions can return a value, similar to how mathematical functions are formed. Or they can execute some statements without returning anything, like we have seen with '`printf`'.

In their more general form, functions have the following elements:

```
<return_type> <function_name>( [optional parameter list] ) {
   <one or more statements>
}
```

The `return_type` is a built-in or custom data type, like `int`, `double`, `String`, etc. When a function is not required to return anything, it is marked with the special return type of '`void`'.

The `function_name` must be a valid, non-reserved keyword. Examples are '`myFunction`', '`f1`', etc. However, good programming practice dictates we use meaningful names for function names, like '`print`', '`computeCircumferance`', etc. The same applies for variables and parameters.

The `parameter_list` is an optional (i.e. possibly empty) list of parameters to be passed to the function. These are specified in brackets and their values are normally used in the body of the function to form the *answer*. It is a good programming practice to avoid *modifying* the parameters. Furthermore, parameters are passed *by-value* by default in C. This means that a *copy* of the variable is created and passed, so even if the value is modified in the body of the function, this is not reflected in the original variable. Note this only applies to standard variables. Variables which are memory addresses, like *pointers* and *arrays* are passed *by-reference* by default which means their modification is reflected in the original variables. Note that explaining the underlying mechanics of pointers and arrays is an important part of C/C++, which however is not covered in this chapter[22].

Finally, the function body is completed with a pair of curly brackets which enclose one or more statements[23]. This is where you specify the code to execute when the function is called. Parameters can be used by name. When a returned value is needed, that can be specified with the '`return`' statement, which naturally also terminates the function.

[22] Pointers and Arrays are significant elements of C/C++. While these are besides the scope of this chapter, readers are encouraged to read about them in additional resources. For example, this is a good starting point: https://books.goalkicker.com/CBook/.
[23] In practice a function can have an *empty* body too, but this is unlikely to be of any use.

For example, consider this function which prints a square with the specified side size.

```
#include <stdio.h>

void square(int side) {
    if(side < 1 || side >> 10) { // if side not 1..10, return immediately
        printf("Side must be 1..10!\n");
        return;
    } else { // side is 1..10
        for(int x=0; x<side; x++) {
            for(int y=0; y<side; y++) {
                printf("*");
            }
            printf("\n"); // move to next line
        }
    }
}

int main() {
    square(4);

    return 0;
}
```

Listing 18: Defining a function for printing a square of the specified size

When executed this program produced the following output:

```
****
****
****
****
```

When the code is changed to call the function with an invalid parameter, e.g. -3, the output is as follows:

```
Side must be 1..10!
```

Note that it is possible to call a function from within itself. This is called *recursion*. In most cases recursion can be avoided, and when possible, it should be avoided, both for performance purposes but also to keep your code clean and comprehensible.

For example, a classic problem to demonstrate recursive function is the Factorial number[24]. This can be defined as a recursive function, and as an iterative function, as shown below.

[24] https://en.wikipedia.org/wiki/Factorial

```
#include <stdio.h>

int factorial_recursive(int n) {
    if(n == 1) return 1;
    else return n * factorial_recursive(n-1);
}

int factorial_iterative(int n) {
    int p = 1;
    for(int x=2; x<=n; x++) {
        p = p * x;
    }
    return p;
}

int main()
{
    printf("Factorial values\n");
    printf("i, recursive, iterative\n");
    for(int i = 1; i<=5; i++) {
        printf("%d: %d, %d\n", i, factorial_recursive(i),
factorial_iterative(i));
    }

    return 0;
}
```

Listing 19: Recursive and equivalent iterative implementations of Factorial sequence functions

The output of this program is as follows:

```
i, recursive, iterative
1: 1, 1
2: 2, 2
3: 6, 6
4: 24, 24
5: 120, 120
```

Note how the corresponding functions return integer values.

2.5 Integrated Development Environment

An *Integrated Development Environment* (IDE) is specialized software which provides various tools to software developers to specify, test, debug, manage, and deploy code.

In practical terms, IDEs are used to improve the performance of software developers, as they can simplify many tasks, minimize the time it requires to do other tasks, and at the same time minimize the risk of errors and bugs.

Individual programming languages and platforms have various IDE options. For example, Java has multiple popular IDEs like IntelliJ IDEA, Netbeans, Eclipse, etc. At the same time, Android

developers use primarily the Android Studio IDE, even though they mostly use Java. Similarly, code written for .NET is predominantly developed using Microsoft's Visual Studio.

In terms of C, and especially for IoT applications, one of the most popular frameworks is the Arduino IDE. Despite its name, it is not limited to Arduino boards, but it can be used for coding other popular platforms too, like ESP8266, etc.

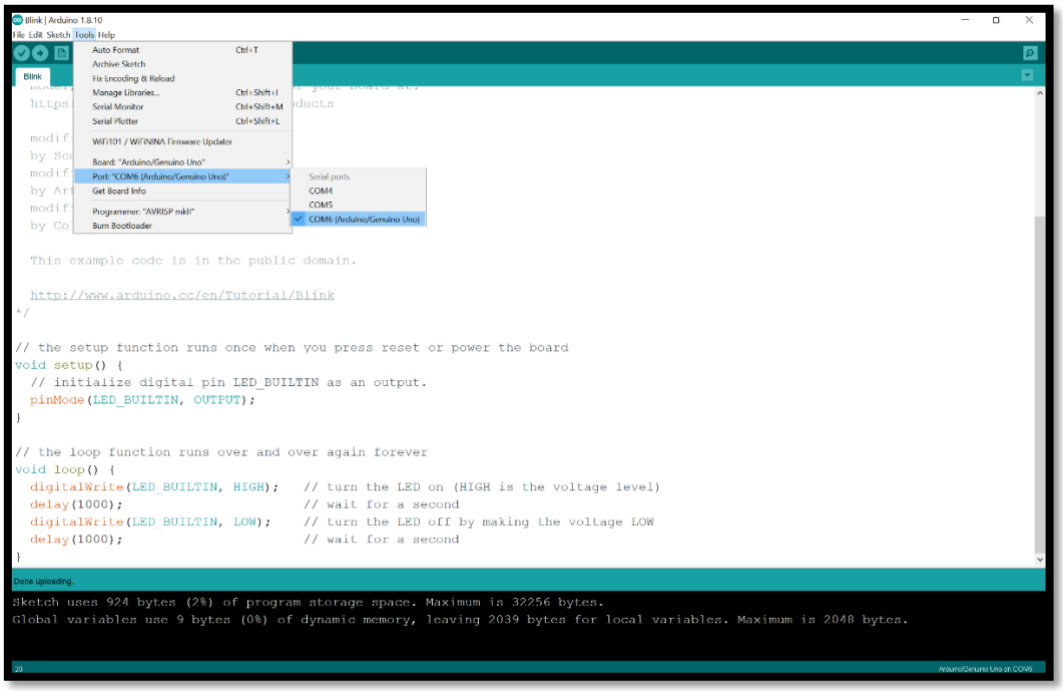


Figure 2-1: The Arduino IDE

The Arduino IDE is available for free both as an online, web-based tool (which requires a web-browser plugin) and as a standalone app for Windows, MacOS and Linux[25]. Another popular, free IDE is Microsoft's Visual Studio Code, also available for Windows, MacOS and Linux[26]. In this book we use the Arduino IDE[27]. A summary of what it looks like is also shown in Figure 2-1.

The next chapter covers in detail the use of Arduino IDE for software development targeting IoT.

[25] You can access both the web and the downloadable versions of Arduino IDE at: https://www.arduino.cc
[26] You can get MS Visual Studio Code from: https://code.visualstudio.com
[27] A description of the features of this IDE are available at https://www.arduino.cc/en/Guide/Environment

2.6 Practice exercises

For practice, complete the following challenges.

- Edit the code in Listing 18 to print a *hollow* square. Extend the allowed range of values to 1..20. A hollow square is one where only the borders are drawn. For example, a hollow square of side 3 would look as follows:

- Learn about Fibonacci numbers[28], and edit the code shown in Listing 19 to compute the corresponding sequence of Fibonacci numbers in both a recursive, and iterative manner.

2.7 Concluding remarks and further resources

This chapter provides a quick introduction to Programming with the C language. Naturally, programming itself is a separate topic and requires its own book (or books).

This chapter aimed at *quickly* introducing the reader to programming. It is expected that most readers are already familiar with the fundamentals of programming and they use this as a refresher, or a revision of the equivalent programming concepts as applied in C.

The following chapter puts this knowledge in practice by guiding the reader through multiple IoT-related programs expressed in C. Readers who feel they need to further enhance their fundamental programming skills before endeavouring to IoT programming are encouraged to go through the following resources.

2.7.1 Further resources

First, the reader is encouraged to refer to these useful online resources for covering the fundamentals of programming in C in more detail.

- **C Interactive Tutorial - https://www.learn-c.org/**

  The "Learn C" is a free interactive tutorial which can quickly introduce you to the basics of C. Conveniently, the tutorial includes interactive elements, which execute directly on the Web Browser, foregoing the need to download and install an programming

[28] https://en.wikipedia.org/wiki/Fibonacci_number

environment. The content of this tutorial is split in two sections: “Learn the Basics” and “Advanced”. The former is highly recommended, while the latter is only for those interested to learn C in higher detail—but not too critical for IoT programming.

- **Kernighan B. and Ritchie D.: The C Programming Language, Pearson; 2nd Edition, April 1, 1988**
  "The C Programming Language" by Brian Kernighan and Dennis Ritchie is considered the seminal book on C. You can use this if you are interested to learn everything about the C language.
- **Stroustrup, B.: The C++ Programming Language, Addison-Wesley 4th Edition, ISBN 978-0321563842. May 2013 - https://www.stroustrup.com/4th.html**
  In case you want to go deep software development with C/C++, this is an extensive book covering many advanced topics, written by the creator of C++, Bjarne Stroustrup.

Additionally, the reader is pointed to the following IDEs (Integrated Development Environments) which provide a modern and powerful approach to software development:

- **Visual Studio Code - https://code.visualstudio.com**
  This is a free IDE developed by Microsoft. It is widely popular as it is free, extensible (via plugins) platform independent, and with a very large user base which makes it easy to get support. Note that one of the many plugins available for Visual Studio Code is one which enables interfacing with Arduino microcontrollers.
- **Arduino Software - https://www.arduino.cc/en/main/software**
- The Arduino website provides two useful tools: First a downloadable editor (IDE) which enables defining the code, compiling it, and 'uploading' it to a connected microcontroller (this includes Arduino hardware but also other popular platforms like the ESP8266-- https://en.wikipedia.org/wiki/ESP8266). Second, an equivalent editor is made available as a Web Based tool (but requires a plugin for enabling connection to hardware components).

3 Software Development for IoT Embedded Systems

Author(s): Nearchos Paspallis

3.1 Introduction

Software development is required throughout the various layers of any IoT architecture (see section 4 for a description of possible IoT architectures and their layers). In most commercial settings, IoT systems comprise of both endpoints (i.e., microprocessors or microcontrollers with attached sensors and actuators) as well as backends (i.e. cloud-based systems for collecting, aggregating, processing and complementing the IoT systems).

While backends can support a wide range of services—from access to large data, storage, and advanced functionality which could include geofencing, Machine Learning (ML), etc.—the endpoints are commonly where the data is collected and processed in the first place.

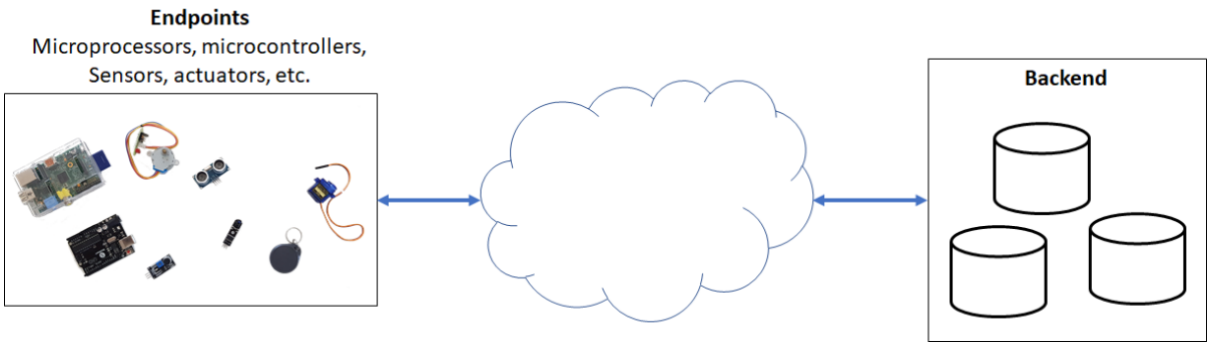


Figure 3-1: Simple IoT architecture: Endpoints, comprising of microprocessors, microcontrollers, sensors and actuators interact with backends to form an IoT system

For the purposes of this section, we focus on software development related to the endpoints. This usually includes low-level code to interface with appropriate hardware (e.g., sensors and actuators), pre-process data (e.g., summarize, filter or compress it), and connect via the network to interface with appropriate Backends.

Microprocessor-based systems, like Raspberry and similar, feature full-scale operating systems enabling them to reuse many of the existing libraries and packages. However, cost and power limitations often render microcontrollers, like Arduino and ESP8266, as the more suitable choice for this purpose.

This section focuses on software development using Arduino, one of the most popular and widely available microprocessors as of today. We cover elements of software development which include interfacing with hardware components—including wiring the respective components—as well as defining logic for realizing IoT embedded systems.

3.2 The development environment

For the purposes of this section, we use the Arduino Software (IDE) which is freely available online[29]. This software is available both as a stand-alone application (for Windows, Mac OS X and Linux) as well as a web-based system—which however requires a Web browser plugin to become fully functional. For the purposes of this section, it is assumed that you have the Arduino IDE stand-alone application available on a computer.

While the Arduino IDE is developed and formally used for Arduino-compatible devices, various IDE plugins allow its use also with similar microcontrollers like the ESP8266.

3.2.1 Tour of the IDE

When launched, the IDE looks as shown in Figure 3-2.

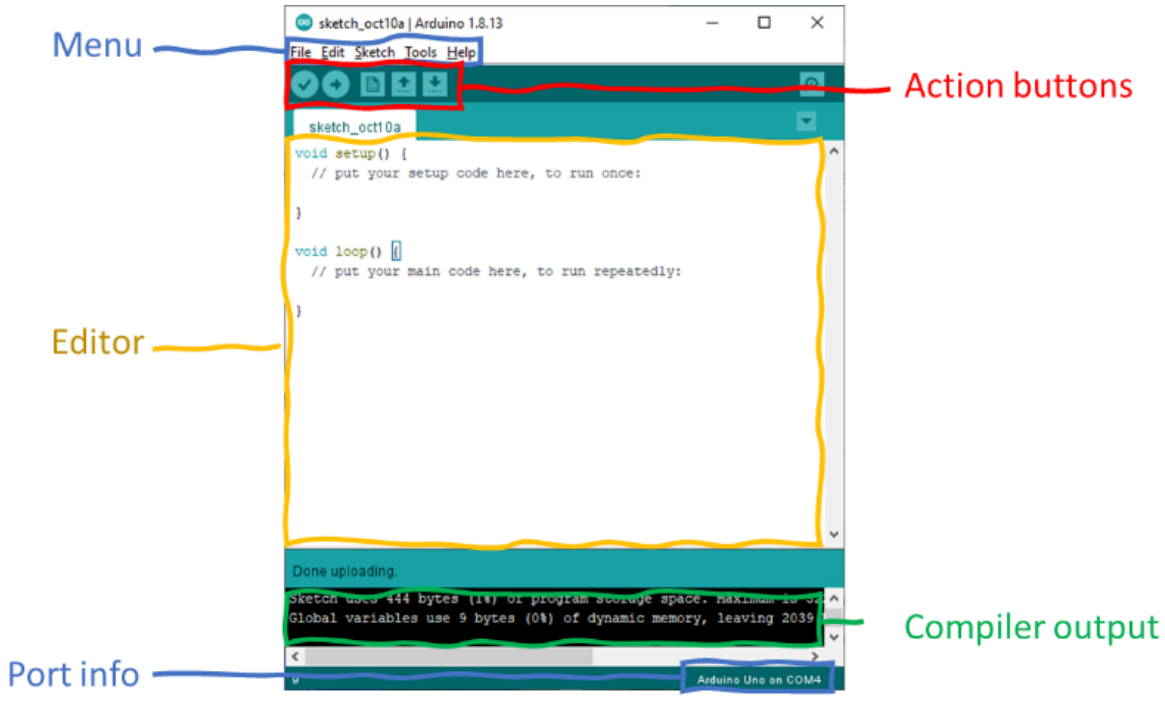


Figure 3-2: The Arduino IDE on launch

The main areas of the editor are the following:

- *Menu*: Provides access to the various functionalities and options available in the IDE.

[29] https://www.arduino.cc/en/Main/software

- *Action buttons*: provides quick access to frequently used actions. Specifically, to *compile/verify*, *upload*, create a *new project*, *open* an existing project and finally *save* the currently edited project.
- *Editor*: is the main area, where you type and edit the code for the targeted hardware.
- *Compiler output*: prints messages related to the compilation of the code ahead of uploading it to the targeted hardware.
- *Port info*: displays information about the targeted architecture (e.g. Arduino, ESP8266, etc.) as well as the port where the hardware is connected to.

3.2.2 Tour of the Arduino UNO

One of the most used microcontroller boards is Arduino UNO. It is popular because of its low cost and of the wide availability of online resources for it, including libraries for interfacing it with sensors and actuators, as well as code examples for both simple and complex projects.

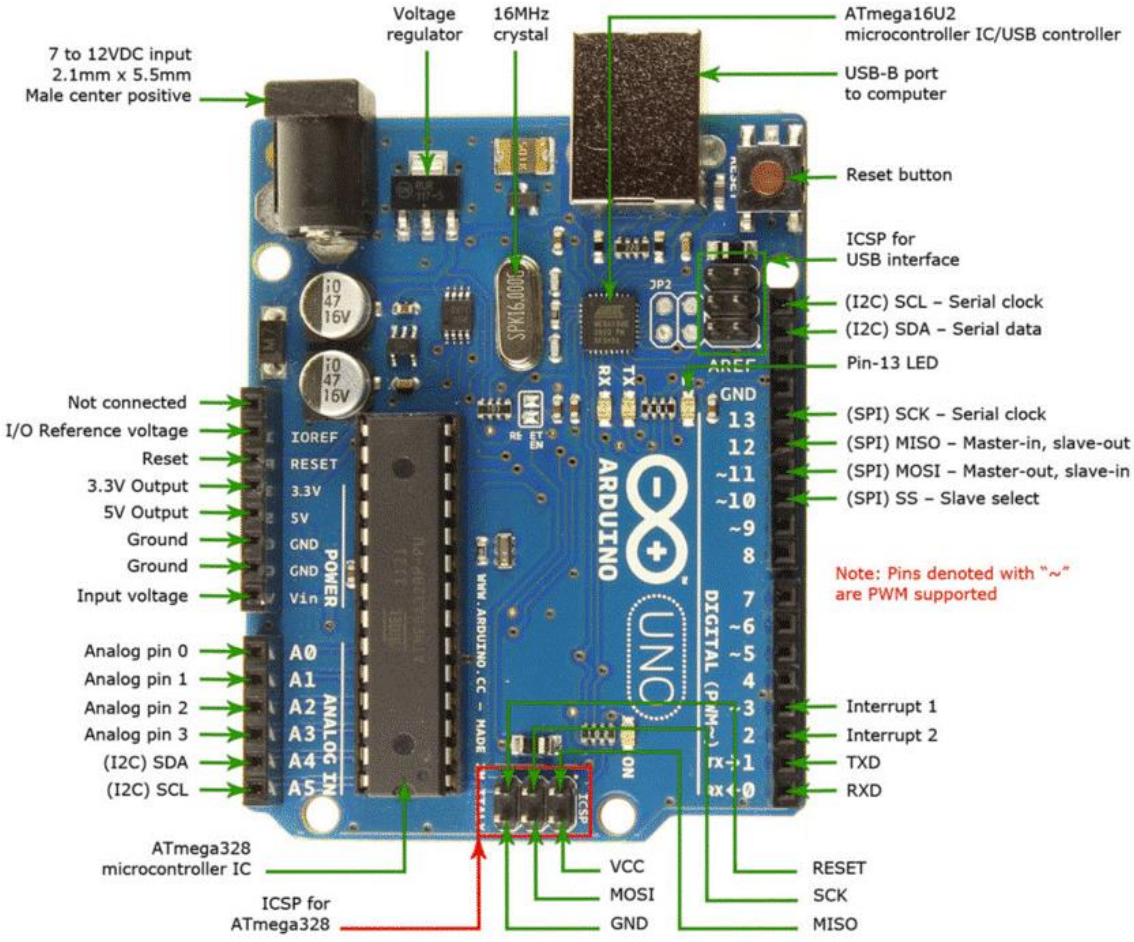


Figure 3-3: Schematic of a typical Arduino UNO (Jameco.com, 2015)

A typical view of the Arduino UNO board along with a description of its main components and pins is illustrated in Figure 3-3. Like most Arduino UNO and NANO boards, the main chip used in them is the ATmega328[30]. An external 7-12 Volt connector (top left) can be used to power up the board. However, while developing on a board, that is typically powered directly by the connected USB cable (top right). Also, at the top right corner, a reset button allows you to restart the board if needed—without the need to unplug and then plug to power again.

The remaining of the pins include methods for interfacing with external circuits. These include standard connections to the *Ground* (0 Volt), as well as to 3.3 Volt and 5.0 Volt. Several pins can be used for analog input[31], and the remaining for digital input or output[32].

Last, note that most boards also include a built-in LED light—in the above diagram fixed to pin 13. This can be quite handy, for instance for running simple programs or testing the *health* of the board, without requiring any external circuitry. This pin is used in the next subsection for demonstrating the simplest possible program on Arduino.

3.2.3 Hello (Blinking) World!

Traditionally, the first program when learning a new programming language or platform is the "Hello World", aiming to produce the simplest possible program, i.e., printing "Hello World" message to the standard output of the computer.

In the Arduino world, the role of this is taken by the "Blink" program, a simple listing of code where the built-in LED (at pin 13) alternates between ON and OFF every one second.

The "Blink"—and many popular, educational and useful programs—are available in the Arduino IDE. You can easily load any of them from the *File* menu, as shown in Figure 3-4.

Once loaded—or edited—the code can be deployed onto the board by pressing the *Upload* button—the one with the arrow pointing right in the *Action buttons* tab, shown in Figure 3-2.

[30] https://en.wikipedia.org/wiki/ATmega328

[31] The analog input pins allow to *read* values in a range of voltage, typically from 0 to 5 Volt. This can be used for example with a potentiometer to read a *range* of values.

[32] Unlike analog input, digital pins can be used to read (sense) or write (power) the pin to either of two values corresponding to 0 and 1 (or false and true). These are usually the values 0 Volt and 5 Volt respectively. Most digital pins can be used for either input or output, but their use must be determined programmatically.

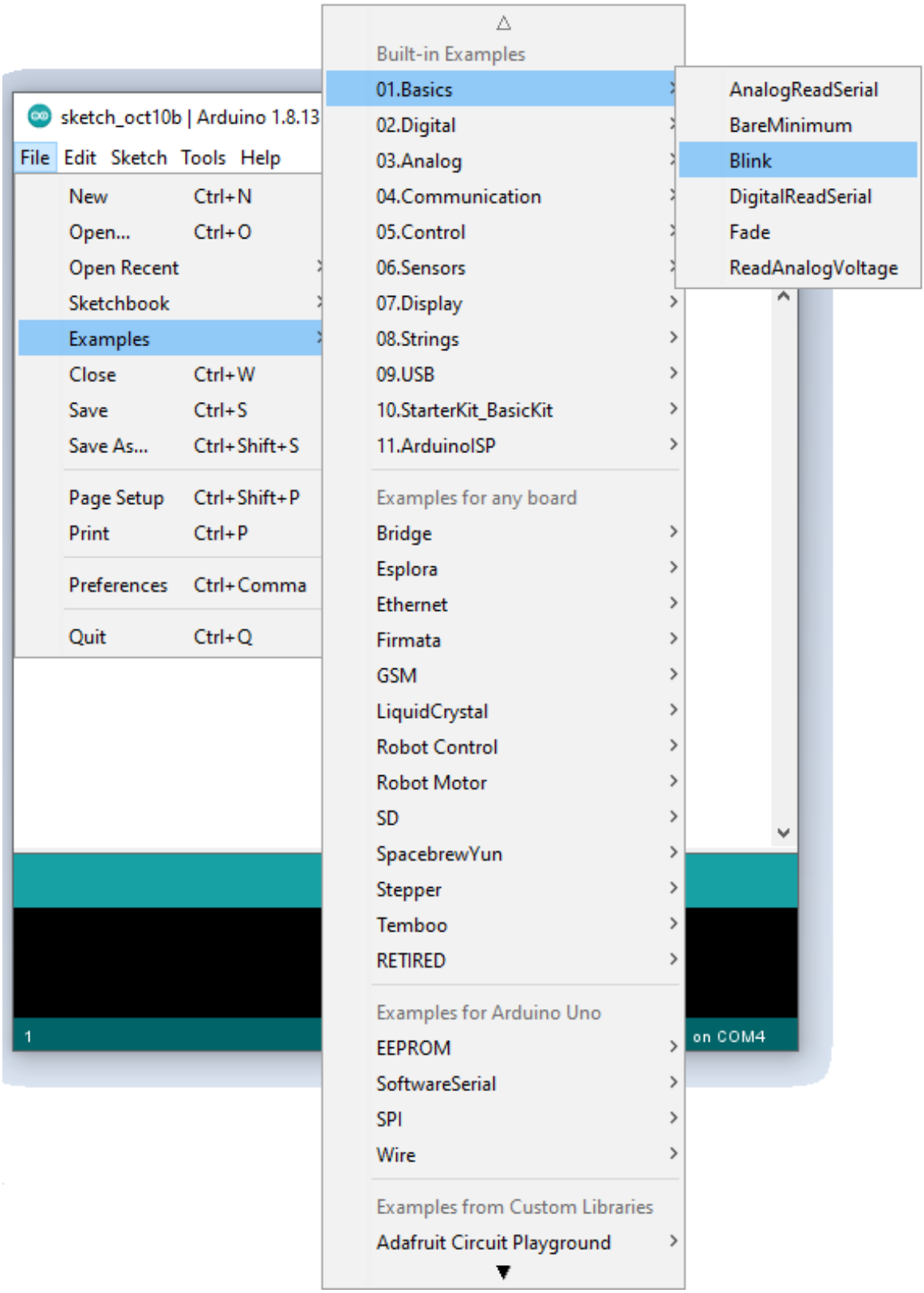


Figure 3-4: Opening Arduino IDE's built-in examples (in this case, "Blink")

The code of the Blink program is shown in Listing 20.

```
/*
  Blink: Turns an LED on for one second, then off for one second,
repeatedly.

  Most Arduinos have an on-board LED you can control. On the UNO, MEGA and
ZERO it is attached to digital pin 13, on MKR1000 on pin 6. LED_BUILTIN is
set to the correct LED pin independent of which board is used.
If you want to know what pin the on-board LED is connected to on your
Arduino model, check the Technical Specs of your board at:
https://www.arduino.cc/en/Main/Products

modified 8 May 2014 by Scott Fitzgerald
modified 2 Sep 2016 by Arturo Guadalupi
modified 8 Sep 2016 by Colby Newman
This example code is in the public domain.
http://www.arduino.cc/en/Tutorial/Blink
*/

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH); // turn LED on (HIGH voltage level)
  delay(1000);                     // wait for a second
  digitalWrite(LED_BUILTIN, LOW);  // turn LED off by making voltage LOW
  delay(1000);                     // wait for a second
}
```

Listing 20: The Blink code

The code shows the basic structure of any Arduino program. The main point is that it defines two special functions as follows:

- *Setup*: This function runs once, at launch—i.e. when the board is first powered up or reset. It is usually used to initialize variables, set up connections, and configure the IO pins—e.g. in this example it sets pin 13 (LED_BUILTIN) as OUTPUT.

- *Loop*: This function is run repeatedly. It is used to realize the *logic* of your program. Typically, this includes *reading values*, *processing* them, possibly *sending* them over a network connection, *using* them to make decisions locally, and finally to *control* the connected actuators. In this simple example the *loop* repeatedly turns on the LED at pin 13, waits for 1000 milliseconds—i.e. 1 second—and then turns it off and wait for 1 more second, etc.

Running the code causes the built-in LED connected to pin 13 to turn on for 1 second, then go off for 1 second, then back on for 1 second, etc. A view of an Arduino UNO-compatible board is shown in Figure 3-5—at the moment the yellow, built-in LED at port 13 is turned on.

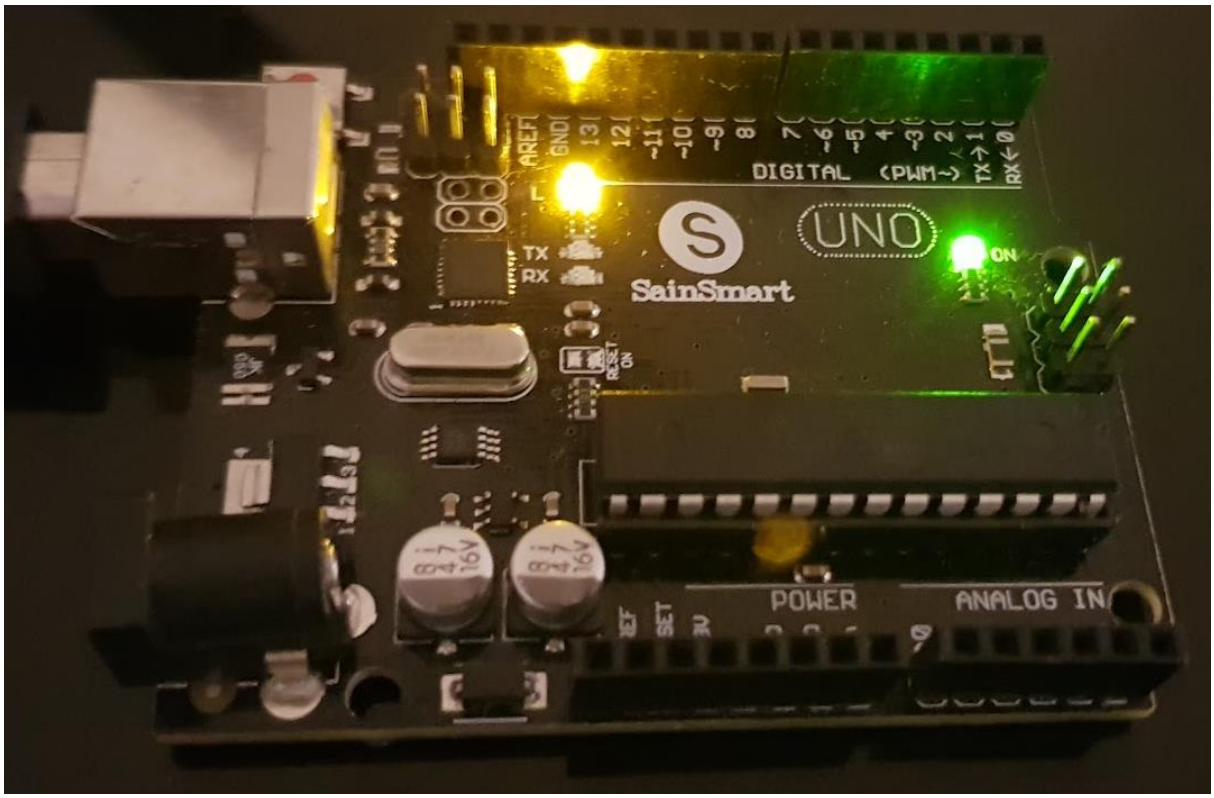


Figure 3-5: Arduino UNO-compatible board with the built-in LED turned on

3.2.4 Monitoring code execution and debugging

While the *Blink* is a trivial program, you can quickly get to a point which requires more careful coding as well as the ability to *monitor* the code execution and possibly *debug* it. As Arduino boards rarely have a standard display themselves, it is very useful to learn how to use the built-in *Serial Monitor* which enables the board to print some information directly in a console on your computer—with which the board is connected via a USB connection.

You can launch this console by selecting *Tools* from the menu, then *Serial Monitor*, as illustrated in Figure 3-6.

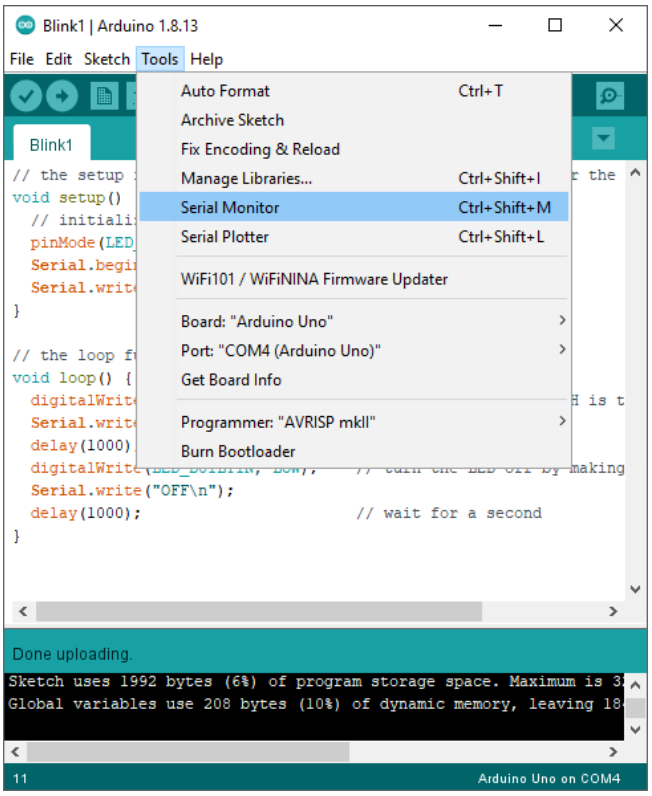


Figure 3-6: Launching the Serial Monitor, a console assisting the developers into monitoring and debugging their code

Inside this console, you can print messages directly, using the `Serial.write()` function. For example, the following statement prints the "Hello World!" message and moves the *cursor* to the next line—using the '\n' special character.

```
Serial.begin(9600);
```

Note that during *setup*, you need to initialize the Serial Monitor using the following statement (where 9600 is the rate with which the Arduino communicates with the connected computer):

```
Serial.write("Hello World!\n");
```

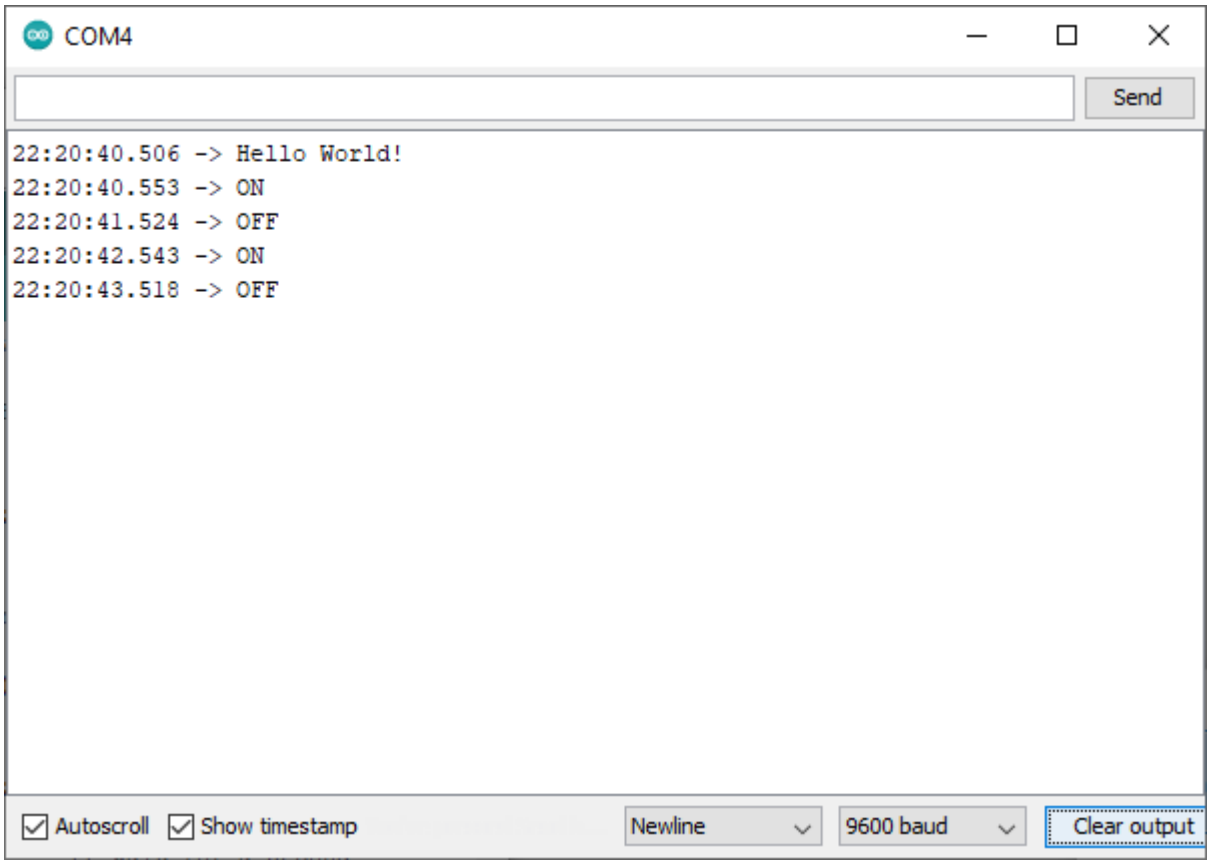
To demonstrate this functionality, edit the original *Blink* code to print messages on *setup* and then in each change of state in the LED. The resulting code is illustrated in Listing 21.

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
  Serial.begin(9600);
  Serial.write("Hello World!\n");
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH); // turn LED on (HIGH is the voltage)
  Serial.write("ON\n");
  delay(1000);                     // wait for a second
  digitalWrite(LED_BUILTIN, LOW);  // turn LED off by making voltage LOW
  Serial.write("OFF\n");
  delay(1000);                     // wait for a second
}
```

Listing 21: The edited Blink code to also print debugging info in the Serial Monitor

And when launched, the code produces output on the *Serial Monitor* as shown in Figure 3-6.



Listing 22: The Serial Monitor showing messages printed by the setup and loop functions

3.3 Examples

3.3.1 Simple traffic lights system

A slightly more advanced—but still simple—example would be a traffic lights system. This uses 3 LED lights—red, yellow and green—and simulates the succession of lights, i.e. red, then red and yellow, then green, then yellow, and back to red[33], as shown in Figure 3-7.

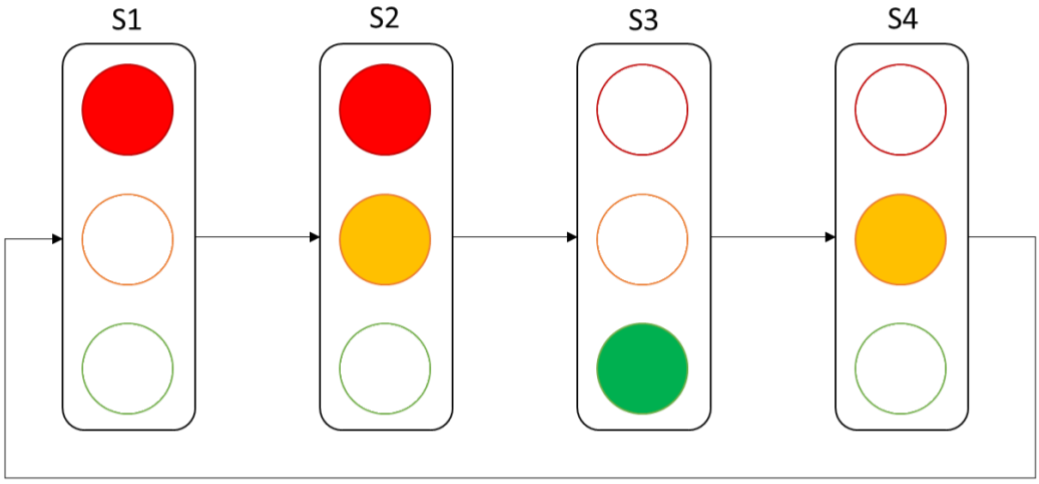


Figure 3-7: The four main states for a simple traffic lights system

Each of the three lights will be connected to one of the digital outputs, say green to 2, yellow to 3 and red to 4. For clarity, we define these as constants, as shown in the code listing.

Typically, states S1 (red) and S3 (green) stay on for a longer period, while transit states S2 and S4 are only activated for a shorter period. For simplicity, we use the same period for transits between any of the states, e.g., 1000 milliseconds.

3.3.1.1 Components, Connections and Code

Obviously, to implement this project we need an Arduino board, a breadboard, 3 LED lights (red, yellow, green), some resistors (e.g. 220 Ohm) to use with the LED lights[34], and some wires to form the circuit. The circuit is formed to realize the simple traffic lights is shown in Figure 3-8.

[33] While this is a typical pattern in many countries, there are places where the sequence is different. See https://en.wikipedia.org/wiki/Traffic_light.

[34] For LED lights, it is recommended that you connect them to Arduino with an inline resistor, otherwise the LED might draw too much current and damage itself and the board.

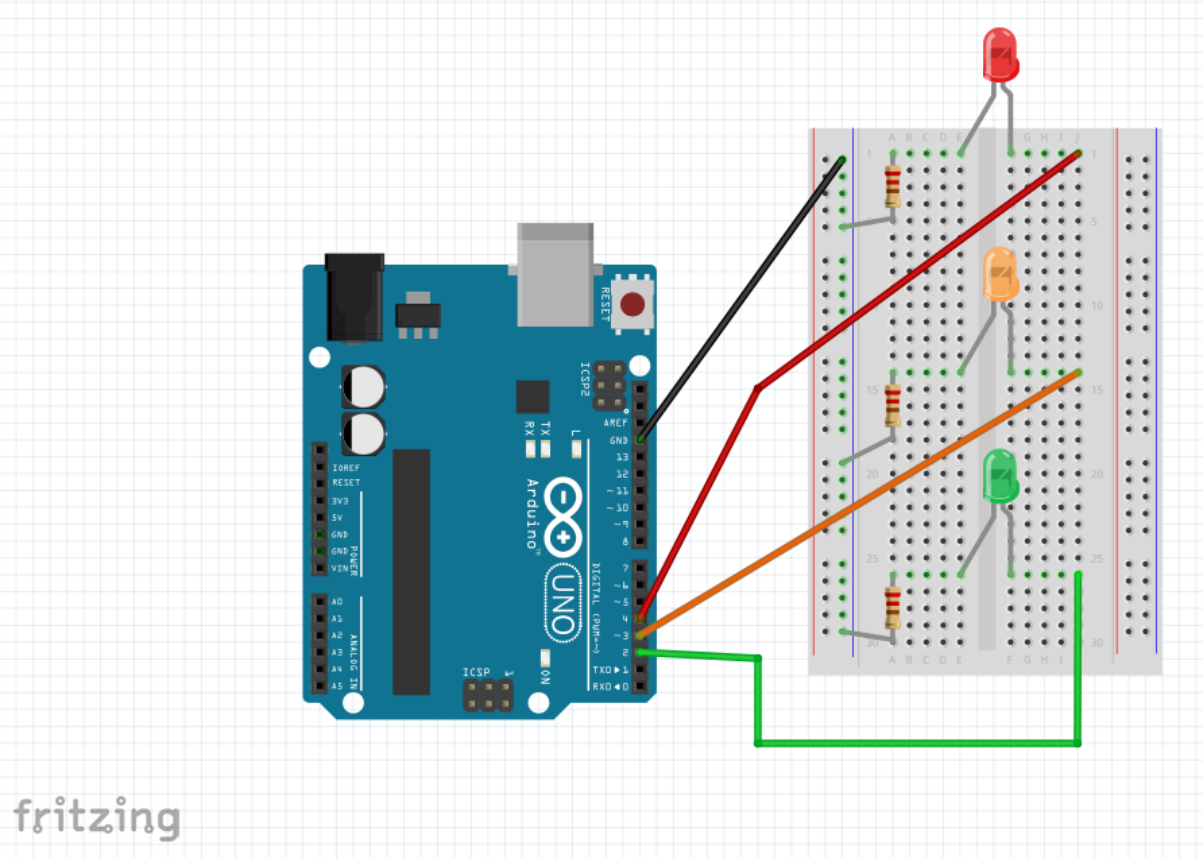


Figure 3-8: The circuit forming the simple traffic lights (designed using fritzing version 0.8.7)

The circuit is rather simple: Each of the 3 coloured LED lights is connected in line with a 220 Ohm resistor. Their anodes (longer pin) are connected to the corresponding ports on the board (i.e. port 2 for green, 3 for yellow and 4 for red). Their cathodes are connected via the resistor to the ground (GND port).

The source code is listed in Listing 23 and is described here:

- Each of the three ports is defined as a constant, integer value with the matching name.
- The state is defined as an integer value to describe the state in which the traffic lights system is in (as per the four states shown in Figure 3-7).
- The setup function initializes each of the three ports to the OUTPUT mode.
- The loop simply increases the state by one (initially it's zero) and then it uses that value to decide which state to activate, using a condition structure with multiple if and if-else statements. Each of the four states is activated with a corresponding function.
- When the state is 4, it resets to 0 (see state diagram in Figure 3-7).
- The last statement simply pauses the loop for 1000 milliseconds.

```
const int GREEN  = 2;
const int YELLOW = 3;
const int RED    = 4;

int state = 0;

void setup() {
  pinMode(RED, OUTPUT);
  pinMode(YELLOW, OUTPUT);
  pinMode(GREEN, OUTPUT);
}

void loop() {
  state++; // move to next state
  if (state == 1) {
    state1();
  } else if (state == 2) {
    state2();
  } else if (state == 3) {
    state3();
  } else { // assume state is 4
    state4();
    state = 0; // reset state
  }

  delay(1000);
}

void state1() {
  digitalWrite(RED, HIGH);
  digitalWrite(YELLOW, LOW);
  digitalWrite(GREEN, LOW);
}

void state2() {
  digitalWrite(RED, HIGH);
  digitalWrite(YELLOW, HIGH);
  digitalWrite(GREEN, LOW);
}

void state3() {
  digitalWrite(RED, LOW);
  digitalWrite(YELLOW, LOW);
  digitalWrite(GREEN, HIGH);
}

void state4() {
  digitalWrite(RED, LOW);
  digitalWrite(YELLOW, HIGH);
  digitalWrite(GREEN, LOW);
}
```

Listing 23: The code for the simple traffic lights

This code can be compiled and uploaded to the board, using the corresponding buttons in the "Action buttons" panel—see Figure 3-2.

Once uploaded, the code starts running, and the lights turn on and off as per the state diagram. A view of an actual such system is shown in Figure 3-9.

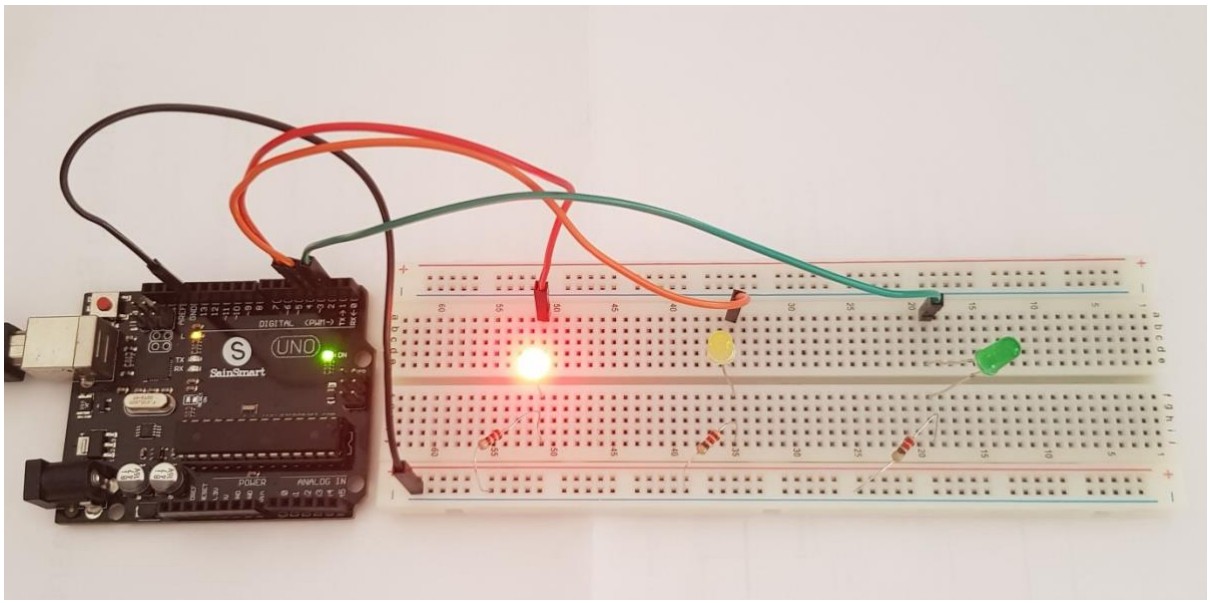


Figure 3-9:View of the simple traffic lights circuit

3.3.2 Adaptive traffic lights system

While the traffic lights system described in section 3.3.1 provides a view of forming a circuit, interfacing it with Arduino, then manipulating it with code, it does not demonstrate how you can read input values from external sensors. In this section, we extend the previous example with a method where the delay (time between states) can be adapted using a hardware component (a potentiometer).

3.3.2.1 Components, Connections and Code

As this example builds on the previous one, it only needs a few additional components: A potentiometer and some wires to connect it to the board.

A potentiometer is an *analogue* component, essentially a variable resistor. By connecting its endpoints to the min (ground) and max (5V) endpoints, we can use one of the analogues input pins of the board to receive input from the user. Specifically, the potentiometer is connected to GND and 5V, as well as to analogue input port 0, as shown in Figure 3-10.

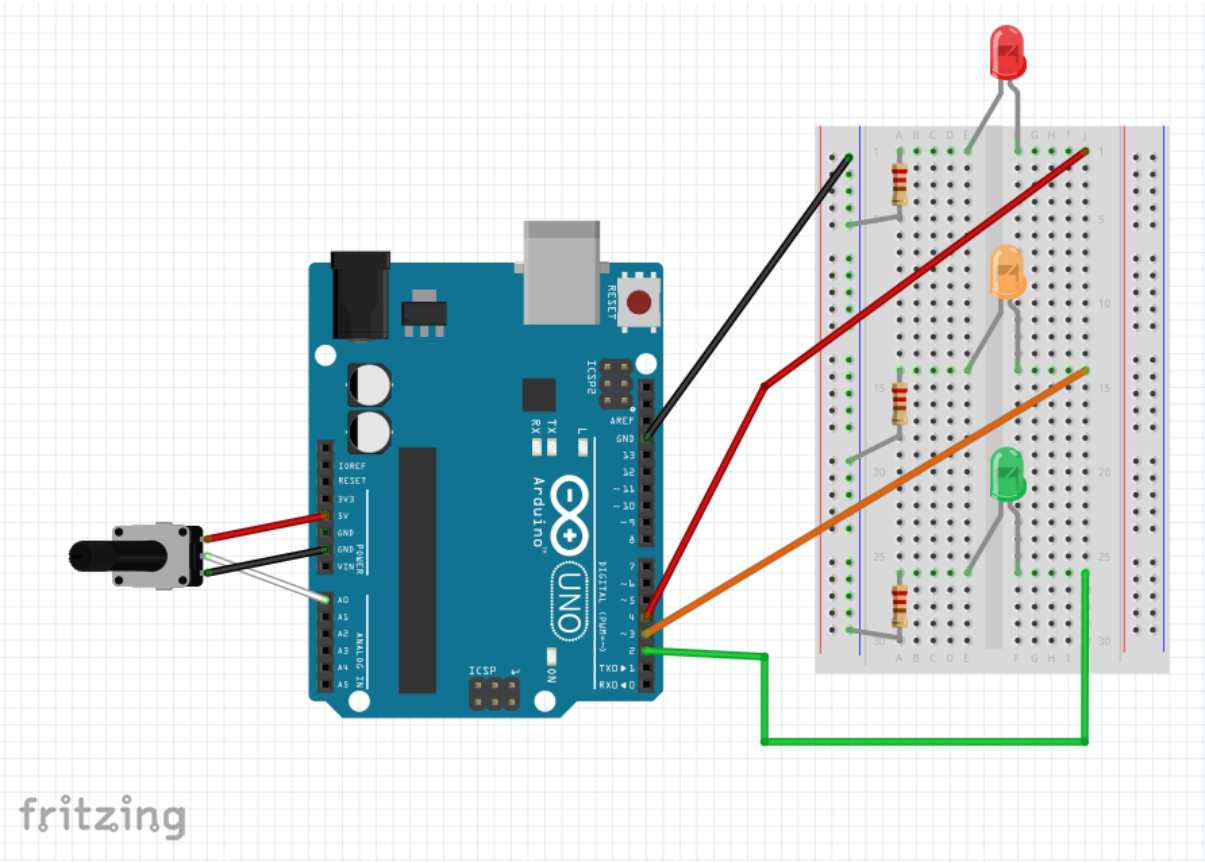


Figure 3-10: The circuit forming the adaptive traffic lights system with a potentiometer to control the delay (designed using fritzing version 0.8.7)

Note that the analogue input ports are used to read voltage. The input range is from 0 (GND) up to a max of 5V. From within the code, this value is accessed using the `analogRead` function, which itself returns an int in the range 0 to 1023 (where 0 corresponds to 0V and 1023 to the max value of 5V).

```
const int RED = 4;
const int YELLOW = 3;
const int GREEN = 2;

const int POT_IN = 0; // port for analogue input

int state = 0;

void setup() {
  pinMode(RED, OUTPUT);
  pinMode(YELLOW, OUTPUT);
  pinMode(GREEN, OUTPUT);
}

void loop() {
  state++;
  if (state == 1) {
    state1();
  } else if (state == 2) {
    state2();
  } else if (state == 3) {
    state3();
  } else { // assume state is 4
    state4();
    state = 0; // reset state
  }

  int val = analogRead(POT_IN); // 0 to 1023
  delay(100 + val*4);
}

// the code for the state1-4 functions is the same and omitted for brevity
```

Figure 3-11: The code for the adaptive traffic lights

The main changes in the code are as follows:

- The POT_IN constant is defined for marking that the potentiometer is connected to analogue input port 0.
- In each loop, the value of the potentiometer is read (in the variable `val`), and that value, which is in the range of 0..1023, is used to decide the delay. At a minimum, the delay is 100 milliseconds (i.e., 0.1 seconds), and at max it is 4192 milliseconds (i.e. 4.192 seconds).

When implemented, the adaptive traffic lights work as before—based on the state transitions shown in Figure 3-7–but the delay between each state is controlled via the potentiometer. The completed circuit is shown in Figure 3-12.

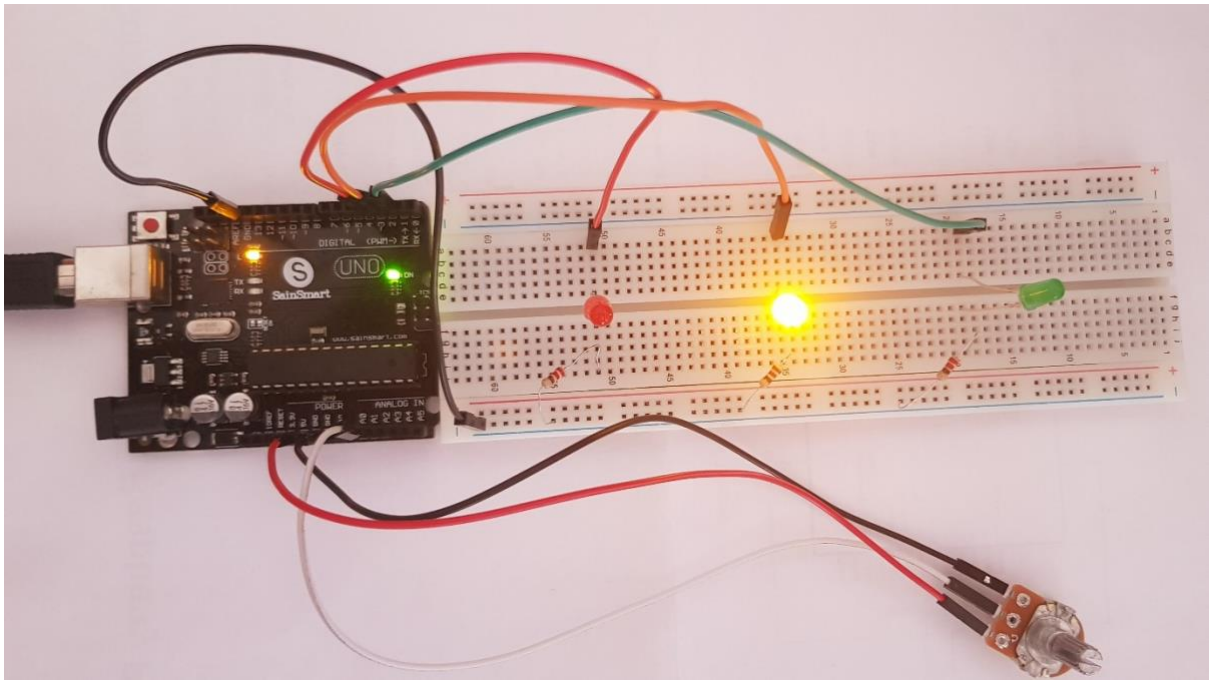


Figure 3-12: View of the adaptive traffic lights circuit

3.4 Additional Resources

Because of its advantages—*inexpensive*, *cross-platform*, *simple*, *open-source*—Arduino is widely popular with many online resources, including tutorials and sample projects. You can find more about Arduino online[35].

3.4.1 Arduino simulator

While Arduino boards and accompanying electronic components are relatively inexpensive and easy to use, it often makes sense to first prototype systems using a software-based simulator. Additionally, it is often faster to use a simulator to form a circuit and test it with some code, compared to using real, hardware components.

One such simulator is the TinkerCad tool, which was introduced in 2011 for 3D modelling, but has since been extended to allow further functionality, notably Circuits simulation.

AutoDesk's TinkerCad is a web-based tool[36]. To access it, it requires that you sign-up for a free account. From its dashboard, you need to select the "Circuits" option. While TinkerCad

[35] https://www.arduino.cc/en/Guide/Introduction
[36] https://www.tinkercad.com

provides a plethora of basic electronic components, it also features Arduino UNO boards, along with the ability to define their code.

A view of the editor, implementing the adaptive traffic lights system example of section 3.3.2, is illustrated in Figure 3-13.

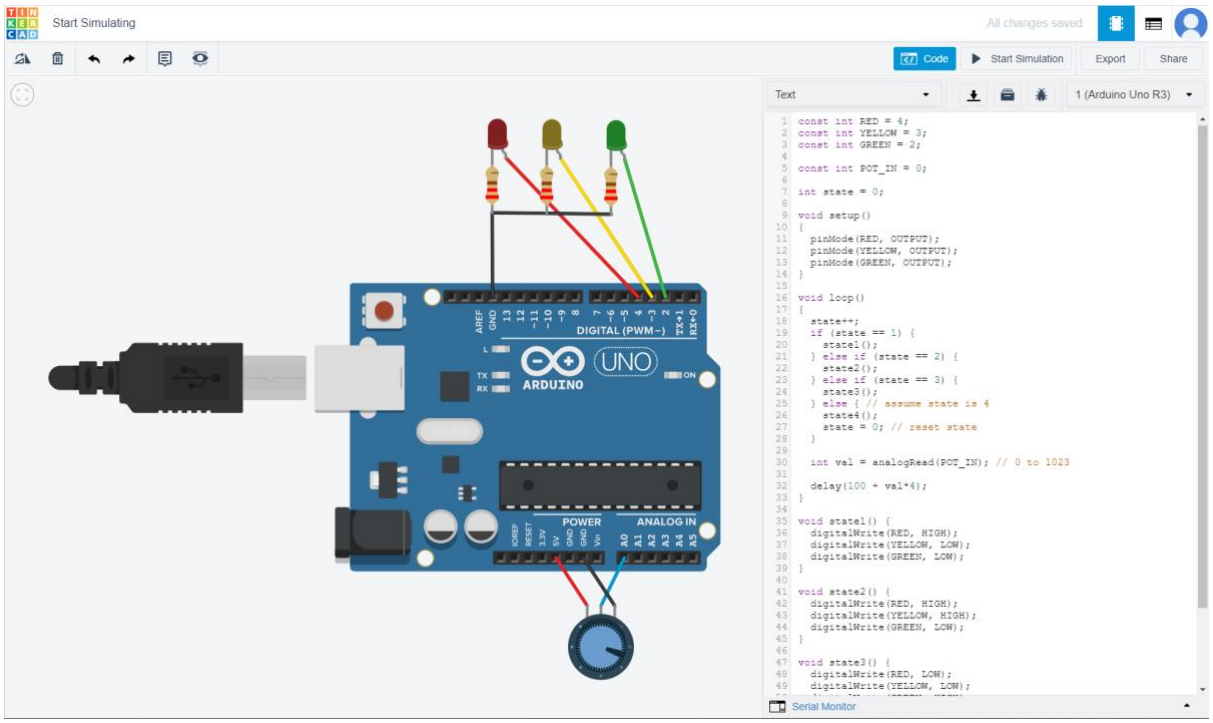


Figure 3-13: TinkerCad implementation of the adaptive traffic lights system

3.4.2 Online tutorials and examples

If you want to learn more about the Arduino IDE (Integrated Development Environment), you can refer to the online reference guide[37], which can also be accessed offline: From the menu, select “Help”, then “Environment”.

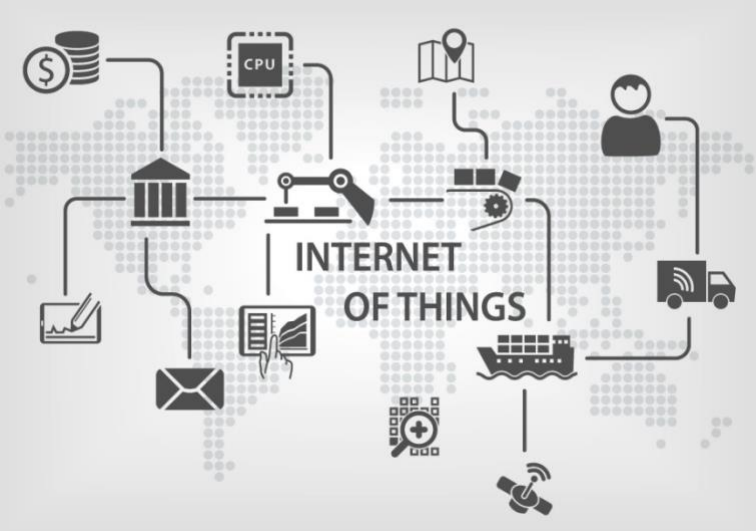
Arduino also offers a convenient *language reference* covering the functions, values and structures of the C language variation used in Arduino. This reference is available online[38], but can also be accessed offline: From the menu, select “Help”, then “Reference”.

Finally, perhaps the best way to further advance your knowledge and skills is by reading and recreating the built-in Arduino examples[39].

[37] https://www.arduino.cc/en/Guide/Environment
[38] https://www.arduino.cc/reference/en/
[39] https://www.arduino.cc/en/Tutorial/BuiltInExamples

4 IoT architecture and components (1 of 2)

Author(s): Marios Raspopoulos
Stelios Ioannou



4.1 Introduction

Internet of Things has become a very popular topic of research and Innovation mainly due to the ubiquitous transformation of computing. Physical devices have become "smart" being able to sense, communicate in a pervasive way and interact with their environment offering useful applications and solutions to the humankind. Today they find applications in wide range of activities like Health, Transportation, Agriculture, Home and Industrial Automation, Retail and many more. The 2005 ITU Internet Report [4] adds a 3rd dimension to the legacy "ANY PLACE" and "ANY TIME" communication; the "ANY THING" communication as shown in Figure 4-1. This has changed the way we perceived the word "telecommunication" to communication between everything rather than communication between people only. This meant that there would be expected an exponential growth of network connections which should be facilitated by powerful networks. A study by Cisco in 2011 [5] predicted that there will be 25 billion devices connected to the Internet by 2015 and 50 billion by 2020.

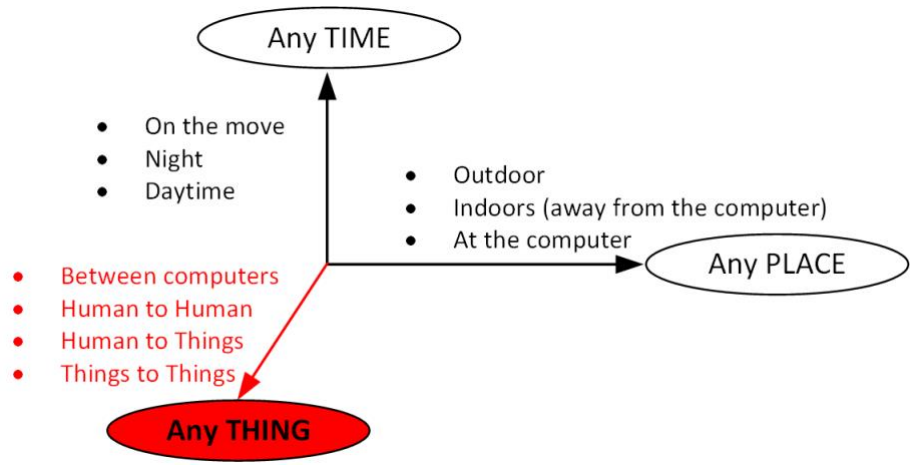


Figure 4-1: The new Dimension introduced in the IoT [4]

To ensure connectivity and interoperability it important that there should exist a reference IoT architecture upon which all IoT applications would be based upon. Nevertheless, there is not a consensus on a single IoT architecture, globally agreed. Literature mainly reports two architectural models for IoT; (1) a 3-layer architecture and (2) a 5-layer [6] [7] architecture but also some specific purpose architectures. In Parallel with the research efforts reported in literature the International Telecommunications Union (ITU) has started in 2012 [8] an effort to standardize the functional architecture model for IoT.

In this chapter the most important architecture models reported in literature and the ITU-T IoT Reference model are overviewed and the most important hardware and software components are identified.

4.2 Characteristics and Requirements of the IoT

4.2.1 Useful Definitions

Before describing the architecture, requirements and models it is important to establish good understanding about some IoT-related definitions

- ***Device***: In the IoT context, this is a piece of equipment must be able to communicate and could optionally sense, act, capture data, store data or process data. Its only mandatory capability is the communication.
- ***Thing***: An object inside the IoT system which is capable of being identified and integrated into communication system.
- ***Physical Thing***: An object of the physical world which is able of being sensed, actuated and connected is known as a physical thing (e.g. industrial robots, electrical equipment etc.)
- ***Virtual Thing***: An object in the information world capable of being stored, processed and accessed is known as virtual thing. For example, multimedia content, application software etc.
- ***Internet of Things***: A global information infrastructure which enables advanced services by interconnecting Things (Physical and/or virtual) based on existing and/or evolving interoperable technologies. The IoT includes functions for identification, data capture, processing, and communication to offer different kinds of applications whilst ensuring security and privacy.

4.2.2 ITU-T Technical Overview of the IoT

Figure 4-2 shows the technical overview of the IoT. A physical thing can be mapped (or represented) by one or more virtual things in the Information domain. Information is being collected by physical devices (or things) in the physical world and is Communication Networks and the Information domain for further processing. Devices may communication with each other either via the communication network (with or without a gateway) or directly without

using the communication network or combinations of these communication links. Exchange of information not only happens between physical things in the Physical world but also between virtual things in the Information World.

The communication networks provide capabilities for reliable and efficient data transfer. The network infrastructure may be implemented or realized via existing networking technologies (e.g., TCP-IP networks) or evolving networks following the current telecommunication trends.

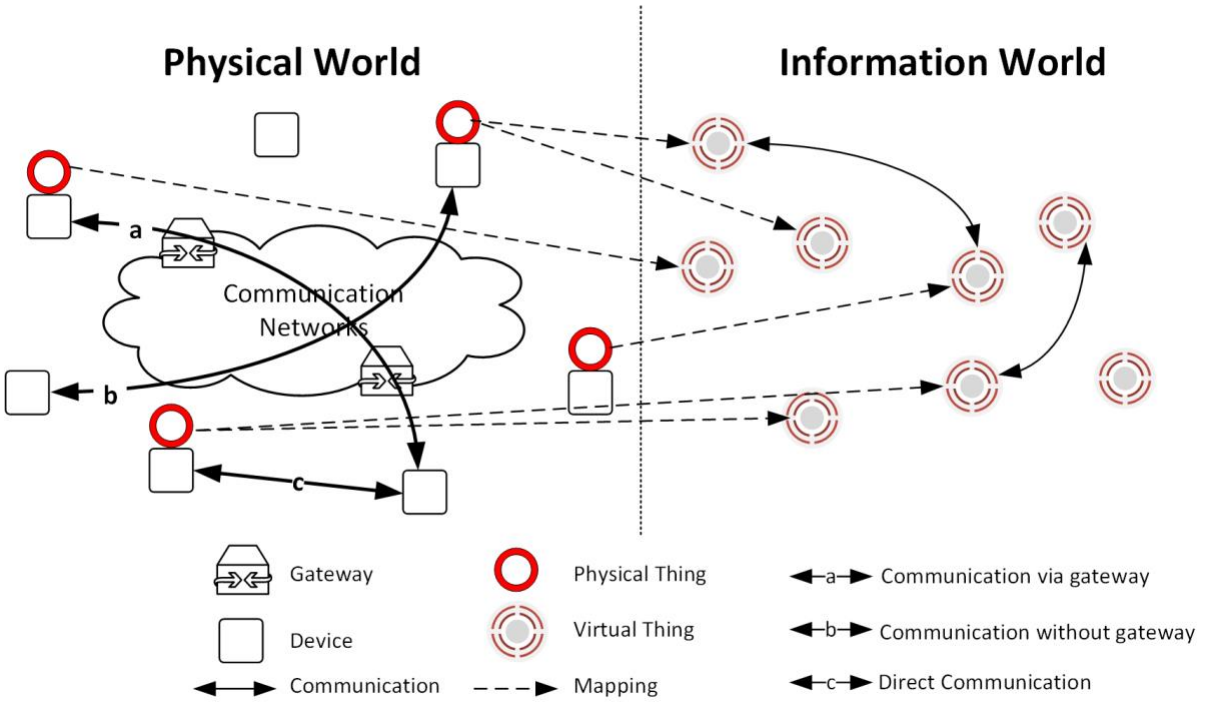


Figure 4-2: Technical Overview of the IoT

4.2.3 Types of Devices

The ITU-T has identified [8] three types of devices in an IoT System and defined their relationships with physical things (see Figure 4-3). As previously mentioned, the minimum requirement for a device in an IoT system is to be able to communicate. Having this in mind, devices are categorized in three main categories:

1. ***Data-Carrying Device***: A device which is directly attached to a physical thing to indirectly connect it the communication network.
2. ***Data-Capturing Device***: A device with reading/writing functionalities capable of interacting with the physical things either directly via data carriers attached to the physical thing or indirectly via a data-carrying device. In the latter case, the data-

capturing device reads the data on a data-carrying device and optionally can write data from the communication network on the data-carrying device. Communication between data-capturing and data-carrying devices can be achieved using Radio Frequency (RF), Infrared (IR), optical and galvanic driving.

3. ***Sensing and Actuating Device***: A device capable of detecting and measuring data within its environment and digitize it. Inversely, it can convert electronic signals from the communication network into actions/operations. Typically, this kind of devices communicate with each other either wirelessly or through wires on a local network and use gateways to connect between different networks.

Generally, a general device has embedded processing and communication capabilities (wired or wireless) and may included equipment or appliances depending on the application domain they are used in (e.g. industrial machines, home electrical appliances, smart phones etc.)

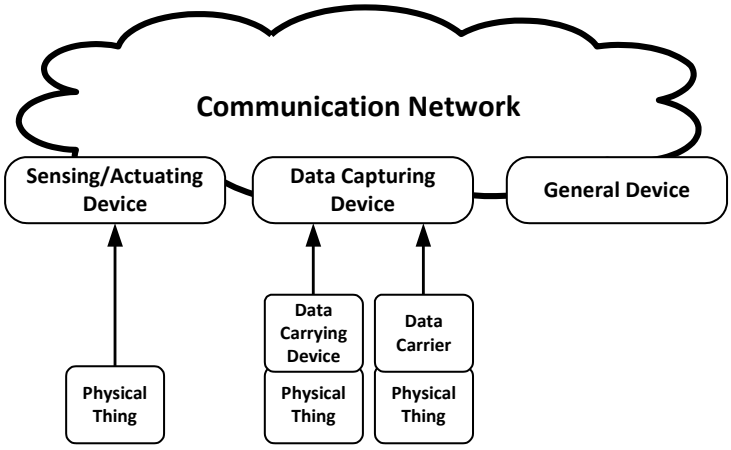


Figure 4-3: Types of Devices and their relationship with Physical Things [8]

4.2.4 Fundamental Characteristics of the IoT

The ITU-T has identified in [8] the fundamental characteristics of IoT systems:

- ***Interconnectivity***: Any IoT device can be interconnected with the global Information and Communication Infrastructure.
- ***Things-related services***: The IoT provisions services which concern the connected "things" within their constrains such as privacy protections and semantic consistency between physical things and their associated virtual things. To provide these thing-related services within the constraints of things requires that both the underlying technologies and the physical and information world change.

- ***Heterogeneity***: Heterogeneous IoT devices with different hardware and networking characteristics get connected and interact with other devices or platforms on various types networks.
- ***Dynamic Changes***: While roaming and interacting in an IoT system, devices change their state dynamically. For example, sleeping and waking up, get connected or disconnected while changing their location and speed. Additionally and equally important the number of connected devices changes dynamically.
- ***Enormous scale***: Usually the number of devices that need to be managed and that of the devices that communicate with each other is significantly larger than the ones that connected to the Internet. This practically means that the communication initialized by devices is much higher than the one that is initialized by humans. Even more important is the management and the analysis of the data generated. This relates to semantics of data, as well as efficient data handling.

4.2.5 IoT Requirements

Based on the above characteristics the ITU-T has defined in [8] a set of high-level IoT System Requirements for the development of an IoT Reference Model:

- ***Identification-based connectivity***: There needs to be a support for the "Things" to be connected to the IoT based on their identifiers. This includes a unified processing of identifiers which might be heterogeneous.
- ***Interoperability***: Interoperability between heterogeneous and distributed systems needs to be ensured so that a variety of information and services is supported.
- ***Automatic Networking***: The IoT network infrastructure should provide control functions for automatic networking including self-management, self-configuration, self-healing, self-optimization and self-protection, to be able to support and facilitate adaptation in different application domains, different communication environments and larger number and types of devices.
- ***Autonomic services provisioning***: Services need to be provided by automatically capturing, communicating and processing of the data of the "Things" according to the rules configured by the operators and/or configured by the subscribers. This

autonomic service provisioning needs to base on data fusion and data mining techniques.

- ***Location-based capabilities***: Localization is a key enabling technology in IoT as location-based services must be supported. Things should be able to track their position to facilitate the provision of services which depend on their location. Location-based communication and services may be constrained by Regulations and Laws and should comply with security requirements.
- ***Security***: The ability of any Thing to connect at any time and any place generates significant security threats against CIA (Confidentiality, Integrity and Authenticity) for both data and services. Therefore, there is an important requirement to integrate different security policy and measures related to the things and their communication in an IoT framework.
- ***Privacy protection***: Data acquired by "Things" may contain private information of their owners and/or their users. Therefore, it is important that privacy protection is supported during transmission, aggregation, storage, mining and processing of this data while not setting a barrier to data source authentication.
- ***High quality and highly secure human body related services***: Services which are based on the capturing, communicating, and processing of data related to human behaviour (e.g. exercise, health, location etc.) automatically or through human intervention should be offered while guaranteeing high quality, accuracy and security.
- ***Plug and Play***: It is important for IoT systems to support plug and play capability in order to enable or facilitate on-the-fly generation, composition and acquisition of semantic-based configurations to seamlessly integrate an internetwork of things with the respective applications and efficiently respond to these applications' requirements.
- ***Manageability***: Applications in an IoT systems usually need to work automatically without the intervention or participation of people and therefore the whole operation process needs to be manageable by the relevant entities in order to ensure normal network operations.

In addition to all the above **scalability** is also an important IoT requirement. Any IoT architecture should be highly scalable and be able to support a very large and progressively

increasing number of devices that are constantly sending, receiving, and acting on data. This kind of architectures usually come with an equally high price – both in hardware, software, and in complexity. So, it is important for any architecture to support scaling from a small deployment to a very large number of devices or vice-versa. Additionally, elastic scalability and the ability to deploy in a cloud infrastructure are essential.

4.3 IoT Architectures

4.3.1 3-Layer Architecture

The 3-layer IoT Architecture [6] [9] [10] [11]shown in Figure 4-4 is the most basic IoT architecture. It was introduced for the first time in 2009 [9] at the early stage of the IoT Research.

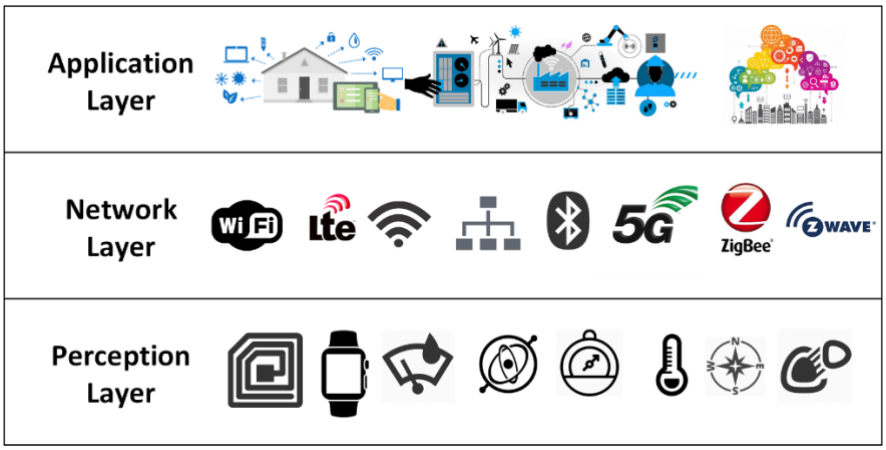


Figure 4-4: 3-Layer IoT Architecture

The three layers of this architecture are:

- ***Perception Layer***: Provides the mechanisms (sensors) through which the Things perceive their environment. This is analogous to the nerve endings of a human being like the eyes, ears, nose, skin, etc. It includes sensing devices that measure different parameters or conditions in their surrounding environment (e.g. thermometers, humidity sensors, inertial sensors etc.). In addition to its sensing capabilities this layer includes functions so that other smart objects in the environment are found and identified (e.g. RFID). In summary its main functions are to recognize things and collect information.

- ***Network Layer***: This layer is responsible for the connectivity of Things to other Things, to network devices (e.g. routers, access points etc.), to servers and to the Internet. In this context it includes functions to connect, associate, authenticate to the attached node, transmit the collected information, and/or receive actions to be performed by the Thing from the attached network. Network Layer is implemented using the current but also the evolving network and mobile technologies (e.g. IEEE802.11 standards, 4G, 5G, Zigbee, Bluetooth etc.) but also different types of networking and data collection protocols (e.g. TCP/IP, MQTT, etc). Besides connectivity and network operations, this layer includes information operations to store and process (or analyse) the massive collected information. Finally, it includes management operation for the seamless and flawless operation of the integrated IoT system.
- ***Application Layer***: This layer is responsible for the delivery of the application services to the users/subscribers. It is responsible of utilizing the collected context from the layers below to deliver intelligent applications to the end users (e.g. smart-home, e-health, smart-transport etc.). It is the final goal of the IoT system which consolidates the input from the underlying technologies to offer useful and user-friendly applications to the users. It therefore mostly includes intelligent software development functions. It can be seen as the means to converge between the social IoT needs and the industrial technology in such a way as to have a broad impact on the global or local economic or social development.

The 3-layer architecture is simple and defines the main idea about IoT but it is not sufficient for research and innovation purposes as research focuses on finer and more detailed aspects of IoT. For this reason, literature reports many multi-layer (more than 3) architectures depending on the IoT application domain. On of them is the 5-layer architecture.

4.3.2 5-layer Architecture

In the 5-layer architecture [12] [6] [10], the Perception and the Application Layers are the same as in the 3-layer one while the Network Layer is renamed to Transport Layer (see section 4.3.1) but 2 new layers are added:, the processing layer and the business layer.

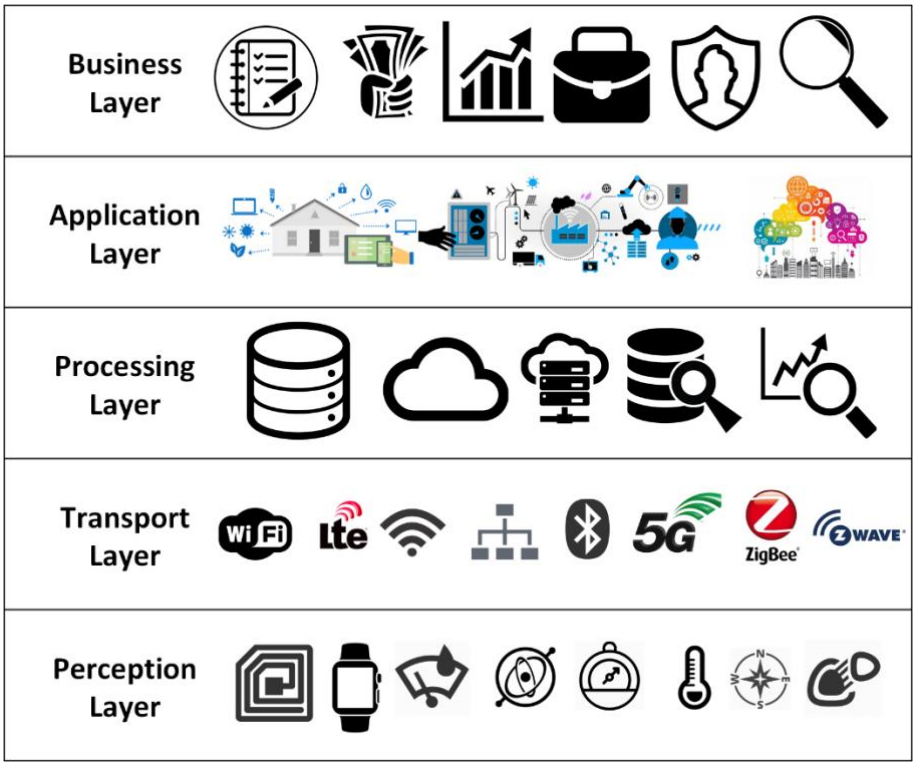


Figure 4-5: 5-layer IoT Architecture

- ***Business Layer***: It is responsible for the management of the whole IoT system, including the business and profit models, the charging, and the privacy of the users. This layer is also concerned with the research and development in the IoT domain.
- ***Processing Layer***: Also known as the middleware layer, the processing layer is responsible for the storage and the analysis of the data collected at the perception layer and communicated over the transport layer. It includes databases, cloud storage and computing capabilities, data analysis modules etc.

4.3.3 Cloud and Fog-Based Architectures

In the two architectures discussed above (3-layer and 5-layer) the discussion was mostly focused on the technologies rather than on the way the data is being collected or processed. Depending on where the processing done, IoT architectures can be classified as either cloud-based or Fob-based [10].

In ***cloud-based architectures***, [13] processing is done centrally at cloud computing servers. This is a cloud-centric approach where all the applications are built around using the communication network to convey the data back and forth. This kind of approach offers the benefits of flexibility and scalability. IoT development can be done using storage tools, data

mining and machine learning tools, visualization tools and others that are available on the cloud. A conceptual IoT framework with cloud computing at the centre is shown in Figure 4-6.

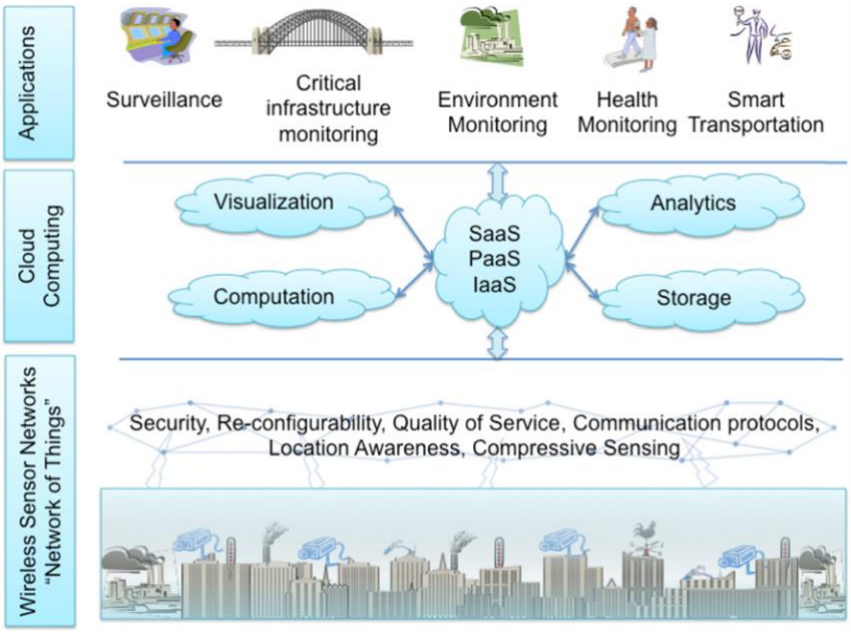


Figure 4-6: Conceptual IoT framework with Cloud Computing at the Centre [13]

In ***fog-based architectures*** [14] [15] [16] the sensors as well as the network gateways do part of the processing and the analysis of the data. In fog computing approaches the capabilities of the cloud computing are extended to the edge of the network which due to the localization of the data the latency is significantly reduced allowing the fast delivery of real-time data and the provision of low-latency and delay-sensitive applications (e.g. real time streaming, e-health applications etc.). As some pre-processing is done at the sensors or the smart gateways before reaching the central cloud there might be interoperability and transcoding problems to solve. This Fog and Smart Gateway-based communication is shown in Figure 4-7. The Gateway in this approach has an enhanced role linking the IoT with the Fogs and the Cloud. A layered architecture of this Smart gateway is shown in Figure 4-8.

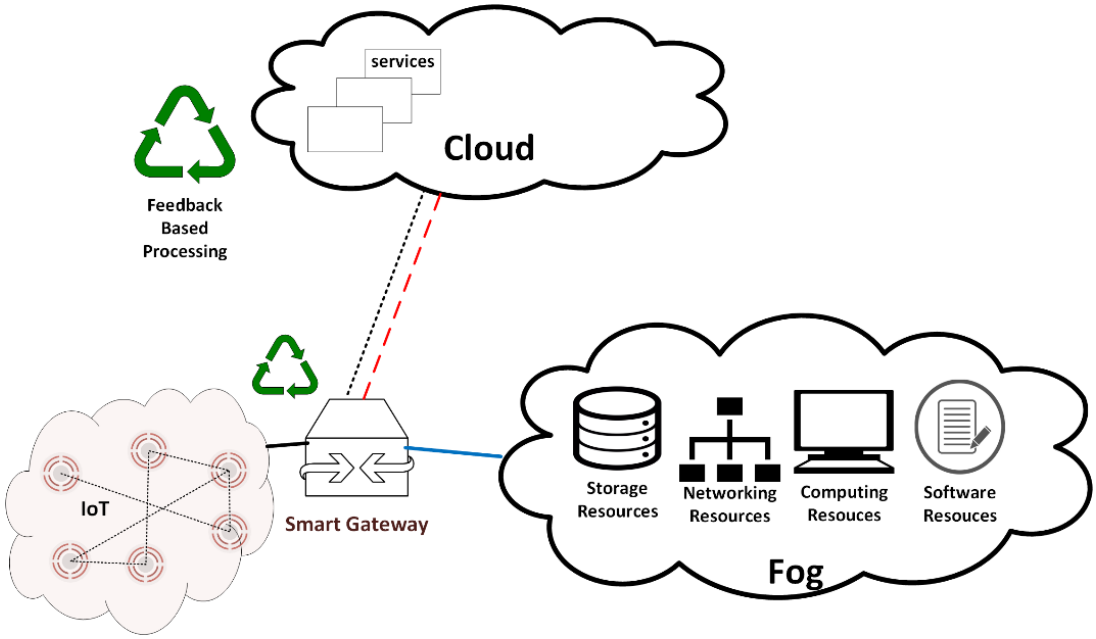


Figure 4-7: Smart Gateway with Fog Computing/Smart Network [16]

Transport Layer
Security Layer
Temporary Storage Layer
Preprocessing Layer
Monitoring Layer
Physical and Visualization Layer

Figure 4-8: Layered Architecture of Smart Gateway in a Fog-based IoT

The *Physical Layer* includes all the physical and virtual Things as well as the physical and virtual networks that interconnect them. The *Monitoring Layer* is responsible for the monitoring of the activities of the nodes and networks in the physical layer. This includes an orchestration and management of activities like which nodes is performing tasks, what kind of task, at what time, what is their required output and input etc. It also includes power monitoring, resource monitoring, response monitoring and service monitoring. The *Pre-processing* layer is responsible of the tasks related to data management such as analysis of the collected data, data filtering, reconstruction and trimming in order to generate more

meaningful and useful data for further processing (typical example is the analysis of inertial data from accelerometers, magnetometers and gyroscope in order to extract navigation information like direction of movement, speed, orientation, acceleration etc. The role of the *Temporary Storage Layer* is to temporarily store the data generated by the Pre-Processing Layer on the Fog resources. This data is kept on the Fogs only until is uploaded on the cloud and then it is deleted. Since there might be generation of private and sensitive data at the underlying layers (e.g. in healthcare, location, military IoTs) there should be functionality to provision security. This is the role of the security layer which includes encryption/decryption, privacy, authentication and integrity measures. Finally, the *Transport Layer* is responsible for the uploading of the pre-processed and secured data to the cloud.

4.3.4 Social IoT

The Social IoT (SIoT) paradigm [17] is based on the notion of social relationships amongst the objects of the IoT systems. This is analogous to the way that people establish social relationships with one another. This approach has 3 main benefits:

- Navigability: Objects can easily discover other objects and establish connections between them easily and in a very scalable way.
- Trustworthiness between friendly objects: Object connected to each other in a friendly relationship can a establish a level of trustworthiness between them.
- Social Networks already in place for humans can be re-used and stented to apply for IoT related solutions and applications

The SIoT model adopts the human social networking approach but it extends it in order to become applicable in the IoT world. In this context the most important components of a SIoT architecture are:

- ***ID Management (ID)***: Objects need to be identifiable therefore an ID is assigned to each object based on typical parameters like MAC address, IPv6 address, the product code etc.
- ***Object Profiling (OP)***: The profile of each object is comprised of information about that object which allows the organization of objects into classes based on their features.

- ***Owner Control (OC)***: the owner defines specific policies regarding the operations that can be performed by the objects. This includes security and access control policies as well as the control of the Relationship Management (RM) component.
- ***Relationship Management (RM)***: Functionality for the creation, termination and updating of object relationships based on human-controlled settings and relationship rules (e.g., what are the conditions for an object to establish a relationship with another one – for instance a temperature sensor with an air-conditioning unit).
- ***Service Discovery (SD)***: To enable objects or services to discover other objects/services. This is analogous to the human world were people search for friends to establish relationships with. Service discovery is performed by each object by querying its social relationship network.
- ***Service Composition (SC)***: The objective of this module is to provide better integrated services to the users based on their preferences and needs. Based on the composition and usage of the services included in this component the service discovery can discover the best service for the users. For this reason, this component should include functions for crowd information processing so that information is gathered from main objects and the best response to a service query is obtained.
- ***Trustworthiness Management (TM)***: Information is collected regarding the behaviour of objects in order to define their reliability and estimate their trustworthiness.

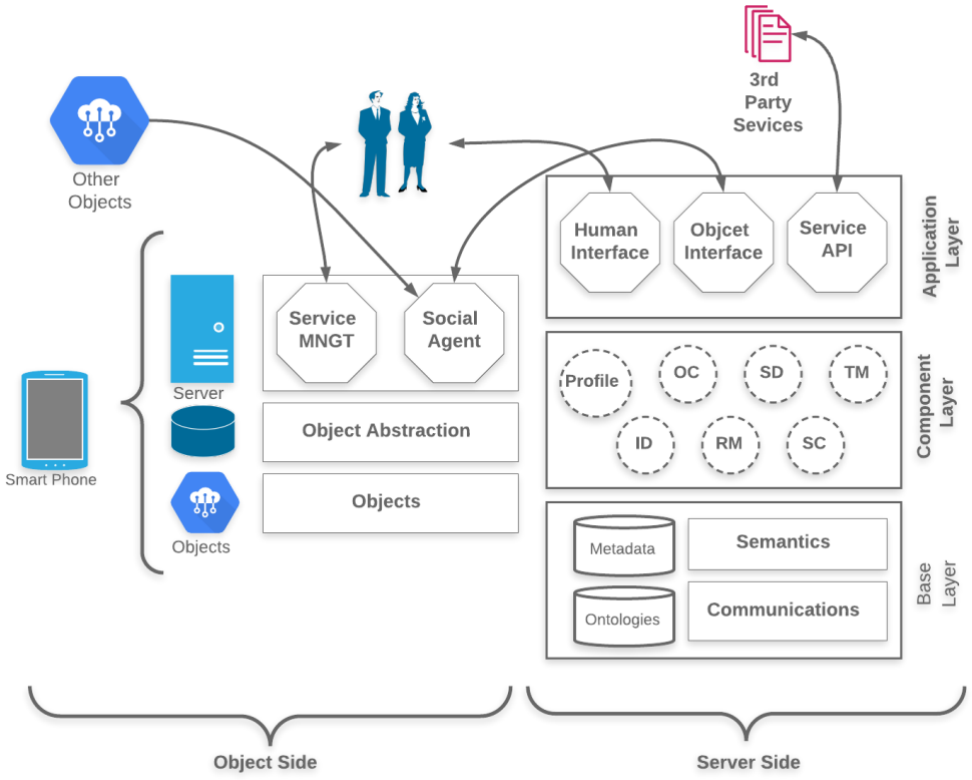


Figure 4-9: Architecture for the SIoT [17]

The architecture of the SIoT paradigm is shown in Figure 4-9. The server side consists of 3 main layers:

- ***Base Layer***: It includes a database that stores information about all the objects (attributes, metadata, relationships, semantic engines and communications between them).
- ***Component Layer***: Functionality for interaction between the objects
- ***Application Layer***: to interface and deliver services to the users.

The object side there are again 3 layers:

- ***Object Layer***: Includes all the physical objects than can be discovered and reached through the communication interfaces:
- ***Object Abstraction Layer***: Common languages and procedures are used to harmonize the communication between different objects.
- ***Social Agent and Social Management Layer***: The social agent handles the communication between objects with regards to updating of the profiles and

relationships and to discover or request services from the social network. The Service Manager provides the interface for humans to control the objects' behaviour.

4.3.5 The ITU-T IoT Reference Model

The ITU-T IoT Model [8] includes 4 Layers as well as Management and Security Capabilities implemented across all the layers as shown in Figure 4-10.

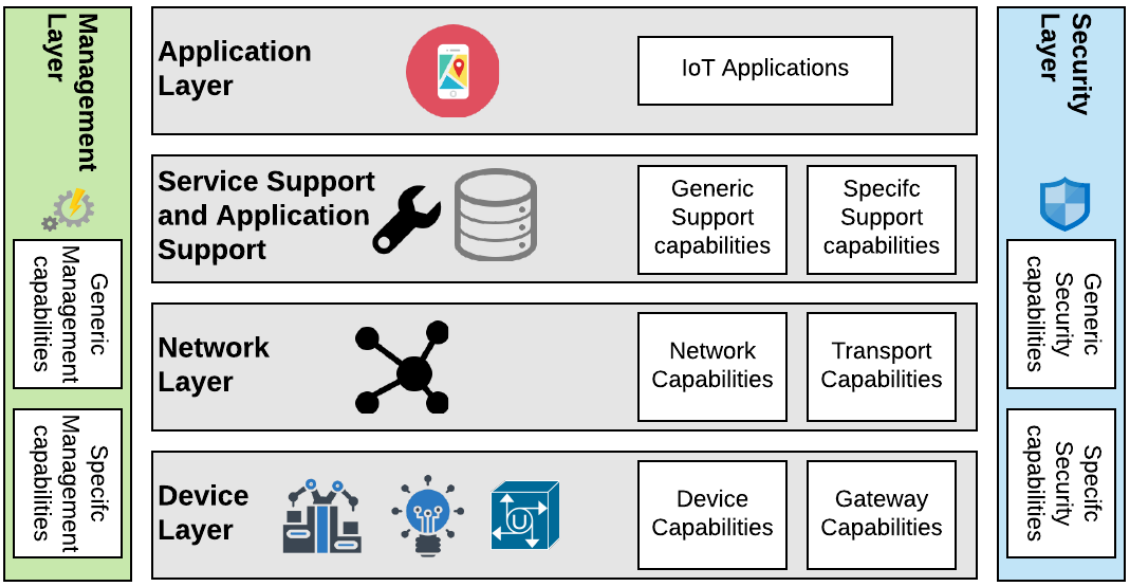


Figure 4-10: ITU-T IoT Reference model [8]

4.3.5.1 Device Layer

The device layer includes the Devices as well as the Gateway that will provide connectivity to the Wide Area Network. For this reason, this Layer capabilities can be grouped in two main categories:

- ***Device Capabilities***: The most important capabilities related to the devices are:
  - Direct interaction of the devices with the communication Network without using the Gateway in order to collect and upload or to receive information from the communication network.
  - Indirect interaction of the devices with the communication Network through the Gateway in order to collect and upload or to receive information from the communication network.

- Ad-hoc networks capabilities so that device can form ad-hoc networks in situations where there is increased need for scalability and quick deployment.
- Sleeping and waking up: These mechanisms are provided to save energy.

- ***Gateway Capabilities***: The most important capabilities of the gateway are:
  - Multi-Interface Support in order to support communication of devices using any wired or wireless technology (e.g. Zigbee, Bluetooth, Wi-Fi etc.). At the network layer the gateway gets connectivity to the Wide Area Network (e.g. the Internet) using mobile technologies (2G, 3G, LTE, 5G), PSTN, DSL etc.
  - Protocol Conversion: To support communication at the device layer when the connected devices use different communication protocols (e.g. Zigbee devices connected with Bluetooth devices) and at the network layer when the connected devices use different technology to connect to the WAN (e.g. DSL and 5G).

4.3.5.2 Network Layer

Includes two sets of capabilities:

- ***Networking capabilities*** to support connectivity to the network such as access and transport resource control functions, mobility management or authentication, authorization and accounting
- ***Transport capabilities*** to provide connectivity for the transport of the IoT Service and application-specific data and the related control and management information.

4.3.5.3 Service Support and Application Support Layer

Includes two sets of capabilities:

- ***Generic Support Capabilities***: Common capabilities usable by different IoT applications (e.g. data processing, data storage).
- ***Specific Support Capabilities*** to support the requirements of diversified applications. In simple words they provide different support functions for different types of IoT applications. Generic Support Capabilities can be re-used in order to build specific Support Capabilities.

4.3.5.4 Application Layer

Contains IoT Applications.

4.3.5.5 Management Layer

Includes capabilities to support traditional networking management functions such as fault, configuration, accounting, performance and security (FCAPS) management. The management capabilities can be grouped in:

- ***Generic Management capabilities*** like:
  - Device Management (e.g., remote device activation and deactivation, diagnostics, software and firmware updating, device working status management etc)
  - Local network topology management
  - Traffic and congestion management
- ***Specific Management capabilities*** which depend on the application requirements

4.3.5.6 Security Layer

It includes two sets of capabilities:

- ***Generic security capabilities*** that do not depend on the application used:
  - At the Application Layer: authorization, authentication, confidentiality of application data, data integrity protection, privacy protection, security audit and anti-virus.
  - At the network Layer: authorization, authentication, confidentiality of use data and signalling data, signalling integrity protection
  - At the Device Layer: Authentication, Authorization, device integrity validation, access control, data confidentiality and integrity protection.
- ***Specific security Capabilities*** which depend on the application used (e.g. security of mobile payments, security of e-health data etc.)

4.4 IoT Devices and Components

An IoT system typically includes a large (and in some cases enormous) number of heterogenous devices with different capabilities and it becomes challenging how all these

devices interoperate. As defined in section 4.2.3 devices are categorized as data-carrying, data-capturing, sensing and actuating. Data-capturing and data-carrying are responsible for the reading and/or writing of information from or to the physical things (e.g., temperature sensors, IR sensors, barcode readers etc.). A general device on the other hand, has embedded processing and communication capabilities (e.g., a micro-controller) to perform more sophisticated functions or facilitate the development of stand-alone IoT systems without the need of connecting to the Wide Area network. Based on this categorization, one can classify devices based on their processing power and their connectivity capabilities [18].

Regarding the processing capabilities devices are classified as:

- **Devices with no processing capability**: In the context of IoT these are considered as passive devices, usually low-cost and with no microcontrollers. A typical example is an RFID.
- **Devices with low processing capabilities**: Their processing capabilities are limited to the reading and writing data from or to sensors and actuators and sending this data to IoT applications, but they are not able to make decisions or run complex algorithm. They are typically low cost and usually embed a very low-power and low-cost microcontroller. Typical example is a smart light or a door sensor.
- **Devices with high procession capabilities**: They have enough processing power to enable them making decisions and running complex algorithms. They are typically high cost as they employ a powerful microcontroller. (e.g. a smart cooling system, or a smart thermostat)

Regarding the connectivity capabilities devices can be classified as:

- **Devices with low connectivity**: This kind of devices do not connect directly to the communication network to transfer the data but instead they rely on additional elements (e.g. gateway) to perform communications tasks (e.g. protocol translation or internet connectivity).
- **Devices with High connectivity**: They have the hardware and ability to directly connect to the network to transfer the data.

The ITU-T recommendation Y.4460 [18] defines the architecture models devices with different capabilities. Specifically, it proposes models for devices with:

- Low Processing and Low Connectivity (LPLC)
- Low Processing and High Connectivity (LPHC)
- High Processing and High Connectivity (HPHC)

Based on the architectures presented in section 4.3 the main components of an IoT can be identified. An ecosystem goes beyond the technology components and includes also the data and monetary/business aspects that that are important when IoT solutions and systems are deployed for economic, social, research or other benefit. These are:

- Sensors/Actuators and Embedded Technology
- Connectivity
- Data Management and IoT Analytics
- IoT Cloud
- User Interface

4.4.1 Sensors/Actuators and Embedded Technology

Sensors and actuators are considered the frontend of any IoT application or system. Sensors facilitate the concept of context awareness so that knowledge about the environment is collected and uploaded for further processing to the attached communication network. Actuators on the other hand, receive instructions from the communication network and perform actions onto the environment they reside.

4.4.1.1 Sensors

A sensor is a device that is used to measure a physical quantity by converting it into a signal that can be read by the system. In IoT physical quantities from the environment (e.g. temperature, humidity, inertia etc.) are measured then they are converted into electronic signals which are then digitized to be sent to the communication network. Sensors typically include transducers which, by definition can convert on form of energy to another. Based on the application there are many possible sensors that can be used in an IoT system (temperature sensors, RFID, light sensors, electromagnetic sensors etc.).

Figure 4-11: Typical IoT Sensors

Sensors can be classified, according to one of the following criteria

- **Power supply requirements**: Passive Sensors or self-generating, directly generate an electrical signal in response to an external stimulus without the need for an external power supply (e.g. thermocouple or piezoelectric sensors). Active sensors require external power supply or an excitation signal for their operation and in this case the output signal power comes from the power supply (e.g. Infrared or Sonar sensors).
- **Nature of the Output signal**: Sensors can be either analogue or digital. Analogue sensors generate signals that are continuous in both their magnitude and temporal or spatial content (e.g. temperature, displacement, light etc.) Digital sensors are ones that generate signals that are discrete in time and amplitude (e.g. shaft encoders, switches etc).
- **Operational Mode**: Deflection mode sensors generate a response that is a deflection or a deviation from the initial condition of the instrument and this deflection is proportional to the measurand of interest (e.g. pressure sensor). A Null mode sensor

exerts an influence on the measured system so as to oppose the effect of the measurand. The influence and measurand are balanced (typically through feedback) until they are equal but opposite in value, yielding a null measurement. Null mode sensors can produce very accurate measurements but are not as fast as deflection instruments. (e.g. Wheatstone bridge sensors).

- **Measurand**: Sensors depending on the quantity they measure (e.g. Mechanical, thermal, magnetic, radiant, chemical etc.)
- **Physical Measurement Variable**: Depending on whether the sensors rely on the variation of resistance, capacitance or inductances they can be classified as resistive, capacitive or inductive.

According to the application as well as accuracy and precision requirements, sensors should be selected while considering the following aspects:

- Accuracy of the input Readings
- Reliability and Repeatability of input
- The conditions of the environment the sensors will be placed in
- Cost and power consumption

4.4.1.2 Actuators

Actuators are devices that can take an effect on the environment they belong by converting electrical signals into different actions or in different forms of energy. Examples include lights, displays, motors, robotic arms, heating/cooling elements etc. Motion-based actuators are typically categorized into electrical, hydraulic or pneumatic actuators. Electrical actuators convert the electric signals into some form of rotation (e.g. motor) or motion, hydraulic ones facilitate mechanical motion using fluids whereas pneumatic actuators use the pressure of compressed air. In the typical example of a smart home automation system we can find actuators that lock/unlock doors, switch on/off the lights, heat up to increase the temperature, etc.

Figure 4-12: Industrial IoT Actuators

4.4.1.3 Microcontrollers and Embedded Systems

While a sensor is a device that turns the received physical conditions or states into signals (analogue or digital) and the actuator is the device that turns the digital signals into some sort of physical effect, the microprocessor is considered to be the computing systems which sits in the middle and processes and/or generates the digital signals. A microcontroller has a central processing unit (CPU), a fixed amount of memory (RAM and ROM) as well as other input/output ports and peripherals all embedded onto a single chip.

When trying to choose a microcontroller for an IoT application or system, the approach "one-size-fits-all" approach cannot be adopted. Here is a list of characteristics that need to be considered when selecting the microcontroller:

- **Bits**: The number of bits that microcontrollers support varies which affects their processing speed. Typical sizes are 8-bit, 16-bit, 32-bit and 64-bit.
- **Random Access Memory (RAM)** is the fast-access memory that does not retain the data when the device is shut down. This memory is embedded in microcontrollers to quickly perform various actions. They come in different sizes. High memory size means better processing capability but also higher cost.

- **Flash or the ROM**: It is the microcontrollers memory that retains the data when power is off. It is smaller than the RAM but it is required in order to support offline storage.
- **General-Purpose Input Output (GPIO) pins**: These are the pins where the sensors and actuators get connected. Depending on the cost and size of the microcontroller, the number of pins can vary from a few tens up to hundreds.
- **Connectivity**: The ability of the microcontroller to establish connections to the network or the Internet. There are various communication technologies that can be used (e.g., Wi-Fi, Bluetooth, Zigbee, Ethernet, etc.)
- **Power consumption**: This is an important characteristic of a microcontroller as it defines the number of actuators and sensors the microcontroller can support and power up especially when the microcontroller is powered but from batteries or solar panels. These days energy efficiency is very important in microcontroller to extent lifetime of the IoT devices' batteries.
- **Development Tools and Community**: Many microcontrollers come with development/programming tools to support and facilitate easy integration onto IoT solution. For a few microcontrollers' families there exist communities and forums that are of great help for integrators and developers when building up an IoT system.

Some popular IoT Microcontrollers are Arduino, ARM, Raspberry Pi and many others.

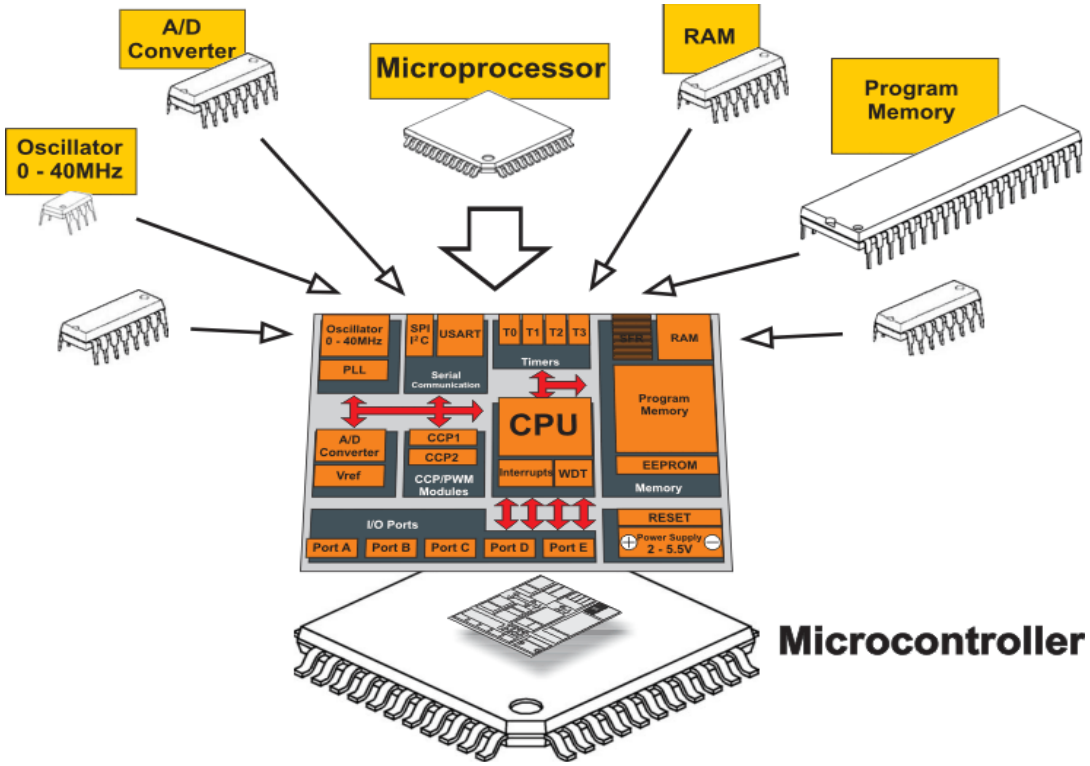


Figure 4-13: IoT microcontroller

4.4.2 Connectivity

Connectivity is a key ingredient of an IoT System since, as defined by the ITU-T [4], the only mandatory capability of an IoT Device is communication. Based on this, any device attached to an IoT platform should be able to send to or receive data from the attached network. There are various challenges that need to be taken into consideration and dealing with IoT connectivity [19]:

- **Identification and Addressing**: That fact that the number of IoT devices attached to a network could be very high necessitates the availability of efficient mechanisms and protocols for device identification through unique addresses. Given the running out of addresses in the IPv4 protocol, IPv6 becomes a necessity in IoT.
- **Low Power Communication**: IoT devices are typically low power devices with many power restrictions. Therefore, it needs to be ensured that the communication technology consumption is kept to the minimum on these devices.
- **Efficient Routing protocols** with low memory requirements
- **High-Speed Communication**

- **Mobility**

More information about IoT communication technologies and protocols can be found in sections 7 and 8

4.4.2.1 Smart Gateway

Data from sensors to the communication network or data from the communication network to the actuators in many cases have to go through gateways especially in cases where multiple networks need to be traversed. Gateways are devices that make network protocol translations to ensure seamless communication between the many (typically heterogeneous) IoT devices. That said, gateways are an integral part and have central and crucial role in an IoT system being responsible for the easy management of data traffic. In some cases, gateways offer security by protecting the system from authorized access and malicious attaches. Moreover, act as pre-processors for the data collected from the sensors before forwarding them to the cloud. In this context, there exist “Smart” or “Intelligent” gateways that analyse the data to either infer new knowledge or compress the data by forward only the important and relevant information to the cloud.

4.4.2.2 Networks, Mobile Technologies and Protocols

In IoT, connection to the Internet is typically and usually achieved using the Internet Protocol (IP) despite the fact the IP protocol stack is power- and memory-demanding for the connected devices. For this reason, it is also possible for devices to connect to the local network using non-IP technologies like RFID, Bluetooth, NFC, etc. however these technologies are limited in range. These low-range technologies are used for personal area networking (PAN) and are quite popular in IoT applications such as wearables. For Local Area Networking (LAN) IP-compatible technologies should be used however the IP-protocol needs to be modified to support low power communications. One of these protocols is 6LoWPAN which incorporates IPv6 with lower power requirements. Other networking technologies that can be used in IoT include IEEE802.15.4, RFID, LTE, 5G, 802.11 standards, Z-wave etc. More details about the networking, mobile technologies and the relevant protocols can be found in chapters 7 and 8.

4.4.3 Data Management and IoT Analytics

IoT is highly associated with a colossal number of data that gets communicated between the IoT components; there is raw sensed data which is being collected by the sensors, pushed to the gateways for prepossessing and maybe inferring of new data and then uploading of this data to the cloud for storage or further processing. In the reverse direction analysed data leads to smart decisions which are forwarded back to the actuators for execution. This requires a system that is capable of storing, processing and analysing a very large amount of data; hence data management and data analytics are critical components of an IoT system.

IoT Analytics is used to make sense of the vast amounts of collected data. For instance, to infer the walking patterns, or most-visited shops of shopping mall visitors whose position is anonymously tracked. Another example could be the estimation of the likelihood of a car accident by monitoring the driving behaviour of cars in a smart transport environment. Once such situations are identified, an immediate decision needs to be taken by the system and a specific action needs to be executed or a warning needs to be issued to prevent undesirable scenarios. In simple words, IoT analytics are concerned with the conversion of the raw sensor-collected data into useful insights (further knowledge inference) which are analysed to lead into smart and useful decisions. Data analysis has storage and computation requirements which in many cases cannot be accommodated in the sensors or even the smart gateways therefore they need to be provisioned in the cloud.

More information about Data Management and Analytics in chapter 9 and 10.

4.4.4 IoT Cloud

The likely limited processing power and storage capability of sensors and smart gateways requires that data travels all the way to the core of the network for either storage or specialized and powerful processing. It can be considered as a smart high-performance entity which combines the various IoT components together with high data-handing, storage and decision-making capabilities. Many IoT applications these days have very low latency requirements in the range of milliseconds (e.g. smart health, smart transport etc.) therefore the IoT Cloud should be able to process the colossal amount of data from multiple source extremely fast.

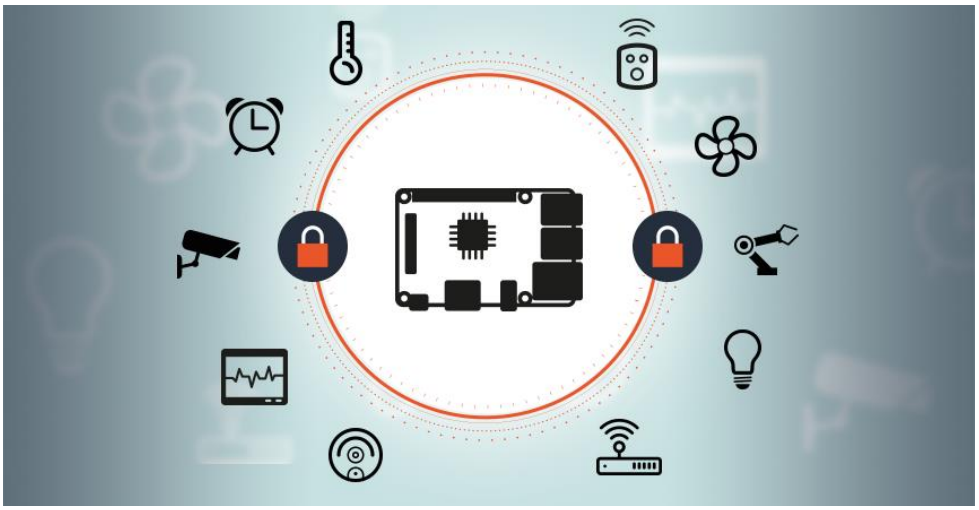
In short, the cloud is the brain of the IoT ecosystem and typically responsible for processing, analysing, decision-making and storing the data. Nevertheless, the cloud is not a mandatory component of an IoT system since (as described in section 4.3.3) Fog or Computing allows processing and storage to happen in a distributed way. The Cloud solution is preferred when massive scalability and decreased operational cost are required whereas the Edge computing solution is the preferred option when large amount of data processing and storage are required on-premises.

4.4.5 User Interface

The user interface is the physical and visible part of the IoT system to the user. It provides an interactive way through which the user can get access to the data, receive alarms, perform actions, give instructions, set preferences etc. It is of high importance to develop an interface which is friendly to the user and does not require extra effort to perform these interactions with the IoT application. The interface could be simply an application implemented on a smartphone or table or it could be a direct interaction with the "Things" (e.g. in Amazon Alexa users interact directly with the devices).

5 IoT architecture and components (2 of 2)

Author(s): Dr. Omar R. Daoud
Dr. Mohammed Bani Younis
Dr. Saleh Saraireh
Eng. Rasha Gh. Freehat



5.1 Cyber-Physical System

5.1.1 Introduction

During everyday life, an interaction with many complex objects and systems is a must during the rises of these days' technologies. Cyber-Physical System (CPS) stands for the direct interaction between the physical world and the computers. The CPS concept was initially supported by the US National Science Foundation (NSF). It is related to the control process of monitoring, sensing, or managing different physical environments that consists of several distributed computing devices. It requires know-how and skills in building algorithms point of view, modelling physical environments, integration and communications, and systems actuations, [20]. Various types of CPSs can be found in our world starting from controlling small instruments into large manufacturing machines or either buildings and cities. Practically, they are interacting not only through interfaces such as touchscreens but also through direct performed actions in the physical life. Thus, it is considered as the key infrastructure for the modern societies, which improves the citizen's life quality. As an example for the most civilized CPS is the latest versions of brand cars, where the computer is considered as a vital element used to control every single action during the movement as well as throughout parking. Moreover, CPS could be presented through fulfilling other targets such as in the energy network of a whole country or among countries, warehouses, and factories. This issue will obviously have a direct impact not only on the citizens' life quality, but also on the economy itself. CPS definition is emphasized to be Cyber-Physical Systems of Systems (CPSoS) in the case of combining several CPSs to fulfil a specific goal under controlling several devices/ machines or components [21, 22].

CPS is considered as a transforming interface of interactions between users and engineered systems. Accordingly, it integrates processes such as sensing, controlling and networking into a physical infrastructure which is based on the internet connection for such interactions [23].

5.1.2 The Rise of CPS

The CPS concept started since early 1980s in Carnegie Mellon University when they connect a Coke machine to the internet which was able to label its Coke with either "cold" or not. From that's date, the powerfulness of using the internet to connect the tools and other

devices has inspired and attracted many researcher's interest. In this occasion the idea of interconnecting was the seed of developing the concept of the Internet of Things (IoT). By this concept some scenarios started to make use of the Radio-Frequency Identification (RFID) to manage the connected objects by computers. This allowed tracing the connected objects by some kind of wireless readers. The RFID is considered as a small tag with a traceable chip, which could be traced, controlled or/and managed through internet. The continuous improvement and advances in the development lead beyond RFID to many new technologies in the computer industry, which could be marked as intelligent tools. Therefore, everything in our lives; homes, buildings, and companies; will soon be connected to the internet which estimated to be more than 26×10^9 connected tools by the year of 2020 [22, 24, 25].

Wireless Sensor Networks (WSNs) is these days an increasing prominent. This is due to that a large number of heterogeneous tools and devices are interconnect through the internet; such as dozens of computerized devices, or even hundreds of small sensor nodes. This interconnection includes electrical machines, electronic devices and many other smart elements. Therefore, the data of the physical world is easily collected, monitored, managed, and controlled by wireless communication. Therefore WSN receives a great potential in every application areas either industrial or domestic and medical. This is due to the fact that these sensing elements permit data collection from the real-world and digital-form handling process, which can easily be distributed making use of Ad-hoc network distribution [5, 26, 27].

Nowadays, these sensors can be distributed on humans in addition to the impeded sensors in the carried smart phones and devices. In this context Bosch Sensory Swarms and the Qualcomm Swarm Lab at UC Berkeley estimated that up to 1000 wireless sensors per person will be deployed over the next 10 to 15 years. This has the consequence that huge data is available for processing, and at the same time a wide range of applications could be deployed and covered such as robotics, mobile computing and IoT. This will result in a monitored, controlled and adaptable environments [22].

As mentioned earlier, the CPS concerns on the control process of monitoring, sensing, or managing different physical environments. This is why improvement of human personal health can be considered as a good CPS example. In this regards integration through the

human vital signs could be monitored and managed to not only to suggest the best places and routes for use but also to make an interoperable personalized medical devices or robotic surgeries. Also, it can be used in transportation, either from cruising control point of view or the securely communications with other smart elements on the roads. This has the ability to reduce the road delays besides inspect the danger and disaster zones and many other crucial issues. Another example for the CPS is the sustainability in many issues such as detecting and deterring fires, combatting the oil spills underwater and many others. Thus, the used sensors, actuators or any other capabilities will be integrated in order to attain the desired results and targets [20] [28].

In CPS, the research is divided into different stages based on the components themselves and the way of linking these parts such as sensors and actuators, the way of communications controlling and networking, and the needed mathematics and software. The research in CPS is divided either to represent the cyber formalism or to cover physical processes; i.e. highlights a specific representation for each of them but not both. As an example, the physical processes will be modelled based on the differential equations while the control flows will be represented by standards frameworks or formal methods such as Petri nets or Automata [23] [29]. This approach will help in modelling and supporting the component-based CPS development; however, the overall system's design verification is very complicated to attain. Therefore, the research needs in CPS could be summarized as follows [23]:

- **Abstraction and Architectures:** Both of abstraction and architectures should has innovative approaches to offer:
    - A controlling process with a unified integration.
    - A rapid design for communication.
    - An accurate deployment for computation.

  Thus, standards for designing and developing the CPSs are urgently needed in order to support the integration between cyber and physical issues.

- **Distributed Computation and Networked Control:** Several challenges appear when taking the design of a networked control into account such as the failure issues, the processing time delay, the distribution schemes and the reconfigurations. This will open the chance for research in order to design both of real-time protocols for quality

of service (QoS), and a real-time implementation from the control point of view. Accordingly, frameworks, algorithms, and tools are needed to fulfil both of reliability requirements and the security visions.

- **Verification and Validation:** Due to the fact that the CPS infrastructure lacks of trustworthiness, its components and structure must be developed and deployed beyond the existing technologies and be fully integrated in terms of:
    - Dependability,
    - Configurability,
    - Certification.

  Therefore, all of the CPS parts such as hardware, software, middleware and the operating systems must be developed to have new models, algorithms, and tools are needed to incorporate verification and validation at the control design stage.

5.1.3 Smart Home Systems as a CPS Case study

Smart home system is a very good example for the CPS, because it gathers different components (home appliances) each of which has its own functionality in a one centric system. Some issues should be taken into consideration in this case as expressed below [30]:

- These components were prototyped using embedded boards,
- Some of them is equipped with a touch screen,
- Some of them have a programmed user interface,
- Smart components (devices) are programmed based on the device profile for web services (DPWS) stack; i.e. they have the smartness behaviour or device operations.

This is in addition to the main challenge that could face the CPS developers and designers namely the huge differences in design practices; because the CPS is an integrated system which contains different disciplines. Therefore, the designers, engineers, and developers need to be able to explore the CPS needs in a collaborative manner. This is besides allocating responsibilities and analysing trade-offs between them. Co-simulation will be one of the best solutions were different disciplines are coupled. As a result, the cooperation will be attained without enforcing new tools or methods.

Figure 5-1 describes the smart home systems from the environment orientation point of view where this approach is considered as one of the simplest designing solutions.

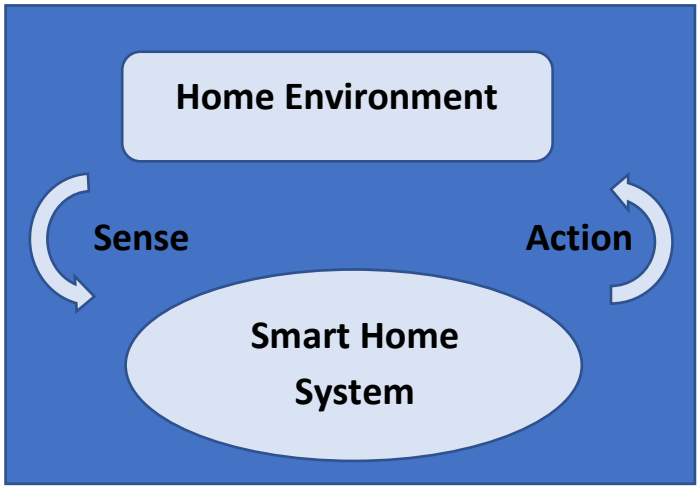


Figure 5-1: Smart Home Systems as a CPS Case study [30]

As depicted in, the home system has been designed from the CPS point of view, this means that the user himself is a part of the system and his behaviour will describe the context-aware service. Then the sensors are the main parts of this system where any action will be sensed and causes a reaction from the system. A decision making process will start as a reaction of the environment changing. These reactions will be drawn a saved context-logic information stack, which has an individualized data for any system.

Figure 5-2 depicts the context-logic stack that helps in an appropriate decision-making process. As described in Figure 5-2, the logic stack consists of layers each of which can be considered as a specific case. This means that according to the preferred data saved in the stack, it should give/propose the appropriate solution(s) intelligently. Furthermore, if the collected data is not enough to draw the solution, then the system should go downgrades to the next layer. Therefore, the lowest layer is the default layer with the factory settings.

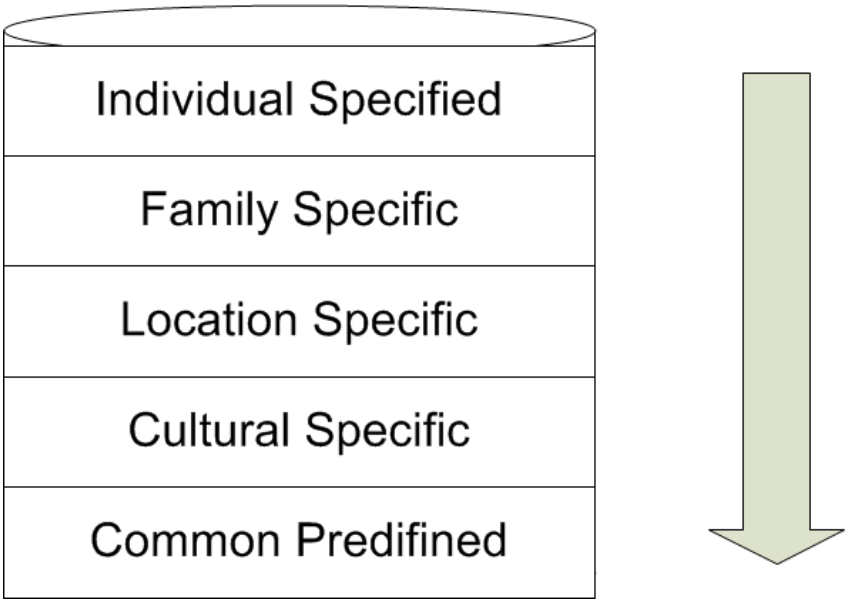


Figure 5-2: Decision Making Process based on Context-Logic Stack [30]

In order to present briefly a case study from different application fields. The following 5.1.3.1 subsection will cover the CPS for range of application domains from the basic level (sensor level) toward the supply chain through the system levels.

5.1.3.1 Real system and controller both mapped and synchronized in virtual environment

One of the main limiting factors in the design issues is of both the system's consideration comprehensive knowledge and the knowledge of technology specifications. This leads to the need of expertise in both of modelling and simulation studies. As an example, a friendly user interface is considered as a crucial element in supporting the decision making effectively. Therefore and to get an intelligent solution, the models and the results of the collected data must be easily accessed and evaluated to appear in an understandable format. This is why the manufacturing systems tend to be equipped with huge number of sensors and data acquisition parts, which will enhance and utilize the communications networks and technology efficiently. Consequently, Supervisory Control and Data Acquisition (SCADA) is prevalence in now days' technologies with same user interface requirements as for the simulation models interfaces [30], [31].

At the case of being the same user who use the simulation model and operate the SCADA system, the SCADA interface could be chosen as a basis interface for both of simulation model

and the graphical user interface (GUI). Thus, the user interface will mirror the manufacturing system SCADA. This will result in a simulation model displaying the actual status of the system. Figure 5-3 depicts the mapping process between the real system and the controllers. This is in addition to the synchronization process in the virtual environment. In this case, executing the scenarios and experiments will be at the same parameters as of the real system. Also, the results will be evaluated using the same performance indicators [32].

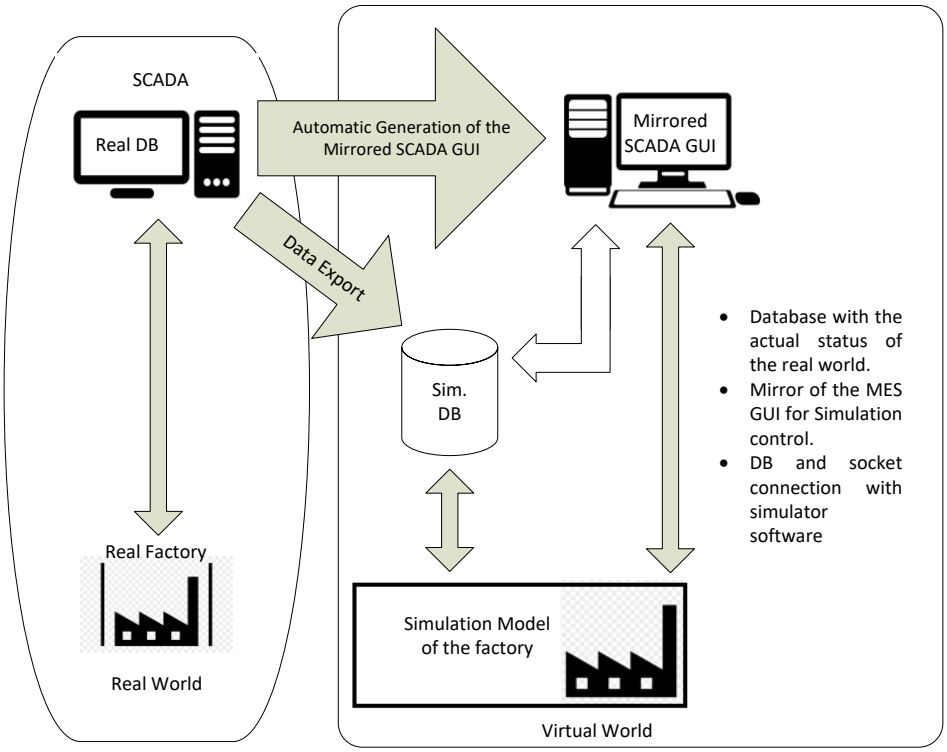


Figure 5-3: Real system and controller both mapped and synchronized in virtual environment for decision support and teaching [32]

The nature of the model building time consuming is another challenge for the simulation modelling. Thus, besides offering a friendly user interface for the daily usage, the building of an efficient model with a periodic updates is important as well.

In order to speed up the modelling process, the automated model technique should be used in the building process. This technique extracted the control codes from the low-level basis into the database of the model definition, which will be attained using a tailored grammar interpreter. As shown in Figure 5-3, the database is used to build up the simulation model

automatically by providing the inputs for the methods; i.e. the stored data in the low level controllers.

As a result, an efficiently controlled maintainable simulation is attained by offered friendly user interface not only in its layout but also in its structure. This will help in achieving the target of decision support for simulation modelling.

5.2 Basic concepts of IoT

The structure of the network of things (IoT) is a system of related figure gadgets, i.e. mechanical machines, advanced machines, objects, creatures or people that have been given Unique Identifiers (UIDs). IoT has also the ability to transfer information in the system without requiring human-human or human-computer operation. IoT structures nowadays are becoming part of different aspects of our lives thanks to their applications. These applications include smart homes, medical services, state control and surveillance, smart city communities and brilliant transportation structures. With the development of the IoT framework, a huge system of systems linking these obligatory gadgets (sensors, actuators, machines, etc.), and with this association, numerous limitations grow, for example, gracefully limited strength and security that develops with any information. A key advance in building IoT frameworks is the reasonable choice of equipment sheets, programming stages and the hidden design regarding the application required. Most IoT applications manage inserted frameworks that were utilized years prior, yet are changed these days into savvy gadgets with arrange network, memory utilization, interoperability, and which are inclined to overhauling highlights. Fundamental ideas in IoT framework will be acquainted with comprehend and look at the realities and constraints in IoT frameworks, which are: memory outline size (RAM), code size (ROM/Flash), handling power, power utilization, UIDs and capacity to refresh programming, accessible force source, bitrate/throughput, cost and size, quickening agents need and peripherals [33].

5.2.1 Storage and Central Processing Units

When the sensor receives physical conditions from the environment and converts them into signals, and the actuator converts the signals into physical actions, the microprocessor is the processing system, which generates these digital signals. Microcontrollers (MCUs) are specifically designed for embedded applications. In these applications, computing is not the

only reason. The MCU has a central processing unit (CPU), a fixed amount of memory (RAM and ROM), all of which are embedded in the chip. MCU acts as an intermediate device to help interconnect and control endpoint devices. The microcontroller is regarded as the brain of the system. When trying to select an MCU for an IoT application or system design, various characteristics need to be considered; as follows [33], [34]:

- **Bits:** MCUs have different functions in terms of the number of supported bits, which will affect their processing speed. Typical sizes are 8 bits, 16 bits, 32 bits and 64 bits.
- **Memories:** Random Access Memory (RAM) is a high-speed memory that does not store information when the gadget is not forced to act. MCUs are installed with this kind of memory to perform various activities quickly. They come in a variety of sizes; however, increase the size of the memory even though it improves the handling ability that expense builds. The MCU has composed the information RAM in several registers (an area in the memory that can compose information or read information) Each register with a unique location, 8-part RAM register can contain 8 parts (one byte) of information, RAM space in particular 265 × 8, which means that 256 registers in RAM contains 8 pieces. Regularly 265 bytes of internal RAM is the most internal RAM for MCU, however some may be usage-based, for example PIC18F452 microcontroller has 1536 bytes of RAM, MCUs can expand RAM amount if needed by including external memory chips.
- **Flash or ROM:** When the power is turned off, it is the microcontroller memory that retains the data stored in it. It is not as large as RAM, but it is necessary to support offline storage. Microcontrollers can read ROM, but cannot write or modify it. ROM is the cheapest program memory. It is recommended to use stable application code. Therefore, ROM contains special instructions and important content, which will never change. On the other hand, flash memory has fast erasing and writing cycles in just a few seconds, so that code development and reprogramming can be performed quickly. The self-programming Flash in some devices can be completed using a specific instruction sequence.
- **General-Purpose Input Output (GPIO) pins:** These are the connection points of sensors and actuators. The number of pins can vary from dozens to hundreds, depending on the size and cost of the microcontroller.

- **Connectivity:** The capacity of the microcontroller to build up an association with the system or the Internet. This should be possible through Wi-Fi, Bluetooth, wired Ethernet or some other correspondence innovation.
- **Power consumption**: This is a significant perspective as it will characterize what number of dynamic sensors and actuators the MCU will have the option to control up and control particularly when the MCU is powered however from elective sources (for example sunlight based, players). It is significant IoT gadgets to be vitality effective so they can perform tasks for quite a while without the need of normally controlling them up.
- **Development Tools and Community**: It extremely valuable for MCUs to accompany advancement instruments and the related documentation to encourage their reconciliation onto IoT arrangement. Having a network or gatherings taking a shot at various sorts of microcontroller makes the integrators/designers work a lot simpler in discovering data identified with their turn of events.

Some popular IoT microcontrollers are Arduino, ARM, Raspberry Pi, Udoo Neo, LightBlue Bean, TinyDuino, etc, where they run the operating system within the following three main options:

- **Bare Metal:** The main advantage, it is the cost-effective and efficient. The main disadvantage is that it provides less support for the developer of the software.
- **Real-Time Operating System (RTOS):** An RTOS system provides exact time operations guarantees. But it has limitation for coordinating physical machinery.
- **Linux:** Linux main advantage it is much easier to program and connect to the Internet, but it does not provide any timing guarantees.

5.2.2 Data Movement

Any device attached to an IoT platform should be able to send or receive data from the attached network. There are various challenges that need to be considered when dealing with IoT communication [35].

- **Identification and Addressing**: There is a very large number of IoT devices attached to the communication network, so efficient mechanisms and protocols must exist for

identifying these devices through unique identifiers (UIDs). Given the running out of addresses in the IPv4 protocol, IPv6 becomes a necessity in IoT.

- **Low Power Communication**: IoT devices are typically low power devices with many power restrictions. Therefore, it needs to be ensured that the communication technology does not consume much of the available power on these devices.
- **Efficient Routing protocols** with low memory requirements
- **High-Speed Communication**

The gateway technology has been widely used in IoT systems, a gateway is deployed to collect, process, and forward the data received from constrained device. The gateway is an essential component on the network connection of an IoT system. Edge devices such as sensors communicate with the gateways and also the gateways take the data and communicate with the cloud. Thus, the gateways act as the bridges between the devices and the cloud and allow communication over a short distances and limited battery. Compelled gadgets utilize a fluctuating transmission conventions, (for example, LPWAN, Wi-Fi, Bluetooth and ZigBee, GPRS and numerous others), gateways interact with these gadgets over a differing conventions and afterward make an interpretation of information to a standard convention, (for example, MQTT, HTTP and CoAP). The gateway engineering comprises of three layers: the detecting layers containing the M2M gadgets, the passage layer offering the types of assistance of system associations, the application layer offering types of assistance to clients. The most well-known standard conventions utilized are MQTT, HTTP and CoAP. Each of these has its advantages and use cases. HTTP gives an appropriate technique to giving two heading information communicate forward and back among gadgets and focal frameworks. It gives web perusing through and extraordinary administrations in Internet of Things gadgets. HTTP is less reasonable if there is a data transfer capacity conditions. MQTT convention for machine-to-machine and IoT arrangements that goes about as an agent, new gadgets or administrations can just interface with the intermediary as they need messages. Whereas MQTT is lighter message size than HTTP, so it is more valuable for executions where data transfer capacity is a requirement issue. Nevertheless, it does exclude encryption as standard so this must be considered independently. The Constrained Application Protocol (CoAP) is another standard convention utilizes with obliged hubs and compelled systems, created for low-force and low-data transmission frameworks. CoAP is focused on coordinated

associations and is not intended for a merchant framework like MQTT, CoAP, it. It is intended to meet the necessities of REST configuration by furnishing an approach to interface with HTTP, CoAP effectively interfaces with HTTP for consolidation with the Web while meeting particular prerequisites, for example, multicast IP support, low overhead, goal locations and straightforwardness for compelled conditions. Yet it satisfies the needs of low-power gadgets and situations. CoAP supports the association model (demand/reaction) between application endpoints, underpins worked in disclosure of administrations and assets, and incorporates key ideas of the Web, for example, URIs and Internet media types [33], [36], [37].

Every one of these protocols supports taking data or updates from the individual gadget and sending it over to a focal area. RFID innovation is radio recurrence recognizable proof forward leap in the inserted correspondence worldview which empowers plan of microchips for remote information correspondence. RFID the specialized strategy for sensors, a RFID reader ought to be structured and associated with the microcontroller. Presently obviously different sides of correspondence are run. The first one is correspondence among microcontrollers, and the other is correspondence between the administration programming and the microcontroller. In any case, where there is a more noteworthy open door, there is the means by which that information is then put away and utilized later on [38], [25].

5.2.3 Input and Output SPI/I2C.

To able to interact with equipment by associate sensors, actuators, and more to make IoT framework bursting at the seams with movement, sensation, sound, and so on to speak with other equipment requires sequential protocol like I2C or SPI. These protocols are the basic language that chip and extra sheets talk. The board knows how to 'talk' with these protocols and control the associated equipment. Two normal sequential protocols, I2C and SPI will present:

- **Serial Peripheral Interface (SPI) Protocol:** It is a bidirectional sequential protocol for two gadgets to send and get information, utilized in short-separation correspondence, essentially in installed frameworks. SPI is known as a four-wire sequential transport, differing with three-, two-, and one-wire sequential transports. SPI is a duplex coordinated sequential information interchanges interface standard which work in a full duplex mode, where information can be send and received all

the times. That makes SPI picked when quick information transmission is required at rate 8Mbits or more (eg. Like speedy difference in thermometers), SPI is quicker than I2C. Gadgets transmit in ace or slave mode where the ace gadget starts the information outline, various slave gadgets are permitted with singular slave select lines with the chip select. The SPI transport has just one ace, which is associated with numerous slaves. SPI can speak with various gadgets through two different ways, the first is by choosing every gadget with a Chip Select line for this situation a different Chip Select line is required for every gadget, which is depicted in Figure 5-4. The second is through multi binding where every gadget is associated with the other through its information out to the information in line of the following as described clearly in Figure 5-5 [39], [40].

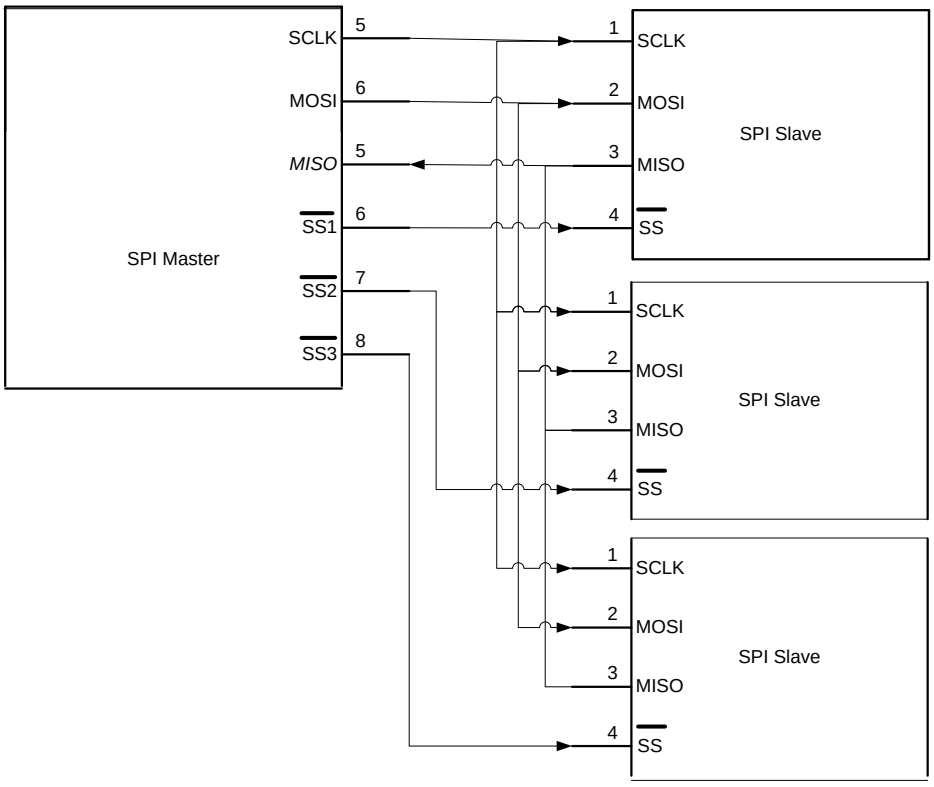


Figure 5-4: SPI Bus Typical Configuration [40]

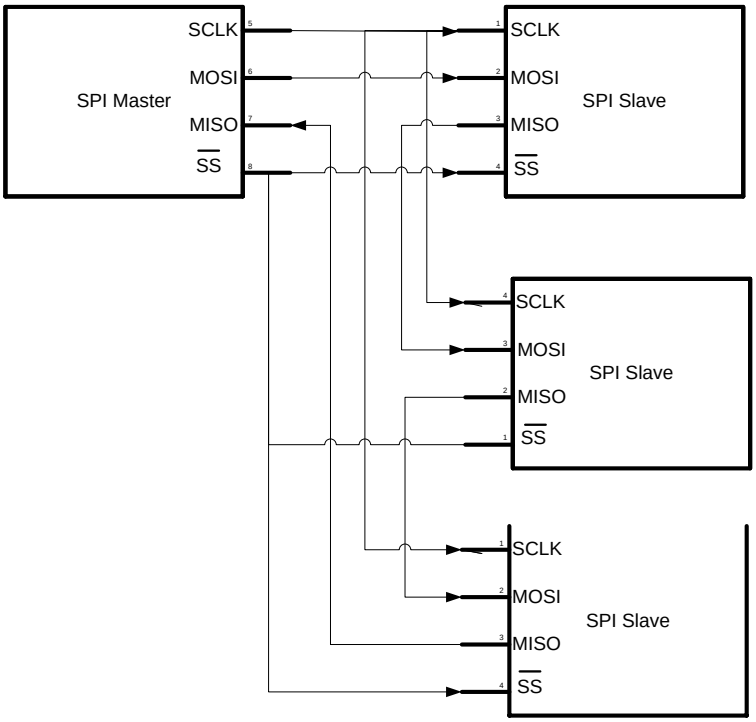


Figure 5-5: SPI Bus Daisy-chain Configuration [40]

- **The Pins: Interface and Function:**

  Coming up next are the SPI transport rationale signals, which has four lines to associate the master to slave, and their function is as follow:

  - **Serial Clock (SCLK):** The output from the master and gives the planning of information trade.
  - **Master Output, Slave Input (MOSI/SIMO):** The output from the master and used to move information from the master to the slave.
  - **Master Input, Slave Output (MISO/SOMI):** The output from the slave and used to move information from the slave to the master.
  - **Slave Select (SS):** The output from the master which is active low. It is utilized by the master in order to address and enact a specific slave so as to begin a correspondence session.

- **Inter Integrated Circuit (I2C) Protocol:** It is a sequential interchanges protocol utilized with embedded frameworks and sensors. I2C is bidirectional two-wire simultaneous sequential transport and require just two wires to communicate data between gadgets associated with the transport. This protocol is helpful for frameworks that

require a wide range of parts cooperating (e.g. low-speed gadgets like microcontrollers, EEPROMs, A/D and D/A converters sensors, pin, extensions and drivers). I2C can associate up to 128 gadgets to the main board and keep up a reasonable correspondence pathway. It utilizes a location framework and a common transport equivalent to the various gadgets, which can be associated utilizing similar wires and all information are sent on an individual wire. Be that as it may, I2C is slower than the SPI; speed of I2C is additionally needy by information speed, wire quality and outer chaos [39] clearly shown in Figure 5-6.

- **The Pins: Interface and Function:**

  The two lines that structure I2C transport have the accompanying functions:

  - **Serial Data (SDA):** It is used to make the I2C as a half-duplex bus by exchanging information among the devices.
  - **Serial Clock (SCL):** It used by the master to constrain the transmission rate.

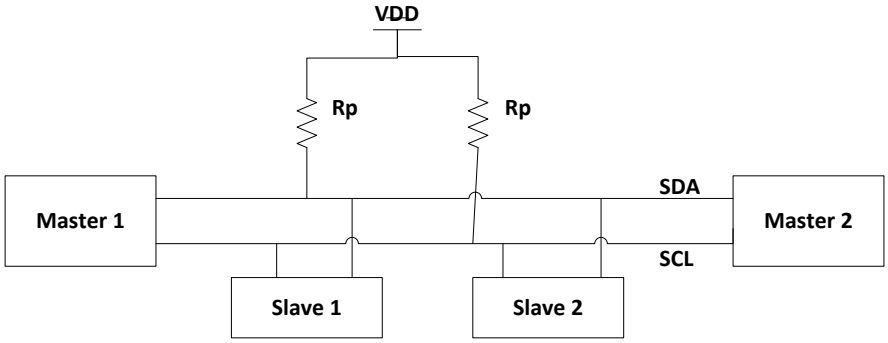


Figure 5-6: I2C Bus Configuration [40]

Every correspondence edge has its own favourable circumstances and burdens and as indicated by vendor and application the best possible correspondence edge is picked. Table 5-1 below summarizes a correlation between principle structure boundaries.

Table 5-1: SPI and I2C comparison

Specifications	SPI	I2C
Complexity	Complex as device increases	Easy to chain many devices
Speed	Faster	**slower**
Number of wires	**4**	**2**
Duplex	Full Duplex	Half Duplex
Number of devices	Many, but there are practical limits and may get complicated	Up to 127 but may get complex as devices increases

Number of masters and slaves	Only 1 master but can have multiple slaves.	Multiple slaves and masters

5.2.4 The instruction cycle/ the fetch-decode-execute cycle.

Fetch-execute cycle is a standard procedure that depicts the means of sending and accepting information from/to the processor and memory that began by press a catch or programming order (turned on) and end with shut-day break. This procedure is finished by the CPU and consists of four primary stages: the get stage, the unravel stage, the execute stage and rehash Cycle. At the point when a memory read activity happens, information is perused from a memory address, at that point taken to the processor, when a memory compose activity happens, information is taken from the processor, at that point kept in touch with the memory. As a matter of first importance, the working framework stacked the information and the program that follows up on that information into primary memory (RAM). At that point the CPU is prepared to accomplish some work as elucidated below [41].

- **Fetching stage:** the initial step where the CPU brings a few information and directions (activity that must be performed by the information) from primary memory. At that point store them in registers (its own inner brief memory zones) this is known as the 'fetch some portion of the cycle'. To accomplish this the CPU utilize fundamental equipment way which is the address bus, the next address of the following thing to be brought on the location transport is set by CPU, from this location information moves from the principle memory into CPU by information transport.
- **Decode stage:** in this sequence, CPU comprehends the brought guidance; this procedure is called decode. CPU comprehend a characterized set of orders called the guidance set, CPU translates the guidance and plans different zone inside the chip prepared of the following stage.
- **Execute:** information preparing is accomplished now, and the guidance is done on the information. The consequence of this procedure is put away in other register. Guidance execution is exceptional for each instruction.
- **Repeat Cycle**: after the execute stage is finished, the CPU sets itself up to start another cycle again.

During the fetch-execute cycle the control unit, information transport and address transport are all being used as follows:

- The control unit will direct the clock speed of the fetch-execute cycle, and enact either the read or write line.
- The address transport will hold the location that is being obtained to in primary memory.
- The information transport will move the information contained in the memory address forward and back between the processor and the memory address.

5.2.5 Accelerators.

IoT is associated universe of capacities, applications and administrations which empower association between physical articles embraced by advancements (for example Sensors, actuators, RFID, remote, portable, cloud and web). This quick increment of advancement, requests and focal points of IoT that emphasizes the requirement for IoT quickening agents to perform the following:

- Service the beneficial devices to get new viewpoints into activities and dynamic help to ventures.
- Provide improvement of uses.
- Unified cooperation for administrators and supervisors.
- Raising cautions dependent on real-time data, connection of different occasions alongside examination and disconnected investigation.

IoT accelerator is a cloud-based IoT arrangement normally utilizes devoted application code and cloud-based administrations to run gadget network, remote monitoring, information handling and examination, and introduction. IoT application accelerator is a light-weight IoT stage that can be utilized to quickly model, create and send esteem included IoT applications, arrangements and administrations.

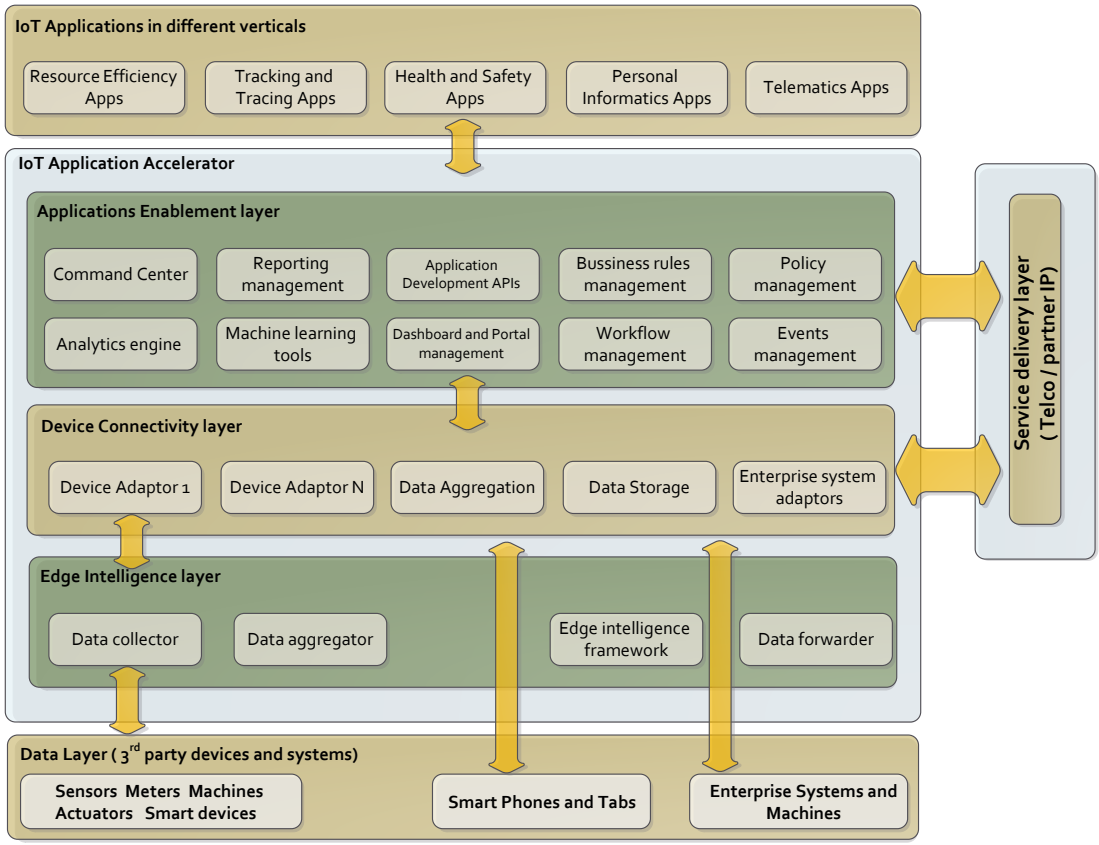


Figure 5-7 shows a stage to quicken IoT administrations engineering which is separated into gadget and resource the executives (equipment/hardware level) layer, middleware, administrations layer, application explicit and customer layers [42].

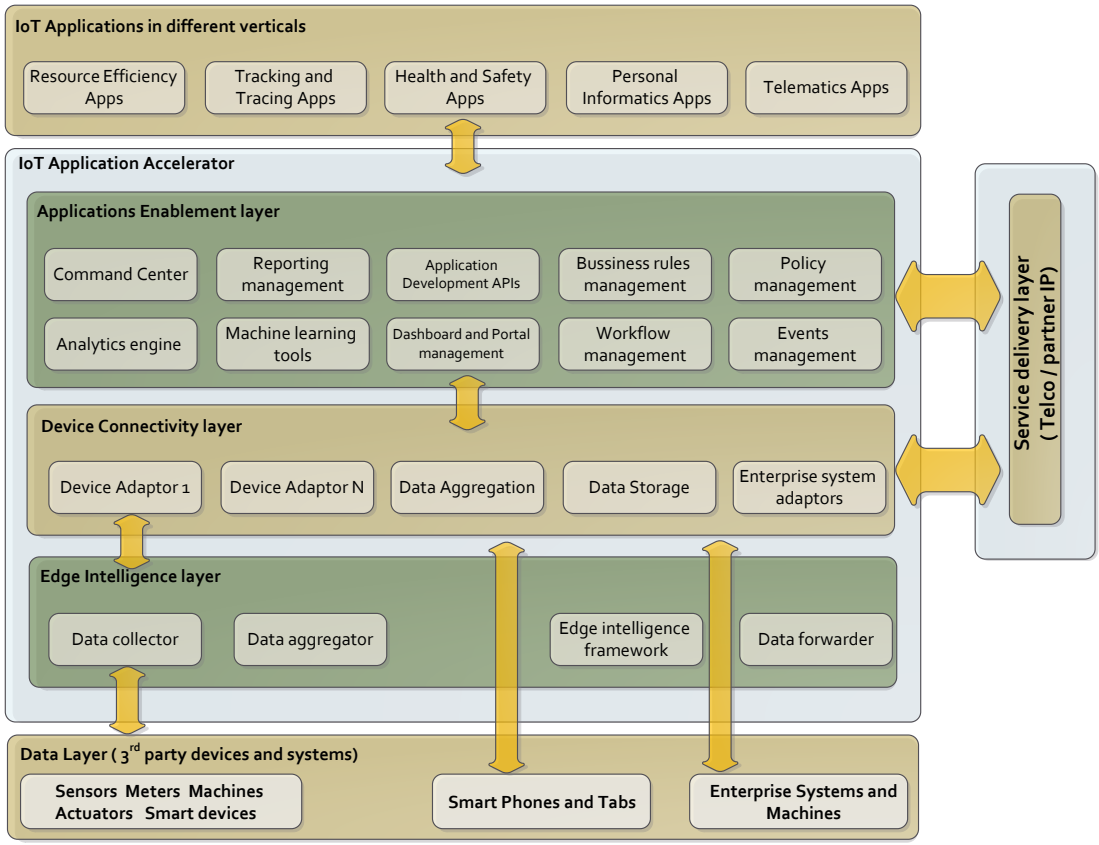


Figure 5-7: The Architecture of the Application Accelerator Platform [42]

IoT Accelerator helps to easy use single unified platforms to provide a global connectivity and device management that will turn business concepts real. Through powerful services, IoT accelerator can accelerate time to market, reduce investment risks, and reduce the complexity of IoT applications and services. This will reduce the barriers to use the IoT. As an example Remote Monitoring (Version 2 is based on micro-services), Connected Factory (support OPC-UA protocol) and Predictive Maintenance are considered as most known accelerators.

5.2.6 Peripherals.

The growth of IoT leads to large number of IoT devices and peripherals. The followings are considered as IoT peripherals [11], [42], [43]:

- Sensing peripherals,
- Motion peripherals,
- Smart peripherals,
- Accelerometers,
- Microphones,

- Switches,
- Screen or speech synthesizers,
- Card readers

These are in addition to many more peripherals which improved and change according to service or application need. Nowadays, more and more peripheral devices and systems (with different resources and functions) are connected to each other to make people's lives much easier and can improve living standards. Among these applications but not all: Mobile phones, building automation equipment, machine-to-machine equipment, smart technology, manufacturing control, climate control, tracking, parking and traffic flow, smart key entry, location, anti-theft, inventory, centralized marketing. .

5.3 Embedded Memory

Due to the increase in the amount of data required for many of these applications (such as video games and communications), embedded memory plays an important role in digital system applications. Moreover, the gap between processor speed, main memory and bus speed (memory wall) is getting wider, so more on-chip memory is needed to keep the processor busy and increase throughput. In addition to increasing the processor frequency, integrating multiple cores or functional units called System-on-Chip (SOC) on the same chip also requires a larger memory size. Embedded memory occupies more than 50% of the chip area and more than 80% of the number of transistors. Due to the expansion of technology scale and the demand for high-density memory, process changes have increased. Therefore, meeting the stringent requirements for performance, power, and yield has brought huge challenges. Embedded memory not only plays an active role in system performance, but also affects yield timing and power consumption. The organization of memory and the early decisions made by the system level and architecture group have a great impact on the role and memory on the entire system. To overcome these issues, we must consider the trade-offs of storage unit type, array organization, storage hierarchy, test design and overall storage subsystem in the early days of [44].

5.3.1 Embedded Systems Memory Types

Many types of storage devices can be used in modern computer systems. You must understand the differences between them and understand how to use each type effectively.

Keep in mind that the development of these devices has taken decades, and the underlying hardware varies rapidly. The names of memory types often reflect the historical nature of the development process and are often more confusing than insightful situations. Figure 5-8 classifies memory devices RAM, ROM, or a mixture of the two [45].

The RAM series includes two important storage devices: static RAM (SRAM) and dynamic RAM (DRAM). The main difference between them is the lifetime of the data they store. As long as power is applied to the chip, SRAM retains its contents. If the power is turned off or temporarily disconnected, its contents will be lost forever. On the other hand, the data life of DRAM is very short-usually about 4 milliseconds. This is true even if the power is continuously applied.

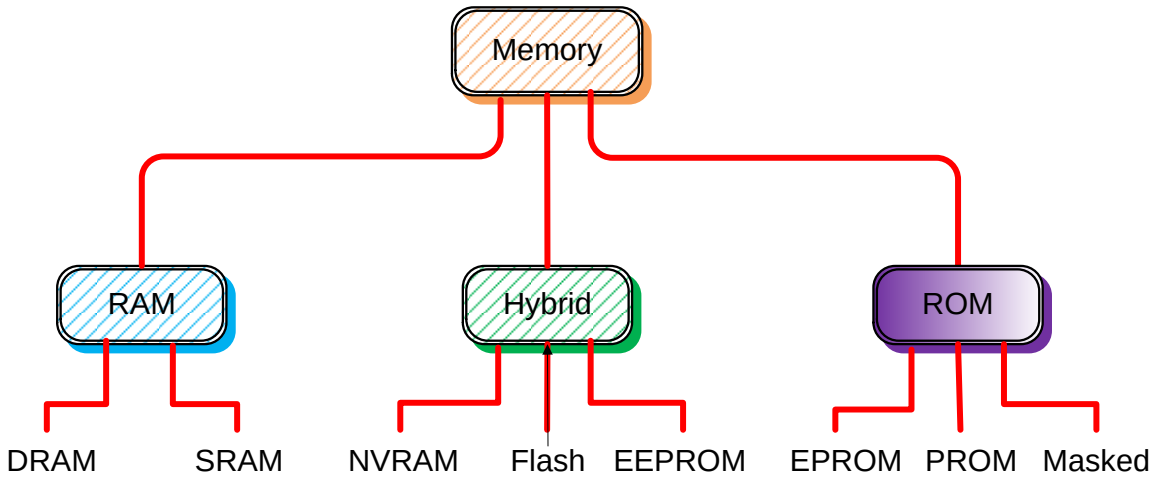


Figure 5-8: Common memory types in embedded systems [45]

SRAM has all the memory attributes to consider when you hear the term Slam. In contrast, Measure seems to be meaningless. By which the activity of the measure supervisor is to occasionally restore the information stored in the measure. By restoring the information before it is terminated, the contents of the memory can be saved for any length of time required. Therefore, considering all factors, Measure is as valuable as SRAM. When choosing which type of RAM to use, the frame creator must consider time and cost. SRAM gadgets can provide amazingly fast access time (about many times faster than Measure), but the production cost is much higher. In general, only use SRAM where access speed is critical. The low cost per byte makes Measure attractive any time a large amount of Smash is needed. Many embedded frameworks combine two types: a small square (a few kilobytes) of SRAM

along the basic information method and a larger measure square (or even megabytes) for everything else [44], [45].

5.3.1.1 Static RAM (SRAM)

SRAM is commonly utilized for fast registers, stores and moderately little memory banks, for example, an edge cushion on a presentation connector. Interestingly, the primary memory in a PC is regularly unique Slam (DRAM, D-RAM). SRAM is designed to meet two requirements: to provide a direct interface to the CPU at a speed that DRAM cannot reach, and to replace DRAM in systems with very low power consumption. In the first role, SRAM is used as cache memory to establish an interface between DRAM and CPU. The subsequent main thrust for SRAM innovation is low force applications. For this situation, SRAMs are utilized in most compact hardware on the grounds that the Measure revive current is a few significant degrees more than the low-power SRAM backup current. Numerous classifications of mechanical and logical subsystems, car gadgets, and comparative, contain static Slam. A few megabytes of SRAM might be utilized in complex items, for example, advanced cameras, phones, synthesizers, and so on. SRAM is additionally utilized in PCs, workstations, switches and edge hardware: interior CPU stores and outer burst mode SRAM reserves, hard plate supports, switch cushions, and so on. LCD screens and printers likewise ordinarily utilize SRAM to hold the picture showed or to be printed. Little SRAM supports are additionally found in CDROM and CDRW drives to cushion track information, which is moved in hinders rather than as single qualities. The equivalent applies to link modems and comparative hardware associated with PCs [44], [46].

A SRAM store comprises of a variety of memory cells alongside edge hardware, for example, address decoder, sense speakers and compose drivers and so on those empower perusing from and composing into the exhibit. A great SRAM memory engineering is appeared in Figure 5-9.

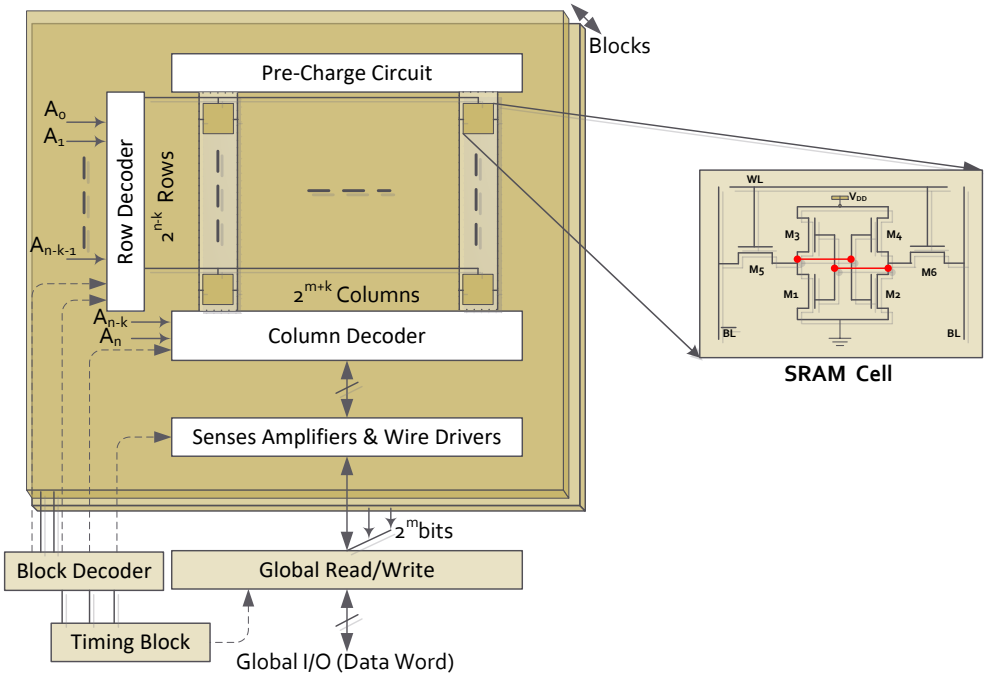


Figure 5-9: SRAM Architecture [46]

The memory cluster comprises of 2^{n} expressions of 2^{m} bits each. Each piece of data is put away in one memory cell. They share a typical word-line (WL) in each line and a piece line sets (BL, supplement of BL) in every section. The elements of each SRAM cluster are restricted by its electrical qualities, for example, capacitances and protections of the bit lines and word lines used to get to cells in the exhibit. Consequently, enormous size recollections might be collapsed into various squares with predetermined number of lines and sections. In the wake of collapsing, so as to meet the bit and word line capacitance prerequisite each line of the memory contains 2^{k} words, so the exhibit is genuinely sorted out as 2^{n-k} lines and 2^{m+k} segments. Each cell can be haphazardly tended to by choosing the fitting word-line (WL) and bit-line sets (BL, supplement of BL), separately, actuated by the line and the segment decoders. The fundamental SRAM cell is appeared in inset of Figure 5-9. It comprises of two cross-coupled inverters (M3, M1 and M4, M2) and two access semiconductors (M5 and M6). The entrance semiconductors are associated with the word line at their particular door terminals, and the bit lines at their source/channel terminals. The word line is utilized to choose the cell while the bit lines are utilized to perform peruse or compose procedure on the cell. Inside, the cell holds the put away an incentive on one side and its supplement on the opposite side. The two complementary bit lines are used to improve speed and noise suppression performance. The voltage transfer characteristics (VTC) passes on the key cell

plan contemplations for peruse and compose tasks. In the cross-coupled design, the put away qualities are spoken to by the two stable states in the VTC. The cell will hold its present status until one of the inside hubs crosses the switching threshold, V_S. At the point when this happens, the cell will flip its inner state. Consequently, during a read activity, we should not upset its present status, while during the compose activity we should constrain the inward voltage to swing past V_S to change the state [46], [47].

5.3.1.2 Dynamic RAM (DRAM)

For more than a quarter century, Dynamic Random Access Memory (DRAM)-ICs have been existed and developed from the most punctual 1-kilobit (Kb) to the ongoing 1-gigabit (Gb) era. This evolution has been attained through advances in both semiconductor procedure and circuit structure innovation. Huge advances in process innovation have drastically decreased element size, allowing ever more elevated levels of joining. These increments in joining have been joined by significant upgrades in segment respect guarantee that general procedure arrangements remain practical and serious. Innovation enhancements, in any case, are not restricted to semiconductor handling. A large number of the advances in process innovation have been joined or empowered by progresses in circuit structure innovation. As a rule, progresses in one have empowered advances in the other. To pick up understanding into how modem DRAM chips are planned, it is valuable to investigate the advancement of DRAM. DRAM types and methods of activity will be outlined as follows [48], [49]:

- **The 1st Generation/1k DRAM:** This type of memories is sorted out as a sensible square of memory components. Its cluster is comprised of 1,024 memory components spread out in a square of $32_{rows} \times 32_{columns}$.
- **The 2nd Generation/4k-64 Meg DRAM:** As a second generation, DRAMs can be distinguished by the presentation of multiplexed address inputs, numerous memory clusters, and the memory cell of a single transistor/ a single capacitor. Besides, they offer more methods of activity for more noteworthy adaptability or higher speed activity.
- The size of this type of memories varies between 4k (4,096address locations x 1 input/output) to 64 Meg i.e. 67,108,864 bits in different sizes as:
    - 16 Meg x 4 ($16{,}777{,}216_{\text{address locations}} \times 4_{\text{input/output}}$)

- 8 Meg x 8 (8 Meg $_{\text{address locations}} \times 8_{\text{input/output}}$)
- 4 Meg x 16 (4 Meg $_{\text{address locations}} \times 4_{\text{input/output}}$)

Furthermore, as a major leap in the second generation is the transition of the power supply to be single 5V power supply at the density of 64kbit. This transition simplifies the system design from different aspects such as the memory and the processor. This is in addition to changing the technology from the NMOS to CMOS in order to fulfil the concerns over the speed, the size and the power, and the density of 1Mbits.

- **The 3rd Generation/Synchronous DRAM:** Its design has been modified to include an interface between 2nd Generation DRAM's core and the off-chip control. This is in addition to reserve the clock (CLK) for executing both of commands and operations.

5.3.1.3 Flash Memory

In the past ten years, in the field of semiconductor recycling, the most important miracle is the unstable development of flash memory showcases, which are affected by PDAs and various electronic convenience devices (multi-function PCs, mp3 sound players, computer cameras, etc.). In addition, in the next few years, convenient frameworks will require more non-volatile recycling, whether it is for information storage applications, with high thickness and high synthesis throughput, or for code execution settings. In other words, it has fast and irregular access rights. In view of these market demands, a significant method of arranging Flash projects and relative progress is to characterize two important parts of the application: – code storage, where the program or work frame is stored and executed by the chip or microcontroller, which continuously record and peruse information files related to pictures, music and voice. Different types of Flash units and structures have been proposed previously. From design point of view, flash memories can be divided as:

- Fowler-Nordheim Tunneling (FN),
- Channel Hot Electrons (CHE),
- Hot Holes (HH)
- Source Side Hot Electronics (SSHE).

Among these designs, there are now two types that can be regarded as industry standards: NOR Flash, a common belief in the field of code and information storage due to its adaptability, and NAND Flash, which is enhanced for the information storage market. In the

accompanying content, we will only introduce the basic concepts of NOR flash memory cells, constant quality issues, improvements and scaling modes. However, most of these considerations are substantial for NAND, this is due to that they both depend on the concept of Sliding/floating gate MOS transistor as shown in Figure 5-10 [50]. It depicts the schematic of floating-gate MOS transistor, where the gate is isolated completely by a dielectric, Control Gate (CG) that has been administrated by a capacitive coupling, and the Floating Gate (FG). Being electrically disengaged, the FG goes about as the putting away cathode for the phone gadget; charge infused in the FG is looked after there, permitting adjustment of the "obvious" limit voltage (i.e., seen from the CG) of the phone semiconductor. Clearly the nature of the dielectrics ensures the non-volatility, while the thickness permits the likelihood to program or delete the cell by electrical heartbeats. Normally the entryway dielectric, i.e., the one between the semiconductor channel and the FG, is an oxide in the scope of 9–10 nm and is designated "burrow oxide" since FN electron burrowing happens through it. The dielectric that isolates the FG from the CG is shaped by a triple layer of oxide–nitride–oxide (ONO). The ONO thickness is in the scope of 15–20 nm of proportional oxide thickness. The ONO layer as interpoly dielectric has been acquainted all together with improves the passage oxide quality. Truth be told, the use of warm oxide over polysilicon suggests development temperature higher than 1100 C, affecting the underneath burrow oxide.

On the off chance that the passage oxide and the ONO act as perfect dielectrics, at that point it is conceivable to schematically speak to the vitality band graph of the FG MOS semiconductor. It very well may be seen that the FG goes about as an expected well for the charge. When the charge is in the FG, the passage and ONO dielectrics structure likely boundaries. The nonpartisan (or decidedly charged) state is related with the consistent state "1" and the adversely charged state, comparing to electrons put away in the FG, is related with the coherent "0".

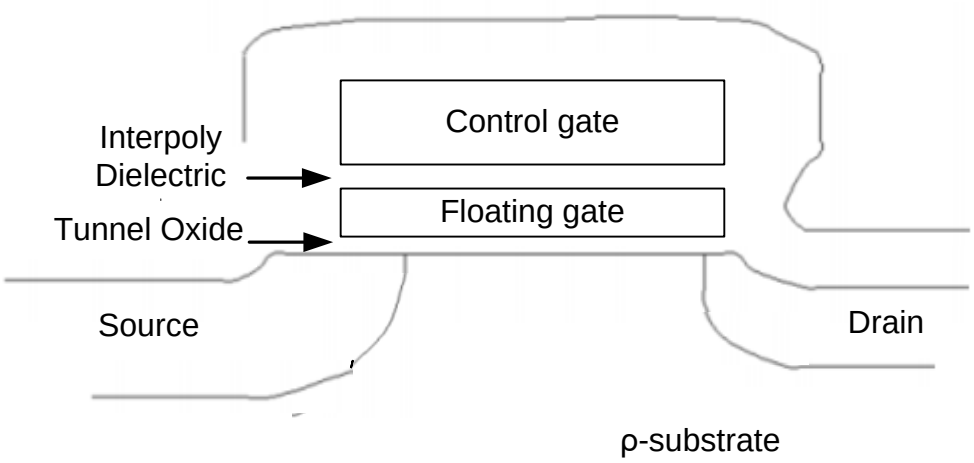


Figure 5-10: A Flash cell schematic cross section [50]

The "NOR" Flash name is identified with the manner in which the phones are orchestrated in a cluster, through lines and segments in a NOR-like structure.

5.4 Causes and Implications of Memory

IoT system classified into three networks first is the Edge network which contains devices (Embedded, sensors, actuators that can be constrained or unconstrained) that operate with external world monitoring, sensing, actuating, etc. Second is the Fog network which is the Gateway and high-end servers that broker, data collecting and processing, commanding, analytics, etc. Third is the cloud network which is the cloud platforms and high-end servers that store and Machine deep learning, without direct active management by the user.

5.4.1 Compute-Constrained Devices

Constrained devices are the edge nodes having (sensors, actuators, smart object or smart devices micro controller) which can handle a specific application target, linked to gateway-like devices and return communication with the IoT cloud platforms. They communicate with low-power and data rate- wireless protocols. IoT have embedded computing devices spread within it and they are considered the resource constrained. This resources constraint affects memory, processing capabilities, the low-power radio standards utilized, and constraint the network interfaces. These embedded constrained devices open opportunities to develop new application such as smart homes, eHealth and many other applications. However, it needs to

have capabilities for performing networking to integrate with the Internet. Table 5-2 presents a few typical low-power constrained devices [51].

Table 5-2: Different Low-Power Constrained Devices Overview

Type	CPU	RAM	Flash/ROM
Crossbow TelosB	16-Bit MSP430	10 KB	48 KB
RedBee EconoTAG	32-Bit MC13224v	96 KB	128 KB
Atmel AVR Raven	8-Bit ATMega1284P	16 KB	128 KB
Crossbow Mica2	8-Bit ATMega 128L	4 KB	128 KB

As seen from the table that these resources are not providing much storage or memory capabilities which can form device restriction.

5.4.2 Constrained Node, Constrain Network and Constrained-Node Network

Some basic terms should be identified in order to help in understanding the work of constrained environment, which is according to the Terminology for Constrained-Node Networks [52] as:

- **Constrained Node:** A node where some of the characteristics that are otherwise pretty taken for granted for Internet nodes at the time of writing are not attainable. This is often due to cost constraints and/or physical constraints on characteristics such as size, weight, and available power and energy. The limits on power, memory (flash and RAM), small computing capabilities, and processing resources lead to upper bounds on state, code space, and processing cycles, so making optimization of energy and network bandwidth usage a dominating consideration in design requirements. Also, some layer-2 services such as full connectivity and broadcast or multicast may be lacking. The constrained Node can be categorized into three classes shown in the Table 5-3, which can help in design.

Table 5-3: Constrained Node Class

Class	RAM	Flash	Network Stack	Security
Class 0	<10 KiB	<100 KiB	No support	No support
Class 1	≈10 KiB	≈100KiB	Specifically designed network stack	Supports security
Class 2	≈50 KiB	≈250 KiB	Capable of supporting the same network stack as used on servers	Supports security

- **Constrained-Node Network:** A network whose characteristics are influenced by being composed of a significant portion of constrained nodes. A constrained-node network is always a constrained network because of the network constraints stemming from the node constraints, but it may also have other constraints that already make it a constrained network. A constrained network is composed of a lot portion of constrained nodes which devolved at the edge of IoT system. Main structure block of Constrained-Node Network is sensors, actuator, cluster, communication channel, Aggregators, eUtility, Decision Trigger. Those building blocks are discussed below and shown in Figure 5-11 [52], [53].

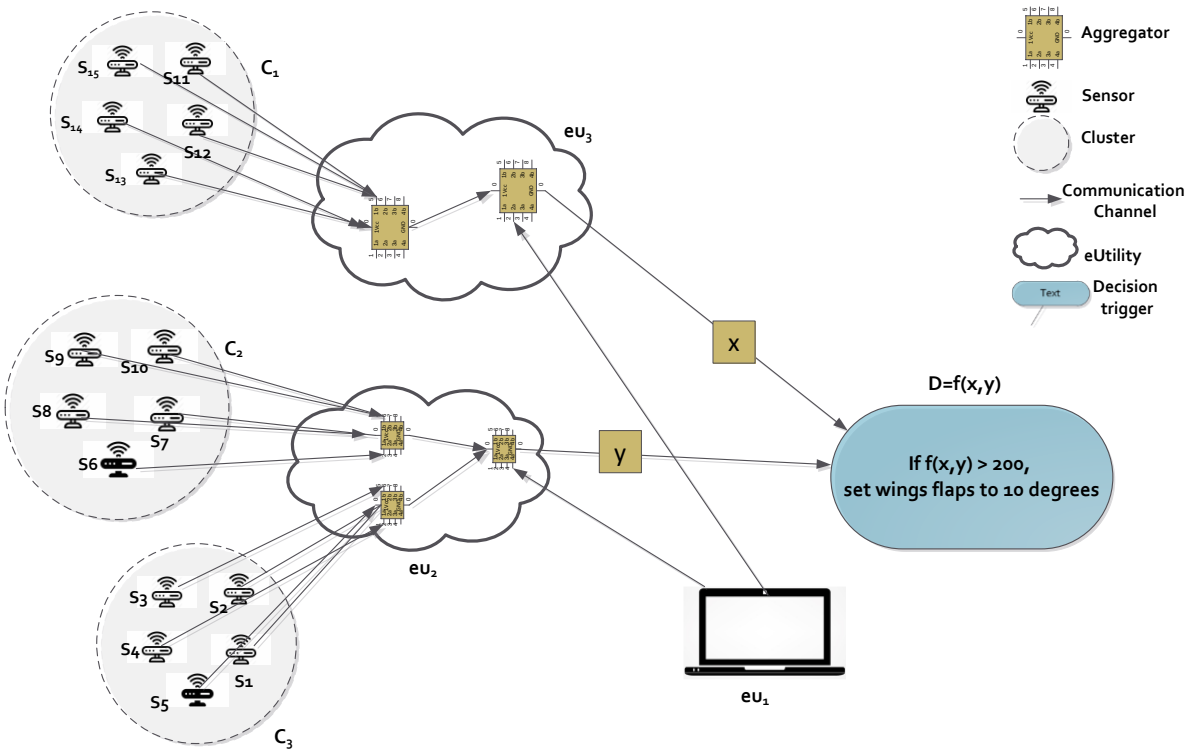


Figure 5-11: Constrained Node Network [52]

- **Sensors:** A sensor is an electronic utility that measures physical properties such as temperature, acceleration, weight, sound, location, presence, identity, etc. All sensors employ mechanical, electrical, chemical, optical, or other effects at an interface to a controlled process or open environment, e.g. temperature sensor and acceleration sensor. Sensors are essential physical component in constrained-node network and the most important aspects in IoT which is not possible without sensor technology. Sensors are mostly small size, low cost and consume low power, and some may have an Internet access capability. Sensors may show these basic characteristics: Accuracy and Inaccuracy, Range, Resolution, Tolerance, Precision, Hysteresis effects and dead space Linearity and Sensitivity, Security and Reliability [52]. Different types could be summarized as [53]:
  a. Mobile Phone Based Sensors (like: accelerometer senses the motion and speed of a mobile phone, camera and microphone sensors which capture visual and audio information, light sensor)
  b. Medical Sensors which can measure many health parameters and monitoring a patient's health when they are alone or not in hospital.

c. Neural Sensors which measure the neural signals of brain that can be used to train the brain to focus, have attention, manage stress, and have better mental health.

d. Environmental and Chemical Sensors which play a major role in monitoring and control environment to have better usage of environments parameters such as temperature, humidity, air pressure, gas and water levels sensors and many other physical environment sensors that are used to reduce energy usage or pollution levels. Chemical sensor is used in food, pharmacy, biomedical applications.

- **Actuator**: Actuator is the unit that obtain physical movement by converting energy usually electrical, air or hydraulic into mechanical action (enables movement), like the muscles in human body. In IoT field actuator change electrical energy into another type of energy according to demand such as light, display, motors, heating or cooling. According to the actuator operation they are classified to three groups (electrical, hydraulic and pneumatic).
- **Cluster:** Cluster is a set of sensors and actuators with the data they output, they are not physical, and can change their sensors and data any time, in constrained-node network grouping similar objects (sensors, actuator). It is called clustering the main purpose of clustering is to easily deal with the objects, maintenance problem and routing decision, and hence clustering is a solution to increase the efficiency.
- **Communication channel:** Communication channel is the physical transmission medium in which data is transferred (. e.g. USB, wireless, direct). Communication channel can be disturbances, delays and interruptions so performance, security and reliability are main issue in communication channel. Communication channels will have a physical or virtual side to them, or both. Protocols and associated implementations provide a virtual side; cables provide a physical side. Dataflow may be unidirectional or bi-directional.
- **Aggregators:** An aggregator is a software implementation based on mathematical function(s) that transforms groups of raw data into intermediate, aggregated data. Raw data can come from any source. Aggregators help in managing 'big' data [52]. Aggregator have a low power communication link with thousands sensors.

Aggregators is assigned to aggregate information which forward to cloud for final action, Aggregators merge large volumes of data into lesser amounts.

- **eUtility: eUtility is a service or hardware or software that support aggregators suppling data and enable computing resources and storage as needed.** It executes processes or feed data into the overall workflow of a Network of Things.
- **Decision Trigger: Decision Trigger is the end-purpose software which computes and takes action if needed** to satisfy the purpose, specification, and requirements. Decision trigger should have a corresponding virtual implementation and may have a unique owner.

5.4.3 The need for Management of constrains devices and constrained devises restrictions

It is now clear that constrained devices have limited storage capabilities CPU memory as depicted in Table 5-2, and power resources, so this network itself considered may be as challenging one. In addition, constrained devices can be connected to unconstrained network, constrained devices responsible of gathering the information, generating and sending the information to one or more station. According to this discussion to manage the network is playing a significant role by monitoring network status, detecting faults, perform actions or remove faults and maintain normal operation to insure efficiency and application performance. Energy-Management for constrained devices that is connected to a network form a challenge. This Energy-Management which is responsible for supplying energy to various devices and component even devices contained batteries they also monitored and managed. Constrained devices are having considerable small size and are not fixed. Because of their size and frequently changing location property they are not able to access the power all the time. So the low power consumption is universal constraint of the constrained devices. Either they use battery technologies or they can use some techniques for taking power from their environment there is a need to manage power which made with long lasting life of devices. Energy management is especially relevant to the Smart Grid. A Smart Grid is an electrical grid that uses data networks to gather and act on energy and power-related information in an automated fashion with the goal to improve the efficiency, reliability, economics, and sustainability of the production and distribution of electricity [54].

Device life cycle and quality with time and environment must consider to not causing a restriction while using constrained devices. The need for access technology rises up with constrained devices to not impose a restriction. So the constrained access technologies are a demand to connect the constrained and traditional unconstrained networks. The need for Constrained Access Technologies arises due to resource restrictions, embedded devices use sensors and actuators use the low-power, low-data- rate wireless access technologies like Digital Enhanced Cordless Telecommunication (DECT), Low-Rate Wireless Personal Area Networks (LR-WPAN), Ultra Low Energy (ULE), or Bluetooth for network connectivity. In this case, it is important for the network management system to be aware of the restrictions imposed by these access technologies to efficiently manage these constrained devices. In particular, such low-power, low data-rate access technologies have small frame sizes. So it would be important for the network management system and management protocol to choose craft packets in a way that avoids fragmentation and reassembly of packets since this can use valuable memory on constrained devices. Devices using such access technologies might operate through a gateway that operates between these access technologies and more traditional Internet protocols. A hierarchical approach to device management is useful, wherein the gateway device is responsible of devices connected to it, while the network management system conducts management operations only to the gateway. A new internet protocol is combined and allows group communication named Constrained Application protocol (CoAP), used with constrained nodes and constrained networks which allow access virtually to huge number of smart objects anywhere and limits the restricted resources such as power, CPU, and memory which turns groups of resources into entities [54], [55].

Security and privacy issue is an essential parameter for reliable and safe communication between constrained-node networks. However protecting the communication between devices with limited resources is a complex issue specially when transmitting sensitive data such military and medical application. On other hand the cost factor will rise up. Some protocols developed for reliable and safe communication between constrained-node network to ensure securing and packet delivery 100% such the ContikiMAC Radio Duty cycle protocols. The National Institute of Standards and Technology (NIST) improve how to manage security guidelines and released a cyber-security framework document. The NIST cybersecurity framework defines five major functions (Identify, protect, detect, respond, recover). Privacy

can only be ensured if data owner has fully control over owned information, owner must know that what data is collected, who is collecting, when it was collected and where it is processed. Another point, the data collector must be authorized by service provider e.g., authorize transportation organizations, medical authorize organizations, authorize research institutions, etc. Furthermore, the data should only be stored for limited time under need and demand otherwise it must be destroyed immediately [55], [56].

5.4.4 Applications for Constrained Devices

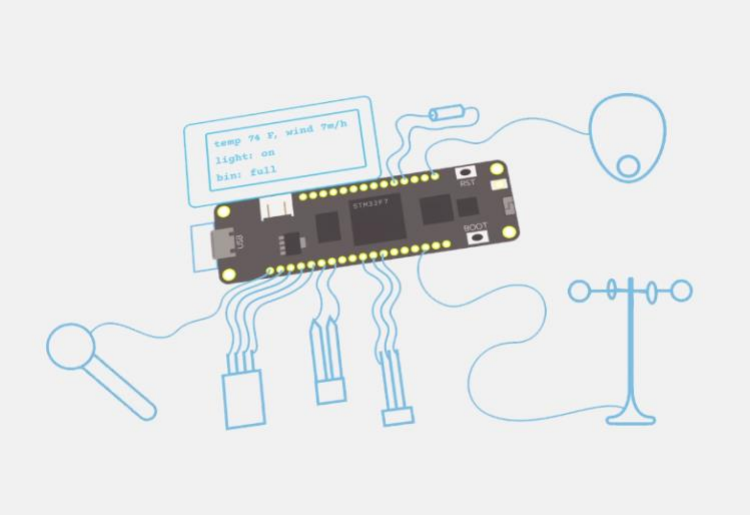
The constrained devices have wide range of applications in my areas, which they are summarized as:

- **Medical Applications:** constrained devices used in wide range for health supervision and control, such as
  - Blood pressure and heart rate application.
  - Pacemaker that help to control heartbeat.
  - Hearing aids that pick up sounds and amplifies the sounds going into ear.
  - Fitness and wellness application that calculate the fitness indicators, motion rate, sleep, and weight control, encourage user to exercise and empower self-monitoring.
  - At the health organization level which provides the services of medical application that used constrained devices not attached to human bodies only, but also used it in its infrastructure to monitor or to ensure treatment applied properly or in emergency- notification systems or remote patient monitoring.
  - Dental Health: records brushing information to study the person's brushing habits and share the statistics with the dentist.
- **Building Automation:** lighting, pumps, boilers, environment temperature, building plants irrigation, Elevator control, air fans, CO2 levels, and more equipment. Unit mechanical, electrical, and electronic inside buildings can be automated by a network of constrained devices to better manage that equipment and optimize energy spend. It requires the deployment of a large number (10 to 100,000) of sensors that monitor the status of devices, parameters inside the building. It needs Controllers with different specialized functionality for areas within the building or the totality of the building.

- **Transport Application:** connecting cars, vehicle-tracking systems, smart parking, electronic toll-collection systems, traffic light, logistic and fleet management, vehicle control, and traffic and transportation management. Safety and roadside assistance are some of application employed constrained devices to monitor and control transportation system. Here some advantages and enhancement for transportation authorities:
  - Improved customer services and dependable transportation.
  - Increased safety: Sensor data tracks speeds, vehicle part conditions and temperature, and the number of cars at an intersection. Authorities can use this information to improve the safety of transit system operations.
  - Reduced congestion, energy use, traveling time by using real time data to Vehicle forwarding and meet demands.
  - Minimize operating costs and improve system capacity.
- **Home Automation:** It controls lighting, heating, doors and windows, air conditioning, appliances; gardens (e.g. detect dryness in the soil to trigger the irrigation system, Detect grass height to trigger Lawn mowers, entertainment and home security devices to improve convenience, comfort, energy efficiency, and safety).

6 IoT Microcontrollers, Sensors for Data Acquisition and Actuators

Author(s): Fabrizio Granelli



6.1 Implementing the Internet of Things

The major feature of the Internet of Things is the fact that it brings the Internet in the real world where we live. With the Internet of Things, objects and places can communicate by using Internet technology. This section sheds some light on how we can enable this interconnection between "our" World and the Digital World of the Internet.

Interaction between the World and computers can be achieved by means of sensors and actuators (that we will discuss later). However, the Internet of Things goes beyond this, and it promises to integrate the Internet in the real world. For doing this, we cannot rely only on common computing platforms such as computers, laptops or mobile phones, but we need simpler, smaller and less consuming devices to "read" sensors and "control" actuators. Those are the microcontrollers.

The Internet of Things can be implemented by integrating the following technologies:

- Microcontrollers: the «brain» capable of interacting with the World (sensing and actuating);
- Real-time systems: IoT systems should be capable of operating in real-time and with bounded delay;
- Embedded software: specific software written for microcontrollers

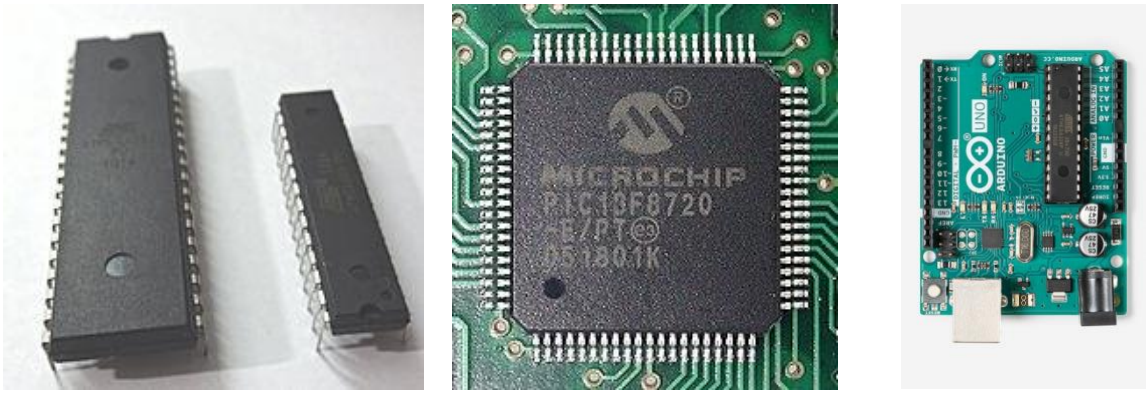
The following sub-sections provide an introduction to those concepts.

6.2 Microcontrollers

A microcontroller, or microcontroller unit (acronym: MCU), is a small computing machine on a single metal-oxide-semiconductor (MOS) integrated circuit (IC) chip. To some extent, a MCU is similar to a system on a chip (SoC), but it is still less sophisticated than that; in practice, a SoC may include a microcontroller as one of its components.

The main difference between micro-controllers and micro-processors, besides the small factor, consists in the fact that micro-controllers are designed for embedded applications, while micro-processors are used in personal computers or general purpose computing architectures.

A microcontroller consists of one or more CPUs (processor cores), with the addition of memory and programmable input/output peripherals. Input/output connections enable the microprocessor to acquire data in input and send data or commands in output.



(a) (b) (c)

Figure 6-1: Microcontrollers examples: two ARM MCU (a), a PIC MCU (b), and Arduino UNO (c).

In a microcontroller, programs are loaded in the memory. Such memory is typically integrated on the chip, as well as some RAM. Technologies for storage are ferroelectric RAM, NOR flash or OTP ROM.

6.2.1 Examples of microcontrollers

When designing IoT systems and services, it is important to have a good knowledge about the available platforms and their specific characteristics, in order to identify the one which is best suited for our goals.

6.2.1.1 Arduino Uno

Arduino Uno is a micro-controller belonging to the Arduino family, which is an open-source prototyping platform, consisting of a programmable circuit board and an Integrated Development Environment (IDE) to upload the code to the physical board. Arduino can be programmed directly by any personal computer using the IDE and a USB cable.

An important and remarkable aspect is that Arduino is completely open-source. Indeed, both its software and hardware specification are available, so that anyone can assemble the simplest Arduino modules themselves by hand or even work at improving and contributing to its design. Pre-assembled Arduino modules are also available and they can be purchased at a

relatively low price. Open source projects like Arduino lower the entry barrier for developers that are looking to experiment with interactive objects, especially in the Internet of things.

As an example from the Arduino family, Arduino Uno [57] is a micro-controller board based on the ATmega328P chipset. Its main features include:

- 14 digital input/output pins (of which 6 can be used as PWM [Pulse Width Modulated] outputs)
- 8 analog inputs
- a 16 MHz ceramic resonator (CSTCE16M0V53-R0)
- a USB connection, a power jack, an ICSP header and a reset button.

6.2.1.2 Arduino Mega

The Arduino Mega 2560 is another micro-controller board belonging to the Arduino family, and it is based on the ATmega2560 chipset. Its main features are:

- 54 digital input/output pins (of which 15 can be used as PWM [Pulse Width Modulated] outputs)
- 16 analog inputs, 4 UARTs (hardware serial ports)
- a 16 MHz crystal oscillator
- a USB connection, a power jack, an ICSP header, and a reset button.

6.2.1.3 Raspberry PI

The Raspberry Pi is a low cost, credit-card sized computer that plugs into a computer monitor or TV, and it uses a standard keyboard and mouse. It is a capable compact device that enables to explore computing, to build IoT software (for its small form factor and limited power consumption) and to learn how to program. As a difference with respect to Arduino, Raspberry Pi is actually a complete computer architecture, not a simple micro-controller.

The Raspberry Pi is targeted to operate in the open source ecosystem. Being a small PC, it runs Linux (a variety of distributions, and recently Windows 10, too), and its main supported operating system, Raspberry OS, is open source and runs a suite of open source software.

The Raspberry Pi Foundation contributes to the Linux kernel and various other open source projects as well as releasing much of its own software as open source. It should be noted that

even if the Raspberry Pi's schematics are released, the board itself is not open hardware, as the Raspberry Pi Foundation relies on related income.

The main features of Raspberry Pi are:

- Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
- 2GB, 4GB or 8GB LPDDR4-3200 SDRAM (depending on model)
- 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE
- Gigabit Ethernet port
- 2 USB 3.0 ports; 2 USB 2.0 ports.
- Raspberry Pi standard 40 pin GPIO header (fully backwards compatible with previous boards)
- 2 × micro-HDMI ports (up to 4kp60 supported)
- stereo audio and composite video port
- H.265 (4kp60 decode) and H264 (1080p60 decode, 1080p30 encode) support
- OpenGL ES 3.0 graphics
- Micro-SD card slot for loading operating system and data storage
- Power over Ethernet (PoE) enabled (requires separate PoE HAT)

6.2.1.4 ARM Microcontrollers

ARM (acronym for Advanced RISC Machine and originally Acorn RISC Machine) is a family of reduced instruction set computing (RISC) architectures for computer processors. As such, their architecture typically requires fewer transistors than those with a complex instruction set computing (CISC) architecture (such as the x86 processors found in most personal computers), which improves cost, power consumption, and heat dissipation.

Today, ARM represents an interesting solution for light, portable, battery-powered devices, but it is also becoming popular for servers and desktops. Moreover, ARM is also growing in relevance for deploying power-efficient solutions in the case of supercomputers or data centers.

For the above reasons, ARM microcontrollers are extremely popular as they represent a building block for several low-power devices. The following tables presents a sample of ARM MCU specifications:

- USART, SPI, I2C
- 2x Quad-SPI
- 16- and 32-bit timers
- SAI + audio PLL
- CAN
- Camera IF
- ART Accelerator™
- Chrom-ART Accelerator™
- 2x 12-bit DACs
- Temperature sensor
- Low voltage 1.71 to 3.6V
- VBAT mode
- Unique ID
- Capacitive touch-sensing

Product line	Flash (KB)	RAM (KB)	Memory I/F	Op-Amp	Comp.	Sigma Delta Interface	12-bit ADC 5 Msps 16-bit HW oversampling	USB2.0 OTG	TFT Display Interface	*Chrom-GRC™	MIPI-DSI	AES 128-/256-bit
STM32L4R9/S9												
STM32L4R9 USB OTG & MIPI-DSI	1024 to 2048	640	SDIO FSMC	2	2	8x ch	1	•w	•	•	•	
STM32L4S9 USB OTG & MIPI-DSI & AES	1024 to 2048	640	SDIO FSMC	2	2	8x ch	1	•	•	•	•	•
STM32L4R7/S7												
STM32L4R7 USB OTG & TFT Interface	1024 to 2048	640	SDIO FSMC	2	2	8x ch	1	•	•	•		
STM32L4S7 USB OTG & TFT Interface & AES	2048	640	SDIO FSMC	2	2	8x ch	1	•	•	•		•
STM32L4R5/S5												
STM32L4R5 USB OTG	1024 to 2048	640	SDIO FSMC	2	2	8x ch	1	•				
STM32L4S5 USB OTG & AES	2048	640	SDIO FSMC	2	2	8x ch	1	•				•
STM32L4P5/Q5												
STM32L4P5 USB OTG	512 to 1024	320	SDIO FSMC	2	2	4 ch	2	•	•			
STM32L4Q5 USB OTG & AES	1024	320	SDIO FSMC	2	2	4 ch	2	•	•			•

Figure 6-2: ARM Cortex MCUs specifications.

6.3 Real-time Systems

A real-time system is described as one which "controls an environment by receiving data, processing them, and returning the results sufficiently quickly to affect the environment at that time".

Real-time systems need to be programmed by proper real-time software environments. For real-time software development may follow the following approaches: synchronous programming languages, real-time operating systems, and real-time networks.

6.4 Embedded Software

Embedded software is used to program and control embedded systems.

Embedded software is a specific type of computer software, specialized for the particular hardware to run onto. Moreover, this kind of software is characterized by time and memory constraints. Embedded software is sometimes used in substitution of the term firmware.

Typically, not all functions of embedded software are initiated or controlled via a human interface, but through machine-interfaces instead.

To make some examples, embedded software is used to control the electronic components, robots, appliances, televisions, set-top-boxes, cars, security systems, digital watches, etc.

6.5 IoT Operating Systems

The main idea of the Internet of Things is connectivity between the web and sensor-based tiny devices on a system. In this scenario, heterogeneity is a key aspect to consider, as each IoT device has its perspective. So, variability is obvious if we consider operating systems for the Internet of Things. Indeed, most of them are focused on bringing interoperability for a heterogeneous set of devices and networks.

An IoT operating system is software that ensures connectivity between IoT applications and embedded devices.

The following subsections propose some open source IoT operating systems, which are practical to use for IoT devices.

6.5.1 Arduino IDE

The open-source Arduino Software (IDE) is written in Java and based on Processing and other open-source software, and it can be used with any Arduino-compatible board [58]. The purpose of the Arduino IDE is to provide a programming interfaces to write code and upload it to the board. The development environment runs on any well-known operating system, including Windows, Mac OS X, and Linux.

6.5.2 ARM MCU Programming

Different Microcontroller Development Kits for ARM based microcontrollers are available (including Arduino IDE). Typically, it is necessary to use a specific hardware programmer to interact with the MCU.

ARM programming libraries are hardware specific.

6.5.3 Contiki

Contiki [59] and latter Contiki-NG is an open-source IoT operating system for low power microcontrollers and other IoT devices. Contiki enables communications across Internet protocol IPv6 and IPv4 networks. Contiki is an operating system for networked, memory-constrained systems with a focus on low-power wireless Internet of Things devices. Contiki is used for managing street lighting, sound monitoring in smart cities, radiation monitoring, and alarms management. The software is open-source software and released under a BSD license. Core language used in Contiki is C language.

Contiki is characterized by an extremely small footprint: only about 10 kilobytes of random-access memory (RAM) and 30 kilobytes of read-only memory (ROM). It natively supports multitasking, the Internet Protocol Suite (TCP/IP stack) and many IoT-related wireless protocols, such as 6lowPAN, RPL and CoAP. The complete system deployment, including a graphical user interface, needs about 30 kilobytes of RAM.

6.5.4 Android Things

Android Things [60] is an Operating System for the IoT developed by Google. Android Things focuses on low-power and memory constrained Internet of Things (IoT) devices, which are usually built from different MCU platforms.

Android Things follows a similar resource-constrained formula as Contiki, with a lightweight implementation requiring only 32-64 Kb of RAM. This lightweight OS supports both Bluetooth Low Energy and Wi-Fi.

On a side note, in addition to Android Things, Google also introduced the Weave protocol, which allows IoT to communicate with other compatible devices.

6.5.5 Riot

Riot [61] belongs to the class of free open source Operating Systems for the Internet of Things, and it is called the Linux of the IoT world, as it is supported by a large development community under GNU Lesser General Public License.

However, Riot is different from other similar OSs. First of all, it is based on a microkernel architecture. Secondly, as a difference with similar lightweight OSs presented above (such as TinyOS or Contiki), it allows the usage of high level programming languages such as C and C++.

Riot runs on 8-bit (such as AVR Atmega), 16-bit (such as TI MSP430) and 32-bit (such as ARM Cortex) processors. A native port also enables RIOT to run as a Linux or macOS process, enabling use of standard development and debugging tools such as GNU Compiler Collection (GCC), GNU Debugger, Valgrind, Wireshark etc.

RIOT provides multiple network stacks, including IPv6, 6LoWPAN, or Content centric networking and standard protocols such as RPL, User Datagram Protocol (UDP), Transmission Control Protocol (TCP), and CoAP. Riot provides full multithreading and real-time abilities, as well as SSL/TLS, which is supported by popular libraries such as wolfSSL.

6.5.6 Apache Mynewt

Apache Mynewt is an IoT OS built for tiny embedded IoT devices [62]. It is a real-time modular operating system dedicated to the IoT ecosystem, made available under Apache License 2.0.

Apache Mynewt supports priority-based scheduling, preemptive multithreading, multistage software watchdog, memory heap and memory pool allocation, etc.

As in previous cases, the OS is extremely lightweight, with a 6 kB kernel. This makes Apache Mynewt a suitable solution for building embedded systems (industrial IoT equipment, medical devices) on top of different microcontroller platforms. The OS is designed for connectivity, and it comes with a full implementation of the Bluetooth Low Energy 4.2 stack. The basic OS, with the inclusion of BLE and various utilities such as the default file system, console, shell, logs, stats, etc., reaches an image size of approximately 96 KB on the Nordic nRF51822 Bluetooth SoC.

6.5.7 Huawei LightOS

LightOS is an IoT Operating Systems by Huawei that provides a standard API for the diverse IoT fields. LightOS [63] is a secure, interoperable, low-power operating system.

LightOS uses middleware to remove the extra cost for the development of IoT devices, and it contains one of the smallest kernels (6kB), comparing with other operating systems.

Various network access protocols are supported: NB-IoT, Ethernet, Bluetooth, Wifi, Zigbee, and more.

6.5.8 Zephyr

Zephyr [64] is a real-time operating system (RTOS) built for IoT applications supported by the Linux Foundation. The size of the OS image is in line with the previously described alternatives, with 8kb of RAM and 512 kb of ROM.

Zephyr includes: a small kernel, a flexible configuration and build system for compile-time definition of required resources and modules, implementation of a relevant set of protocol stacks (IPv4 and IPv6, OMA LWM2M, MQTT, 802.15.4, Bluetooth Low Energy, CAN), a virtual file system interface with several flash file systems for non-volatile storage, and management and device firmware update mechanisms.

Zephyr inherits the Kconfig and device tree functionalities for its configuration from the Linux kernel. However, such functionalities are implemented in Python for portability to non-Unix operating systems. Zephyr software is based on the CMake package, which allows Zephyr applications to be built on Linux, macOS and Microsoft Windows.

6.5.9 Snappy

Snappy [65] is a Ubuntu core IoT OS, especially developed for Raspberry Pi. It derives from the Linux package “snap”, which includes libraries, kernel, and major applications.

Snappy guarantees strong security to IoT devices with the help of Ubuntu community research. Snappy distributes applications as snaps, being its native packaging system.

6.5.10 TinyOS

TinyOS [66] is a component-based open-source operating system, developed by the TinyOS Alliance. The core language of TinyOS is nesC, which is a variation of the well-known of C language. A specific and characterizing component of TinyOS is the availability of abstractions to support IoT systems functionalities, such as, for example, sensing, packet communication, routing, etc.

TinyOS targets low-power wireless devices, such as those used in wireless sensor networks (WSNs), smart dust, ubiquitous computing, personal area networks, building automation, and smart meters.

6.5.11 Fuchsia

Fuchsia [67] is a microkernel-based operating system with effective connectivity solutions.

Fuchsia runs well in low powered devices. The use of Node.js on operating system ensures application to run on smartphones, tablets as well as IoT devices.

This OS is quite flexible in terms of programming, since it supports Dart, Go, Rust, C, C++.

6.5.12 Windows IoT

Based on the former Windows Embedded, Windows IoT is a family of operating systems designed by Microsoft for use in embedded systems. Windows IoT is not an open source OS. Windows Embedded operating systems are provided to original equipment manufacturers (OEMs), in order to pre-load it on their hardware.

Windows 10 IoT represents basically an ARM version of Windows 10. Windows IoT is aimed at IoT connectivity and cloud integration.

6.5.13 TizenRT

TizenRT [68] is a Linux based operating system for both mobile applications and small embedded systems, based on a shared infrastructure called “Tizen Common”. TizenRT can support IoT devices and services, and upgraded versions of TizenRT can be integrated in smart TV, vehicles, home appliances, etc.

TizenRT is compatible with different programming languages for developing apps and services, including C, C++, and Html5.

6.5.14 Raspbian or Raspberry Pi OS

Raspberry Pi is one of the most used devices for IoT development, and Raspbian (now called Raspberry Pi OS) is its main operating system [69]. Raspberry Pi OS represents the Debian based IoT Operating System for all models of Raspberry Pi, even though other OSs can be deployed on Raspberry Pi devices, such as Ubuntu.

Raspberry Pi OS is highly optimized for the ARM CPUs within the Raspberry Pi line of compact single-board computers. In general, Raspberry Pi OS uses a modified LXDE as its desktop environment with the Openbox stacking window manager. Raspbian provides a large number of pre-installed IoT software for general use, experimental, educational purposes, etc. and it is possible to adapt various Linux software to run on the Raspberry Pi platform.

6.5.15 FreeRTOS

FreeRTOS [70] is an open-source microcontroller-based operating system for IoT development. FreeRTOS provides methods for multiple threads or tasks, semaphores and software timers. Moreover, it supports a tick-less mode for low power applications as well as thread priorities.

FreeRTOS is designed with simplicity in mind. Indeed, the kernel itself consists of only three C files. This makes the code readable, easy to port, and maintainable, even though there are a few assembly functions included where needed (especially in architecture-specific scheduler routines). As a consequence, the memory footprint is only around 6-15kb, making Free RTOS suitable for small and low power microcontrollers.

FreeRTOS has also an Amazon extension, that uses the cloud service of Amazon Web Service, called AWS IoT Core, to run IoT applications.

6.5.16 Embedded Linux

Embedded Linux [71] is an operating system built for embedded devices based on the Linux kernel. The Linux kernel has been historically ported to a variety of CPUs which are not only primarily used as processors of a desktop or server computer, but also ARC, ARM, AVR32, and others. In this framework, the Embeddable Linux Kernel Subset is a Linux distribution that fits on a floppy disk for outdated or low resource hardware.

Embedded Linux requires 100kB space in memory which makes it faster and reliable, even if larger than other IoT OSs. Moreover, embedded applications such as SQL Lite, Boa, httpd, PEG, NANO are supported.

6.5.17 mbed OS

Mbed is a free and open source operating system working on an ARM processor [72], thus appropriate for IoT projects. Mbed OS provides the Mbed C/C++ software platform and tools for creating IoT microcontroller firmware. The Operating System consists of core libraries that provide the microcontroller peripheral drivers, networking, RTOS and runtime environment, build tools and test and debug scripts. The network connections can be secured by using SSL/TLS libraries such as Mbed TLS or wolfSSL.

mbed OS supports a large number of network connectivity standards, including Wifi, Bluetooth, 6LowPan, Ethernet, Cellular, RFID, NFC, and others.

6.6 Sensing components and devices

IoT devices are built from hardware tailored to interact with the physical world. Simple stand-alone electronic circuits can be made to repeatedly flash a light or play a musical note. However, those are not the kind of electronic circuits and component that we are looking for in the Internet of Things scenario.

Indeed, in the framework of IoT applications and services, the electronic components need to be capable to communicate with the "real world". Such capability can be expressed by

- reading an input signal from an "ON/OFF" switch, or
- activating some form of output device to illuminate a single light

Depending on quantity being managed and whether the device senses or perform actions in the physical world, we can derive the table below that, for different physical parameters, provides a short overview of the technological devices to deploy.

Quantity being Measured	Input Device (Sensor)	Output Device (Actuator)
Light Level	Light Dependant Resistor (LDR) Photodiode Photo-transistor Solar Cell	Lights & Lamps LED's & Displays Fibre Optics
Temperature	Thermocouple Thermistor Thermostat Resistive Temperature Detectors	Heater Fan
Force/Pressure	Strain Gauge Pressure Switch Load Cells	Lifts & Jacks Electromagnet Vibration
Position	Potentiometer Encoders Reflective/Slotted Opto-switch LVDT	Motor Solenoid Panel Meters
Speed	Tacho-generator Reflective/Slotted Opto-coupler Doppler Effect Sensors	AC and DC Motors Stepper Motor Brake
Sound	Carbon Microphone Piezo-electric Crystal	Bell Buzzer Loudspeaker

Figure 6-3: Common sensors and actuators.

6.6.1 Sensors

Sensors are one of the basic components of IoT systems. Sensors can be used to sense a wide range of different energy forms such as movement, electrical signals, radiant energy, thermal or magnetic energy etc.

In order to perform actions in the real world, we also need actuators. Actuators can be used to switch voltages or currents.

More in general we can define:

- Sensors: devices which perform an "Input" function. They "sense" a physical change in some characteristic that might be modified in response to some excitation, for example heat or force, and convert that into an electrical signal.
- Actuators: devices which perform an "Output" function, and are used to control some external device, for example movement or sound.

In general, both sensors and actuators are electrical transducers. Indeed, they are used to convert energy of one kind into energy of another kind. For example, a microphone (input device) converts sound waves into electrical signals for the amplifier to amplify (a process), while a loudspeaker (output device) converts these electrical signals back into sound waves.

Analog Sensors produce a continuous output signal or voltage which is generally proportional to the quantity being measured. Indeed, most physical quantities to be measured as continuous in nature, including Temperature, Speed, Pressure, Displacement, Strain, etc.

Digital Sensors produce a discrete digital output signals or voltages that are a digital representation of the quantity being measured. Digital sensors produce a binary-encoded output signal in the form of a sequence of logic "1s" and a logic "0s". Depending on the specific sensor, the values might represent a single "bit" at a time (serial transmission) or a sequence of bits representing a digital sample of the input signal (parallel transmission).

6.6.1.1 Position Sensors

Position sensors are sensors capable of measuring position information. Position can be measured as distance, angle, geographical coordinates, etc.

The most commonly used of all the "Position Sensors" is the potentiometer because it is an inexpensive and easy to use position sensor. Basically, a potentiometer enables to measure the angle of rotation of an object.

The potentiometer is characterized by a wiper contact linked to a mechanical shaft that can be move in either an angular (rotational) or a linear (slider type) movement. The resistance value between the wiper/slider and the two end connections changes due to the different positions, providing an electrical signal output that is proportional to the actual wiper position on the resistive track and its resistance value. Therefore, resistance is proportional to the corresponding position.

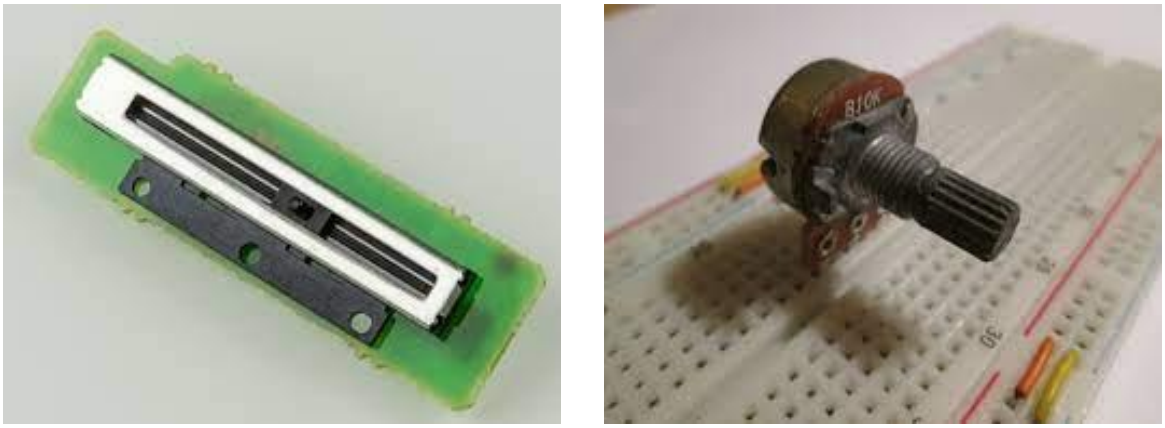


Figure 6-4: Different types of potentiometers.

Another position sensor is the proximity sensor, that detect nearby objects. The proximity sensor can be built in the form of an inductive proximity sensor. In this case, the system consists of the following main components: the oscillator, which produces the electromagnetic field, the coil which generates the magnetic field, the detection circuit which detects changes in the field (e.g. when an object enters in the promixity) and the output circuit which produces the output signal, either with normally closed (NC) or normally open (NO) contacts.

Inductive proximity sensors allow for the detection of metallic objects in front of the sensor head without any physical contact of the object itself being detected.

6.6.1.2 Temperature Sensors

Temperature sensors are used to measure the temperature of the environment. A thermistor is a special type of temperature sensor, which consists of a resistor which changes its physical resistance when exposed to changes in temperature.

Figure 6-5: The thermistor.

As an alternative, it is possible to use a thermocouple. The operating principle of a thermocouple is very simple and basic. When fused together, the junction of the two dissimilar metals (such as copper and constantan) produces a "thermo-electric" effect. Such effect provides a constant potential difference of only a few millivolts (mV) between themn and it varies based on the operating temperature.

6.6.1.3 Light Sensors

Light Sensors are photoelectric devices that convert light energy (photons) whether visible or infra-red light into an electrical (electrons) signal. Typically, they provide low voltage or low current measurements, therefore proper amplification is required in order to "read" the corresponding values from the sensor.

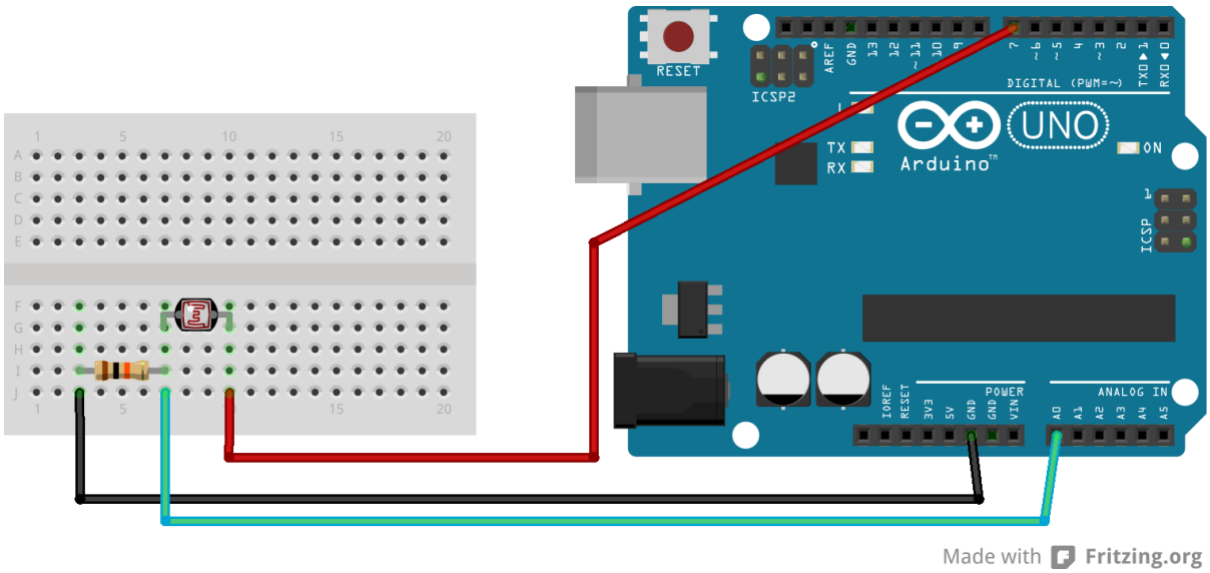


Figure 6-6: LDR circuit including the Arduino microcontroller.

6.6.1.4 Sound Sensors

Sound sensors can also be defined as sound transducers. A well-known sound transducer that can be classed as a "sound sensor" is the microphone. The microphone measures the "acoustic" pressure against its flexible diaphragm and it produces an electrical analogue output signal which is proportional to the pressure value.

6.6.1.5 Distance Sensors

Distance sensors are used to measure the distance between a target and the sensor.

A typical way to compute the distance from physical objects is to use ultrasonic sensors. Such sensors operate by emitting sound waves with a frequency that is too high for a human to hear. Such sound waves travel through the air with the speed of sound, which corresponds to approximately 343 m/s.

If there is an object in front of the sensor, it will reflect the sound waves back to the transmitter and the receiver of the ultrasonic sensor detects them. The resulting distance between the sensor and the object can be calculated by measuring how much time passed between sending and receiving the sound waves, e.g.:

Distance (cm) = Speed of sound (cm/µs) × Time (µs) / 2

For example, if the sensors measures a flight time equal to 2ms:

Distance (cm) = 0.0343 (cm/µs) × 2000 (µs) / 2 = 34.3 cm

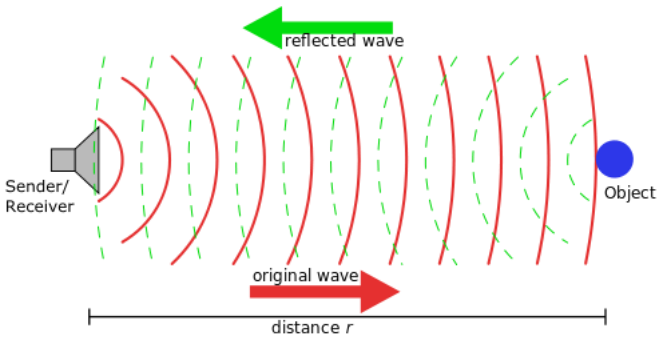


Figure 6-7: Basic concept of sonar (image from Wikimedia).

An alternative to sound waves is to use light. In this framework, one of the most relevant sensors is the LiDAR. The LiDAR, or light detection and ranging (sometimes also referred to as active laser scanning), sensor is one remote sensing method that can be used to map structure including vegetation height, density and other characteristics across a region.

By analysing the LiDAR output it is possible to directly measures the height and density of vegetation on the ground, making it an ideal tool for scientists studying vegetation over large areas.

6.6.2 Actuators

Actuators convert an electrical signal into a corresponding physical "effect" on the environment, such as movement, force, sound etc. As discussed earlier, an actuator can be also classified as a transducer, since it transforms one type of physical quantity into another. Actuators are typically activated or operated by a low voltage command signal.

Actuators can be classified as either binary or continuous devices based upon the number of stable states their output has.

6.6.2.1 Electrical Relays

Electrical relays and contactors use a low-level control signal to switch a higher voltage or current supply using a number of different contact arrangements.

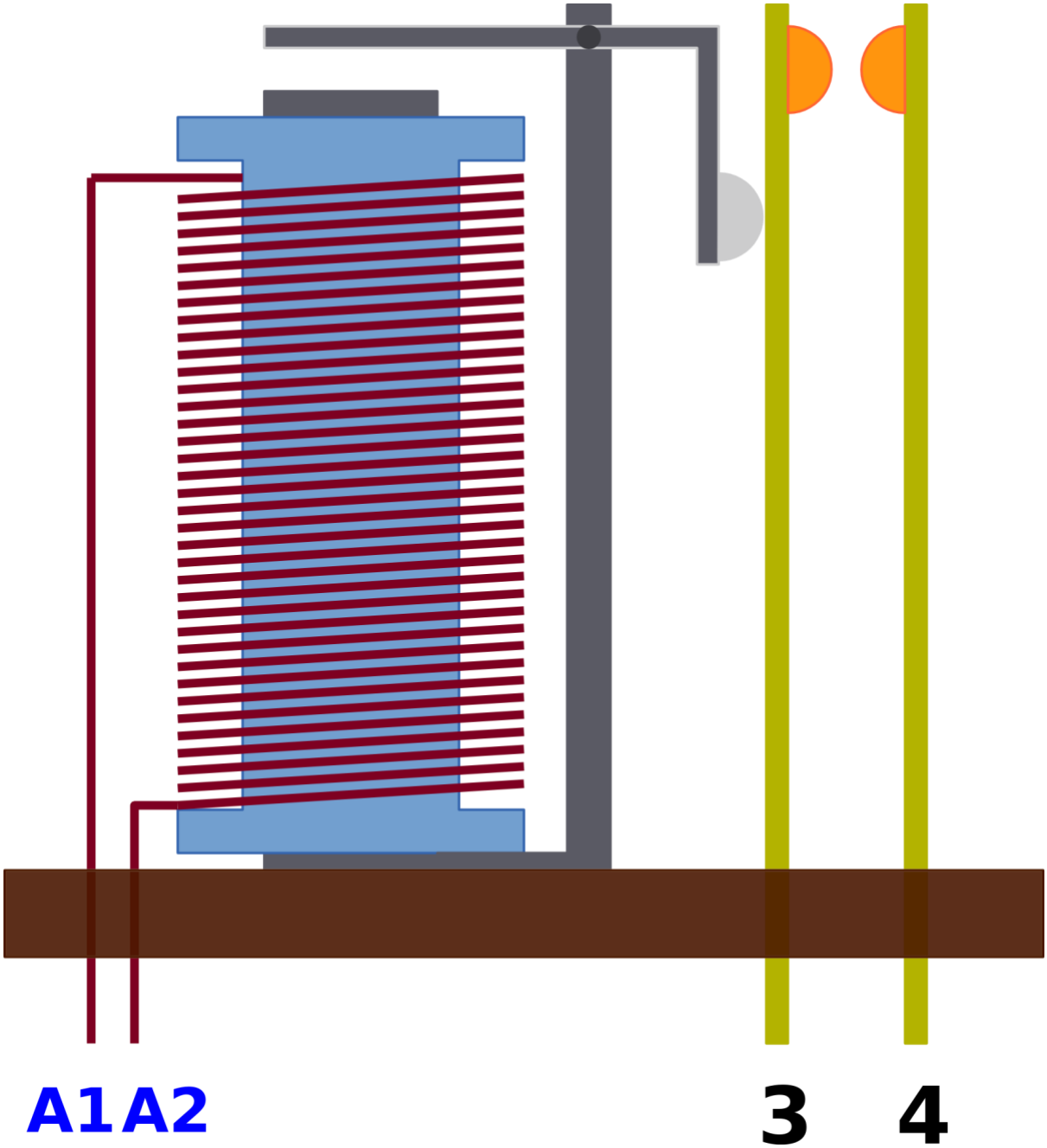


Figure 6-8: The electrical relay structure.

6.6.2.2 DC Motors

DC (Direct Current) Motors are electromechanical actuators, which use the interaction of magnetic fields and conductors to convert the electrical energy into rotary mechanical energy.

6.6.2.3 DC Servo Motor

A servo motor consists of a DC motor, reduction gearbox, positional feedback device and an error correction mechanism. The rotation speed or position is controlled by a positional input signal or reference signal applied to the device.

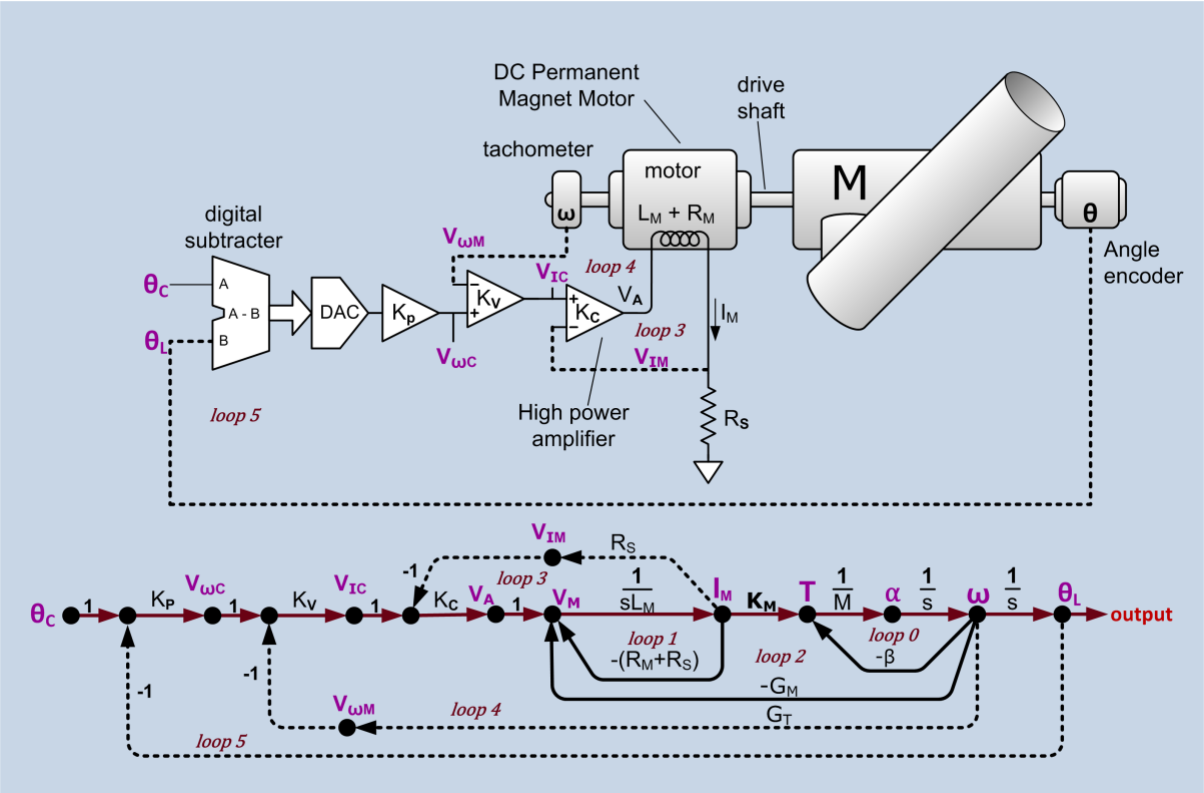


Figure 6-9: The architecture of a servomotor (image from Wikimedia).

6.6.2.4 Loudspeaker

Loudspeakers belong to the family of audio actuators. They are indeed sound transducers and basically represent the exact opposite of microphones. The function of loudspeakers is to convert complex electrical analogue signals into sound waves as close as possible to the original input audio signal.

Loudspeakers are available in different shapes, sizes and frequency ranges. Frequency ranges are defined depending on the intended application (e.g. voice, music, other). The most common types of loudspeakers include moving coil, electrostatic, isodynamic and piezo-electric ones. Moving coil type loudspeakers are by far the most commonly used speaker in electronic circuits, kits and toys.

7 IoT Connectivity Technologies

Author(s): Cristina López Bravo
Felipe Gil Castiñeira
René Lastra Cid

7.1 Introduction

An essential part of IoT is the connectivity that allows devices to “talk” among them and with backend services. One of the key characteristics of IoT is the “interconnectivity”. IoT devices should be able to be connected with other devices and the infrastructure [6]. Connectivity and networking are also one of the three layers for the IoT architecture [6] [9] [10] [11].

Nevertheless, there is not just one “Internet of Things”. There is a diverse range of use cases and applications that specify very different requirements, so it is not possible to find a “one-size-fits-all” solution for communications.

Different technologies jcoexist and selecting the right communication technology for a new IoT deployment requires defining strategic, financial and other objectives. The communication technology requirements can be divided in three categories:

- Technical requirements, such as coverage, energy efficiency, data rate, mobility support, etc.
- Commercial requirements, such as Quality of Service, cost, security or scalability.
- Ecosystem requirements, such as the readiness of the technology or its global reach.

As stated before, no one single solution is ideal for all the use cases. Furthermore, there are different competitors trying to establish their market dominance and ecosystem. Probably, in the future, some technologies will emerge among the others and become leaders.

7.1.1 Technologies for connectivity

Wireless communication technologies satisfy the requirements for most IoT use cases. Although wired technologies may be useful for certain situations, wireless solutions provide clear advantages:

- Simpler installation
- Facilitate the reconfiguration of the architecture (replacing nodes, changing their location, adding new devices, etc.)
- Support for mobility

Thus, this course is focused on the different wireless technologies.

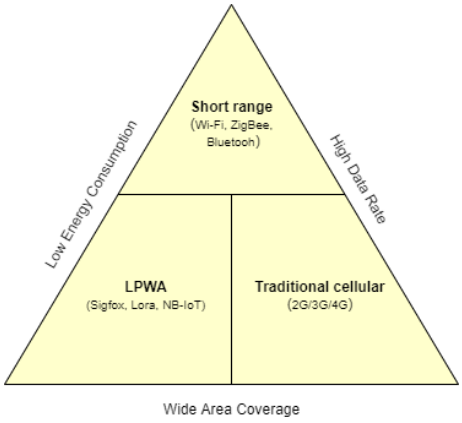


Figure 7-1: Wireless communications design constraints (source: Telenor)

There are different design constraints we have to consider when selecting a wireless technology (Figure 7-1):

- Power consumption
- Coverage range
- Bandwidth
- QoS requirements (usually related to the usage of licensed spectrum)

Usually there is a trade-off among the different constraints. For example, a large coverage range and/or bandwidth usually requires more energy. The usage of licensed spectrum makes it easier to satisfy QoS requirements, but usually at a higher cost.

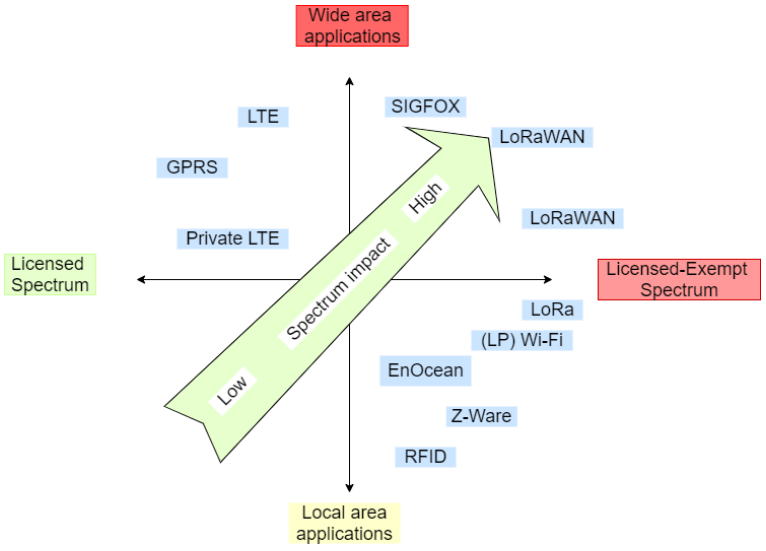


Figure 7-2: Spectrum impact and coverage for connectivity technologies for IoT (source ITU)

One of the parameters that causes a higher impact is the communication range (Figure 7-2). Thus, a typical classification for wireless IoT technologies is in "short range wireless communications" (or Wireless Personal and Wireless Local Area Networks –WPAN and WLAN), and in "wide are wireless communications" (Wireless Wide Area Networks or WWAN).

Table 7-1: Communications stack

APPLICATION LAYER		AMQP, CoAP, DDS, MQTT, XMPP, REST, etc.
NETWORK	ENCAPSULATION	6LowPAN, Thread
	ROUTING	CARP, RPL, DSR, AODV, etc.
DATALINK		Bluetooth / BLE, Wi-Fi / Wi-Fi HaLow, LoRaWAN, SigFox, Z-Wave, ZigBee, USB, LTE/5G

In this chapter we describe different communication technologies which may be used to implement applications directly (for example, Bluetooth and Zigbee also standardize the application layer) or to be used as "layer 2" or "datalink" technologies, and to build a complete communications stack with other higher layer protocols (as shown in Table 7-1: Communications stack).

7.2 Short range communications

IoT devices are usually small, battery powered and low-cost devices. Thus, the communication technologies should also be adapted to those restrictions, and initially only low range communications satisfied them. In fact, originally Wi-Fi modules were too expensive and power hungry to be used as the communication technology for IoT devices. It required several years for Wi-Fi to evolve to be embedded in phones (smartphones) and a few more to be part of cheaper devices whose battery should last.

In order to guarantee the interoperability among devices, it is necessary to use standards. This way, devices created by different vendors can exchange information. Nevertheless, many standards emerged just to satisfy a particular need. For example, Bluetooth was developed originally by Ericsson to provide means to connect mobile phones and their peripherals (such as headsets or hands-free devices in cars) or other devices such as

computers. That is, Bluetooth was designed just as a “cable replacement” for phones. The protocol was designed and implemented long before IEEE 802.15.1 was created.

7.2.1 Wireless Local Area Networks (Wi-Fi)

Along with cellular technology, Wi-Fi is the best-known connectivity protocol and is present in almost every home in the world.

It was originally designed to provide connectivity to computers "on the road" in airports, hotels, Internet cafes, and shopping malls, or to avoid connecting cables at home or the office (a convenient feature for laptops). In those scenarios, the goal was to provide web browsing, email and, for business users, access to the office network through Virtual Private Network (VPN) applications.

Now, Wi-Fi is a technology that is available in many devices: computers, printers, games consoles, media servers, scanners, etc. Wi-Fi modules can be embedded in almost any type of device with a processor, ranging from devices as small as a smartphone or as large as a screen in an auditorium.

Wi-Fi can be used to easily link together IoT devices, as well as connecting them to wireless access points that in turn connect to cloud-base systems.

7.2.1.1 802.11 standard

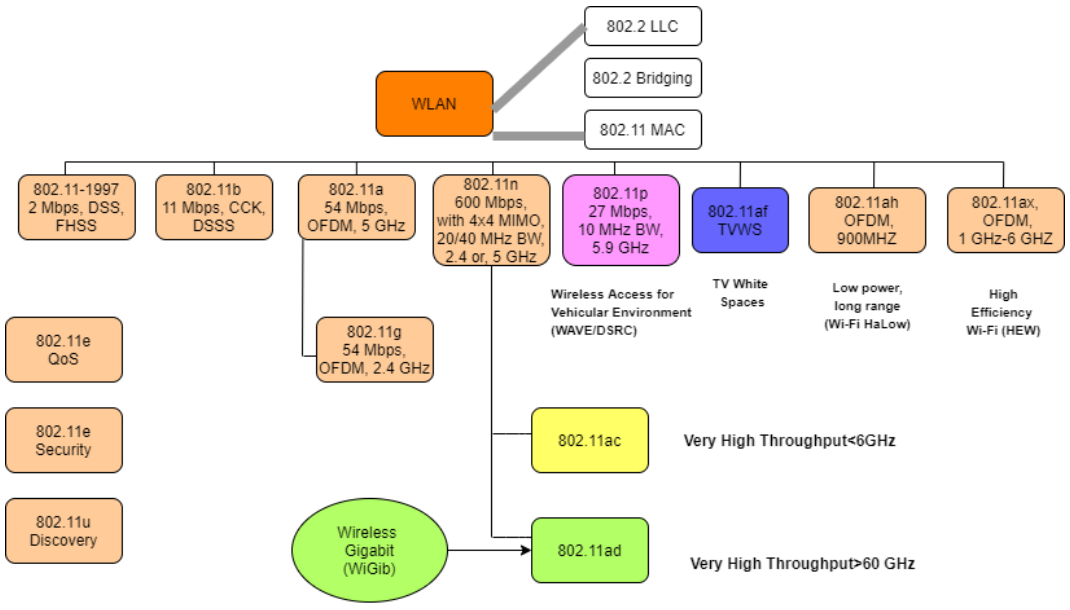


Figure 7-3: 802.11 Family of standards

Wi-Fi is a term that originally meant “Wireless Fidelity”, somewhat remembering the term “Hi-Fi” used to describe “High Fidelity” audio devices. Nowadays, it is almost a synonym of “wireless internet”. Wi-Fi is not really a protocol, but a trademark of the Wi-Fi Alliance, a non-profit organization that certifies wireless products as interoperable and promotes the technology. The standard is defined by the IEEE through the IEEE 802.11 specification family [39]. The first version was published in 1997, but it was modified and extended through the years, as shown in Figure 7-3, being the main goal to improve standard performance. Most of the amendments are related to changes in the physical layer. Changes have affected both modulation and transmission frequency, bringing faster speeds, higher density, additional frequency bands, and faster throughput to the users; the changes also bring the adaptation to specific environments such as vehicle communication, or the transmission over the free TV band.

In addition to these amendments related to the physical layer. The IEEE has published other amendments related to aspects such as security and authentication (802.11i), quality of service (802.11e), direct communication and discovery, indoor locations energy saving, etc.

In order to maintain compatibility with legacy devices, new standards include extra headers and reserve the channel with techniques that make legacy devices understand that the channel is busy.

Power consumption is still an issue for IoT devices, but this technology is used in many non-battery powered devices, or to implement gateways to connect IoT devices using other technologies to the Internet.

7.2.1.2 Architecture

Although the prevalent use of Wi-Fi is for providing wireless connectivity to the Internet, this technology can be used to provide peer-to-peer connectivity between devices.

This way, Wi-Fi networks can also be used independently of wired networks, by creating a LAN that can be used as stand-alone network anywhere to link multiple devices (known as stations or STA) together without having to build or extend wired networks.

In this peer-to-peer topology, client devices within a cell communicate directly to each other when they are in the same transmission range. To reach further, forwarding of data is required (that is, it would be necessary to create a multi-hop network).

Nodes are more complex since they need to incorporate management, forwarding and routing functions.

Multiple networks (different IBSS or "Independent Basic Service Set") are possible by spatial separation or by using different carrier frequencies.

Ad hoc communications are helpful in public safety and search and rescue scenarios when emergency teams need to communicate quickly and efficiently in these situations.

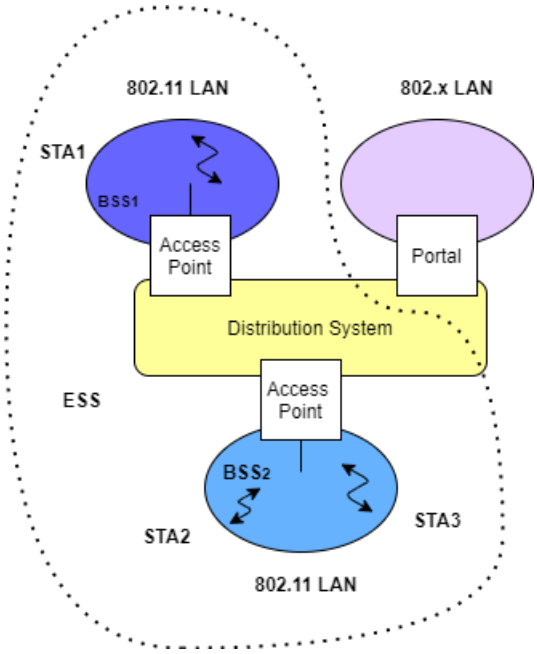


Figure 7-4: Wi-Fi infrastructure network architecture [30]

Nevertheless, the most conventional architecture we will usually find for Wi-Fi networks is the one shown in Figure 7-. In this case, Access Points (APs) are used to bridge traffic between the wireless connection and a wired backbone. A wireless client computer can connect with any other wired or wireless device on the network thanks to access points.

A Basic Service Set is made up of linked stations (STA) and an access point (AP) (BSS). Two separate BSSs are seen in Figure 7-4 and are bound by a distribution grid. A delivery system links several BSSs to form an Extended Service Set (ESS), which has its own identifier or ESSID. This is often referred to as the network's tag.

Each AP manages communications within its range. Among other, the principal functions completed by the AP are:

- Medium Access Control functions
- Mobility Management functions
- Authentication functions

Stations may interact with a specific AP. When APs share the same ESS, stations can also roam between them.

Wi-Fi networks can also be deployed using a mesh architecture, where the distribution system is also implemented with a Wi-Fi link. By following this approach, it is possible to reduce the dependence on wired networks, simplifying and reducing the time and cost of the installation and making it easily expandable. On the other side, this configuration may reduce the performance (the wireless channel is used also as a backbone network) ad be more complex to maintain (if a mesh node does not mesh to the rest of the network for some reason, you cannot access it remotely from the support centre).

In some cases, it might be beneficial to implement a mesh network indoors, but the primary application for mesh networks is in large outdoors areas.

7.2.1.3 Layers and functions

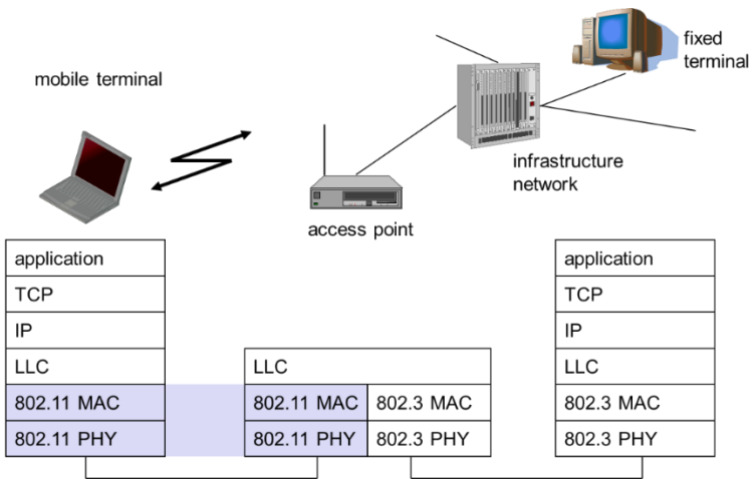


Figure 7-5: 802.11 protocol architecture [30]

The 802.11 protocol is a member of the IEEE 802.x family of protocols for local area networks (LANs). This means that the standard specifies how the physical and de medium access control layer should behave – adapted in this case to the wireless environment- while keeping the

same interface at the higher layer as the other standards in the family, to ensure interoperability. As shown in Figure 7-5, APs are usually connected to a switched Ethernet network (802.3) which is completely transparent for applications (an application running in a Wi-Fi station is not aware of the wireless nature of the connection).

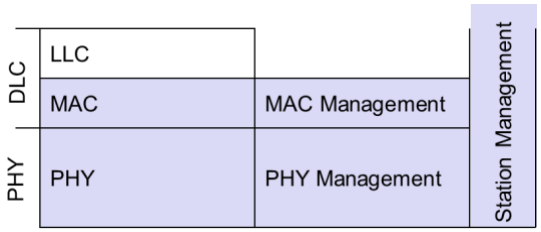


Figure 7-6: Detail of 802.11 protocol architecture [30]

The 802.11 standard only covers the physical layer (PHY) and the Medium Access Control layer (MAC). The standard specifies different functionalities:

- MAC: Provides access mechanisms, fragmentation and encryption.
- MAC Management: Supports the association of a station to the access point, synchronization, roaming, maintenance of a MAC management Information Base (MIB) and power management.
- PHY: Provides a clear channel assessment (CCA or carrier sense signal), handles modulation and encoding/decoding of signals.
- PHY Management: Its main tasks are channel selection, maintenance of the MIB,
- Station Management: Interacts with the other management layers and coordinates their management functions.

7.2.1.4 Wi-Fi HaLow

Wi-Fi HaLow for IoT	
Features	**Benefits**
Sub-1 GHz spectrum operation	Long range: approximately 1 Km
Narrow band OFDM channels	Penetration through walls and other obstacles

Native IP support	Supports coin cell battery devices for months or years
Latest Wi-Fi security	No need for proprietary hubs or gateways

Figure 7-7: Wi-Fi HaLow main characteristics

Among the new standards, IEEE 802.11ah is especially interesting for the IoT world. It is marketed under the name Wi-Fi HaLow and is designed to have a better coverage and a lower consumption than other protocols of the family.

It offers range, data rates, penetration, and low power consumption profiles expected in IoT use cases, such as industrial automation, logistics and transportation, agriculture, home and building automation or smart cities.

One of the main differences with the other physical layers is its operation in frequency bands below 1GHz. The usage of 900 MHz bands helps to obtain extended coverage. It implements different techniques to reduce power consumption. Other functionalities in the MAC also help to reduce congestion and increase both capacity and device density.

The MAC extends the amount of time a client computer is required to sleep until the AP initiates the disassociation in order to extend battery life. This retained status saves resources by preventing power-sensitive sensors from having to re-authenticate once they have gone offline. The functionality allows for a limit of five years of inactivity. The idle time would, in fact, be determined by the implementation and application specifications.

Other features to reduce consumption are Target Wake Time (TWT) and Restricted Access Window (RAW).

Client devices that plan to sleep for lengthy stretches of time will use TWT to sign a TWT deal with the AP, which will store all traffic intended for the client before the agreed-upon activation time. The client computer listens for its beacon and contacts the AP to collect and send the requisite data after the agreed-upon time has passed.

The number of wakeup times agreed by the customer and the AP can be extremely brief (microseconds) to extremely long (years). RAW is commonly found in systems with predictable operation cycles. An AP can assign RAW privileges to a subset of clients, allowing them to move data while others are forced to sleep, buffering non-urgent data.

Client devices have the potential to save energy by freeing up network bandwidth for other time-critical traffic.

Network designers can minimise channel contention to save system-wide power by combining TWT and RAW functions.

The coverage area of the BSS can be partitioned into sectors, each one with a subset of stations. Each section is covered by a set of antennas to reduce medium contention and interference.

7.2.2 Wireless Personal Area Networks (Bluetooth)

Bluetooth is a wireless networking standard that allows phones, computers, and other network equipment to communicate over short distances without the use of wires.

It was created as a universal radio interface for ad-hoc wireless communication between computers and peripherals, as well as mobile devices, PDAs, and mobile phones. It was originally designed by Ericsson to replace cables in phones for voice and data transmission (audio cable, cable with the computer to transfer data, etc.). Thus, it was a technology that had to be embedded in other devices and the cost should be very low (< 1 $).

Also, this use case (cable replacement) didn't impose a large range for the communications. It was considered that 10m (the most typical range, although the standard defined also classes of devices with a range of 100m) was enough. To make the integration in phones and other battery powered devices required a low power consumption.

As a result, Bluetooth is a Wireless Personal Area Network (WPAN) system that has very restricted coverage without requiring infrastructure. It's a short-range technology that's built into a microchip and is always attached (always on).

Bluetooth has a wide range of potential applications, but the most prominent application is providing a convenient and uncontrolled means of interconnecting electronic equipment.

Throughout the 1990s, most of the wireless interconnection between PDAs, mobile phones and laptops was done with online infrared technology. By using wireless RF communications, Bluetooth does not require direct line of sight (SLO) and can support multipoint communications in addition to the point-to-point communication that was available with

infrared. The short-range receiver-emitters that are built into mobile devices to provide Bluetooth compatibility are designed to operate in the 2.4 GHz unlicensed radio band. Bluetooth was designed to operate in a range of 10 to 100 meters. By hopping to a different frequency after sending or receiving a packet, a Bluetooth transceiver can mitigate the impact of interference from other signals. Bluetooth also supports forward error correction (FEC) and automatic repeat request (ARQ), which involves computing a CRC for data packets and retransmitting received packets with errors.

The following is the goals of Bluetooth technology:

- The device should be uniform and operate on a global scale.
- The machine would be able to create contact between two Bluetooth-enabled devices, regardless of their nature: a PC, a cell phone, car gadgets, and so on.
- Since the radio transmitter must be built into battery-powered equipment, it must use little electricity.
- The transmitter microchip should be inexpensive.
- It would be a device that is built on a reliable and stable protocol.

It should be possible to use it for both data and audio transmission. Transmission rates should be around 1 Mbps.

Bluetooth was created to eliminate the need for cables, especially serial cables that linked various devices. File sharing, listening to music, copying notes, navigation, and making cell phone calls via Bluetooth technology have also been added to the list of use cases over time.

With the addition of **Bluetooth 4**, the variety of applications is even larger, including applications such as **IoT**, health-related devices such as thermometers, blood pressure monitors, glucose meters; home automation, home training (remote controls, wireless keyboards); Smart energy; payment with mobile devices; security; automotive-related devices such as sensors to measure tire pressure, motion sensors, temperature or pollution sensors, etc.; sports-related equipment such as pedometers, GPS locators, heart rate monitors, etc.

The **Bluetooth devices** that we will find today can be divided into three categories: classic Bluetooth devices (BR/EDR), Bluetooth Low Energy (LE) devices or Bluetooth Smart, and Bluetooth Smart Ready devices.

Within the classic devices, Bluetooth technology allows wireless communication and information exchange between devices of different nature that meet the specifications of the standard. The following are examples of Bluetooth-enabled computers, grouped by category:

- Audio: Stereo headset, hands-free
- Car: integrated systems, hands-free, GPS modules
- Laptops with embedded Bluetooth, Bluetooth USB adapters, and link servers to other networks are examples of personal computers.
- Peripherals: Wireless keyboards and mice, printers
- Telephony: Mobile phones, smart phones, PDAs
- Video and image: Photo cameras, video cameras, projectors.

The ability to connect several devices and exchange voice and data opens up a plethora of realistic Bluetooth scenarios and applications.

- Exchange files and synchronized information between personal computers.
- Connecting peripherals.
- Hands-free capability for phone conversations via headsets, car kits, or integrated systems.
- Proximity marketing by sending advertising.

  Authentication is the process of verifying someone's identity. In Universal Second Factor (U2F) authentication, Bluetooth Smart can be used instead of USB dongles.

  Find misplaced devices, such as the remote or to send notifications.

  Detect the presence or absence of a device and react appropriately.

7.2.2.1 Protocol stack

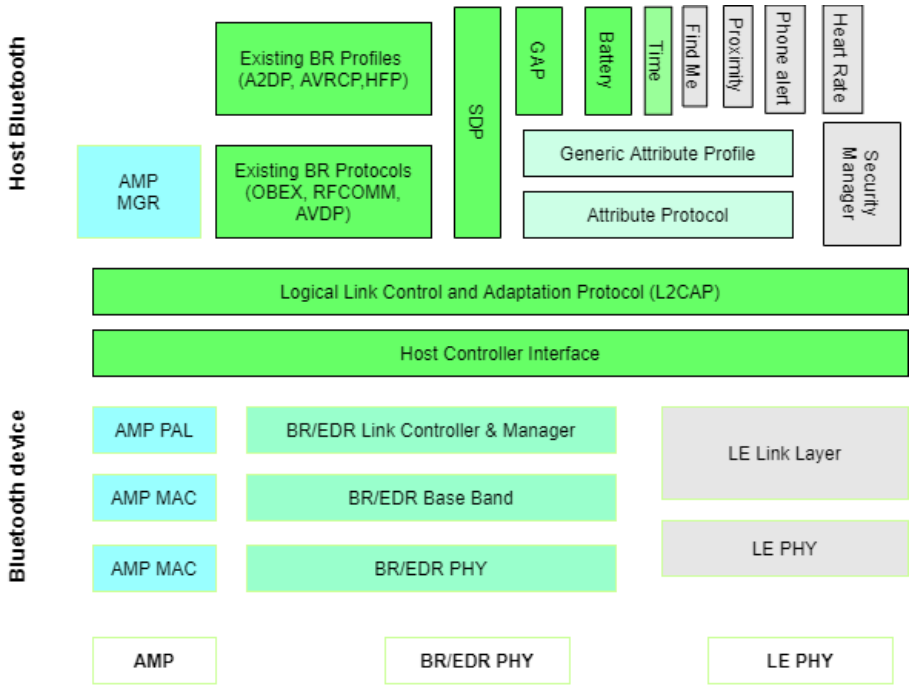


Figure 7-8: Bluetooth stack

The Bluetooth protocol stack is divided into two sections: a "controller stack" that contains the timing-critical radio interface, and a "host stack" that handles high-level data. In most cases, the controller stack is implemented in a low-cost silicon computer with a Bluetooth radio and a microprocessor (hardware). The host stack is usually deployed as part of an operating system or as a standalone package that can be installed on top of one (software). The Host Controller Interface connects the two subsystems (HCI). The Bluetooth controller can be swapped with limited adaptation using this interface [47].

Physical Layer (PHY). The PHY layer is responsible for transmitting and receiving packets of information on the physical channel. Bluetooth technology supports three radio versions: the physical layer that allows high-rate transmissions using IEEE 802.11 (AMP = Alternative MAC/PHY, available from the Bluetooth High Speed specification), the basic rate & enhanced data rate (BR/EDR) radios and the Low Energy (LE) radio.

Layer of the baseband. The baseband layer in Bluetooth includes the simple ARQ protocol which serves two purposes. One of the features is a scheduler that allocates physical channel time to all organisations that have agreed to an access deal. Negotiating access contracts with

these entities is the other feature. An access contract is a promise to provide a certain quality of service (QoS) in order to offer expected performance to a customer application.

Link controller and Manager Layer. The Link Controller is in charge of encoding and decoding Bluetooth packets based on the data payload and physical channel, logical transport, and logical link parameters. The Link Controller is responsible for signalling the flow control, acceptance, and retransmission request signals using the link control protocol. The connection manager is responsible for developing, altering, and releasing logical links as well as updating the specifications for physical links between devices. The Link Manager does this by using the Link Management Protocol to communicate with the Link Managers in the remote Bluetooth systems (LMP).

Where necessary, the LM protocol allows for the construction of new logical links and logical transports between devices, as well as the general regulation of connection and transport attributes such as allowing encryption on the logical transport, adjusting transmit power on the physical link, and adjusting QoS settings on a logical link. Other functions of this layer are authentication of the communication with partners (pairing) and cyphering, QoS parameters negotiation, latency and speed control, and power control.

Logical Link Control and Adaptation Protocol (L2CAP) Layer. The L2CAP is in charge of handling the order in which PDU fragments are sent to the baseband and some relative scheduling between channels to assure that L2CAP channels with QoS compromises are not denied access to the physical channel because of exhaustion of Bluetooth controller resources. Optionally, the L2CAP layer can provide a further error detection and relaying to the L2CAP PDUs.

Host Controller Interface (HCI). HCI is characterized as a collection of commands and events that the host and controller use to communicate with one another, as well as a data packet format and flow control rules.

These five levels form the Bluetooth core specification. Above them there are other protocols, that allows to complete the functionality of the Bluetooth devices. Among them we remark the following:

Service Discovery Protocol (SDP). Using Service Discovery Protocol (SDP), the services provided by remote devices can be discovered, and their characteristics can also be known. These services are registered with an SDP server. This server maintains a list of services in the form of service registers where the SDP client can query these services.

Security Manager Protocol (SMP). The peer-to-peer protocol SMP is used to generate encryption and identity keys. This protocol is implemented using a set L2CAP channel. The SMP block is also in charge of generating random addresses and resolving random addresses with known user identifiers, as well as storing encryption keys and identity keys.

During encryption or pairing procedures, the SMP block collaborates with the controller to supply the stored keys that are used for encryption and authentication. In LE structures, this block is used. In the BR/EDR scheme, the Link Manager block in the Controller provides similar features. On LE platforms, SMP functionality is in the Host to reduce the expense of implementing LE only Controllers.

Attribute Protocol (ATT). The ATT specifies the client/server protocol for data exchange once the link has been created. Attributes are grouped into meaningful services using the Generic Attribute Profile (GATT). ATT is commonly used in LE and BR/EDR implementations.

Generic Attribute Protocol (GATT). GATT provides services that encapsulate part of a device's behaviour and explain a use case, features, and general behaviours based on GATT functionality. Its service architecture defines the procedures and formats for services as well as their functionality, such as discovery, read and write, notification and function indication, and feature broadcast configuration. Only Bluetooth LE implementations use GATT.

Generic Access Protocol (GAP). In order to define the procedures and functions regarding Bluetooth device discovery and information exchange, as well as the link management aspects of connecting to Bluetooth devices, the GAP works together with GATT on Bluetooth LE implementations. Related to device discovery and connection to devices.

Each vendor may add their own proprietary application protocol layer on top of the Bluetooth specific protocol layer. As a result, the Bluetooth open standard significantly extends the range of devices that can take advantage of this wireless technology's capability. Despite the presence of separate proprietary application protocol stacks, the Bluetooth specification

demands that interoperability between devices using different stacks be maintained. Interoperability is ensured thanks to Bluetooth Profiles.

Profiles describe general behaviours that Bluetooth activated devices use to connect with other Bluetooth devices and are used to define possible applications. Profiles are used to determine what kind of data a Bluetooth module transmits and are based on the Bluetooth format. Hands-free functionality to heart rate monitors, alarms, and other profiles are defined by the user application [49].

7.2.2.2 Bluetooth Low Energy

Bluetooth version 4.0 implemented Bluetooth Low Energy (BLE). It was designed for applications where power consumption was crucial (such as battery power devices) and where small amounts of data (states) were transfer infrequently (such as in sensor applications). The goal from the start was to create a radio standard with the least amount of power usage possible, one that was optimized for low cost, low bandwidth, low power, and low complexity. A series of decisions were made in order to do this:

- Using shorter packets.
    - TX peak current is reduced by using short packets.
    - RX time is cut in half for short packets.
- Designed for the transmission of small pieces of data (1 Mbps, but not optimized for data transmission)
- Using less RF channels to improve discovery and connection time
- Simplifying link layer state machine

Technical details

- Data Transfers. Bluetooth Smart (low energy) accepts very fast data packets (between 8 and 27 bytes) at 1 Mbps. To reach very short service cycles, connections use advanced sniff-sub grouping. The estimated maximum throughput that BLE can have is 1 Mbps. Owing to a variety of reasons, including bidirectional flow, protocol overhead, CPU and radio limits, and artificial device restrictions, this restriction has been reduced:

- The SoC may limit the number of packet exchanges per connection event
- The smartphone or tablet (device to which we are sending the data) may also be busy talking to other devices

Ideally, a maximum data throughput of around 5-10 kbps should be assumed.

- Frequency Hopping is a technique for hopping from one frequency to the next. To keep interference from other technologies in the 2.4 GHz ISM band to a minimum, Bluetooth Smart (low energy) uses adaptive frequency hopping, which is similar to all models of Bluetooth technology. Multipath's advantages include increased budgets and connection range.
- Host Control. The controller has a lot of intelligence thanks to Bluetooth Smart (low energy), which helps the host to sleep for longer periods of time and be woken up by the controller when the host wants to execute an operation. Since the host uses more power than the transmitter, this makes for the biggest power savings.
- Latency is a term used to describe a period of Bluetooth Smart (low energy) will establish a link and transmit data in as little as 3 milliseconds. This allows an application to establish a link and then send authenticated data in milliseconds for a brief contact burst before disconnecting.
- Range. A Bluetooth Smart (low energy) range of more than 100 meters is possible thanks to the improved modulation rate.
- BLE is focus on very short-range communication (to save battery lifetime). Typical operating rates is probably closer to two to five meters.
- Robustness. Both packets in Bluetooth Smart (low energy) use AFH and a solid 24 bites CRC (larger than in Bluetooth BR/EDR) to ensure optimum interference resistance.
- Strong Security. To ensure good encryption and authentication of data packets, full AES-128 encryption is used in conjunction with CCM.
- Topology is a term that refers to the study of Each packet for each slave in Bluetooth Smart (low energy) uses a 32-bit connection address, allowing billions of devices to be attached. This technology is enhanced for one-to-one connections, while a star topology allows for one-to-many connections.
- Maximum output power: 10 mW.

- The transmitter output power is described by LE as being between 0.01 mW (-20 dBm) and 10 mW (+10 dBm). The computer will adjust the output power dynamically to reduce power usage and interaction with other devices.
- LE defines the transmitter output power in the range of 0.01 mW (-20 dBm) to 10 mW (+10 dBm). The device is able to change the output power dynamically to optimise power consumption and minimise interference to other equipment.
- Range, Max. Current and Sleep Current are implementation specific issues. Based on the output power defined, LE support a range of about 30m to 100m.

Table 7-2: Bluetooth LE main technical characteristics

Range	~ 150 meters open field
Output Power	~ 10 mW(10dBm)
Max Current	~ 15 mA
Latency	3 ms
Topology	Star
Connections	> 2 billion
Modulation	GFSK @ 2.4 GHz
Robustness	Adaptive Frequency Hopping,24 bit CRC
Security	128bit AES CCM
Sleep current	~ 1µA
Modes	Broadcast, Connection, Event Data Models, Reads, Writes
Packet size	8-27 bytes

Bluetooth classic and Bluetooth Low Energy are incompatible with each other. This is why some devices such as smartphones have chosen to incorporate both types of Bluetooth (Dual Mode Bluetooth Devices), while others such smart wristbands only incorporate BLE (Single Mode Bluetooth Device).

7.2.2.2.1 Bluetooth Low Energy Architecture

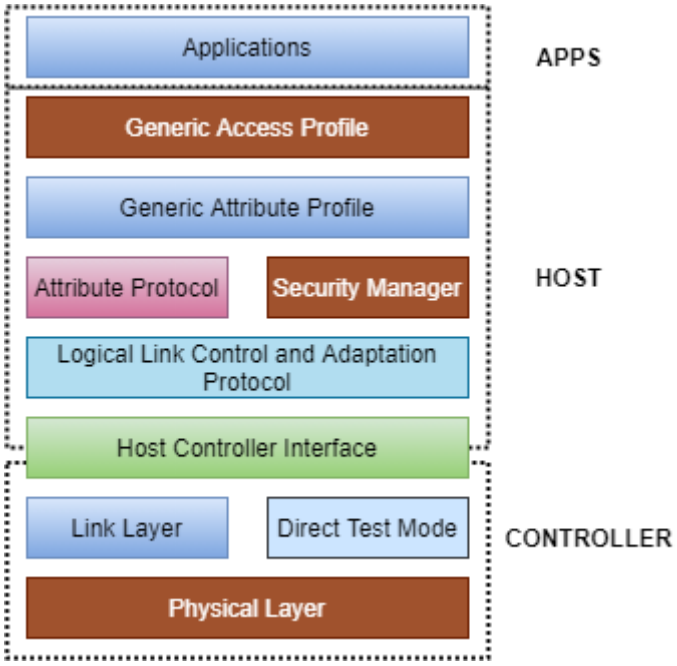


Figure 7-9: Bluetooth Low Energy Protocol Stack

In BLE, the physical layer operates in the ISM band (2.4 GHz spectrum). It's split into 40 radio frequency channels, each separated by 2 MHz (Figure 7-).

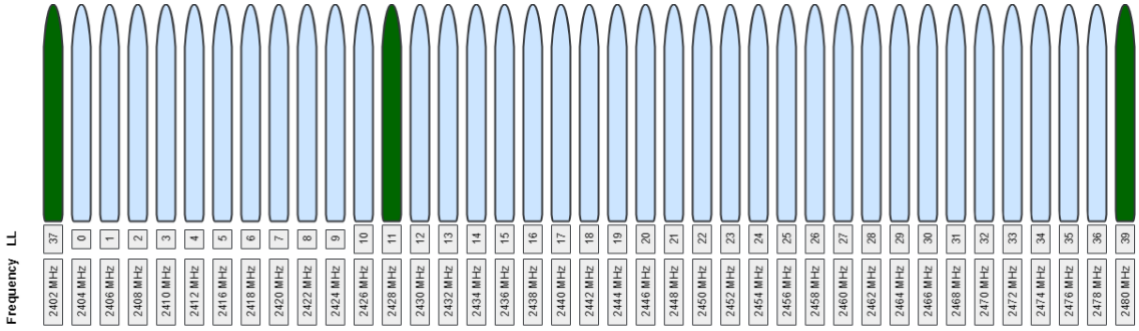


Figure 7-10: Bluetooth LE channels

Primary Advertising Channels refers to three of these channels. Secondary Advertisements and data transmission during a link are handled by the remaining 37 networks. The connection layer is in charge of maintaining the radio status and synchronization conditions that must be met in order to comply with the BLE specification. It's also in charge of hardware-accelerated operations like CRC, random number generation, and encryption.

The relation layer's operation is represented using a simple state machine with five states.

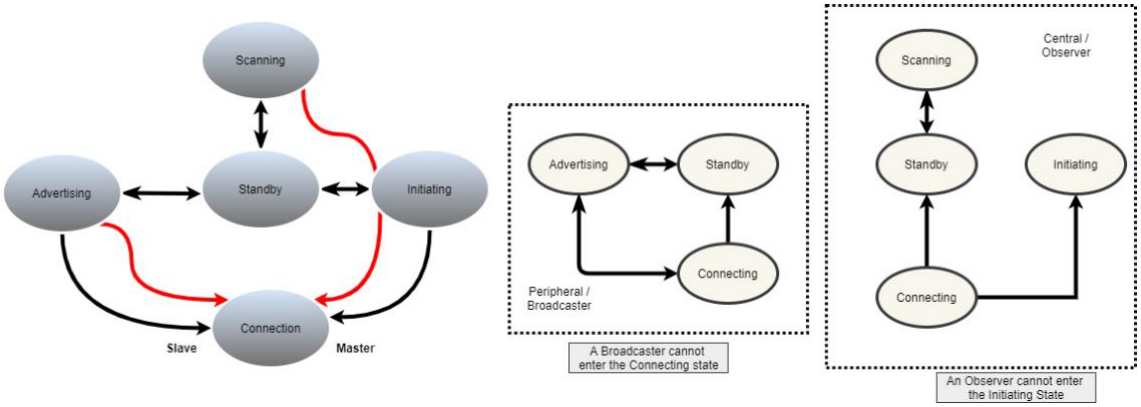


Figure 7-11: Bluetooth LE link layer states

- Standby: When no packets are received or exchanged, this is the normal link layer condition. This state can be reached by a computer from either of the other nations.
- Advertising state: Sends out advertising packets and will listen for devices that respond to those packets, then respond accordingly to those devices. When the connection initiates ads, it is possible to reach this state from the standby state. An Advertiser is a connection layer that is in this state.
- Scanning state: Listens for packets from the Advertiser and can request additional information from the Advertiser. When the connection layer wishes to start scanning, this state can be entered from the standby state. Scanner is the name of a relation layer in this state.
- Initiating state: The link layer is in the initiating state when it listens for packets from the Advertiser and responds by initiating a connection. When the Scanner initiates a communication with the advertiser, it will reach this state from the standby state. The Initiator is a connection layer that is in this state.
- Connection state: The computer is attached to another device, and two functions have been assigned to it: master and slave. This state can be accessed from either the initiation or the advertising states. The computer assumes the role of Master when entering from the initiation state. The computer behaves as a Slave when it enters from the advertisement state.

A connection layer in the Slave role will only connect with one system in the Master role, implying that the LE does not accept sparse network situations like the BR/EDR. The following functions are specified by the connection layer:

- Advertisement (a device sending advertising packets)
- Scanning (a device scanning for advertising packets)
- Educator (a device that initiates a connection and manages it later)
- Slave (a computer that acknowledges a communication request and keeps time with the master).

When there is no active connection, the roles are grouped into (announcer and scanner) while when there is a connection they are grouped into (master and slave).

Advertising and Scanning

There is just one packet format and two kinds of packets in Bluetooth Low Energy (advertisement and data packets). This makes the protocol stack's deployment easier.

Advertisement packets are used to broadcast data for programs that don't need the overhead of a complete connection establishment, as well as to discover and link to slaves.

Each advertisement packet will contain up to 31 bytes of advertising data as well as simple header information. The advertiser's interval determines the rate at which these packets are delivered (between 20 ms and 10.24s). They're delivered at a predetermined pace determined by the advertiser's interval (between 20 ms and 10.24s). The shorter the interval, the more often commercial packets are transmitted, increasing the likelihood that these packets will be handled by a scanner. An increase in the number of transmitted packets leads to an increase in energy demand.

There are two categories of scanning procedures defined in the specification: passive scanning and active scanning. The scanner simply listens for ads packets while passive scanning. The advertiser is completely unaware that a scanner has received one or more packets. After processing an advertisement packet, the scanner sends a scan request packet to the advertiser, who receives it and replies with a scan response packet. This extra packet effectively doubles the amount of data that the advertiser will send to the scanner. The scanner is unable to transmit any consumer details to the advertiser as a result of this.

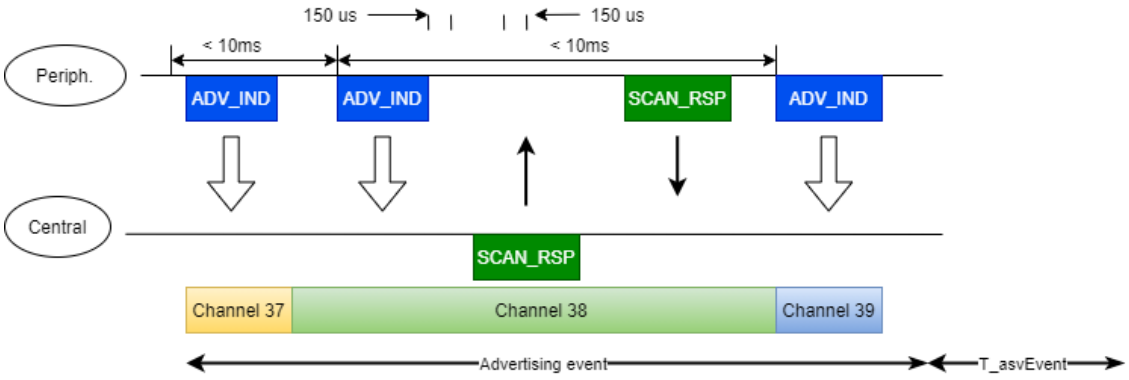


Figure 7-12: Bluetooth LE link layer advertising

Connecting

To make a connection, a master begins by looking for advertisers that are willing to entertain connection requests. The advertiser is either filtered depending on the BT address or the advertisement data itself. The master sends a communication request packet to the slave after detecting a fitting advertiser slave and creates a link. The frequency hop increment is included in the link request packet, and it determines the hop series that the master and slave will obey over the connection's lifetime. Another number of main variables shared by the master during the creation of a link is the connection parameters:

- The time before the start of two consecutive connection events is known as the connection event interval (7.5 ms and 4s).
- The number of link events that a slave may opt to miss without causing a disconnection is referred to as slave latency. If the slave latency is set to ten, for example, the slave must listen for every tenth communication occurrence. If it's set to 0, the slave will have to listen for each and every communication occurrence.

The maximal interval between two legitimate data packets received until the link is deemed lost is known as the connection supervision timeout. The communication is deemed broken and no further packets are transmitted if a packet is not sent during the monitoring timeout. The host is notified that the link has been lost. This timeout is used by both the master and the slave.

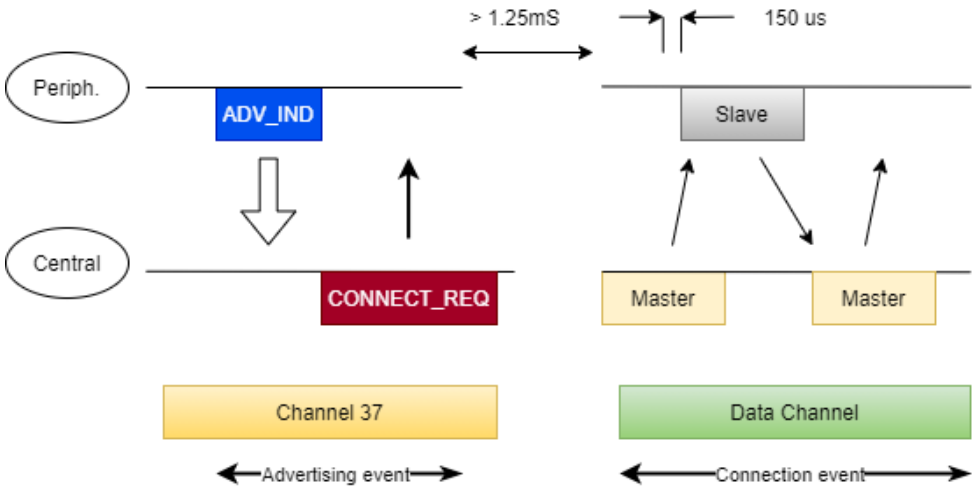


Figure 7-13: Bluetooth LE link layer connection

Topology

The possible topologies for the LE are shown in Figure 7-. An Advertiser sends advertisement packets over the physical advertising channel on the left side of the figure. Two scanners listen to these advertisement packets and may either ask the Advertiser for more details or submit a connection request over the advertising channel.

A piconet is depicted on the right side of the figure, with a Master attached to three slaves. The physical data channels are used to transmit data between the Master and the Slaves. The Master works like a printer, scanning the advertisement platform for packets from advertisers.

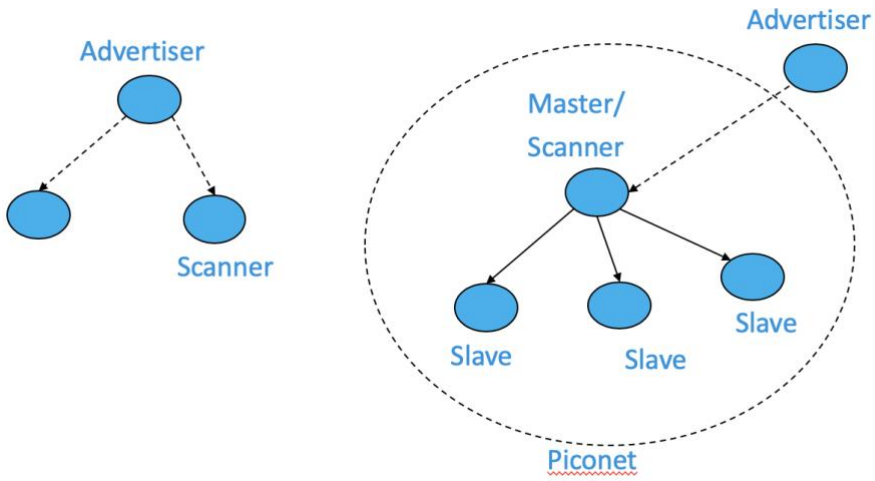


Figure 7-14: Bluetooth LE Topology

Bluetooth LE is based on the previously described Attribute Protocol (ATT) and Generic Attribute Profile (GATT).

ATT. The attribute protocol is a process for finding, interpreting, and writing the attributes of a remote computer. Is a stateless client/server protocol based on the following attributes provided by devices:

- The client discovers attributes, reads and writes attributes, requests data to server
- The server contains attributes

Each attribute contains the following information:

- Handle: identifier to access the attribute
- The form and quality of the data in the value field is determined by the UUID.
- Permissions: Permissions include readable, writable, notify, coding, authentication, permission, and more. Describes the operations that can be performed on this attribute.
- Value. Represents the value of the attribute.

GAT: Built on top of the Attribute Protocol (ATT), GAT adds a hierarchy and a data abstraction model. Establishes how to share both profile and account data over a BLE network in great detail. It's a client/server protocol as well:

- A client sends requests to a server, and the server responds.
    - *First: Service discovery*
    - *Reads and writes attributes found in the server*
    - *Receives server-initiated updates*
- The server is in charge of storing and rendering user data, which is structured in attributes, accessible to the client.

7.2.2.3 Bluetooth Mesh

Packet forwarding in a Bluetooth mesh network is based on the managed flooding communication model. In this way, a message injected into the mesh network could be forwarded to several relay nodes at once. This approach offers a high flexibility in the

deployment and operation of the network, but as a counterpart it generates a high congestion, which could lead to packet loss.

One of the positive aspects of Bluetooth is that it allows devices to establish multiple connections. It is possible to establish a point-to-point connection with a heart rate monitor through which data can be transferred. It is also possible to establish a point-to-point connection with an activity monitor. On the other hand, a mesh network has a many-to-many topology. Each device can communicate with all other devices in the mesh via messages, and devices can relay messages to other devices. In this way the end-to-end communication range can be extended beyond the radio range of each individual node.

When a node queries the status of other nodes it sends a message of an appropriate type. To inform other nodes of its status, it sends a message. Communication in the mesh network is "message-oriented". Many types of messages are defined, each with its own unique opcode. We can divide messages into two categories: acknowledged and unacknowledged.

Bluetooth Mesh networks were created because mesh topologies offer the best option to satisfy several increasingly common requirements:

- Very large coverage areas
- The ability to monitor and control a large number of devices
- Low power consumption, and optimized consumption
- Efficient use of radio resources, leading to greater scalability
- Compatibility with current terminals (smartphones, tablets, PCs)
- Industry standards and high safety levels set by governments

The mesh network operation defined by this specification is designed to:

- Enable messages to be sent from one element to one or more elements;
- Allow messages to be relayed via other nodes to extend the range of communication;
- Secure messages against known security attacks, including eavesdropping attacks, man-in-the-middle attacks, replay attacks, trash-can attacks, brute-force key attacks, and possible additional security attacks not documented here;
- Work on existing devices in the market today;
- Deliver messages in a timely manner;

- continue to work when one or more devices are moved or stop operating; and
- have built-in forward compatibility to support future versions of the Mesh Profile specification.

Bluetooth mesh features:

- The publish/subscribe model is as follows: The publish/subscribe model is used to describe data sharing within the mesh network. Nodes that generate messages send them to a specific address, and nodes that want to receive them subscribe to that address. This enables address assignment and party casting to be done in a variety of ways.
- Two-layer security: Messages may be authenticated and encrypted using two different forms of encryption keys. The network layer key ensures that all communication within a mesh network is safe. The application key is used to ensure that application data sent between intended devices is kept private and authenticated. The application key enables the use of intermediary devices to relay data, allowing messages to be authenticated for retransmission while preventing intermediate devices from reading or changing the application data.
- Flooding is one of the simplest and most straightforward methods of propagating messages in a network via broadcast. A message retransmitted by one device can be received by multiple retransmitters that, in turn, retransmit it. The Bluetooth mesh contains rules to restrict devices from retransmitting messages they have seen and to avoid messages being retransmitted over many hops.
- If devices need low-power support they can be associated with an always-on device where messages are stored and transmitted on their behalf. This concept is known as friendship. Friendship is defined as the special relationship between a low-power node and a neighbouring friendly node. The low-power node is the first to establish friendship. The friendly node takes actions to help reduce the energy consumption of the low energy node. Incoming messages are cached and directed to the low-power node. The friend node delivers security updates to the low-power node.

The Bluetooth Mesh specified advertisement bearer may not be supported natively by certain Bluetooth devices, such as smartphones. The Bluetooth Mesh profile defines a proxy protocol for mesh messages to be shared between devices using legacy Bluetooth connections.

7.2.3 Personal Area Networks (Zigbee)

ZigBee is a Wireless communication standard designed by the ZigBee Alliance. It was originally created by Ember Corporation (later acquired by Silicon Labs) to build low-power and low-data-rate WPANs that could incorporate more nodes than Bluetooth [44]. It was designed thinking on the simplicity of the implementation and in low power consumption, targeting applications requiring secure communications, low data transmission rates and that maximizes batteries lifetime. Now, it is controlled by the ZigBee Alliance, a consortium of software, hardware and service companies such as: Motorola, Philips, Mitsubishi, Invensys or Telefonica.

This technology targeted industrial and residential applications, especially for sensors and control devices and for scenarios where power consumption and/or implementation costs are critical. Zigbee is designed to optimize low bit-rate applications and low duty cycles.

The ZigBee specification includes a protocol stack divided in two principal layers. The lower layers are tied closely to the IEEE 802.15.4 standard, which offers a wide range of physical layer specifications, but only a subset is standardized under the ZigBee specification. Nevertheless, sensor and controller applications needed a meshed network and standard syntax for application level messages. Thus, ZigBee Alliance created the standards for the missing layers, which would be needed to enable the deployment of multi-vendor networks, relying on the radio layers of the IEEE 802.15.4 standard. The Network layer (NWK) manages routing tasks and the maintenance of network nodes, and the Application Support Sublayer (APS) establishes an interface between network layer and ZigBee Device Objects (defined by the standard or the manufacture.

ZigBee has been designed to have the following features and functions:

- Ease of implementation (Using DSS allows analog circuitry to be very simple).
- Low power consumption: device batteries are expected to have a life span of several months to several years.

- The power control is simple. There are only two possible states: active (transmitting or receiving) and asleep. This allows applications to focus on the application itself rather than on what is the optimum power mode for each aspect of its operation.
- Low cost (both devices and their installation and maintenance).
- High density of nodes in ZigBee networks Both the physical level and the MAC level allow a high number of nodes per network, which is critical in sensor networks. The network can grow spatially, without the need to use higher power transmitters.
- Simple protocol, global in scope (the 868 MHz band is used in Europe, 915 in N America, Australia, etc., and the 2.4 GHz band is accepted in most countries as a global band).
- Small devices.
- Short range (for each node).
    - 50 m typical (between 5 and 500 m depending on the environment).
- Flexibility in network configuration (supports multiple topologies without adding complexity).

ZigBee is especially suitable for situations where energy consumption and/or implementation costs are critical. Therefore, it is suitable for home automation applications where it is not desired or possible to create a wired communication network. It is also suitable for those cases where the controlled elements or sensors are mobile. It is intended to be used in general purpose applications with self-organizing and low-cost features (mesh networks, in particular). It can be used for industrial control, embedded sensors, medical data collection, smoke or intruder detection or home automation. The network as a whole will use a very small amount of energy so that each individual device can have an autonomy of up to 5 years before needing a replacement in its power system.

IEEE 802.15.4 has been designed to be useful for a number of applications, including industrial control and monitoring; public safety, including listening, detection and location of disasters; smart bands and tags; in-car control, e.g., tire pressure monitoring. Precision agriculture, control of the level of pesticides, herbicides, soil moisture (sprinkler irrigation systems, using just the right amount of water to keep plants green), pH levels; fleet control, reporting on the state of vehicle maintenance, tire conditions, mileage. However, one of the main

opportunities of the technology can be found in domotic applications, building automation and network interconnection.

Inside the home, different sectors could be considered: connection of peripherals (wireless mice, keyboards, joysticks, PDAS, games); consumer electronics such as remote controls, televisions, CDs, VCRs, and the possibility of controlling all equipment with a single universal remote control; home automation, ventilation and air conditioning, heating, security, lighting and the control of objects such as curtains, doors, windows and locks; health control, medical monitoring of elderly people living alone and warning medical staff of changes that could mean health problems; toys and interactive games. The transmission rate required for this type of application is expected to be a maximum of 115.2 kbps for some peripherals, and less than 10 kbps for home automation and consumer electronics applications. Similarly, the maximum acceptable latency is expected to be 15 ms for peripherals up to 100 ms or even higher for home automation applications.

7.2.3.1 Protocol stack: PHY, NET, Application layers

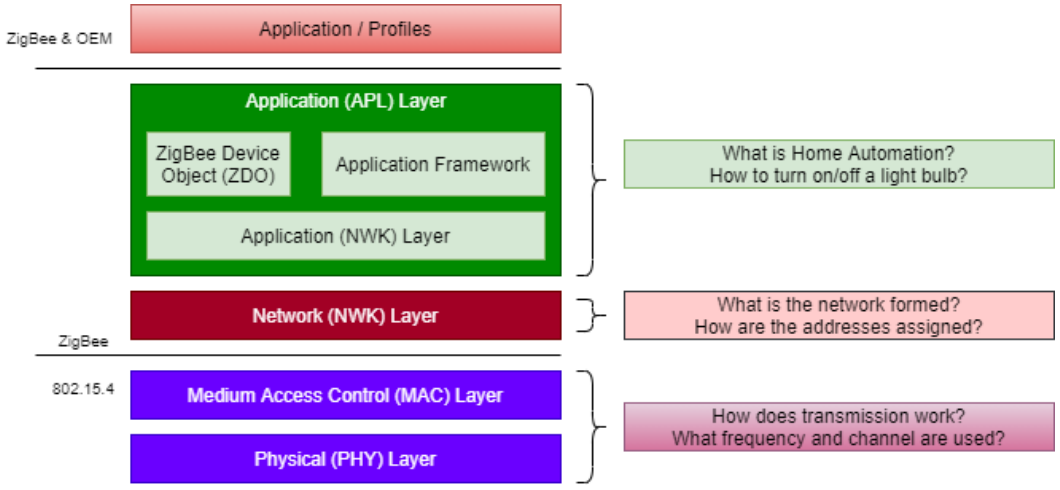


Figure 7-15: ZigBee stack (source: Xbee)

As stated before, the ZigBee protocol stack (Figure 7-) is built upon the physical and MAC layers defined by IEEE 802.15.4.

The physical layer distinguishes three frequency bands: the 2.4 GHz ISM band, the 915MHz ISM band, and the 868MHz Short Range Device band (SRD).

The various features of the implementation for various frequencies may be used to accomplish various objectives. The lower rate used at 868/915 MHz PHY frequencies, for example, can be converted into greater sensitivity and a wider coverage area, reducing the number of nodes needed to reach a given physical area, whereas the higher limit of the 2.4 GHz PHY can be used to achieve higher throughput, lower latency, or lower duty cycle.

The locations where ZigBee is deployed may contain multiple types of wireless networks competing for the same frequency bands, so it is necessary to have mechanisms to relocate within the spectrum. ZigBee offers this possibility through the cross-layer cooperation between PHY, MAC and NWK. PHY layers implement receiver energy detection, link quality indication, and channel switching, which enable channel assessment and frequency agility. The MAC layer includes a scan function that steps through the list of supported channels in search of a beacon, and finally the NWK layers establishes the initial operation channel and changes it in response to a breakdown or prolonged failure.

Regarding coverage, it will depend on the transmitted power. Each device shall be capable of transmitting at least 1 mW, covering a 10–20 m range. With good sensitivity and a moderate increase in transmit power, a star network topology can provide complete coverage for the desired area. Nevertheless, for applications allowing more latency, mesh network topologies provide an alternative to cover larger scenarios.

The range depends also on the frequency used. For example, the typical values for the 2,4 GHz band are 10 m indoor and 200 m outdoor, and for 868/915 MHz bands 30 m indoor and up to 1000 m outdoor. Also, as the devices operate in the ISM band, they must be able to accept interferences caused by other devices.

In ZigBee we can create different topologies by using the “star” and “Peer-to-Per” configurations. With the “Peer-to-Peer” configuration, we can build mesh networks.

Not all the nodes share the same functionality. They can be classified according to their capabilities:

- Reduced Function Device (RFD): Limited devices suitable only for simple applications (light switches and infrared sensors) that do not need to send or received large amounts of data. They have a reduced stack size with limited functions so they can

only be “leaves” in a star topology. They cannot work as network coordinator, and talk only to Full Function Devices (FFD).

- Full Function Devices (FFD): Are capable of being network coordinator, link coordinator or a simple communication node. They can talk to any other device, route messages and complete discovery procedures. They implement a full stack, so need more power

- ZigBee Network Coordinator (ZC): Are the nodes that set up a network (selecting the channel and PAN ID), transmit network beacons, manage network nodes, store network node information. They distribute addresses, allowing routers and end devices to join the network and assist in routing data. For children that are sleeping, they buffer data packets. Manages the other network functions, so it cannot sleep and must be powered on at all times.

The NWK layer satisfies different functions. It is responsible for starting or creating a network (usually the coordinator), allowing devices to join or leave a network, assign addresses to devices, implement synchronization using beacons or by polling, implement security, implement routing protocols.

The Application Support Sublayer (APS) also implements high level functions that simplify the operation of the final applications, such as the generation of the application level PDU (APDU) by adding the appropriate protocol overhead, binding (devices can be bound, and then this layer is able to transfer a message from one to other), group address filtering (the ability to filter group-addressed messages based on endpoint group membership), reliable transport (increases the reliability of transactions above that available from the NWK), duplicate rejection (messages offered for transmission will not be received more than once), fragmentation (enables segmentation and reassembly of messages longer than the payload of a single NWK layer frame), AIB (APS Information Base) management (manages the information about bounded devices), security (the ability to set up authentic relationships with other devices through the use of secure keys), and group management (the ability to declare a single address shared by multiple devices, to add devices to the group, and to remove devices from the group).

7.2.3.2 ZigBee profiles

ZigBee is a protocol that standardizes the application layer, allowing devices from different manufacturers to participate in the implementation of an application. Thus, profiles are the key element for the communication between ZigBee devices.

Profiles define the offered services and describe a common language for exchanging data: type of messages, available commands and their responses, etc. This way, they allow communication between separated devices to build a distributed application, guaranteeing device interoperability even across different manufactures. For example, tasks for joining the network or discovering devices and services are supported by ZigBee Device Objects (ZDO).

One example is home automation. This ZigBee profile allows a number of computer types to exchange control messages, allowing for the creation of a wireless home automation program. These systems are designed to send and receive known messages in order to affect power, such as turning on or off a lamp, reporting a light sensor calculation to a lighting controller, or sending a warning message if an occupancy sensor detects motion [46].

7.3 Wide Area Networks: Cellular connectivity

In previous sections we have studied short range communication protocols which perfectly fit many IoT applications. Nevertheless, the IoT devices using Bluetooth, ZigBee or even Wi-Fi require a local gateway that provides connectivity to a backend through the Internet.

This approach is not always valid. For example, if we want to install sensors for metering (utilities such as water, gas, etc.), detect fires in forests or measure other parameters in areas without a pre-existing communications infrastructure, we will require a mechanism or a technology to reach large areas.

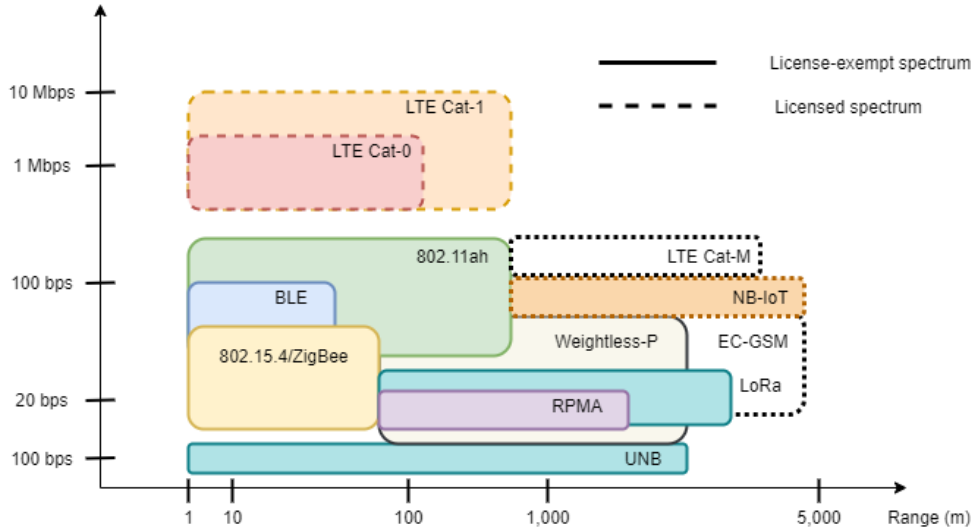


Figure 7-16: IoT technologies. Range vs data rates.

There are different alternatives, for example:

- Create a network covering a large area with a multi-hop network over links created with short range technologies (such as ZigBee, Bluetooth LE, Wi-Fi HaLow, Z-Wave, ANT+, ISA 100, WirelessHart, etc.).
- Use traditional cellular technologies, such as GSM, GPRS, 3G, LTE, etc.
- Use the new cellular protocols specifically designed for IoT: LTE-MTC (LTE-M), LTE-eMTC, NB-CIoT, NB-LTE
- Use the new Low Power Wide Area Network (LPWAN) technologies: Sigfox, LoRaWAN, NWAve (Weighless-N), Platanus (Weighless-P), Weighless-W, Ingenu (OnRamp), etc.

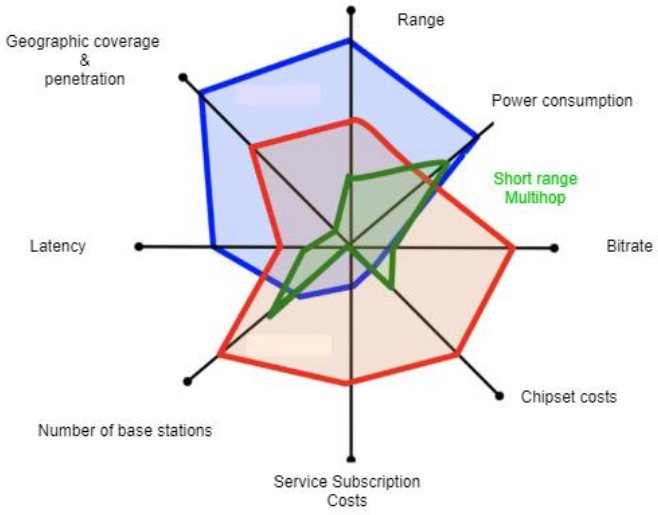


Figure 7-17: Comparison between IoT technologies for wide area coverage

A multi-hop network with short range technologies will be hard to configure and maintain, and it will require placing the nodes at a distance that makes it possible the communication. This approach will be limited to very specific situations.

Traditional cellular networks are being used nowadays to connect IoT devices, but that technology was not designed for scalability or low power. Also, the costs are usually high.

Then, for those scenarios, new cellular protocols designed for IoT are the best choice. LPWAN is a relatively recent category of wireless communications technologies designed to support IoT deployments that require strong coverage over large areas even when devices are underground or deep within buildings. They also provide great power efficiency, allowing devices to run on batteries for 10 years or more. LPWAN protocols are also designed to be massive scalable, making it possible the connection of millions of devices at once in a single deployment, allowing each cell or base station the connection of more than 10k devices. Also, as any other system for IoT, the communications hardware is cheap (< 10 $). In return, LPWAN usually provides a low bandwidth (few bytes of data to be transmitted per device per day).

7.3.1 Sigfox

Sigfox was the first LPWAN technology launched to the market. It was created by Sigfox, founded in 2009 which developed the technology and also provides a communication service for IoT.

Sigfox employs an Ultra Narrow Band (UNB) modulation. The information regarding the physical layer was opened to the public in 2019. Nevertheless, the there is no public information available regarding the network layer protocols.

Early versions of the technology only allowed unidirectional communication, from the device to the aggregator, but now bi-directional communication is supported. Every base station can manage up to one million connected objects, with a coverage area of 30-50 km in rural areas and 3-10 km in urban areas.

Sigfox uses a BPSK modulation for uplink (differential binary phase shift keying or D-BPSK), with Ultra Narrow-Band (UNB) technology, which provides benefits such as improved scalability, reduced interference, less power, etc. For downlink, it uses a GFSK modulation.

The physical layer is designed to operate in license-free ISM bands, at 868 and 900 MHz.

Usually, devices are configured to transmit information to a server (data collection), but some devices are also able to receive downlink messages, which can transport up to 8 bytes data to the device.

Users can purchase connectivity. In low volumes, the cost is approximately 16€/year per device, providing up to 140 messages/day (12 bytes payload).

According to Sigfox, the typical life of a message is as follows:

1. *The device "wakes up" at most 6 times per hour. During that awakening, its sensor(s) activate and retrieve information. Therefore...*
2. *The device has something to report. It creates a message. Sigfox allows for messages with a 12-byte payload (at most), itself wrapped into a 26-byte data frame (at most).*
3. *To send its message, the device broadcasts radio signals. To do so, it transfers a small amount of energy on a random frequency, with no protocol overhead. The device sends every frame three times and on three random frequencies: this is "frequency hopping". It is one of Sigfox's security feature: it protects radio frames from sniffing because it is impossible to know where the device is going to send in the operation band. It also enables broadcasting diversity and transmission resilience.*
4. *These signals get to the base stations listening to the Sigfox frequency. Sigfox base stations listen to the whole radio spectrum, and interpret all UNB signals they receive on the legal frequencies. The signals from Sigfox devices are therefore detected on-the-fly.*
5. *The base station retrieves the message from the signal and checks that it is valid. Base stations demodulate the received signals in order to assert that there is indeed a message within.*
6. *The base station uploads the message to the Sigfox Cloud.*
7. *The Sigfox Cloud receives the messages and authenticates the sending object. Once it has asserted the sender's ID, Sigfox Cloud knows the action to perform with the message.*

8. *The Sigfox Cloud sends the message to the device maker's server. Event callbacks created on the Sigfox Backend create a "push" procedure, which makes that Sigfox Cloud only contacts the remote server when there is a message to transmit. That remote server can of course be an integration with a major Cloud solution: Amazon AWS, Microsoft Azure, theThings.io, etc.*
9. *Finally, the data is displayed on the customer platform.*

Because of the operation mode, according to the Libelium company, Sigfox is recommended for long-range device communications in cities where their base stations are deployed, but it is not recommended for some use cases. For example, when it is necessary to send one frame every few minutes, or if it is necessary to exceed the 140 packets per day limit, it is necessary to transmit a large amount of information (the maximum payload is 12 bytes) or bi-directional communications are necessary.

Sigfox is not recommended also for real-time streaming, because transmissions are not performed in real time as there is a minimum delay for packet arrival

7.3.2 LoRa

LoRa is a physical layer technology created by Semtech, the company that builds the physical devices and that owns the proprietary radiofrequency technology and intellectual property rights. Thus, Semtech built its business following the opposite approach to Sigfox. While Sigfox opens its physical layer to third party manufacturers and controls the network to offer the connectivity service, Semtech keeps ownership of the physical layer and allows third parties to deploy networks using the LoRaWAN standard.

The network's communication protocol and device architecture are described by LoRaWAN (LoRa is the physical layer that enables the long-range communication link).

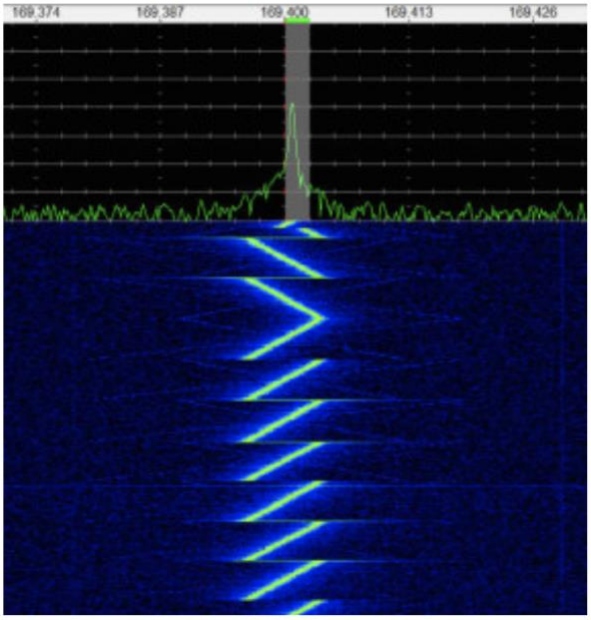


Figure 7-4: Chirp modulation

LoRa uses its own spread spectrum modulation scheme (LoRa Spread Spectrum), variant of Chirp Spread Spectrum (CSS). LoRa is purely a physical layer protocol (the higher layers are defined by LoRaWAN), implemented in ISM bands: 868 (Europe), 915 (USA) and 433 MHz (Asia). It is designed for long range communications (up to 22 Km in rural areas or 2 to 5 Km in urban environments). One of the requirements is also the low power consumption, so it keeps low data rates (0.3 to 50 kbps for uplink) and only replies to previous transmissions in downlink (the sending node remains listening for a while before going to sleep).

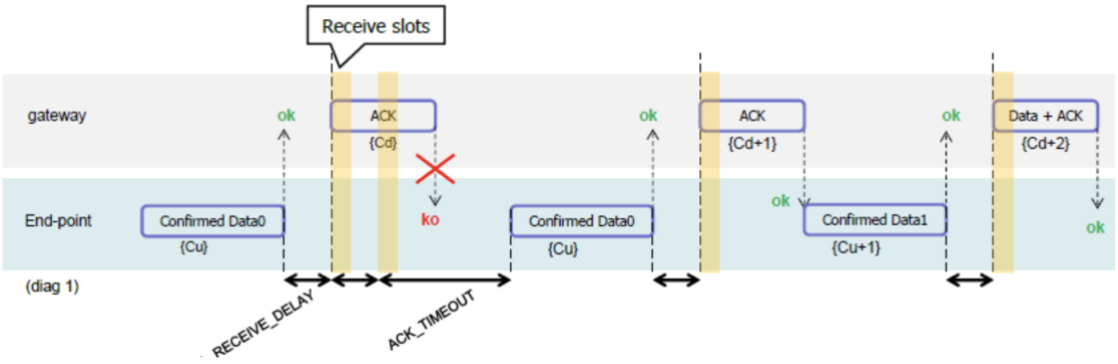


Figure 7-5: LoRa MAC uplink timing diagram for confirmed data message [41]

The MAC layer is defined by the LoRa Alliance and openly published as a part of LoRaWAN. It is based on ALOHA. The transmission slot is scheduled by the end-device depending on its

needs, adding a random time to avoid interferences. Figure 7-5 shows a timing diagram where the end-device first transmits a confirmed data frame containing the Data0 payload at an arbitrary instant and on an arbitrary channel. The network receives the frame and generates a downlink frame with the ACK exactly after RECEIVE_DELAY1 seconds. In this example the ACK is not received, so the information has to be retransmitted.

A system address (DevAddr), an application identifier (AppEUI), a network session key (NwkSKey), and an application session key are all stored on each computer (AppSKey). The keys are used to encrypt and decrypt the payload area of application-specific data messages, as well as to validate the integrity of messages.

There are three types of nodes:

- Bi-directional end-devices (Class A): This is the most popular operation model. In the very least, all systems must be Class A. They make it possible to communicate in both directions. Two brief downlink receive windows are opened after a system sends data over the uplink channel. In the other time, downlink messages from the cloud would have to wait for the next planned uplink.
- Bidirectional end-devices with scheduled receive slots (Class B): This type of devices allows for a greater number of receive slots. They receive a coordinated Beacon from the gateway, which helps the server to know when a Class B device will be listening, and they open extra receive windows at predetermined times. This class will be used to create actuators that must be activated from the server.
- Bi-directional end-devices with maximal receive slots (Class C): This category of devices have a higher consumption because they are expected to have nearly continuously open receive windows. This way, there is no latency for downlink communication, making this operation suitable for actuations with strict time constrains, such as main power actuators, automotion, control, etc.

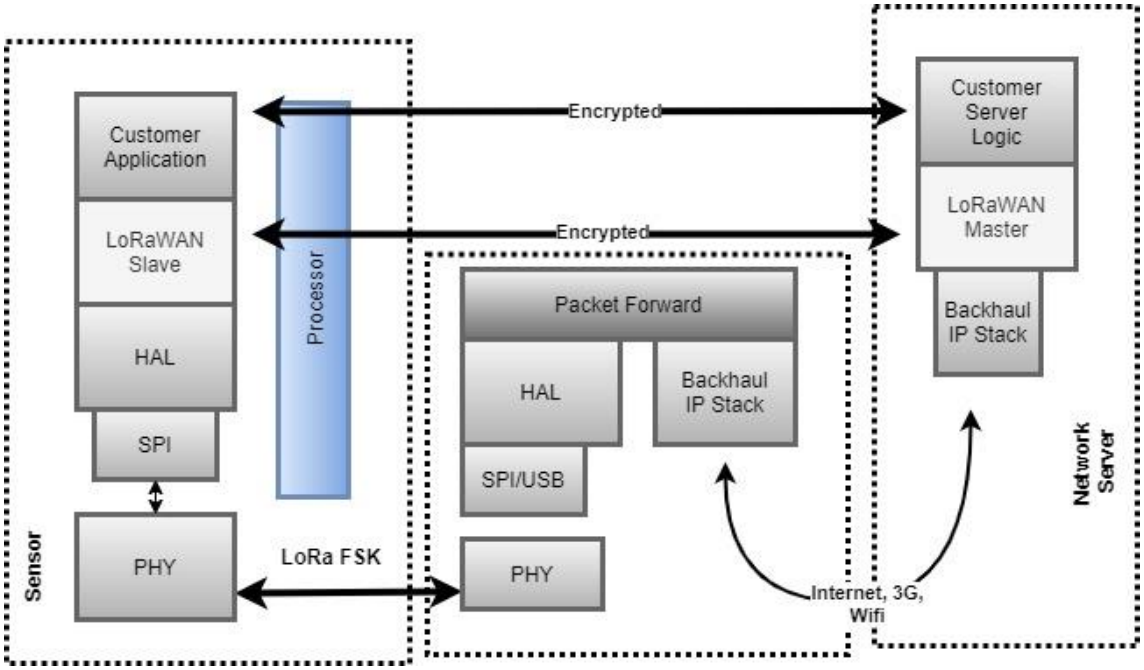


Figure 7-6: LoRa WAN architecture and communication stack (source: https://lora-alliance.org)

LoRaWAN not only defines a MAC layer. The protocol defines a complete networking protocol, including the topology and the details for the technical implementation (Figure 7-6).

LoRaWAN compatible devices are connected using the LoRa physical layer to one or more Gateways, following a star-of-stars topology. Gateways are connected to an IP network and relay messages to a central network server.

While gateways must listen to all frequencies, communication between end-devices and gateways is spread out on various frequency channels and data rates (the frequencies and data rates depend on the distance and length of the transmissions). The end-devices are not directly linked to a specific gateway.

User applications are running in the cloud, directly in the network server or in any other location using an additional protocol/API (e.g., MQTT). LoRaWAN also includes mechanisms to implement roaming between networks, updates over the air (OTA), or to maintain QoS.

LoRa can be deployed as a public network (following an approach similar to Sigfox) or as a private network. Anyone can buy a LoRa gateway and connect devices to implement an application.

Furthermore, the LoRa physical layer can be used to implement other communication stacks (different to LoRaWAN). LoRa can be us as the physical layer, and implement any protocol

over it, ranging from custom protocols, implementations of existing protocols or solutions (such as 802.15.4 MAC or 6LoWPAN). For example, Simphony-Link uses LoRa as the physical link, but uses the IEEE 802.15.4 MAC supporting bidirectional communications and low latency. Virtual Extension VEMesh creates a mesh topology with multihop bidirectional communications.

7.3.3 NB-IoT

Narrowband IoT (NB-IoT) is an open 3GPP standard based on LTE defined in the 3GPP Release 13 Release 13 in June 2016. It allows a flexible and quick deployment, because it is compatible with existing network infrastructure by using a small portion of the available spectrum in LTE. Most eNB (LTE base stations) can be upgraded to support NB-IoT.

Although NB-IoT is a modern radio protocol (only devices programmed for it would be able to connect), it heavily borrows from LTE technologies, including numerologies, downlink orthogonal frequency-division multiple access (OFDMA), uplink single-carrier frequency-division multiple access (SC-FDMA), channel coding, rate matching, interleaving, and so on. [42].

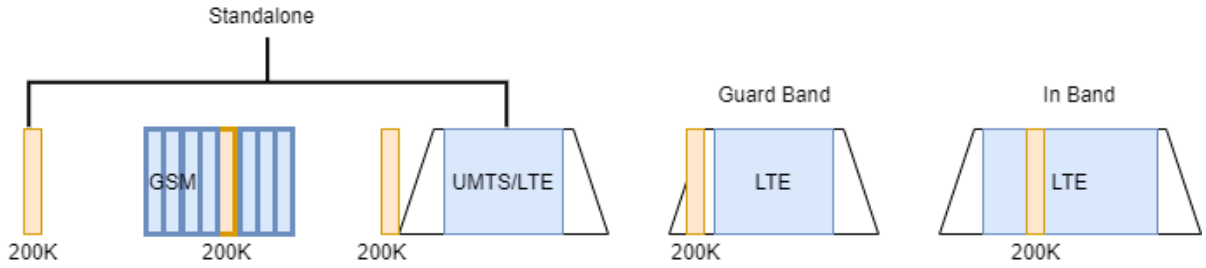


Figure 7-21: NB-IoT deployment alternatives

Different alternatives may be used to deploy NB-IoT by an operator in his licensed LTE band (Figure 7-). It may be deployed as a standalone carrier or within the LTE spectrum. In this case, it can be deployed inside an LTE carrier or in the guard band. In any case, IoT devices would be not aware of the option selected.

NB-IoT takes advantage of the modern cellular communication technologies by adding a narrowband signal that can be used to communicate low powered devices. Features such as the multiple access and the modulation schemes from LTE are adapted. For example, for uplink two new channels using SC-FDMA were standardized: Narrowband physical random

access channel (nprach) and narrowband physical uplink shared channel (npUSCH). For downlink, also six new channels were created: Narrowband primary synchronization signal (NPSS), Narrowband secondary synchronization signal (NSSS), Narrowband physical broadcast channel (NPBCH), Narrowband reference signal (NRS), Narrowband physical downlink control channel (NPDCCH), Narrowband physical downlink shared channel (NPDSCH)

NB-IoT is designed to build low complexity devices. For example, it allows only half-duplex frequency-division duplexing operation. The coverage objective is achieved with 20 or 23 dBm power amplifier, making it possible to use an integrated power amplifier in the UE.

7.4 Wireless Sensor Networks

7.4.1 Introduction

A Wireless Sensor Network (WSN) is a self-configuring network of small sensor nodes that communicate using radio signals and are used to feel the physical environment in large numbers [43].

The concept of WSN was popular before IoT, that is, before sensors could be directly connected to the Internet to provide services through a backend. Nevertheless, we can say that WSNs are a subset of the IoT. WSNs are formed by a large collection of sensors that are not directly connected to the Internet, so they form a network for distributing the information.

We can classify devices participating in WSNs in three categories:

- Sources of data: Responsible of measuring data and report them “somewhere”. This class of devices are typically equipped with different kinds of actual sensors.
- Sinks of data: Interested in receiving data from the WSN. For example, a smartphone or a gateway that transmits the information to a backend.
- Actuators: Use data to control a computer, which is typically also a drain.

This kind of networks have sensing and/or actuation faculties, in combination with computation and communication abilities that can be used to build their own infrastructure,

they can be built when there is no infrastructure available (e.g., disaster zones), building an infrastructure is too expensive or there is no time to build an infrastructure (e.g., military operations).

- Disaster recovery: This is one of the most often mentioned application for WSNs. A typical scenario is wildfire detection where sensor nodes, with thermometers or other devices to detect fire and a location mechanism (GPS or a relative positioning system), are deployed in a forest, for example from a plane, and collectively produce a map to determine the area affected from a wildfire, so firefighters get that information in real time.

- Biodiversity mapping: WSNs can be used to control the environment (sensing pollutants, erosion) or observe wildlife.

- Intelligent buildings (or bridges): WSNs can be deployed to improve the efficiency of buildings by controlling humidity, ventilation, or air conditioning (HVAC). In this area, they can also be used for measuring room occupancy, temperature, air flow, etc. In seismic active areas, they can also be used to monitor mechanical stress after earthquakes.

- Facility management: For large facilities with several buildings WSNs can be used for different purposes, ranging from intrusion detection into facilities or access control, detecting leakages in chemical plants, etc.

- Machine surveillance and preventive maintenance: Sensor nodes to can be attached in places hard to connect with a wire to detect abnormal operation patterns or unexpected stops.

- Precision agriculture: WSNs can be used to analyze the soil and apply fertilizer/pesticides/irrigation only where needed.

- Medicine and health care: Postoperative and intensive care, or long-term surveillance of chronically ill patients or the elderly.

- Logistics: Tracking goods with IoT devices, knowing the state of the good (temperature, impacts, etc.).

- Telematics: Sensors can be embedded in the streets or roadsides to gather information about traffic conditions (number of vehicles, speed, etc.) creating an intelligent roadside or use cars as the sensor nodes.

This type of sensor networks has to find mechanisms to create a network that is able to support the specific quality of service, lifetime, and maintainability requirements specific of the particular application. Usually the communication range is small, so it is necessary to create a multi-hop network, but this kind of networks have their own challenges. First of all, there is no central entity to organize/orchestrate the whole network, so participants must organize themselves and implement decentralized MAC mechanisms and distributed routing protocols. If the participants are mobile, then it is necessary to implement specific mechanisms (route repairing, location, etc.) in what is called a Mobile Ad-hoc NETwork (MANET).

7.4.1.1 Multi-hop networks

The term "ad-hoc network" means that the network was setup for a specific purpose without a previous planning. In this case, the self-configuration of the network is essential to make everything work without manual intervention. Mobility is another ingredient in MANETs. WSNs and MANETs share the same restrictions, but MANETs have additional challenges. For example, the mobility changes multihop routes in the network, and routing protocols have to handle such changes, but protocols have to be also energy efficient for battery powered nodes.

7.4.1.2 Requirements for WSN

The applications supported by WSNs and MANETS usually share a set of requirements:

- Data-centric networks: It is not (really) relevant which node provides the information, but the information itself. Thus, it is possible to measure the information from different nodes and forward it to the sink. If the information from one of the nodes arrives, it will be enough.
- Traditional quality of service standards for WSNs, such as bounded delay or minimum bandwidth, are obsolete. In some cases, simply delivering a packet is sufficient; in

others, the time it takes to respond after an incident is observed is important, and so on.

- Fault tolerance: Nodes may run out of energy or even get destroyed, so the network has to be robust against node failure.
- Lifetime: In many situation nodes will be powered by batteries that will not be replaced after being emptied, so it will be necessary for the network to operate at least for a mission time. Nevertheless, what is important is the lifetime of the network, more than the lifetime of each particular node.
- Scalability: Since a WSN can have a huge number of nodes, the architecture and protocols must be scalable.
- Wide range of densities: The number of nodes per unit area may vary, even over time. The protocols should support vast or small number of nodes per unit area.
- Programmability: Nodes may have to be reprogrammed with new versions or upgrading their functionality.
- Maintainability: The network must adapt to changes, so it should include self-monitoring and mechanisms that adapt the operation of the WSN to the actual state, including the appearance of new resources.

7.4.2 6LoWPAN

WSN (Wireless Sensor Networks) require (high level) standard protocols to interoperate. Thus, the IETF has been working in different working groups (WGs) to develop standards for WSNs, being the most known 6LowPAN [46].

IPv6 over Low-power Wireless Personal Area Networks is abbreviated as 6LoWPAN. It specifies the requirements for IPv6 communication over IEEE 802.15.4 wireless technology.

IPv6 over Low Power Wireless Personal Networks, or 6LoWPAN, is an acronym that stands for IPv6 connectivity over IEEE 802.15.4 wireless communication technologies.

IPv6 over Low-power Wireless Personal Area Networks is abbreviated to 6LoWPAN. It specifies the requirements for IPv6 connectivity over IEEE 802.15.4 wireless technology.

The IPv4 protocol is not supported by this specification. Low power lossy networks is a concept that refers to networks made up of heavily compressed nodes linked by a set of " lossy " connections (LLNs). Low speed, low throughput, low cost, and unreliable networking are typical characteristics. A LoWPAN is a kind of LLN made up of IEEE 802.15.4-compliant modules.

The following are the features of LoWPANs [46]: Sleeping mode; small packet size Several addressing modes are specified by IEEE 802.15.4. a limited bandwidth; Star and mesh topologies are examples of topologies. Usually, the location of the instruments is not predetermined. LoWPAN devices are notoriously inefficient. A large number of sensors are expected to be installed during the technology's lifespan.

IPv6 has a number of benefits. Because of the design of IP networks, existing infrastructure may be used. IP-based systems have been shown to work and are readily available, making deployment faster and less expensive. IP networking infrastructure is documented in open, publicly accessible specifications.

IP networking tools are available. Without the use of intermediary organizations like protocol translation gateways or proxies, IP-based devices may bind to other IP-based networks. IPv6 allows for a large number of addresses and allows for self-configuration of network parameters (SLAAC). This is important for 6LoWPANs, which would accommodate a huge number of computers.

The small packet size is one of the characteristics of 6LoWPANs. As a result, IPv6 and higher layer headers are compressed wherever possible.

A considerable number of LoWPAN instruments have been deployed. They'll just need a small monitor and input system.

These devices' locations can be difficult to find. The protocols used in LoWPANs should have a minimal setup, according to the recommendations. To discover, monitor, and manage the services offered by the devices, LoWPANs include basic service discovery network protocols.

7.4.2.1 6LoWPAN protocol stack

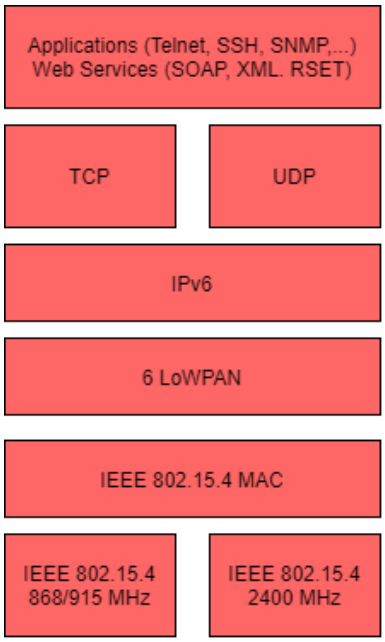


Figure 7-7: 6LoWPAN protocol stack [46]

The primary explanation for improving IETF specifications is that there are some critical problems that need to be resolved by using an adaptation layer between the IP (network layer) and the lower layer: 6LoWPAN 6LoWPAN (Figure 7-7). Some features of 6LoWPAN are: Fragmentation and Reassembly layer; Header Compression; Address Autoconfiguration, Mesh Routing Protocol.

IEEE 802.15.4 defines four types of frames: beacon frames, MAC command frames, acknowledgement frames, and data frames. IPv6 packets must be transported in data frames.

It is recommended that IPv6 packets be transported in data frames.

Both source and destination addresses must be included in the IEEE 802.15.4 frame header.

The source or destination PAN ID fields may be included.

Both 64-bit extended addresses and 16-bit short addresses are supported.

IPv6 level multicast packets are transported as link layer broadcast frames in IEEE 802.15.4 networks.

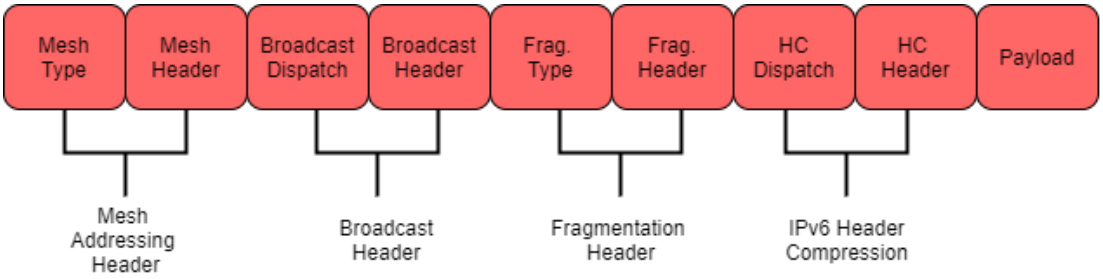


Figure 7-8: IPv6 frame

Due to limited bandwidth, power or memory resources, the 6LoWPAN adaptation format was specified to transport constrained links. 6lowpan offers a solution for each of these objectives and requirements:

- A mesh addressing header to support sub-IP forwarding.
- A fragmentation header to support the IPv6 minimum MTU requirement.
- A broadcast header to be used when IPv6 multicast packets are to be sent over the IEEE 802.15.4 network.
- Stateless header compression for IPv6 datagrams to reduce IPv6 and UDP headers. These headers are used as LoWPAN encapsulation.

8 IoT Connectivity Protocols

Author(s): **Fabrizio Granelli**
Claudio Sacchi

8.1 IoT Connectivity Protocols

IoT promises to interconnect and network “objects”, from smart meters to household machines, from cars to tiny sensors. However, the network needs to follow up and to satisfy such unprecedented demand, and this represents especially for the mobile industry a big obstacle to overcome. The traditional centralised mobile network architecture may simply not be up to the job of handling the huge traffic volumes and massive connectivity expected to result from the billions of IoT devices predicted to come on stream over the next few years. Indeed, the architecture of today’s mobile networks, based on a centralised star topology, needs to be redesigned to cope with billions of devices rather than the millions of smartphones for which they were originally intended.

IoT is expected to enable new levels of wireless connectivity to a broad range of things by bringing devices, people and analytics platforms together. The combination of sensor and aggregated data with other related information will spark new services, applications and opportunities.

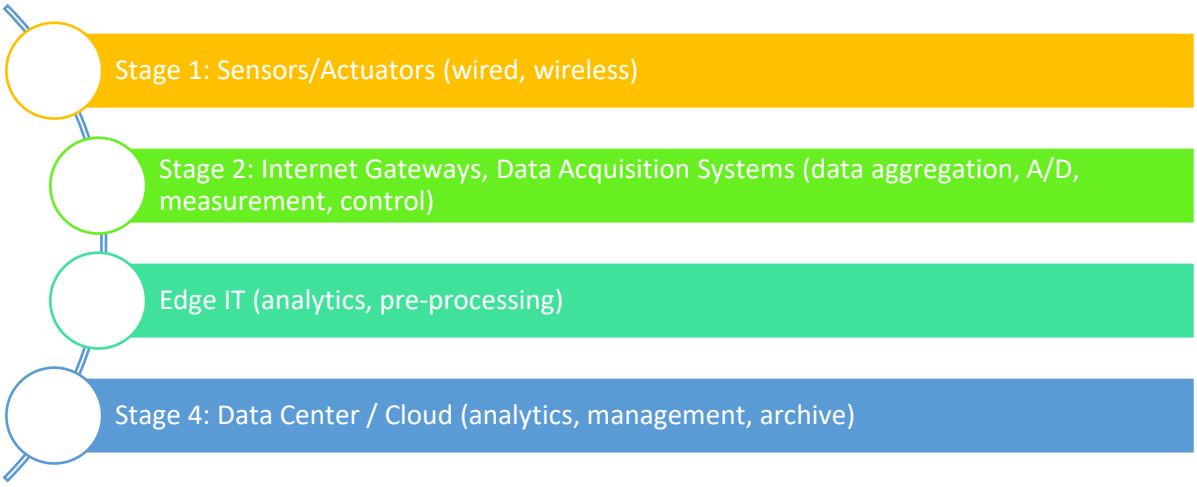
This section will analyze the different aspects related to connectivity of IoT devices.

8.2 IoT Connectivity Paradigms

The Internet of Things is a complex system-of-systems. Indeed, the implementation of different levels of interactions among sensors and actuator might involve different communication technologies and/or a combination of several communication networks.

A general view of the IoT architecture is proposed in **Error! Reference source not found.**.

1. Sensors and actuators (providing access to the outer world via sensing and actuating)
2. Internet getaways and Data Acquisition Systems (processing the enormous amount of information collected on the previous stage and converting)
3. Edge IT (performing enhanced analytics and data pre-processing)
4. Data center and cloud (providing in-depth processing and service delivery)



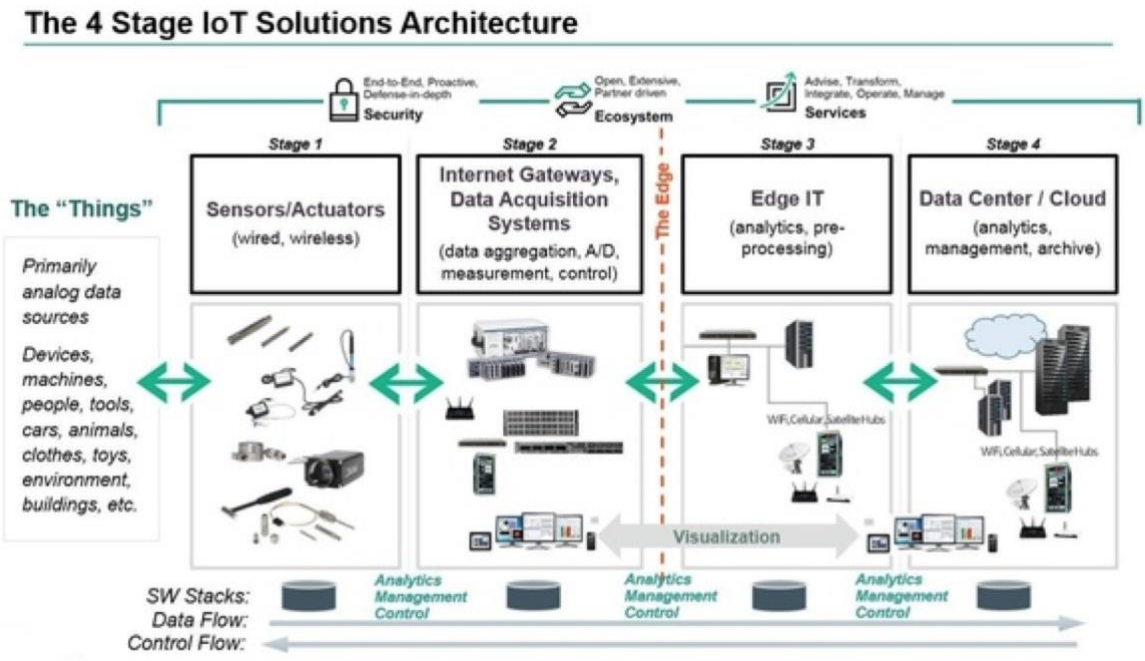


Figure 8-1: General stages of an IoT architecture.

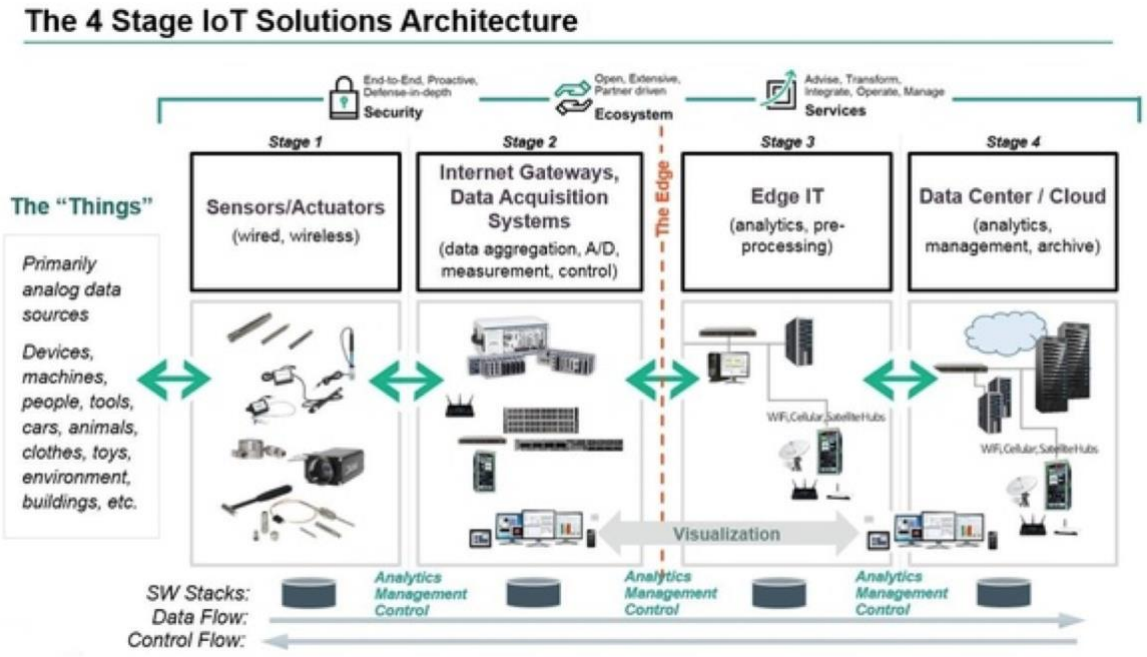


Figure 8-2 provides a high-level illustration of different data flows paradigms that might be triggered in an IoT application or service.

Some applications might require actions that may be taken on sensors locally or via processing at an Application provider location or via the end user's operation centre, while other services might require global access, e.g. global cellular access – through a Mobile Network Operator or even a Mobile Virtual Network Operator (MVNO).

This will depend on the conceptual location where decisions are made. Decision making may be local (e.g. via local computing in proximity, thus implying usage of direct communication technologies) or remote (e.g. via a cloud server accessible through a WAN connection), or even a combination of local and remote.

Moreover, it is important to notice that for interoperability reasons most sensors and actuators are accessible by proper IoT gateways. The gateways enable to "translate" the IoT-specific communication standards (e.g. LPWAN, LoRA, etc.) into LAN/WAN standards (e.g. defining data formats that can be encapsulated into IP datagrams).

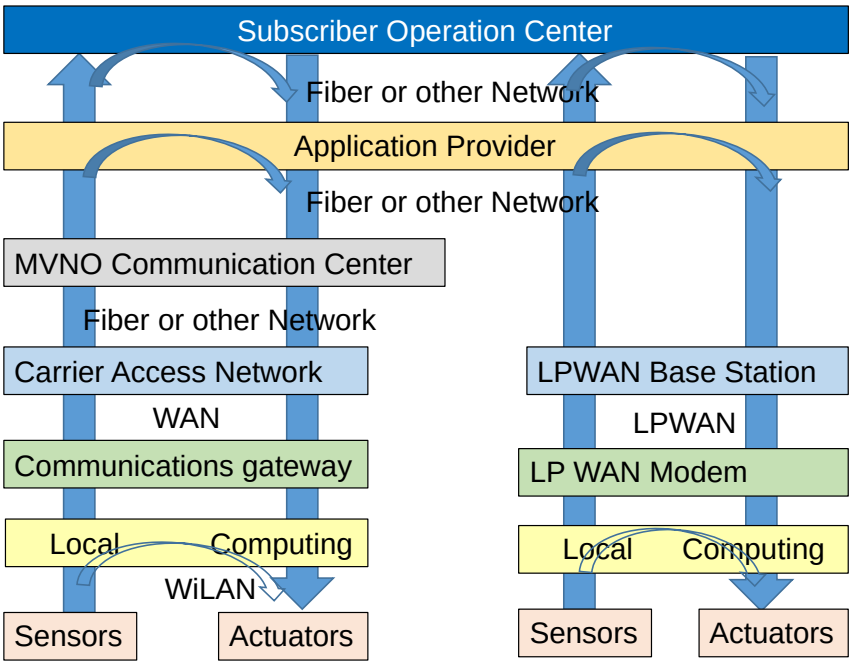


Figure 8-2: Different operations among sensors and actuators in the Internet of Things.

Typically, long range connectivity is provided by existing IP networks, in order to integrate IoT services within the public Internet. Nevertheless, for a reliable operation, the Internet of Things require proper architectures to adapt the general Internet to its requirements and specific features.

In this framework, the most relevant models are those proposed by ETSI and IETF.

Figure 8-3 illustrates the ETSI model, which proposes a model focused on the perspectives of telecom carriers.

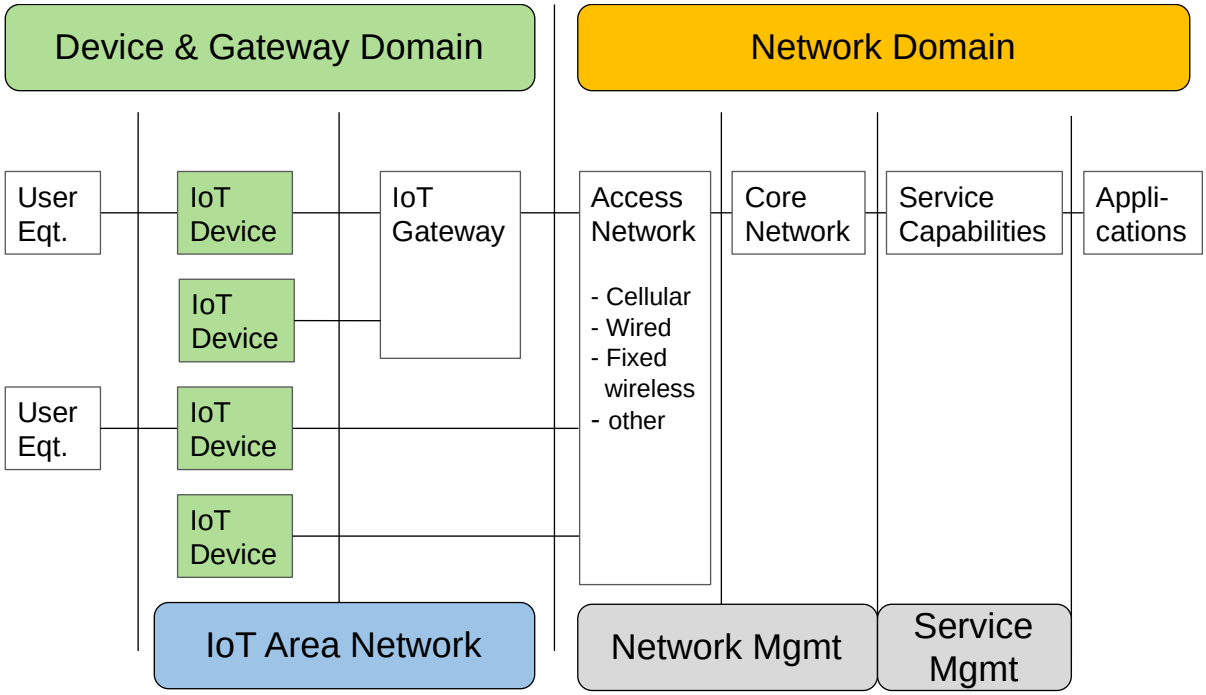


Figure 8-3: Internet of Things connectivity model by ETSI.

The figure shows the pieces of an IoT connectivity solution from a technical perspective. It does not show the business pieces which can be even more complex, with various companies supplying pieces of the various sections. For example, one might buy Network management from a third party who then interacts with both the carrier's network, your devices and gateways and your back ends.

The model clearly outlines the boundaries between the IoT Area Network, which is typically IoT-specific (e.g. communication is performed by using IoT standards), and the Network Domain, which represents the WAN connectivity and includes the services.

Like all networking solutions, IoT wireless networks can be built in a variety of configurations – following a modular approach. The IoT device may attach to the user equipment – health monitoring of equipment, process or environmental sensors – or it may be the user's equipment (location tracking device).

The model assumes both the possibility for the network carrier to directly provide application services as well as to run many applications "over the top" of the proposed model.

An IoT device may connect directly to the WAN or connect to a local gateway. In those cases where there is a gateway, two wireless choices need to be made – the cellular

side of the gateway and the IoT Area Network side – star, tree, mesh/ad-hoc architectures, standards-based or proprietary.

The IETF model represents an Internet-centric model, which is based on the 6LowPAN architecture. 6LoWPAN is the acronym of IPv6 over Low-Power Wireless Personal Area Networks. The 6LoWPAN concept originated from the idea that "the Internet Protocol could and should be applied even to the smallest devices," and that low-power devices with limited processing capabilities should be able to participate in the Internet of Things.

The 6LoWPAN standard provides encapsulation and header compression mechanisms that allow IPv6 packets to be sent and received over IEEE 802.15.4 based networks. In particular, IPv4 and IPv6 are used for data delivery in common packet-switched network architectures such as the Internet, while IEEE 802.15.4 standard provides equipped devices with sensing/ communication ability in the wireless access domain.

6LoWPAN main specification is provided by IETF group in RFC 4944 [73]. The RFC was updated by RFC 6282 introducing header compression, and by RFC 6775 including neighbour discovery optimizations. IPv6 over Bluetooth Low Energy (BLE) is instead defined in RFC 7668.

RFC 4944 proposes an IPv6 over Low-Power Wireless Area Networks approach based on direct end-to-end Internet integration. The benefits of 6LoWPAN include: openness, availability of standards, transparent Internet integration, global scalability, end-to-end data flow management and small device memory footprint.

6LoWPANs represent stub networks. By definition, "*a stub network is a computer network, or part of an internetwork, with no knowledge of other networks, that will typically send all of its non-local traffic out via a single path, with the network aware only of a default route to non-local destinations*".

Indeed, the transition/integration of IoT on the public Internet architecture is not trivial, since IoT is characterized by constrained resources (bitrate, power, computational power), massive connectivity and highly intermittent traffic (Figure 8-4).

To this goal, the following protocol stack changes are required:

1. The general Internet protocol stack is not the best fit for IoT applications, so changes are required to make for reliable and properly functioning IoT systems;

2. Modified TLS to support UDP data transfer is preferred in many IoT applications over TCP, which requires relevant overhead due to the session management;
3. CoAP is a tiny subset of HTTP with interworking via proxy server;
4. RPL routing supports a multitude of router selection approaches including link reliability and power source.

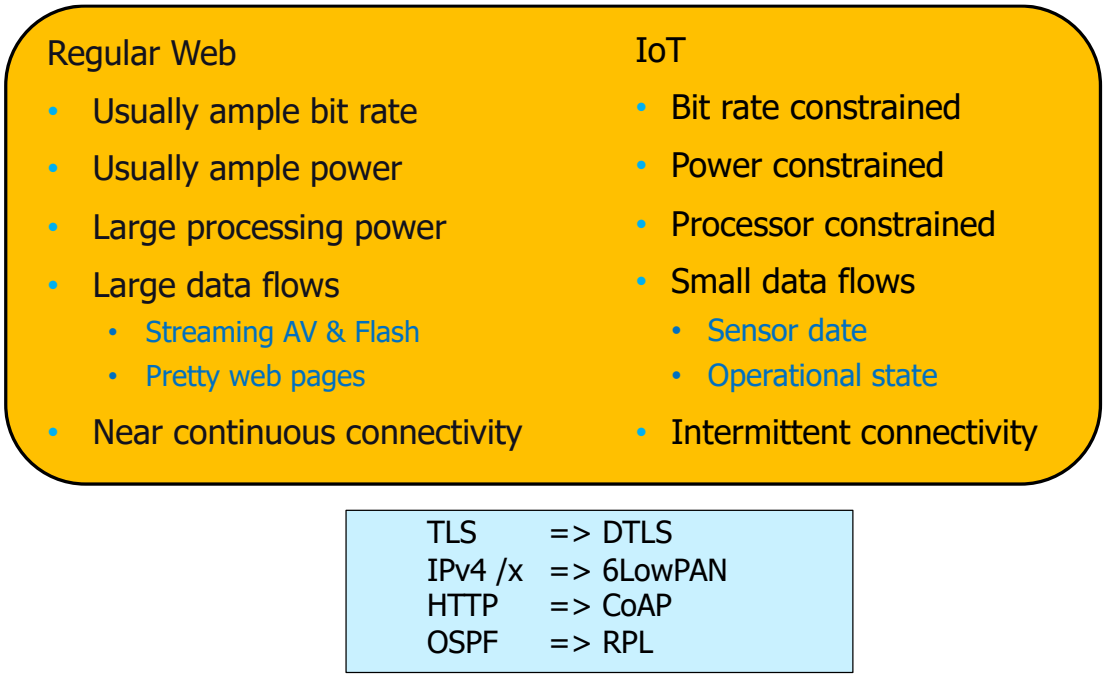


Figure 8-4: Internet of Things connectivity model by IETF (from regular web to IoT).

8.3 Application Layer Protocols for the IoT

In general, the application layer protocols for the Internet of Things employ the REST paradigm, which represents the reference communications architecture for IoT devices. However, typically, the IoT devices are resource constrained, and there may be data loss or a high memory requirement in this type of communication.

For this reasons, several protocols might be used at application level, with the most common being of course HTTP. Nevertheless, other suitable protocols for IoT communications are Message Queue Telemetry Transport (MQTT), Constrained Application Protocol (CoAP), WebSocket, Extensible Messaging and Presence Protocol (XMPP), and Advanced Message Queuing Protocol (AMQP).

8.3.1 HTTP

Hypertext Transfer Protocol represents a possible alternative to support IoT services and maintain compatibility with the World Wide Web. HTTP is based on a client-server paradigm, where the client requests data from the server through a TCP connection.

HTTP messages are text-based and are characterized by a relevant level of redundancy, both in the HTTP headers as well as in the transported format (typically html text or binary data converted to text format). For this reason, header compression is a common functionality to reduce the communication overhead and data size in IoT scenarios.

8.3.2 MQTT

The Message Queuing Telemetry Transport (MQTT) protocol is an open OASIS and ISO standard (ISO/IEC 20922). MQTT allows lightweight message transport between devices through a publish-subscribe network protocol. Indeed, the purpose of MQTT is to provide connections with remote locations, under constraints of "small code footprint" and/or limited network bandwidth. The protocol was designed to run over TCP/IP; however, any network protocol that provides ordered, lossless, bi-directional connections can support MQTT.

The MQTT protocol allows communications between two types of network entities: a message broker and several clients. The MQTT broker acts as a server, and it receives all messages from the clients. The messages are then routed to the appropriate destination clients. By definition, an MQTT client is any device that runs an MQTT library and connects to an MQTT broker over a network. This includes micro controllers up to full-fledged servers.

In the MQTT protocol, collected information is organized in a hierarchy of topics. When an MQTT client (in this case, the publisher) acquires a new item of data to distribute, it sends a control message with the data to the connected broker. The MQTT broker then distributes the information to any clients (in this case, the subscribers) that have subscribed to that topic. The publisher does not need to have any knowledge on the number or locations of subscribers, and subscribers, in turn, do not have to be configured with any data about the publishers.

An example of an MQTT connection is provided in Figure 8-5.

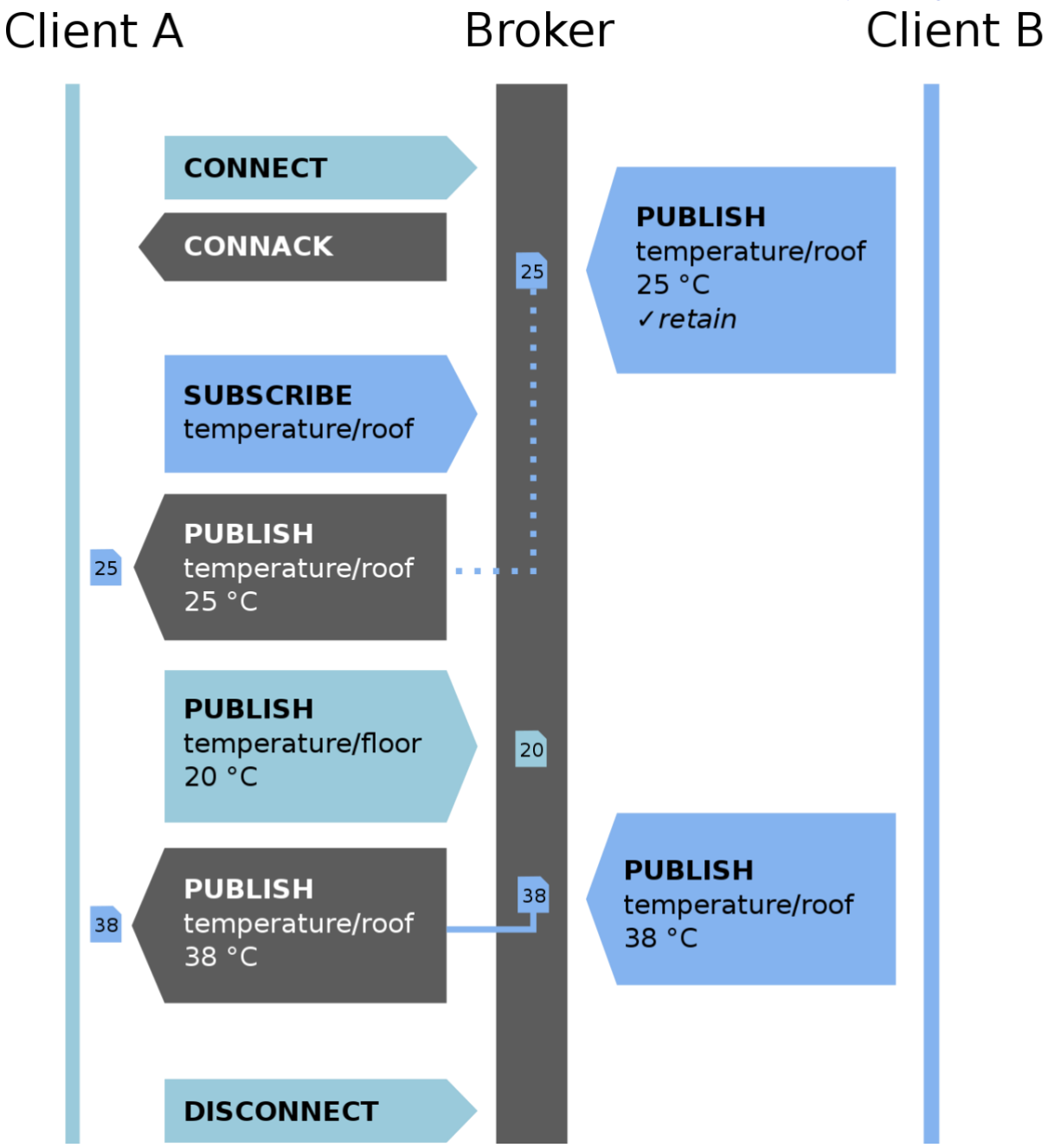


Figure 8-5: Example of an MQTT connection (Source: Wikipedia).

8.3.3 CoAP

The Constrained Application Protocol (CoAP), defined in RFC 7252, is a specialized Internet Application Protocol for constrained devices. CoAP is designed for use between devices on the same constrained network (e.g., low-power, lossy networks), called “nodes”, between devices and general nodes on the Internet, and between devices on different constrained networks both connected through the Internet. CoAP can be implemented via other mechanisms, such as SMS in the case of mobile networks.

CoAP is designed to be converted to HTTP for simplified integration with the web. Neverthless, CoAP supports specialized requirements such as multicast support, very low overhead, and simplicity. CoAP can run on most devices that support UDP or similar transport layer protocols.

Most of the standardization efforts for CoAP were done by the Internet Engineering Task Force (IETF) Constrained RESTful Environments Working Group (CoRE), with the specification include in RFC 7252. Several CoAP extensions are planned and in different phases of standardization in order to make the protocol suitable to IoT and M2M applications.

8.3.4 WebSocket

WebSocket is a computer communications protocol, standardized by the IETF as RFC 6455 in 2011. WebSocket provides full-duplex communication channels over a single TCP connection.

Even though both WebSocket and HTTP are located at layer 5 (application) of the TCP/IP layered model and request services to TCP at layer 4, RFC 6455 states that WebSocket "*is designed to work over HTTP ports 443 and 80 as well as to support HTTP proxies and intermediaries*," thus making it compatible with the HTTP protocol. Compatibility is achieved by enabling the WebSocket handshake to exploit the HTTP Upgrade header, in order to change from the HTTP protocol to the WebSocket protocol operation.

In practice, the WebSocket protocol enables interaction between a web browser (or other client application) and a web server with lower overhead than half-duplex alternatives such as HTTP polling, facilitating real-time data exchange between client and server.

8.3.5 AMQP

The Advanced Message Queuing Protocol (AMQP) is an open standard application layer protocol for message-oriented middleware. AMQP provides the following services: message orientation, queuing, routing (including point-to-point and publish-and-subscribe), reliability and security.

AMQP is a wire-level protocol. A wire-level protocol provides a description of the format of the data that is sent across the network as a stream of bytes. Consequently, any tool that can create and interpret messages that conform to this data format can interoperate with any other compliant tool irrespective of the specific implementation implementation language.

As a consequence, AMQP do not specify the protocol for data transfer between the messaging provider and client, in order to enable implementations from different vendors to be interoperable, in a similar way as SMTP, HTTP, FTP.

8.3.6 Test cases for the IoT Protocols

The following table provides a summary of the IoT protocols discussed in the previous sub-sections and some potential use cases for each one.

Name	Description	Use Case
MQTT	Simple and lightweight IoT protocol designed for constrained devices and low network bandwidth. See mqtt.org.	Small medical devices with limited network connectivity, mobile apps in mobile devices, sensors in remote locations that communicate with a gateway.
CoAP	Protocol based on the REST model and is suitable for constrained devices such as a microcontroller or a constrained network because it functions with minimum resources in the device or the network. See http://coap.technology/.	Smart energy applications and building automation applications.
WebSocket	A full-duplex communication channel over a TCP connection.	Implement WebSocket in runtime environments or libraries that act as servers or clients. You can apply WebSockets in an IoT network where chunks of data are transmitted continuously within multiple devices.
XMPP	Uses the XML text format for communication and runs over TCP. It's not fast and uses polling to	Use XMPP to connect your home thermostat to a web server so that you can access it from your

	check for updates when needed. See https://xmpp.org	phone. It's used in consumer-oriented IoT applications.
AMQP	The message queue asynchronous protocol is for communication of transactional messages between servers. See https://www.amqp.org/	AMQP is best used in sever-based analytical functions. It's effectively used in the banking industry.

8.4 Integrating IoT within current networks

After reviewing the application-level protocols, it is then necessary to discuss how to interconnect the objects that we want to use for building IoT services. In this scenario, we need to consider existing dominant technologies at the different layers of the protocol stack, from the network down to the physical layer of the TCP/IP protocol stack.

Since detailed description of IP, Ethernet or other specific technology is out of the scope of this textbook, the following subsections provide a short summary about the functionalities and usage of the different technologies that might be integrated in the IoT.

8.4.1 IPv4/IPv6, Ethernet/GigE.

The reference scenario is the current and future Internet. This implies the need to use the Internet Protocol (IP) as the common layer 3 technology, in order to foster interoperability and enable interconnection of different underlying technologies. Currently, two versions of IP are available: the “old” IPv4 and the “new” IPv6. As already mentioned, several approaches to IoT connectivity already focus on IPv6, but we should consider that currently IPv4 is still diffused especially at the access/home level.

Local Area Network (and in some cases also Metropolitan Area Network) connectivity is dominated by the Ethernet standard and its extensions. Most diffused versions of Ethernet include Fast Ethernet (100Mbps) and Gigabit Ethernet (1Gbps), even though the standard is evolving towards higher data rates. Gigabit Ethernet (GbE or 1 GigE) is based on the variant 1000BASE-T defined by the IEEE 802.3ab standard.

8.4.2 Cellular/WAN connectivity

Long range connectivity represents a necessary building block of the Internet of Things. Indeed, in most cases, services will be hosted in cloud data centers, or they might require access to different and distant geographically distributed areas.

Technologies to be used at this stage can be wired or wireless.

Wired long range connectivity is clearly provided by optical fibers. Optical fibers are used most often as they permit transmission over longer distances and at higher bandwidths (data transfer rates) than electrical cables.

Wireless WANs are represented by cellular networks. Most diffused cellular networks today belong to the 4G LTE standard, while 5G is being deployed.

Today's LTE networks are based on a star topology with a centralised EPC (Evolved Packet Core) for handling authentication, signalling and network traffic, with the average base station typically capable of handling signalling for up to 1000-1500 devices. Such numbers might become critical under the assumption of massive connectivity required by IoT.

The future 5G is expected to address massive machine-type communications as one of the main deployment scenarios, therefore it is expected to represent the most appropriate cellular wireless technology to use for the Internet of Things.

8.4.3 Dedicated standards

This section illustrates an overview of the standards that are specific to the Internet of Things domain. Such standards are typically dedicated to low-power communications and WLAN/WPAN scenarios, and they cover layers 1 (physical layer) and 2 (link layer) of the TCP/IP protocol stack.

8.4.3.1 IEEE 802.15.4

IEEE 802.15.4 [74] is a technical standard maintained by the IEEE 802.15 working. It specifies the physical layer and media access control for operation of low-rate wireless personal area networks (LR-WPANs). IEEE 802.15.4 is the basis for other standards, such as Zigbee, ISA100.11a, WirelessHART, MiWi, 6LoWPAN, Thread and SNAP specifications, each of which further extends the standard by developing the upper layers (which are not in the scope of

IEEE 802.15.4). As an example, 6LoWPAN defines a binding for the IPv6 version of the Internet Protocol (IP) over WPANs.

The basic IEEE 802.15.4 specification focuses on a 10-meter communications range with a transfer rate of 250 kbit/s. Tradeoffs are possible to balance performance against power consumption, through the definition of multiple alternatives at the physical layer. Lower transfer rates of 20 and 40 kbit/s were initially defined, with the 100 kbit/s rate being added in the current revision.

Relevant features of IEEE 802.15.4 include: the implementation of random access with collision avoidance through CSMA/CA, available of real-time traffic management by reservation of Guaranteed Time Slots (GTS), and integrated support for secure communications. Devices also include power management functions related to link quality and energy measurements. The standard can support time and rate sensitive applications because of its ability to operate alternatively in pure CSMA/CA or TDMA access modes. The TDMA mode of operation is supported via the GTS feature of the standard.

IEEE 802.15.4-compatible devices may use three possible frequency bands of operation (868 MHz / 915 MHz / 2450 MHz).

8.4.3.2 M-PHY

MIPI (Mobile Industry Processor Interface) Alliance is a global business alliance that develops technical specifications for the mobile ecosystem, especially smart phones.

MIPI specifications address only the interface technology, focusing on signaling characteristics and protocols. Indeed, MIPI standards do not address entire application processors or peripherals. The main purpose of MIPI is to enable products to share common interfaces, thus facilitating system integration.

Because MIPI specifications address only the interface requirements of application processor and peripherals, MIPI compliant products are applicable to all network technologies, including GSM, CDMA2000, WCDMA, PHS, TD-SCDMA, and others. In this sense, MIPI is agnostic to air interface or wireless telecommunication standards.

M-PHY [75] is a high speed data communications physical layer protocol standard developed by the MIPI Alliance - PHY Working group, and targeted at the needs of mobile multimedia

devices. The specification's details are proprietary to MIPI member organizations. M-PHY was incorporated into several industry standards, such as Mobile PCI Express, Universal Flash Storage, and as the physical layer for SuperSpeed Inter-Chip USB.

M-PHY supports a broad range of signaling speeds, ranging from 10 kbit/s to over 11.6 Gbit/s, by using two different major signaling/speed modes, a simple low-speed (using PWM) mode and high speed one (using 8b10b). Communication is implemented in bursts, and the design of both high-speed and low-speed modes allows for extended periods of idle communications at low-power, making the design particularly suitable for mobile devices.

8.4.3.3 UniPro

UniPro (or Unified Protocol) [76] is a high-speed interface technology for interconnecting integrated circuits in mobile and mobile-influenced electronics, created within the MIPI Alliance (see section 8.4.3.2).

UniPro aims to provide high-speed data communication (gigabits/second), low-power operation (low swing signaling, standby modes), low pin count (serial signaling, multiplexing), small silicon area (small packet sizes), data reliability (differential signaling, error recovery) and robustness (proven networking concepts, including congestion management).

UniPro version 1.6 concentrates on enabling high-speed point to point communication between chips in mobile electronics. UniPro supports networks consisting of up to 128 UniPro devices (integrated circuit, modules, etc.). Couples of UniPro devices are interconnected via so-called “links” while data packets are routed toward their destination by UniPro switches. These switches operate in a similar way with respect to the routers used in wired LAN based on gigabit Ethernet. However, UniPro technology was designed to connect chips within a mobile terminal, and not hosts across a network.

The UniPro protocol stack follows the classical OSI or TCP/IP reference layered architecture. OSI's Physical Layer is split into two sub-layers: Layer 1 (the actual physical layer) and Layer 1.5 (the PHY Adapter layer) which abstracts from differences between alternative Layer 1 technologies.

Layer #		Layer name	Functionality	Data unit name
LA		Application	Payload and transaction semantics	Message
DME	Layer 4	Transport	Ports, multiplexing, flow control	Segment
	Layer 3	Network	Addressing, routing	Packet
	Layer 2	Data link	Single-hop reliability and priority-based arbitration	Frame
	Layer 1.5	PHY adapter	Physical layer abstraction and multi-lane support	UniPro symbol
Layer 1		Physical layer (PHY)	Signaling, clocking, line encoding, power modes	PHY symbol

Figure 8-6: UniPro Protocol Stack (Source: Wikipedia).

The UniPro specification itself covers Layers 1.5, 2, 3, 4 and the DME (Device Management Entity). The Application Layer (LA) is out of scope, while he Physical Layer (L1) is covered in separate MIPI specifications in order to allow the PHY to be reused by other (less generic) protocols.

8.4.3.4 SPMI

The System Power Management Interface (SPMI) is another standard specified by the MIPI Alliance. SPMI is a high-speed, low-latency, bi-directional, two-wire serial bus suitable for real-time control of voltage and frequency scaled multi-core application processors and its power management of auxiliary components.

8.4.3.5 SuperSpeed USB Inter-Chip (SSIC)

InterChip USB is an addendum to the USB Implementer Forum's USB 2.0 specification. This standard is identified by multiple names and acronyms, including IC-USB, USB-IC, Inter-chip USB, or High-Speed Inter-Chip (HSIC). The USB 3.0 successor of HSIC is called SuperSpeed Inter-Chip (SSIC).

IC-USB is intended as a low-power variant of the standard physical USB interface, aimed at direct chip-to-chip communications, with a maximum length of 10 cm for reduced power requirements.

IC-USB is being used primarily in embedded systems and standards, especially on mobile phones where, for instance, ETSI (in specification TS 102 600) has standardized on IC-USB as the official high-speed interface for connections between the phone's main chipset and the SIM card or UICC card.

USB 2.0 High-Speed Inter-Chip (HSIC) is a chip-to-chip variant of USB 2.0 that eliminates the conventional analog transceivers found in normal USB, and it uses about 50% less power and 75% less board area compared to traditional USB 2.0. HSIC uses two signals at 1.2 V and provides a throughput of 480 Mbit/s.

8.4.3.6 Mobile PCIe (M-PCIe)

PCI Express (Peripheral Component Interconnect Express) is a high-speed serial computer expansion bus standard, designed to replace the older PCI, PCI-X and AGP bus standards. PCIe (or PCI-e) is the common motherboard interface for personal computers' graphics cards, hard drives, Solid State Drives, Wi-Fi and Ethernet hardware connections.

Mobile PCIe specification (abbreviated to M-PCIe) [77] allows PCI Express architecture to operate over the MIPI Alliance's M-PHY physical layer technology, with the target to enable mobile devices to use PCI Express.

8.4.3.7 SPI

The Serial Peripheral Interface (SPI) is a synchronous serial communication interface specification used for short-distance communication, primarily in embedded systems. Typical applications include Secure Digital cards and liquid crystal displays.

SPI devices communicate in full duplex mode using a master-slave architecture with a single master. The master device manages access to the transmission medium for reading and writing. Multiple slave-devices are supported through selection with individual slave select (SS) lines.

8.4.3.8 MODBUS

Modbus [78] is a data communications protocol originally published by Modicon (now Schneider Electric) in 1979 for use with its programmable logic controllers (PLCs). It is extremely diffused in industrial environments because it is openly published and royalty-free, becoming a de facto standard communication protocol and is now a commonly available means of connecting industrial electronic devices. Modbus uses the RS485 or Ethernet physical layers. Modbus supports communication to and from multiple devices connected to the same cable or Ethernet network, e.g., enabling a device that measures temperature and a different device to measure humidity to communicate with a computer.

The typical application of Modbus is in Supervisory Control and Data Acquisition (SCADA) systems in the electric power industry, where Modbus is used to connect a plant/system supervisory computer with a remote terminal unit (RTU).

8.4.3.9 OBD

On-board diagnostics (OBD) is an automotive term referring to a vehicle's self-diagnostic and reporting capability, as well as a specification for communication with the vehicle on-board devices. OBD systems provides the vehicle owner or repair technician access to the status of the various vehicle sub-systems. The amount of diagnostic information available via OBD has varied widely since its introduction in the early 1980s versions of on-board vehicle computers.

Modern OBD implementations use a standardized digital communications port to provide real-time data in addition to a standardized series of diagnostic trouble codes, or DTCs, which allow a person to rapidly identify and remedy malfunctions within the vehicle.

8.5 Test cases for Connecting the Internet of Things

There are several options to connect devices to the Internet of Things. The selection of the proper method is based on the types of devices, protocols, and existing environment.

The following diagram and table describe the available options to connect IoT devices to the Internet of Things based on different scenarios and protocols.

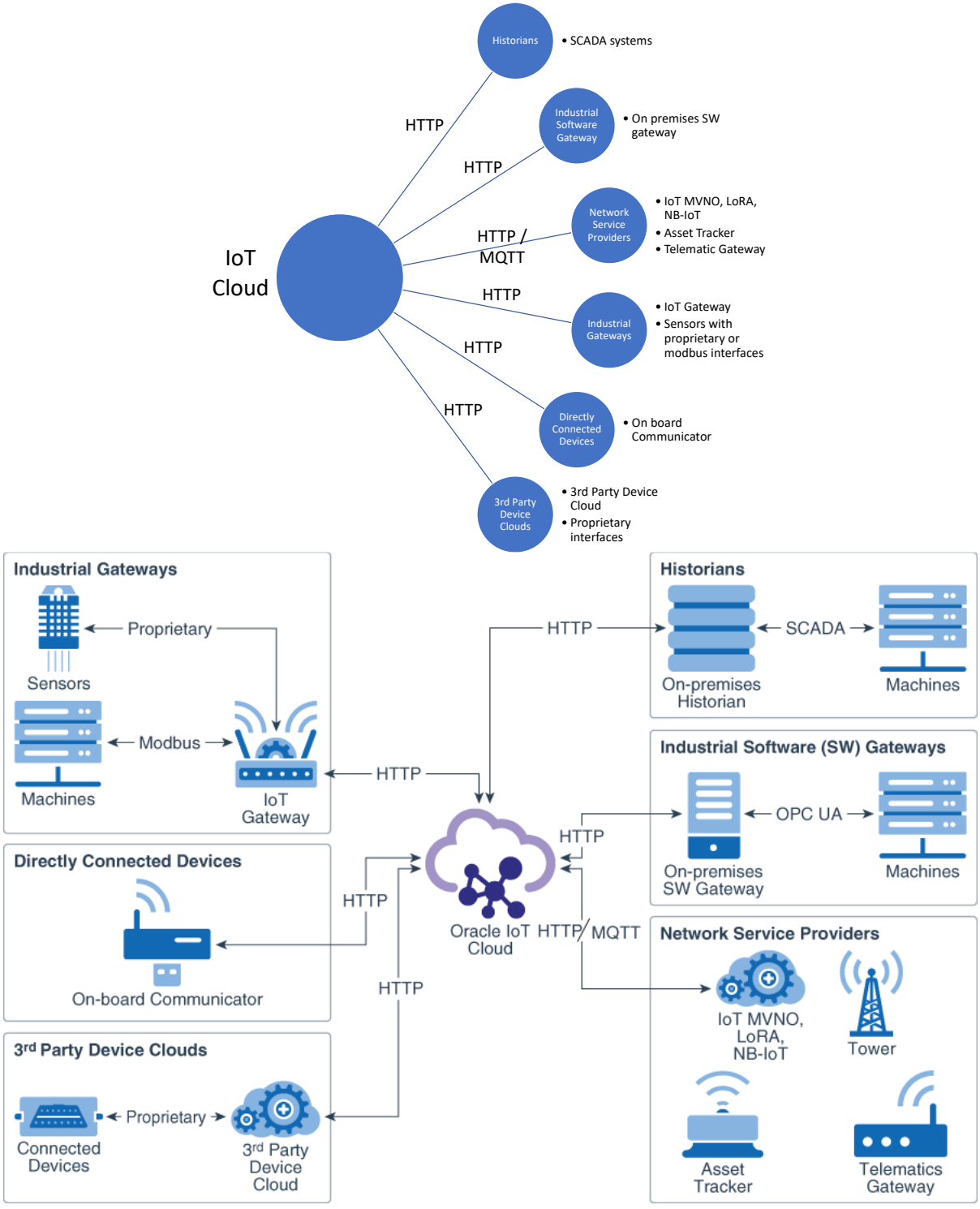


Figure 8-7: IoT connection scenarios (based on the Oracle IoT Cloud Service [79] concept).

Scenario	Option to Connect the Devices	Example
Devices that support proprietary protocols.	Sensors and machines connect indirectly to Internet of Things Cloud Service through a standardized industrial gateway using proprietary protocols.	A device uses the MODBUS protocol to connect with an industrial IoT gateway which, in turn, connects to Internet of Things Cloud Service using the HTTP protocol
Machine with a powerful in-built sensor that supports HTTP protocol.	Directly connect with Oracle Internet of Things Cloud Service using HTTP.	A device such as an on-board communicator directly exchanges messages with the Internet of Things Cloud Service by using the HTTP protocol.
Devices that are already connected to an existing third-party cloud service using proprietary protocols.	Third-party clouds can connect with the Internet of Things Cloud Service using HTTP. This results in the devices getting indirectly connected to Internet of Things Cloud Service.	A device such as an OBD II data logger connects to a third-party cloud service using its proprietary protocol. The third-party device cloud connects to Internet of Things Cloud Service using the HTTP protocol.
Devices and gateways that are already connected to existing network providers that provide wireless connectivity to the devices.	The network providers can transmit the device data to Internet of Things Cloud Service using HTTP or MQTT.	Network providers use LoRA, NB-IoT, or IoT MVNO for wireless connectivity with devices such as an asset tracker or a telematics gateway. The network provider can transmit the device messages to Internet of Things Cloud Service using HTTP or MQTT.
Existing database with device data on events, time stamps, and alarms.	The on-premises database can connect to Internet of Things Cloud Service using HTTP and transfer the device messages.	Machines using the Historian service save and store messages in a database or multiple databases over the supervisory control and data acquisition

Scenario	Option to Connect the Devices	Example
		(SCADA) network. An on-premises Historian service connects to Internet of Things Cloud Service and provides the machine data for analysis.
Existing on-premises industrial gateway software	On-premises industrial gateway software applications manage machine data. The Internet of Things Cloud Service can connect to these gateway applications using HTTP to obtain the machine data.	Machines that use OPC Unified Architecture (UA), a machine-to-machine protocol to transfer machine data to industrial gateway software and that, in turn, can connect to the Internet of Things Cloud Service and send device messages received from the machines.

9 Data Storage and Cloud Systems

Author(s): **Fabrizio Granelli**

9.1 Introduction

Cloud computing and data storage systems represent extremely relevant services to support modern Internet of Things applications. Indeed, IoT and cloud are nowadays two related Internet technologies, which go hand-in-hand in IoT deployments. Typically, cloud computing represents a key component for the development and deployment of scalable Internet-of-Things business models and applications.

Several modern massive IoT ecosystems are based on the cloud support, as it will be described in the next sub-section of this part of the course. Nevertheless, before describing how IoT and cloud computing can be integrated, we will introduce some basic cloud computing concepts.

Most modern IoT ecosystems are cloud-based, as it will be described in the next sections of this course. The purpose of the next section is to briefly discuss the main concepts within the cloud computing paradigm as well as the integration between IoT and cloud computing architectures.

9.2 Cloud Computing Basics

Cloud computing provided a big step forward in the delivery of ICT resources and services, and it represented conceptually an evolution of the well-known client-server paradigm. In cluod computing, ICT services and infrastructure can be delivered as services, including computing power, computing infrastructure (e.g., servers and/or storage resources), applications, business processes and more. This is allowed by the availability of a powerful computational and communications infrastructure in the form of datacenters, and the implementation of proper virtualization technologies to enable flexibility and scalability in the management of such huge amounts of resources.

Cloud services can be offered through infrastructures (clouds) that are publicly accessible (i.e. public cloud services), but also by privately owned infrastructures (i.e. private cloud services). Furthermore, it is possible to offer services supporting by both public and private clouds, which are characterized as hybrid cloud services.

Cloud computing infrastructures and services have the following characteristics:

- Elasticity and scalability: The elastic nature of the cloud computing paradigm facilitates the implementation of flexibly scalable business models, e.g., enabling

enterprises to use more or less resources as their business grows or shrinks. Indeed, cloud computing services can automatically be scaled upwards during high periods of demand and downward during periods of lighter demand to adapt the service needs.

- Self-service provisioning and automatic deprovisioning: Cloud computing provides easy access to cloud services without a lengthy and complex provisioning process. In cloud computing, both provisioning and de-provisioning of resources are represented by automated functionalities due to the implementation of virtualization and abstraction methodologies.
- Application programming interfaces (APIs): Cloud services are accessible via high level APIs, which enable and facilitate applications and data sources to communicate with each other.
- Billing and metering of service usage in a pay-as-you-go model: Cloud services are associated with a utility-based pay-as-you-go model. To this end, they provide the means for metering resource usage and subsequently issuing bills.
- Performance monitoring and measuring: Cloud computing infrastructures provide a service management environment for enabling transparent and flexible control on the system performance and collecting information about its state.
- Security: Cloud computing infrastructures offer security functionalities towards safeguarding critical data and fulfilling customers' compliance requirements.

Depending on the types of resources that are accessed as a service, cloud computing is associated with different service delivery models (Figure 9-1):

- Infrastructure as a Service (IaaS): IaaS deals with the delivery of storage and computing resources towards supporting custom business solutions. In this case, the user is the owner of the service and it has the capability to set up its operating environment based on his/her needs. The most relevant example of IaaS service Amazon's Elastic Compute Cloud (EC2), which uses the Xen open-source hypervisor to create and manage virtual machines.
- Platform as a Service (PaaS): PaaS provides development environments for creating cloud-ready business applications. It provides a deeper set of capabilities comparing to IaaS, including development, middleware, and deployment capabilities. PaaS enables direct deployment of applications on a standardized set of interfaces and

services. Typical examples of PaaS services are Google's App Engine and Microsoft's Azure cloud environment, which both provide a workflow engine, development tools, a testing environment, database integration functionalities, as well as third-party tools and services.

- Software as a Service (SaaS): SaaS services enable access to purpose-built business applications in the cloud. In this case, the typical user is interested only in accessing a specific service, which is delivered on the Internet with the pay-as-you-go model, it enables to reduce CAPEX and it is supported by the elastic properties of cloud computing infrastructures.

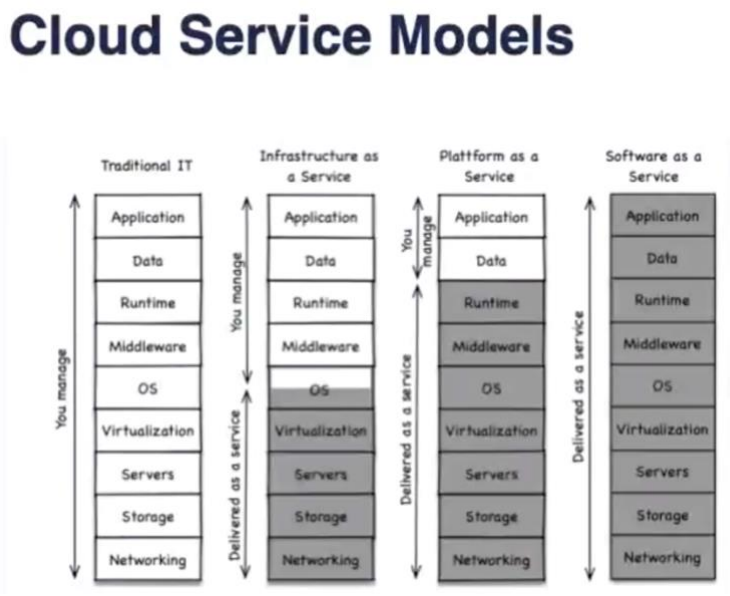


Figure 9-1: Cloud computing service models.

9.3 Processing for the Internet of Things Services

Summarizing what we discussed in the previous sections of this course, we can divide an IoT service into three basic components, the device, gateway, and cloud:

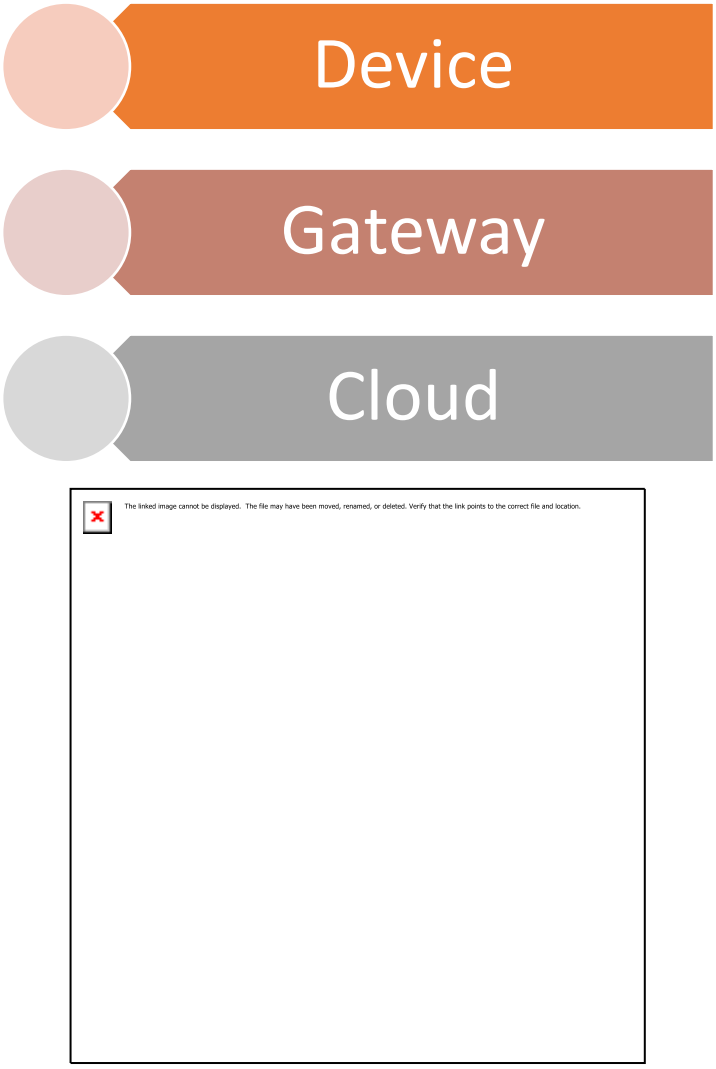


Figure 9-2: Simplified IoT service model (based on a concept by Google).

The *device* component represents hardware and software that directly interact with the world, i.e. the sensor or actuator. Devices need to connect to a network to communicate - either with each other or to centralized applications. In this scenario, devices might be directly or indirectly connected to the Internet (i.e. connected through a gateway).

A gateway enables devices that are not directly connected to the Internet to reach cloud services. The reader should note that even though the term *gateway* in the context of networking describes a specific function, in the context of IoT the term might also be used to describe a device that processes data on behalf of a group or cluster of devices. Ideally, once the data from each device is sent to the *cloud*, the cloud servers process and combine it with data from other devices, and potentially with other business-related data.

Those three components represent different location where to perform processing and storage of data.

9.3.1 On-device Processing

Besides collecting data from a sensor, an IoT device can provide data processing functionality before sending the data to the cloud. This procedure is called on-device processing. Following this paradigm, multiple devices might handle and process the data before it gets to the cloud, therefore already introducing some amount of processing.

The following diagram illustrates sample on-device processing tasks.

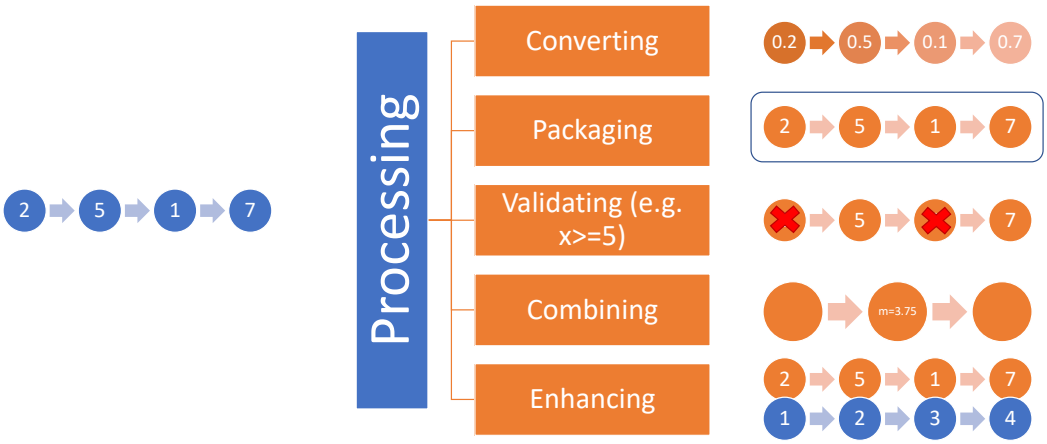


Figure 9-3: On-device sample processing operations.

Processing might include the following functionalities:

- Converting data between different formats
- Packaging and organizing data in a way that's secure and combines the data into a practical batch
- Validating data to ensure it complies with a set of rules
- Sorting data to create a requested sequence
- Enhancing data to decorate the core value with additional related information
- Summarizing data to reduce the volume and eliminate unneeded or unwanted detail
- Combining data into aggregate values
- On-device analysis can combine multiple processing tasks to provide an intermediate, synthesized interpretation that enables more information to be transmitted in a smaller data footprint.

9.3.2 Gateway Processing

An IoT gateway device manages traffic between networks that use different protocols, it is responsible for protocol translation and other interoperability tasks, and it can be employed to provide the connection and translation between devices and the cloud.

As it was discussed earlier in the course, since some devices might not contain the network stack required for Internet connectivity, a gateway device acts as a proxy, receiving data from devices and packaging it for transmission over TCP/IP.

A dedicated gateway device might be a requirement in case of absence of connectivity to the Internet, lack of support for transport-layer security (TLS) communications, or even by power limitations.

There are setups in which a gateway device might be used even in the case IoT devices are capable of communicating without its direct need. In this situation, the added value provided by the presence of the gateway might be to provide pre-processing of the data across multiple devices before sending it to the cloud.

In general, the following tasks can be re-allocated (or delegated) to a gateway device:

- Compressing/analyzing data to maximize the amount of useful information that can be sent to the cloud over a single bandwidth-limited link
- Storing data in a local database, and then forwarding it on in case the connection to cloud is not always available
- Providing a real-time clock, with a battery backup, used to provide a consistent timestamp for devices that can't manage timestamps well or keep them well synchronized
- Performing IPV6 to IPV4 translation
- Collecting other flat-file-based data from the local network that is relevant and associated with the IoT data
- Acting as a local cache for firmware or other software updates

9.3.3 Cloud Processing

After an IoT project is up and running, many devices will be producing lots of data. Those require an efficient, scalable, affordable way to both manage those devices and handle all that information. When it comes to storing, processing, and analyzing data, especially big data, the cloud represents the most appropriate location.

In order to provide an example on the services provided by the cloud for IoT, the following diagram shows the various stages of IoT data management in the Cloud, focusing on the specific case of the Google Cloud Platform.

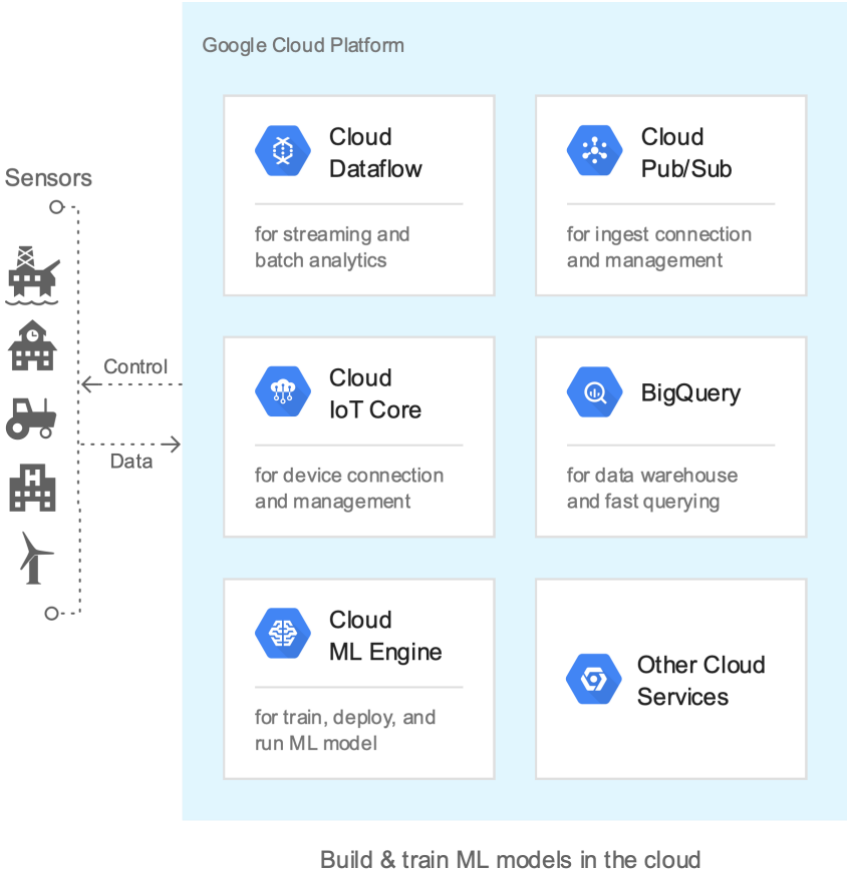


Figure 9-4: Cloud processing operations in the case of Google Cloud (Source: Google [80]).

9.3.3.1 Dataflow

Dataflow is developed for managing the high-volume data processing pipelines required for IoT scenarios, as it provides a set of functionalities to enable processing data in multiple ways, including batch operations, extract-transform-load (ETL) patterns, and continuous, streaming computation.

9.3.3.2 IoT Core

The cloud will typically offer a native service for collecting and managing IoT data. This functionality is sometimes defined as Cloud IoT Core service [81].

The Cloud IoT Core provides a secure MQTT (Message Queue Telemetry Transport) broker for devices managed by the IoT Core. As previously discussed in this course, this efficient binary industry standard allows for constrained devices to send real-time telemetry as well as immediately receive messages sent from cloud to device by using the configuration management feature. The IoT Core MQTT broker directly connects with the Pub/Sub service.

9.3.3.3 Publish/Subscribe Services

Publish/Subscribe services (Pub/Sub) provide a global framework for managing message exchange among the components of an IoT service. Publish/subscribe service operate through the creation of *topics* for streams or channels, and to enable different components of the IoT application to *subscribe* to specific streams of data. The advantage of this approach is that subscription is performed without explicitly needing to construct subscriber-specific channels on each device. Pub/Sub services can also be used to connect to other cloud services, helping to connect ingestion, data pipelines, and storage systems.

Another important feature of Pub/Sub services is that they can act like a "cushion" and rate leveller for both incoming data streams and application architecture changes. Pub/Sub scales to handle data spikes that can occur when swarms of devices respond to events in the physical world, and it buffers these spikes to help isolate them from applications monitoring the data.

9.3.3.4 Cloud Monitoring and Cloud Logging

Cloud Monitoring and Cloud Logging are two services that enable service providers to control the operating infrastructure. The corresponding information is provided by such services through their dedicated interfaces.

9.3.3.5 Pipeline processing tasks

Pipelines implement a data management paradigm similar to how parts are managed on a factory line. In particular, after data arrives in the Cloud, they provide the following services:

- **Transforming data.** These services is used to convert the data into another format, for example, converting a captured device signal voltage to a calibrated unit measure of temperature.
- **Aggregating data and computing.** Aggregation enables to manage large amounts of data in a unified way, for example enabling to detect outliers or erroneous samples. By adding computation to your pipeline, it is possible to apply streaming analytics to data while it is still in the processing pipeline.
- **Enriching data.** Some pipeline tasks can be used to combine device-generated data with other metadata about the device, or with other datasets, such as weather or traffic data, for use in subsequent analysis.
- **Moving data.** Those services allow to store the processed data in one or more final storage locations.

9.3.3.6 Analytics

In several cases, performing analytics on data obtained through IoT sources the final purpose of deploying IoT in the physical world. This requires proper analytics services capable of analyzing streamed data in a processing pipeline, while the data accumulates. Over time, such data provides a rich source of information for looking at trends, and it can be combined with other data, including data from sources outside of the IoT devices.

9.4 Storage

It is worthy to dedicate a separate sub-section to the storage of IoT data, and in particular to the existing databases that might be used to efficiently store and operate IoT data.

Indeed, managing and organizing access to a massive amount of data represents one of the key issues in the design of IoT applications. IoT is about data, services, connectivity: data are gathered by objects, transferred, analyzed, and translated into services. In such scenario, large-scale IoT deployments can produce huge amounts of data, leading to the concept of *big data*.

The word *big-data* is currently used to identify: (i) data sources with specific characteristics, as well as (ii) novel technologies to manage huge amounts of data.

In facts, big data cannot be managed using conventional technologies of information systems, because of its main characteristics:

- Volume: in the order of Petabytes
- Velocity: data is produced at high rate
- Variety: big data is heterogeneous (text, image, video, etc)
- Value: valuable information is stored within such huge “container” and it can be extracted using proper IT techniques

In terms of storage and processing, we might consider IoT and big data highly related concepts. Therefore, IoT might use big data storage concepts in order to address this issue.

An example of big data IoT application can be the Energy@Home project [82] by Telecom Italia, which was aimed at collecting data by smart plugs installed in customers’ houses for implementing data classification (i.e. identify the appliance consuming power), profiling (i.e. identify users’ habits), prediction (i.e. predict energy consumption) and scheduling (i.e. intelligent ON/OFF schedule).

In general, the unifying characteristic of IoT data is that they represent a time-series: a sequence of timestamp plus values. This has the following implications:

- Data are immutable.
- Writing data is done by “appending” to previous values.
- Reading is performed for contiguous sequences of samples data.
- Data is highly compressible.
- Deleting usually happens across large time period.
- High precision might be desired for a short period of time, but single values are not so important.

The following sub-sections will provide some databases models which might be used to store IoT data.

9.4.1 SQL Databases

Relational Database Management Systems (RDBMS) are a family of databases based on a relational model. They are also called SQL databases, since they employ the SQL querying language.

A relational database is a scheme-based (structured) database, with the following components: tables (relations), primary keys, foreign keys, NULL values.

A time series implemented in a relational database would lead to a structure such as the one described in the following table.

Time	Time series ID	Value
16:17:00	101	10.1
16:17:05	102	0.5
16:17:10	103	2.3
16:17:15	104	1.2

The main problems of SQL databases for IoT are:

- Scalability (i.e. need to store large amount of time-series data)
- Performance (e.g. support for range-based operations)

9.4.2 NoSQL Databases

An alternative is to use not relational databases, or NoSQL Database Management Systems. NoSQL databases represent a set of tools and logic models, alternative or complementary to RDBMS. They do not employ the SQL language and provide a scheme-less (un/semi-structured) database.

There are different families of not relational databases, including: Key-values DB, Document-based DB, Column-based DB, Graph-based DB [83].

NoSQL databases have achieved a large popularity thanks to their high performance, flexibility in scaling, and high availability. In this framework, the most popular NoSQL databases for IoT services are: Redis, Cassandra, MongoDb, Couchbase and Neo4j.

9.4.3 Time Series Databases

Time-series Databases are dedicated DBMS optimized for managing large volumes of time-series data. They provide:

- Optimized data storage and sharding
- Operational support (e.g. range-based queries)
- Time-granularity management
- Time-series analytics and mining

Time series databases are gaining popularity in the last months, most likely because of the need of IoT for dedicated databases (Figure 9-5).

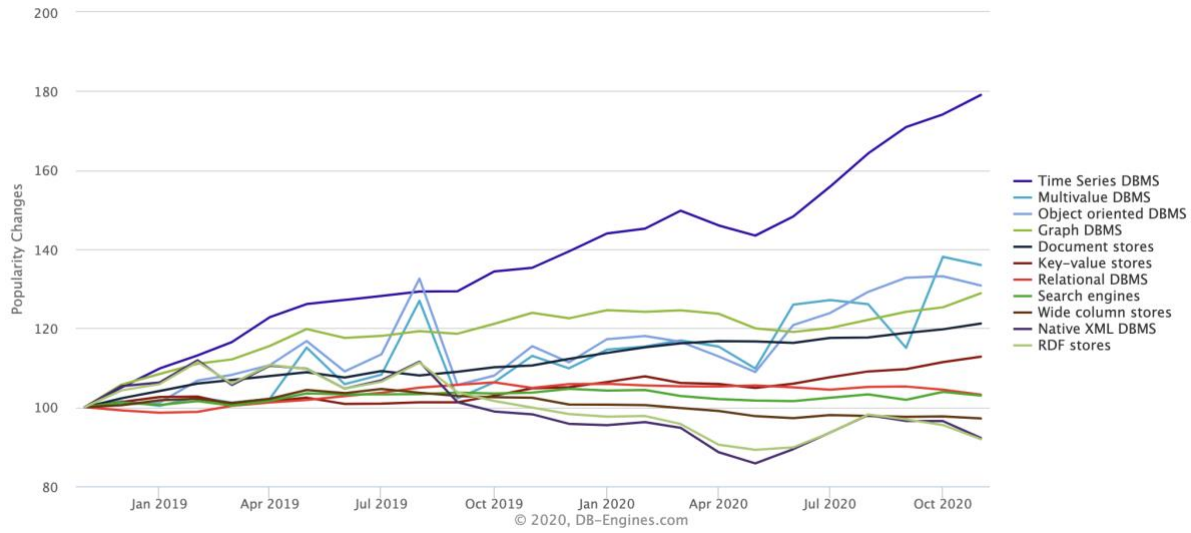


Figure 9-5: Database popularity changes in the last months (Source: [84])

As an example of a popular solution in this area, InfluxDB [85] is an open-source time-series database (InfluxData) that supports an SQL-like query language (InfluxQL) (<1.8) and a JS-like query language (Flux) (>=1.8). It provides GUI, Command Line Interface (CLI) and HTTP APIs and it supports distributed deployments. Moreover, it eases integration with time-series tools for data acquisition, analytics and visualization (e.g. Telegraf, Grafana).

InfluxDB is based on the following parameters/fields:

- Time-Structured Merge Tree (TSM): data structure used to contain sorted, compressed series data.
- Time Series Index (TSI): it can address millions of unique time series, regardless of the amount of memory on the server hardware.
- Automatic downsampling and data retention procedures.
- Timestamp: in RFC3339 UTC format (yyyy-mm-ddThh:mm:ssZ)
- Field keys: string metadata, similar to column name
- Field values: actual measured data
- Tag-sets: optimal, extra-information about the measurements
    - Tag keys: string meta-data, similar to field keys
    - Tag values: string values
- Measurement: Container to hold the timestamps, fields and tags (similar to a table in a RDBMS)
- Buckets: collection of data-points, containing:
    - Measurement
    - Tag-sets
    - Retention policy: period that datapoints are being stored in InfluxDB, which is called DURATION but also the number of versions that should be kept on the cluster, as REPLICATION.

10 Data Analytics and Applications

Author(s): Fabrizio Granelli

10.1 Data Analytics

Data analytics surely represents an essential feature in IoT services. Nevertheless, probably due to its novelty, there is no specific methodology to solve this problems of Data Science for IoT. On one hand, a Data Science for IoT problem is a typical Data Science problem. On the other hand, there are some unique features that makes IoT services different – for example in the use of hardware, high data volumes, impact of verticals (see later), streaming data etc.

In principle, the development of a proper methodology for IoT analytics (Data Science for IoT) should include the unique aspects of the different steps in "traditional" Data Science. The choice of the model family (ANN, SVM, Trees, etc.) represents only one of several choices to make, that might include [86]:

a. Choice of the model structure - optimisation methodology (Bootstrap, etc)
b. Choice of the model parameter optimisation algorithm (joint gradients vs. conjugate gradients)
c. Pre-processing of the data (reduction, functional reduction, log-transform, etc.)
d. How to deal with missing data (case deletion, interpolation, etc.)
e. How to detect and deal with suspect data (distance-based outlier detection, density-based, etc.)
f. How to choose relevant features (filters, wrappers, etc.)
g. How to measure prediction performance (mean square error, mean absolute error, misclassification rate, precision/recall, etc.)

Moreover, the selected methodology should consider or incorporate the unique aspects of the IoT scenario:

a. Complex event processing: IoT is typically used to analyze and perform actions in a complex environment, characterized by multiple variables and events with typically unknown relationships.
b. Need for deep learning approaches: the amount and complexity of the collected data requires the deployment of deep learning methodologies to enable data processing and extraction of useful information.

c. Anomaly Detection: What is the triggering event, how much has the machine deviated from the plan, are there any external factors affecting the system performance, how do I know that I should trust IoT data? Is there a recommended plan of action? How is the Data visualized? Does the Data have missing elements? How do we detect failure in other processes?
d. Specific IoT vertical domains (such as Smart Grid, Smart Cities, Smart Energy, Automotive, Smart factory, etc.) might introduce additional choices or constraints on the data analytics process.

10.2 Interpretation of IoT Data

Interpretation of the data is an extremely relevant issue in IoT applications. Indeed, IoT applications will typically generate huge amounts of data, where it is necessary to identify anomaly or extract information to make the application significant and useful to the clients.

Kalman filters and sensor fusion are common methodologies that can be used to enable interpretation of IoT data.

Kalman filtering is a data filtering algorithm used to produce estimates of unknown variables more precise than those based on a single measurement. To perform this task, a Kalman filter uses a series of measurements observed over time, containing statistical noise and other inaccuracies. The Kalman filter has numerous applications in technology – including IoT.

In the IoT scenario, Kalman filters are often used in Sensor fusion. Sensor fusion helps to determine the state (and also the overall context) of an IoT based computing system based on distributed measurements by several sensors.

Kalman filters can be introduced using an example:

Let's assume you have a model that predicts river water level every hour (using the usual inputs). You know that your model is not perfect, and you don't trust it 100%. So, you want to send someone to check the river level in person. However, the river level can only be checked once a day around noon and not every hour. Furthermore, the person who measures the river level cannot be trusted 100% either. How do you combine both outputs of river level (from model and from measurement) so that you get a 'fused' and better estimate?

In general, in IoT applications, we can state the following:

- The system state cannot be measured directly
- The application can collect multiple measurements – each of which might not be 100% accurate. Then, those need to be fused/combined.
- Fusion across different sensors can mean different things: example: several temperature sensors, fusion across different Attributes, (example: temperature, pressure humidity to determine air refractive index), fusion across different domains, example: different ranges / domains, fusion across different time (Example: Sampling over space and time)

The reader should note at this stage that sensor fusion does not merely imply 'adding' values, i.e. not just adding temperature values. On the opposite, sensor fusion aims at about understanding or estimating the overall 'State' of a system based on multiple sensors.

Kalman filters are typically used to predict the state of dynamic systems. Dynamic systems are systems that change or evolve in time according to a fixed rule. For many physical systems, This behaviour is described by a set of first-order differential equations. This set of input, output and state variables related by first-order differential equations is called a state-space representation of a physical system.

In dynamic systems, it is possible to define a minimum set of variables, defined as state variables, that can fully describe the system and its response to any given set of inputs. In a state-determined system, given such minimum set of variables $x_i(t)$, i = 1,…,n, together with knowledge of those variables at an initial time t_0 and the system inputs for time $t \geq t_0$, it is possible to predict the future system state and estimate the outputs for all time $t>t_0$. Several engineering, biological, social and economic systems can be represented by state-determined system models, including Internet of Things applications.

In some cases, we might be interested in estimating the covariance (or correlation) between the two measurements. If the greater values of one variable mainly correspond with the greater values of the other variable, and the same holds for the smaller values, i.e., the variables tend to show similar behaviour, and the corresponding covariance is positive. In the opposite case, when the greater values of one variable mainly correspond to the smaller values of the other, i.e., the variables tend to show opposite behaviour, and the related

covariance is negative. The sign of the covariance therefore shows the tendency in the linear relationship between the variables.

To summarise, Kalman filters can be used to determine context for IoT systems in the following way:

- Sensor measurements can generate fragments of context information with varying degrees of confidence. Such values are fed into the overall context estimation of the state. Sensors are highly distributed, and their individual performance varies, so it is likely to experience overlaps and conflicts among such measurements.
- In the typical situation, sensed information from individual sensors will include the sensor's confidence level and timestamp.
- The Kalman filter averages a prediction of a system's state with a new measurement using a weighted average. In this computation, values with smaller estimated uncertainty are trusted more. The relative weights between the sensor inputs and their impact on the overall state are calculated using covariance. This enables the estimation of a new state of the system.
- This process is repeated every time step, with a new estimate and its covariance influencing the prediction produced at the following iteration.

10.3 Visualization of Data

Efficient visualization of data might be relevant in an IoT scenario for the following reasons:

- It helps to make real-time decisions by displaying a combination of multiple data sources into a single insightful dashboard with multi-layered visual data.
- It combines the updated IoT data transmitted from data sensors with the existing data to analyze and lead to the identification of new business opportunities.
- It enable better support to monitor IoT devices and infrastructure for a clearer interpretation of the overall IoT data flow.
- It helps to analyse any existing multiple data correlations in real-time.

The following are some examples of data visualization tools commonly used in the IoT applications.

10.3.1 Grafana

Grafana [87] is an open-source data visualization tool, especially built for time-series data. As described in the previous chapter, time series data are measured over a duration of time. There are several time-series data storage backends where Grafana supports with specific labels, data sources.

Grafana offers a visual dashboard which covers multiple functionalities in form of a single and powerful interface. Several panels in the dashboard are responsible for analyzing the data and presents the data in different visual formats like heat maps, geo maps, histograms, charts – Pie/Bar, graphs.

Figure 10-1: The visual interface of the Grafana tool.

10.3.2 Kibana

Kibana [88] is an open-source data visualization tool for analyzing large volumes of log data. Kibana requires joint operation with Elasticsearch and Logstash, leading to the globally known ELK stack for log management.

In the Kibana workflow, Logstash is responsible to collect all the data from the remote sources. Then, these data logs are then pushed and sent to the Elasticsearch. Elasticsearch acts as the database to the Kibana tool with all the log information. Finally, Kibana tool presents these log data in the form of pie charts, bar or line graphs to the user.

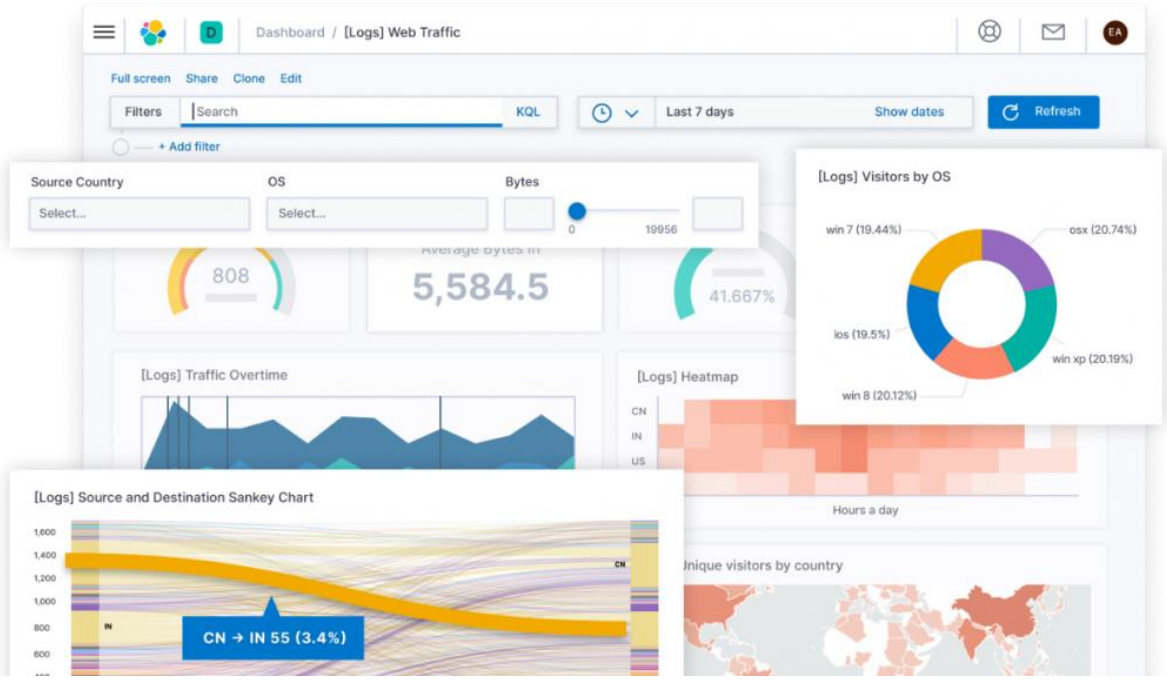


Figure 10-2: The visual interface of the Kibana tool.

10.3.3 Power BI

Microsoft PowerBI [89] is a popular Business Intelligence Tool that provides a detailed analysis reports for large Enterprises. Power BI comes with a suite of products with Power BI desktop, mobile Power BI apps and Power BI services for SaaS.

The first step consists of data collection from the external data sources. The 'Get Data' option allows to collect information from various sources including Facebook, Google Analytics, Azure Cloud, Salesforce etc. It provides ODBC connection to get ODBC data as well.

In Power BI, there are 2 ways to a visualization: (1) by adding from the right-side panel to the report canvas which is in a table type visualization format, or (2) by simple drag and drop of value axis under visualization. Once the report is developed, it can be published to web portal with the help of the Power BI service and then exported in pdf, excel or any preferred format.

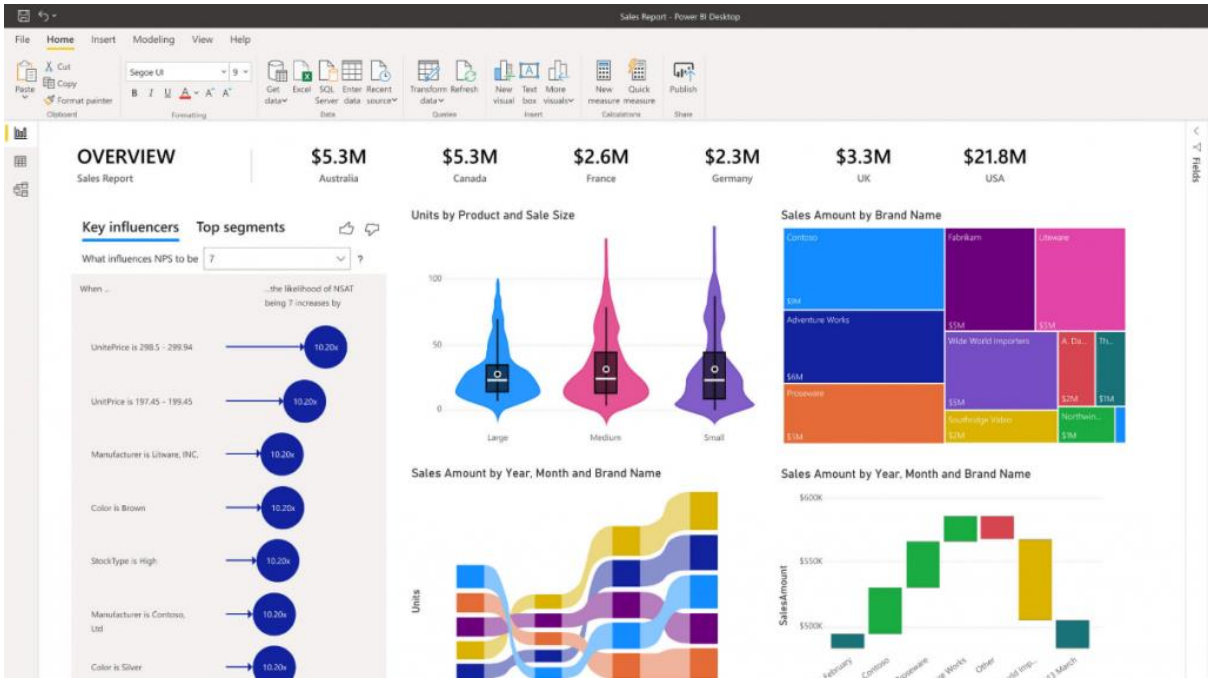


Figure 10-3: The visual interface of the Power BI tool.

10.4 A case study of a simple sensor, broker, app application deployment

As an example of IoT, let's consider a simple scenario. In this case, we have one or many sensors, an MQTT broker and one or more applications. Sensors can measure any relevant information from the environment (e.g. temperature, humidity, etc.) and provide data in input to the broker. IoT applications subscribe to the data flows of interest and then analyze and process the corresponding streams of data.

The MQTT Broker collects data from the sensors (publishers) and provides them to subscribers. Subscribers can subscribe to multiple topics and receive the corresponding data flows.

MQ Telemetry Transport (MQTT) is a lightweight broker-based publish/subscribe messaging protocol designed to be open, lightweight, and easy to implement. It was designed as an extremely lightweight protocol to allow the application in remote locations with a limited Internet connection and small low power devices. These principles also turn out to make the protocol ideal of the emerging "machine-to-machine" (M2M) or "Internet of Things" world of connected devices, and for mobile applications.

The protocol runs over TCP/IP and typically on port 1883, which is assigned by the Internet Assigned Numbers Authority (IANA). For using MQTT over SSL, port 8883 is used.

A communication example, with a sensor publishing temperature over a topic and a client subscribed to that topic receiving it, is provided in Figure 10-4.

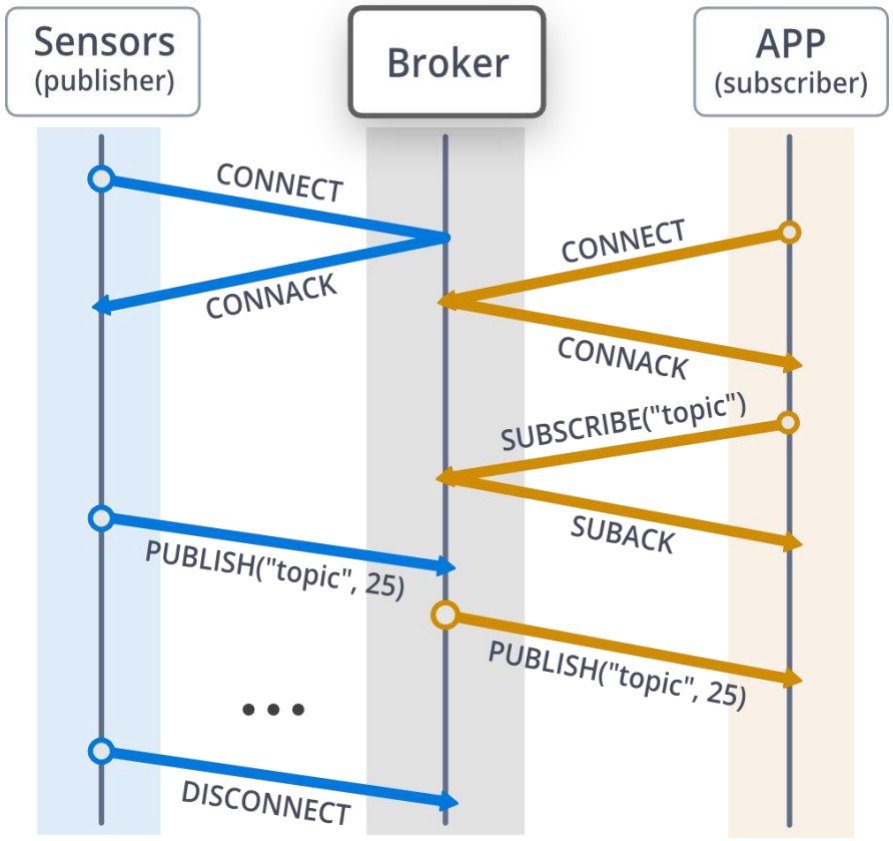


Figure 10-4: MQTT communication: A sensor connects to the broker and publishes to a topic some value. Meanwhile, an app has connected to the broker and subscribe to the same topic. So, the broker forwards the message published by the sensor to the app.

11 IoT Security and security standards

Author(s): Maria Papaioannou
Georgios Mantas
Claudia Barbosa
Jonathan Rodriguez

11.1 The Internet of Things (IoT) – An Overview

The Internet of Things constitutes the most recent advancement in the continuing revolution of computing and communication [90].

``*The Internet of Things (IoT) is a term that refers to the expanding interconnection of smart devices, ranging from appliances to tiny sensors* [90].´´ In particular, the IoT enables innovative communication forms between (i) persons and things, and (ii) things themselves, by embedding short-range mobile transceivers into multiple gadgets and everyday items. These days, the Internet, through advanced computing and the cloud systems, supports the interconnection of billions of devices - both industrial and personal objects. These objects receive, handle, and deliver sensing data (usually coming from sensor devices), act on their environment, and may alter themselves, in order to overall manage a larger environment or system, such as a factory or a town.

The IoT primarily consists of low-bandwidth, low-repetition data capture, and low-bandwidth data-usage embedded devices [90]. These devices connect and communicate with each other and deliver data via user interfaces.

11.1.1 Evolution

With reference to the end systems supported, the Internet has gone through roughly four generations of deployment culminating in the IoT. As detailed in [90]:

1. **Information technology (IT):** IT devices such as PCs, servers, routers, firewalls, bought by enterprise IT people, primarily using wired connectivity.
2. **Operational technology (OT):** Machines and electrical devices with embedded IT manufactured by non-IT companies, such as industrials machines, medical machinery, SCADA (supervisory control and data acquisition), and process control, bought by enterprise OT people, primarily using wired connectivity.
3. **Personal technology:** Personal devices such as smartphones, tablets, and eBook readers bought as IT devices by consumers/organizations, exclusively using wireless connectivity and often multiple forms of wireless connectivity.

4. **Sensor/actuator technology:** Single-purpose devices bought by consumers, IT, and OT people exclusively using wireless connectivity, for a wide of applications and as part of larger systems. The fourth generation is marked by using billions of interconnected embedded devices, and thus is typically considered as the IoT itself.

11.1.2 IoT Components

The key components of an IoT-enabled device are the following:

- **Sensor:** A sensor device is responsible for gathering information of any kind of systems, for instance, biological, physical, or chemical systems, and delivering an electronic signal (i.e., as a digital signal, or an analog voltage level) that is related to the observed parameters. Afterwards, this output – i.e., the electronic signal, is usually input to a management component like a microcontroller.
- **Actuator**: An actuator receives an input from a controller in the form of an electronic signal and responds to this input by interacting with its environment. The product of this interaction is an effect on a certain parameter of biological, physical, or chemical entity or system that the actuator is been a part of.
- **Microcontroller:** The microcontroller is a deeply embedded device and thus provides the "smart" or intelligence in a smart device.
- **Transceiver:** A transceiver consists of all the necessary electronics to transmit and receive data. The majority of IoT devices include a wireless transceiver. Typically, the transceivers are capable to communicate employing Wi-Fi, ZigBee, or other wireless technologies.
- **Radio-frequency Identification (RFID):** RFID is increasingly becoming an enabling technology for the IoT. RFID utilizes radio waves to identify objects. The tags and the readers constitute the main components of an RFID system. RFID tags may be in a wide variety of shapes, sizes, functionalities, and costs, and they are small programmable devices mainly used for object, animal, and human tracking. On the other hand, RFID readers acquire and on occasion rewrite information stored on RFID tags that come within operating range – from a few inches up to several feet. Typically, RFID readers are connected to a computer system. The computer system receives,

records and formats the acquired data for further processing or applications in general.

11.1.2.1 Edge

The edge of a classic enterprise network includes a number of IoT-enabled devices. These devices can be sensors and maybe actuators. The devices are usually organized in a network structure and are sometimes capable of communication with one another. An example would be a cluster of sensors that send their data to another sensor. The latter sensor may aggregate the data before transmitting them to a higher-level device. Also, an edge network may include gateway devices. Each gateway device acts as a bridge to enable the connection between the IoT-enabled devices and the higher-level communication networks. Since the communication networks and the IoT devices may use different protocols, the gateway performs the necessary transformations and may also execute a basic data aggregation function.

11.1.2.2 Fog

In many IoT deployments, a distributed network of sensors may be capable of producing a large amount of data. For example, offshore oil fields and refineries can create a terabyte of data every day. Another example is that of an airplane producing multiple terabytes of data every hour. One method for these data is to keep it permanently (or at least for a long time) in central storage, ensuring that it can be accessed by IoT applications. However, it is sometimes preferred to perform as much data processing close to the sensors as possible. This type of processing is often referred to as processing done at the edge computing level. The purpose is the transformation of the sensor data into information suitable for storage and higher-level processing. Specifically, the data transformation can result in a significant decrease of the initial high volume of sensor data. Some of the fog computing operations are described below:

- **Evaluation**: Data can be evaluated based on different criteria such as whether it should be sent to a higher-level entity device for processing.
- **Formatting**: Data can be reformatted as this can improve efficiency of higher-level processing.
- **Expanding/decoding:** Handling cryptic data with additional context (such as the origin).

- **Distillation/reduction:** Data can be filtered and summarized. Only the important information will be kept to reduce the impact that the transmitted data will have on the network traffic and the higher-level processing systems.
- **Assessment:** Information can be assessed to decide if it indicates a threshold or alert. The type of decision may also result in redirecting data to additional destinations.

In general, fog computing devices can be found near the edge of the network of IoT sensors and other data-generating devices. This contributes to the fact that some basic processing operations of the produced data can be performed close to the data source. At the same time, this means reduced resource requirement for the higher-level processing systems.

Fog computing and fog services are some of IoT 's distinct characteristics. As a philosophy, fog computing and cloud computing stand in opposite sides. On the one hand, cloud computing is synonymous to massive, centralized storage and processing resources. Utilizing cloud networking facilities, the storage and resources are available to distributed customers. On the other hand, fog computing entails massive numbers of individual smart objects that communicate with each other through fog networking facilities. These objects perform processing and storage operations close to the IoT edge devices. The activity of thousands or millions of smart devices raises multiple issues, pertaining to security, privacy, network capacity constraints and latency requirements. Fog computing is capable of addressing those issues. The term "fog computing" originates from the observation that fog tends to hover low to the ground, whereas clouds are high in the sky.

11.1.2.3 Core

The core network is often called as a **backbone network**. Its purpose is the interconnection of geographically dispersed fog networks. Furthermore, it can provide access to networks outside of the enterprise network. Typically, in order to achieve its purpose, the core network utilizes very high-performance routers, high-capacity transmission lines, and multiple interconnected routers to increase redundancy and capacity. The increased redundancy is also achieved if a part of the core routers is purely internal without acting as edge routers. Additionally, the core network may communicate with high-performance, high-capacity servers such as large database servers and private cloud facilities.

11.1.3 IoT Security

IoT may be considered the most complex and at the same time most undeveloped area of network security [90].

Typically, the centre of the IoT network consists of all the application platforms, databases, data storage servers, and network and security management systems. These central systems are in charge of collecting information acquired by the sensor devices and sending control signals to actuators. On top of that, they are also responsible for the overall management of the IoT devices and their communication networks. Furthermore, the edge of the IoT network consists of IoT-enabled devices. Some of these devices might be quite simple constrained devices, while others might be more intelligent unconstrained devices. In addition, gateways may perform networking services on behalf of IoT devices like protocol conversion.

Figure 11-3 illustrates a number of typical scenarios for interconnection and the inclusion of security features. The shading in Figure 11-3 indicates the systems that support at least some of these functions. Typically, gateway devices are the elements of the IoT networks that usually will implement secure functions, such as TLS and IPsec. On the contrary, unconstrained devices may or may not implement security capabilities. Constrained devices generally have limited or no security features. Therefore, gateway devices enable secure communication between the gateway and the central systems (e.g., application platforms and data storage servers). However, any constrained or unconstrained IoT device connected to the gateway is beyond the zone of security established between the gateway and the central systems. In some cases, unconstrained devices can communicate directly with the central systems and they might support security functions. However, constrained devices that are not connected to gateways have no secure communications with the devices of the central system.

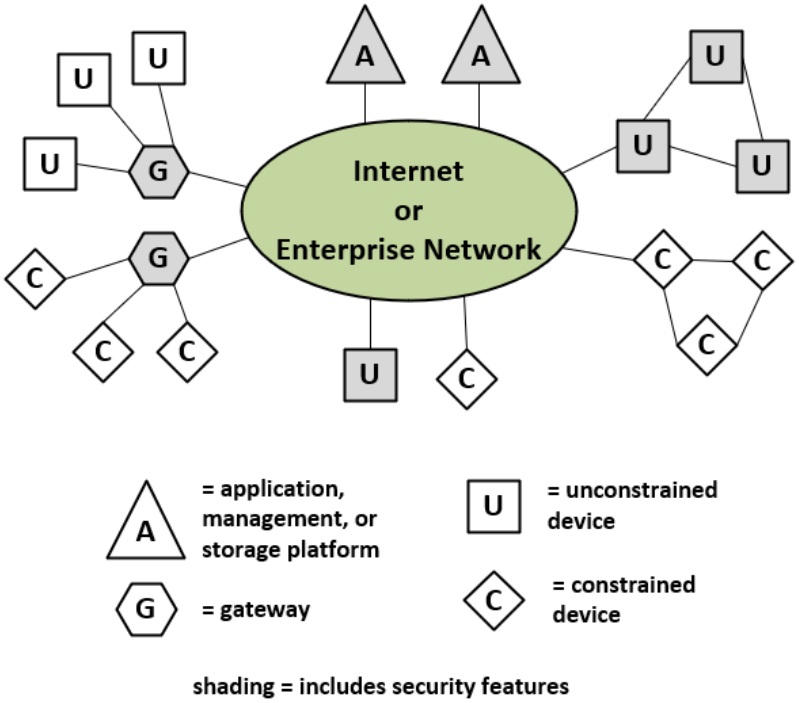


Figure 11-1: IoT Security: Elements of Interest [90]

11.1.3.1 Patching Vulnerability

In 2014, security expert Bruce Schneier stated that we are at a crisis point regarding the security of embedded systems, and particularly the IoT devices. De facto, the Internet of Things has increased rapidly leading to an increasingly need for embedded devices. Therefore, the electronics manufacturers had powerful incentives to produce their chips - with the firmware and software, as cheaply and quickly as possible. Their primary focus was on the price and the feature of the chip. In particular, they focused of the functionality of the device itself and they neglect anything related to the software and firmware of the chip. As a result, the embedded devices are riddled with vulnerabilities and so far, there is no efficient way to patch them. Since the information about how to patch the vulnerabilities is inadequate, and apparently the end user may have no means of patching the system, the hundreds of millions of interconnected IoT devices are vulnerable to malicious attacks. For instance, an attacker may take control of an IoT sensor and insert false data into the whole IoT network. In the case

of the actuators, where the attacker can potentially affect the operation of devices like a machinery, the consequences of a potential security attack are much graver.

11.1.3.2 IoT Security and Privacy Requirements

ITU-T Recommendation Y.2066 (Common Requirements of the Internet of Things, June 2014) sets a list of security requirements for IoT networks. This list constitutes a useful baseline for understanding the scope of a suitable and efficient security implementation for an IoT deployment. These requirements are covering all the functional operations of an IoT system that are capturing, storing, transferring, aggregating, processing the data of things, as well as providing services which involve things. On top of that, they are also related to all the IoT actors. The requirements are the following, as defined in ITU-T Recommendation Y.2066:

- **Communication security:** Secure, trusted, and privacy protected communication capability is required, so unauthorized access to the content of data can be prohibited, integrity of data can be guaranteed, and privacy-related content of data can be protected during data transmission or transfer in IoT.
- **Data management security:** Secure, trusted, and privacy protected data management capability is required, so unauthorized access to the content of data can be prohibited, integrity of data can be guaranteed, and privacy-related content of data can be protected when storing or processing data in IoT.
- **Service provision security:** Secure, trusted, and privacy protected service provision capability is required, so unauthorized access to service and fraudulent service provision can be prohibited and privacy information related to IoT users can be protected.
- **Integration of security policies and techniques:** The ability to integrate different security policies and techniques is required, so as to ensure a consistent security control over the variety of devices and user networks in IoT.
- **Mutual authentication and authorization:** Before a device (or an IoT user) can access the IoT, mutual authentication and authorization between the device (or the IoT user) and IoT is required to be performed according to predefined security policies.
- **Security audit:** Security audit is required to be supported in IoT. Any data access or attempt to access IoT applications are required to be fully transparent, traceable and

reproducible according to appropriate regulation and laws. In particular, IoT is required to support security audit for data transmission, storage, processing, and application access.

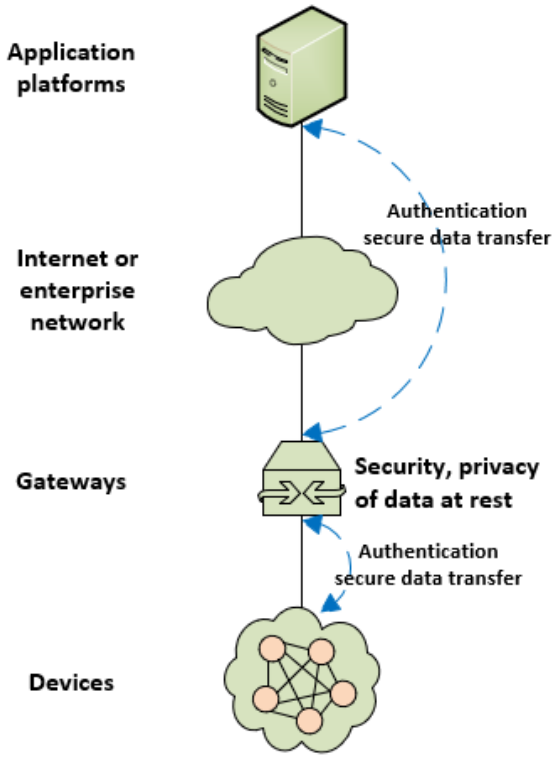


Figure 11-2: IoT Gateway Security Functions [90]

There is no doubt that the gateway is a key element in providing security in an IoT deployment. Therefore. ITU-T Recommendation Y.2067 (Common Requirements and Capabilities of a Gateway for Internet of Things Applications, June 2014) details specific security functions that the gateway should implement, some of which are illustrated in Figure 11-4. These consist of the following, as stated in ITU-T Recommendation Y.2066:

- Support identification of each access to the connected devices.
- Support authentication with devices. Based on application requirements and device capabilities, it is required to support mutual or one-way authentication with devices. With one-way authentication, either the device authenticates itself to the gateway or the gateway authenticates itself to the device, but not both.
- Support mutual authentication with applications.

- Support the security of the data that are stored in devices and the gateway, or transferred between the gateway and devices, or transferred between the gateway and applications. Support the security of these data based on security levels.
- Support mechanisms to protect privacy for devices and the gateway.
- Support self-diagnosis and self-repair as well as remote maintenance.
- Support firmware and software update.
- Support auto configuration or configuration by applications. The gateway is required to support multiple configuration modes, for example, remote and local configuration, automatic and manual configuration, and dynamic configuration based on policies.
- It is worthwhile to mention that when security services for constrained devices are required, some of the detailed requirements may be challenging to deploy. For instance, the gateway should support security of data stored in devices. However, this may be impractical to achieve when the constrained device has no encryption capability.
- Finally, the Y.2067 requirements also detail a number of requirements related to privacy. In Fact, privacy constitutes one additional critical area with the rapid widespread deployment of IoT-enabled things in environments like homes, vehicles, and even human beings. As more and more things are getting interconnected, governments and private enterprises will be able to gather huge amounts of sensitive personal information, such as medical data, location information, and application usage.

11.1.3.3 An IoT security framework

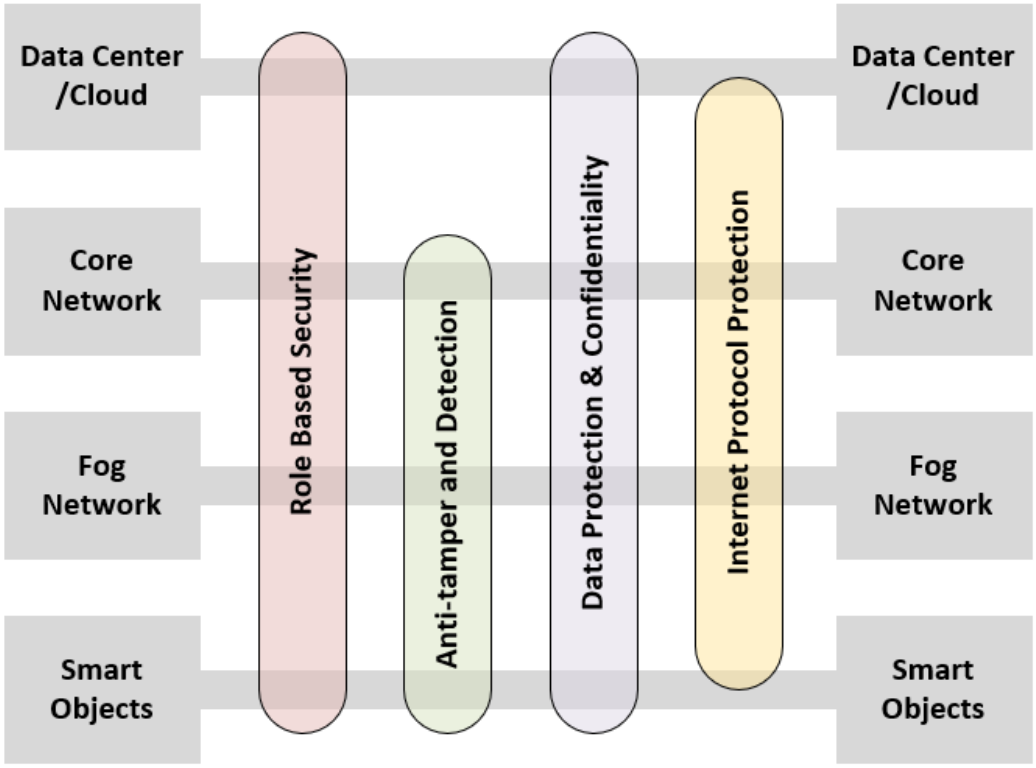


Figure 11-3: IoT Security Environment [90]

Cisco created a framework about IoT security that can be utilized as a guide to the IoT security requirements. The security environment regarding the logical structure of an IoT is depicted in figure 11-5. The IoT model constitutes a simpler version of the World Forum IoT Reference Model and its levels are the following:

- **Smart objects/embedded systems:** Contains sensors, actuators and other embedded systems located at the edge of the network. This is where most of IoT vulnerabilities lie, since the devices may have to be placed in a physically insecure environment and operate for years. Among the various issues, network managers need to consider availability, authenticity and integrity of the sensor data and protection of actuators and other smart devices from unauthorized use as well as privacy and protection from eavesdropping.
- **Fog/edge network:** This level relates to the wired and wireless interconnection of IoT devices. Also, this level may include a certain degree of data processing and consolidation. A great deal of issues is caused due to the variety of network

technologies and protocols of the different IoT devices. Thus, the development and enforcement of a uniform security policy is necessary.

- **Core network:** This level provides data connections between the network center platforms and the IoT devices. Although the security concerns in this level are commonly found in traditional core networks, the security burden increases due to the large number of endpoints to interact with.
- **Data center/cloud:** This level involves the application, network and data storage management platforms. At this level, no new security issues emerge because of IoT, except form the need to interact with a vast number of individual endpoints.

The four-level architecture of the Cisco model contains four general security traits that extend to multiple levels:

- **Role-based security:** In RBAC systems, access rights are assigned to roles and not to individual users. Those users acquire various roles, either statically or dynamically, based on their responsibilities. RBAC techniques are commercially used in cloud and enterprise systems and in general, they constitute a well-known and useful tool to manage the access to IoT devices and their generated data.
- **Anti-tamper and detection:** This function is essential to the IoT device, the fog network and the core network levels. The reason is that some components of these levels may physically lie outside the area that is covered by the physical security measures of the enterprise.
- **Data protection and confidentiality:** These functions extend to all levels of the architecture.
- **Internet protocol protection:** It is important to protect data in motion from snooping and eavesdropping.

Figure 11-5 maps specific security functional areas across the four layers of the IoT model.

A secure IoT framework is also proposed by the Cisco model. The framework describes the components of a IoT security facility and encompasses all the IoT levels. The four components are presented below:

- **Authentication:** Contains the elements that identify the IoT devices and then determine the type of access. Contrary to typical enterprise network devices, where

human credentials (e.g., usernames and passwords or tokens) are used for identification, the IoT endpoints must be designed in order to not require human interaction. Examples of identifiers, that can be used, are RFID, x.509 certificates, or the MAC address of the endpoint.

- **Authorization:** Determines a device 's access rights throughout the network fabric. This layer handles access control. In combination with the authentication layer, necessary parameters are set to allow the information exchange between different devices and between devices and application platforms. These functions are essential to permit the operation of IoT-related services.
- **Network enforced policy:** Includes all elements that ensure that endpoint traffic is routed and transported securely over the infrastructure, whether control, management, or actual data traffic is concerned.
- **Secure analytics, including visibility and control:** Encompasses all the elements for central management of IoT devices. The first function to consider relates to visibility of IoT devices, meaning that central management services perceive in a secure way the existence of the various IoT devices as well as the identity and attributes of each device. By utilizing this visibility, the central management services can perform operations on the IoT devices such as configuration, patch updates, and threat countermeasures.

Moreover, this framework considers as important the concept of trust relationship. Trust relationship relates to the situation where the two partners to an exchange have confidence in the identity and access rights of the other. The authentication mechanisms initially establish a level of trust and the authorization mechanisms increase this level of trust. As per the example of the Cisco model, a car may form a trust relationship with a different car of the same vendor. Through this trust relationship, the cars may only be allowed to exchange their safety capabilities. However, in the case that the same car establishes a trust relationship with its dealer 's network, additional information such as the car 's odometer reading and last maintenance record may be permitted to be shared.

11.2 Baseline Security Recommendations for IoT

11.2.1 Security considerations

As time passes, we are becoming increasingly dependent on smart, interconnected devices for a lot of tasks in our everyday lives. Nevertheless, the same devices or "things" can be the target of attacks and intrusions. The attacks can cause the malfunction of the device and endanger our personal privacy and public safety. Thus, it is evident that security is one of the main IoT issues, that should be seriously considered together with safety. These two matters are always closely connected with the physical world. Furthermore, one more issue concerns the administration of IoT devices, meaning who will be the supervisor and manage the devices. The difficulty of the administration task can be better understood, considering the inherent complexity and diversity of the IoT ecosystem and its scalability issues.

There are a lot of different concerns that limit the consolidation of secure IoT ecosystems. Below, some of these concerns are presented:

- **Very large attack surface:** IoT-related risks and threats are many in number and are constantly changing. Also, IoT devices and services affect citizens' health, safety and privacy since devices gather, exchange and process data from various sources sometimes including sensitive data. Because of the afore-mentioned, the attack range related to IoT is extremely wide.
- **Limited device resources:** Technical constraints in IoT means that conventional security practices cannot be applied as they are but significant reengineering will be required. A characteristic of a majority of IoT devices is their inherent limited capabilities as far as processing, storage and energy supply are concerned. Therefore, advanced security controls cannot be implemented.
- **Complex ecosystem:** One more reason that security concerns regarding IoT are enhanced is that IoT is often depicted as a collection of independent devices. In reality, it should be visualized as a large and diverse ecosystem that includes devices, communications, interfaces and people.
- **Fragmentation of standards and regulations:** IoT security concerns are additionally complicated due to the fact that standards and regulations about IoT security

measures are slowly adopted and simultaneously new technologies are constantly emerging.

- **Widespread deployment:** With the exception of commercial IoT applications, Critical Infrastructures (Cis) have recently started to migrate toward Smart ones. This is achieved by implementing IoT on top of legacy infrastructures.
- **Security integration:** The potentially opposing viewpoints and requirements from all involved stakeholders complicate matters relating to security integration. An instance of that would be IoT systems with different authentication methods, which should be able to communicate and operate with each other seamlessly.
- **Safety aspects:** The presence of actuators or other devices which operate on the physical world makes safety aspects very relevant in the IoT context. Examples, such as the recent cybersecurity attacks on connected vehicles, proved that security threats can easily turn into safety threats.
- **Low cost:** As IoT and its advanced functionalities are employed in several sectors, the potential for considerable cost savings is further highlighted. The reduced costs can be achieved by implementing features such as data flows, advanced monitoring, and integration. However, the low cost of IoT devices and systems can become an important obstacle in implementing security solutions. Manufacturers tend to care more about decreasing production costs. As a result, security features become more limited and product security possibly cannot protect against specific IoT attacks.
- **Lack of expertise:** Since the IoT domain is a comparatively new one, not a lot of people possess the suitable skillset and experience in IoT cybersecurity.
- **Security updates:** It is extremely challenging to apply security updates to IoT systems. IoT User interfaces, in their majority, do not allow traditional update mechanisms. Securing those mechanisms, as well as implementing Over-The-Air updates, is a really difficult task.
- **Insecure programming:** The “time to market” pressure for products of the IoT domain is higher compared to other domains. Consequently, limitations are imposed on the efforts to integrate security and privacy into the design of IoT devices. More emphasis is directed towards the functionality of the devices rather than their integrated security. This behavior is also enhanced sometimes due to budget issues.

- **Unclear liabilities:** The assignment of liabilities is unclear. Therefore, in case of security incidents, there are many ambiguities and conflicts, which are further enhanced due to the wide and complicated supply chain related to IoT.

11.2.2 Challenge of defining horizontal baseline security measures

ENISA together with the vast majority of the experts interviewed agree that studying IoT security in a horizontal way is an extremely complex task. The study of the security measures and the impact of the security threats should take into account the various assets, which differ based on the specific application and usage scenario.

Each IoT environment requires a risk assessment to be carried out in order to understand the threats of the various assets, define the plausible attack scenarios and associate them with the context of the specific IoT service which will indicate the critical hazards and the ones that can be tackled. Therefore, it is easy to perceive the intricacy of approaching the IoT in a horizontal way, in contrast to dealing with a specific IoT problem vertically such as Smart Cars, Smart Airports, Smart Homes, Intelligent Public Transport etc.

Despite the above-mentioned, this section considers the horizontal aspects of IoT that are observed across vertical sectors and focuses on defining baseline security measures for IoT across Critical Information Infrastructures. Therefore, this section compliments the actions of ENISA in the vertical sectors and shows a holistic approach towards IoT security.

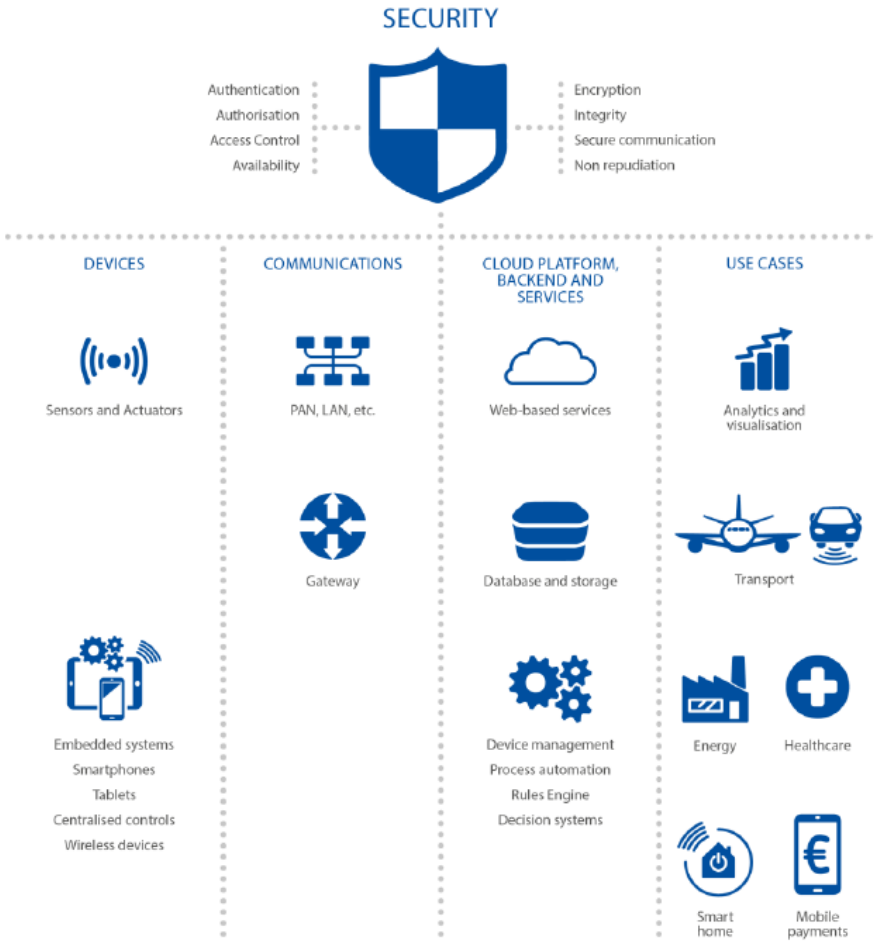


Figure 11-4: IoT High Level Reference Model [91]

11.2.3 Security measures and good practices

In this section, security measures and good practices are listed in detail. These measures and practices aim to tackle the threats, weaknesses and risks associated with IoT devices and environments. They were defined while taking into account the various IoT environments and deployments that they could be applied. Therefore, a wide range of security matters, such as security by design, data protection, risk assessment and others, is covered. The list of security measures / good practices of this section is the result of a very extensive and thorough desktop research. The research considered different security guidelines, standards, etc.

The security measures and good practices can be divided into several security domains. The purpose of this split is the coverage of every IoT environment horizontally and the classification of the security measures based on the IoT ecosystem that they apply. Below, the proposed security domains are presented:

- **Information System Security Governance & Risk Management:** Includes security measures relating to information security risk analysis, accreditation, policy, human resource security and indicators and audit.
- **Ecosystem Management:** Includes security measures relating to ecosystem mapping and ecosystem relations.
- **IT Security Architecture:** Includes security measures relating to system configuration, system segregation, asset management, cryptography and traffic filtering.
- **IT Security Administration:** Includes security measures relating to administration information systems and administration accounts.
- **Identity and access management:** Entails security measures relating to identification, authentication and access rights.
- **IT security maintenance:** Entails security measures relating to IT security maintenance procedures and remote access.
- **Physical and environmental security**
- **Detection:** Entails security measures relating to detection, logging and log analysis and correlation.
- **Computer security incident management:** Entails security measures relating to system security incident report, incident analysis and incident response.
- **Continuity of Operations:** Entails security measures relating to business continuity management and disaster recovery management.
- **Crisis Management:** Entails security measures relating to crisis management process and organization.

As it was stated earlier, the proposed security domains arrange the security measures according to the area of IoT ecosystem that they apply to. Except the area of application, the security measures can be classified based on their nature. The first category is policies that must be considered during the devices' development. The next category consists of organisational business measures and employees that should be adopted by the organisation. The last category refers to the technical measures to limit the potential risks that target the IoT devices and other elements of the IoT ecosystem. Thus, the identified IoT baseline security measures are placed into the three categories and are presented below:

- Policies (PS)

- Organisational, People and Process measures (OP)
- Technical Measures (TM)

11.2.3.1 Policies

The first category of security measures is the policies relating to information security and how to make it more concrete and robust. The policies need to contain well documented information and they should cover all of the organisation's activity.

Additionally, it should be mentioned that concerning the security and privacy by design, the security measures need to take into account the unique traits and context that the IoT device or system is deployed in. For example, different specifications are considered when an IoT device is used at a home environment, in comparison to a critical infrastructure. When applying specific security measures, it is always worth keeping in mind that the cyber risk associated to IoT depends highly on context.

11.2.3.2 Organisational, People and Process measures

It is important for all businesses to have organisational criteria for information security. The established personnel practices should promote adequate security and ensure that processes are safely managed and that information is safely used. In addition, organisations need to make sure that contractors and suppliers are responsible and accountable for the functions considered. In case of a safety incident in the organisational, everyone in the organisational should be prepared, understanding what they are responsible for, how to evaluate the incident and how to respond to it.

11.2.3.3 Technical Measures

The security measures should take into consideration the technical elements to reduce the vulnerabilities of IoT. While there are horizontal technical measures that can be introduced across multiple vertical sectors/CIIs, the unique traits of each vertical means that more specific measures can be employed for each vertical/CII. An instance of these unique traits is scalability, meaning that the large amount of involved devices may point towards specific measures that should be applied at the level of specialised architectural components, e.g. gateways.

11.2.4 Gaps analysis

In this section, the main gaps of cyber security in IoT are analysed. Before addressing cyber security in IoT, it is necessary to identify and define the gaps, namely how much the present state deviates from the desired state. This deviation can be used to determine solutions to close those gaps.

11.2.4.1 Gap 1: Fragmentation in existing security approaches and regulations

Currently, there is neither an established EU-wide approach nor a common multi-stakeholder model regarding cyber security in IoT. Interviews with experts have shown that most of them identified the deficit of concrete security frameworks and the breadth of security considerations as large obstacles for the enhancement of security. Thus, each company or manufacturer has its own approach when applying security features into IoT. This means standards and good practices, that could guide the adoption of IoT security measures, are slowly embraced. In this context, it could be more beneficial if initiatives were established to stimulate the development of security in private companies. Additionally, it is crucial that both the public and private sectors understand that security is a shared responsibility. It does not only concern a single company, manufacturer, customer or IT professional, but rather everyone involved. One more obstacle for the development of IoT security measures is the fragmentation of regulations. There is no regulation which enforces the employment of security measures and protocols in the various levels of an IoT ecosystem, such as the devices, network etc. Establishing such regulations could lead to a more complete integration of safety and security in the development lifecycles. On the other hand, creating a common standard for all cases might hinder innovation and research in the area. As it was stated previously, different applications areas may have different security requirements and this should always be kept in mind.

Furthermore, the issue of unclear liabilities needs to be dealt with. The responsibilities of everyone should be defined and enforced in order to overcome the existing problem of non-responsibility, both moral and legal. As different manufactures and parties keep developing and operating the various elements of an IoT ecosystem, a perfect isolation between these elements will unavoidably be formed. Thus, it is necessary to determine and clarify the liability of each actor in case that a security incident occurs.

11.2.4.2 Gap 2: Lack of awareness and knowledge

A knowledge gap exists as far as the move towards connected and interdependent systems and devices is concerned. When interviewing with IoT experts, variations in fundamental terminology were observed. One instance concerns the concepts of safety and security, where security experts have adapted to the term of “business IT” security, but not that of IoT security.

An overall lack of awareness in relation to the need of security in IoT devices can be observed. Additionally, a majority of IoT consumers do not understand their IoT devices and lack knowledge about the threats these devices are exposed to. This may lead to devices not being updated, which increases the possibilities of potential security breaches.

Moreover, companies need to update their employees in good security practices, since technological expertise is not necessarily the same as security expertise. Generally, consumers, developers, manufacturers and others need to be educated about the IoT use, its associated security risks and how everyone should prepare themselves. Also, security awareness can be raised with training in both safety and cyber security.

Developers and manufactures could avoid many security events if they knew about the risks affecting not only the IoT devices but also the whole IoT environment. This is necessary to raise awareness about existing threats and risks and to learn how to prevent, protect and act when a security incident happens.

11.2.4.3 Gap 3: Insecure design and/or development

Many studies have focused on the design and development concerns of IoT security. While interviewing experts, the findings of these studies were verified. Thus, some important issues about IoT design and development are presented below:

- No defence-in-depth strategy during the design of the system, such as a secure boot process, isolation of a Trusted Computing Base, limitation of the number of open ports, self-protection, etc.
- No security-by-design or privacy-by-design. In some instances, there is information exchange with a third-party. It should be assured that the information, sent outside of the IoT environment, is only as much as is strictly needed.
- Lack of communication protection, on internal as well as external interfaces.

- Lack of strong authentication and authorisation:
  - No validation or signing of firmware updates,
  - Software updates without server authentication and file trust verification,
  - No secure boot mechanisms.
- Lack of hardening:
  - No data execution prevention or attack mitigation technologies used on the firmware,
  - Public vulnerabilities (DNS proxy, HTTP service…) left unfixed,
  - Some services are exposed through different entry points, with unnecessary communication portsleft open – services such as Telnet or ssh are sometimes bound to all network interfaces,
  - Weak passwords policies or default passwords left unchanged,
  - Configuration flaws.
- Lack of diagnosis / response capabilities.

11.2.4.4 Gap 4: Lack of interoperability across different IoT devices, platforms and frameworks

In most IoT ecosystems, IoT devices are connected with legacy systems, especially when there are Critical Information Infrastructures. Moreover, as stated previously, the lack of common regulation means that companies and manufacturers design IoT devices in their own way. This leads to the emergence of interoperability issues between devices of different manufacturers and to the existence of diverse and maybe incompatible security models, concepts, taxonomies etc. Thus, measures need to be adopted to ensure a proper and secure interconnection and interoperability between the IoT environment, legacy systems and other IoT devices produced by third-parties.

A majority of IoT devices employ proprietary protocols created by their manufacturers so that they can interconnect with devices. Devices of the same manufacturer can communicate in this way without any problems. However, interconnecting devices from different manufacturers is an issue. Therefore, standard protocols, that will be supported by all manufacturers, need to be established in order to enhance interoperability without sacrificing efficiency or security. In this context, the use of close-source and proprietary protocols should

be avoided. Their security cannot be verified and many events have shown that obscurity in security does not necessarily mean proper security coverage.

In the same spirit, using common frameworks can also raise the efficiency and security of the IoT devices when there is a need to interconnect multiple devices produced by different manufacturers.

11.2.4.5 Gap 5: Lack of economic incentives

For IoT manufactures and vendors, functionality and usability are considered as more significant factors compared to security. They are not interested in spending much money on security which in many instances is not considered at all. The companies do not dedicate much budget to security since it is generally believed that there is no immediate return-on-investment for applying security measures. The reason for this belief stems from the difficulty of assessing the financial impact of possible security weaknesses.

Moreover, there are not enough economic incentives that would encourage the companies to improve security. These incentives could have the form of economic benefits (e.g., more grants to integrate better security in the devices), resources, perceived reputation, etc. The few existing forms of economic support are limited to very competitive research and development programs like the H2020 project.

Generally, in the interviews with IoT experts, they acknowledge that the impact of the various risks, threats and hazards are usually underrated and set aside due to budget issues. Security concerns are usually addressed after an incident has occurred.

11.2.4.6 Gap 6: Lack of proper product lifecycle management

In general, the lack of safety measures from the design stage is discovered in a later development stage. The various IoT devices and networks connect with each other and are, in most times, exposed to the Internet becoming the target of a lot of threats. Consequently, the product lifecycle of the different assets of a specific IoT environment should be managed properly.

Since IoT encompasses a large number of products, the vulnerabilities of those products can affect negatively the entire surface of the traditional supply chain. IoT extends the global

attack area and thus, the various devices need to be improved in order to provide their services consistently and in a secure way through their whole lifecycle.

In this process, the vendors, who design and develop the devices, should take action to add new security features and characteristics proficiently and cost-efficiently. Nevertheless, this does not only rely on manufacturers that implement the new features, but also on organisations that bear the associated costs. A balance between cost and security should be sustained.

Throughout their lifecycle, the IoT devices should maintain the ability to be updated in order to ensure that they operate properly and that new vulnerabilities are amended. As stated previously, most IoT consumers lack knowledge about their IoT devices and their associated threats which may lead to devices not being updated and increases the possibilities of a potential security breach.

In addition, the deployment stage is a significant stage in the device lifecycle management. Best practices to deploy IoT devices can be defined. In these practices, recommendations may be included about specific configurations of devices and networks or about the necessity to employ cybersecurity monitoring systems to detect anomalies in the deployed infrastructure.

ID	DESCRIPTION
1	Promote harmonization of IoT security initiatives and regulations
2	Raise awareness for the need for IoT cybersecurity
3	Define secure software/hardware development lifecycle guidelines for IoT
4	Achieve consensus for interoperability across the IoT ecosystem
5	Foster economic and administrative incentives for IoT security
6	Establishment of secure IoT product/service lifecycle management
7	Clarify liability among IoT stakeholders

Figure 11-5: High-level recommendations to improve IoT cybersecurity [91]

11.3 Guidelines for Securing the Internet of Things: Secure supply chain for IoT

11.3.1 Supply chain reference model for IoT

The IoT supply chain includes the actors, processes and assets that participate in the realization (e.g., development, design, maintenance, patch management) of any IoT device.

This study considers the supply chain for IoT is composed of two main aspects: the physical aspect and the logic aspect. The physical supply chain relates to all the physical objects (e.g., devices, electronic components, appliances) moved through the supply chain phases, as well as the associated manual processes (e.g., manual assembly, distribution processes). The logic aspect of the supply chain for IoT is associated with the software development and deployment, network-based communications, and virtual interactions between the IoT objects and the supply chain stakeholders.

IoT supply chain risks, and more generally IT supply chain risks, are associated with an organisation's decreased visibility into, and understanding of, how the technology they employ in their product or solution is developed, integrated, and deployed. An overview of the IoT supply chain is provided, presenting all its different stages with a detailed mapping of them that can be found after the following subsections. This aims to give an approximation of the stages sequence and the interactions between actors to identify where the security concerns might arise.

Although the stage layers are presented as being separated, it should be taken into account that sometimes they are treated as a single entity due to project constraints or other business realities.

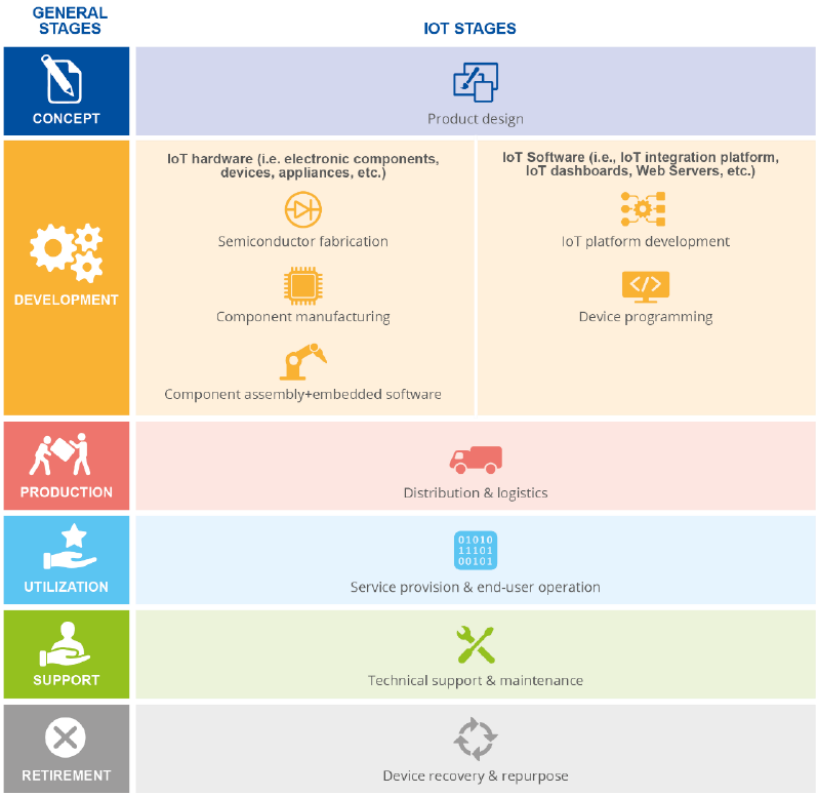


Figure 11-6: Supply Chain Reference Model for IoT [92]

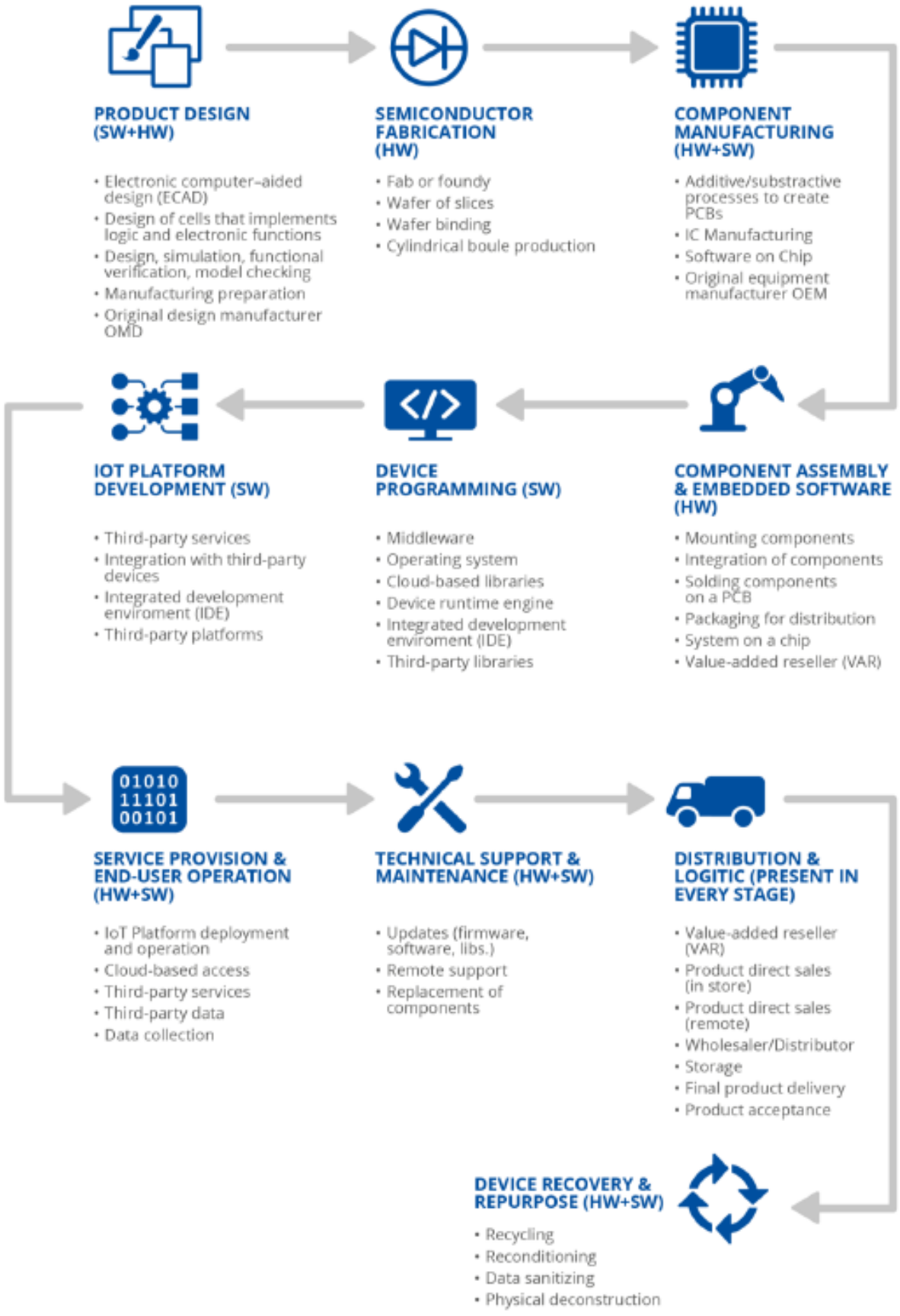


Figure 11-7: Mapping of the IoT supply Chain [92]

11.3.2 Good practices for security of IoT supply chain

Development of good practices for securing the supply chain for IoT is one of the key objectives of this section. The aim is to provide recommendations for the target audience to assist in countering and mitigating the threats that might impact the supply chain for IoT. Recommendations focus on covering the overlapping issues, as most practices are not effective across all industries and users.

To organise the domains in a logical manner, good practices were classified into the following three main groups: actors, processes and technologies. Please note that there may exist a degree of overlap between groups and some good practices could be classified into multiple categories due to the strongly integrated nature of the supply chain for IoT.

Actors: guidelines related to the principles that shape how actors in the supply chain are expected to think about, perceive and approach security in the supply chain for IoT; whether it is in the context of a clearly defined and previously agreed framework or from a personal standpoint. Industry professionals (e.g. managers, engineers), end users and organizations can be identified as actors in the supply chain.

Processes: addresses security in the processes involved when an IoT project is designed, developed, deployed and maintained. These processes are not limited to the context of a single organization and include interactions between stakeholders, especially in those cases where trust cannot be clearly established.

Technologies: potential technical measures and elements that could be applied in order to predict, detect and reduce vulnerabilities and threats. These include hardware components, design recommendations, techniques, libraries or other software components to support the process throughout the entire supply chain.

11.4 Secure Software Development Lifecycle

11.4.1 IoT Secure Software Development Lifecycle (SDLC)

ENISA establishes good practices regarding IoT security, focusing on software development guidelines to ensure the security of IoT products and services throughout their lifetime. Introducing secure development guidelines across the IoT ecosystem, is an absolute necessity for the improvement of IoT security. The good practices, provided in this section, can help to achieve security by design, a key recommendation that was highlighted in the Baseline Security Recommendations section which presented the security of the IoT ecosystem from a horizontal point of view.

Software consists the core of all IoT systems and services. It empowers their operation and provides their features. The ways that software supports IoT are multiple such as the IoT devices firmware, implementations of IoT communication protocols and stacks, Operating Systems (OSs) for IoT products, Application Programming Interfaces (APIs) supporting connectivity of different IoT services, IoT device drivers, backend IoT cloud and virtualization software, as well as software implementation of various IoT service operations. A great deal of attention is paid to supply chain issues, such as the integration of hardware and software.

Following the secure Software Development Life Cycle (SDLC) guidelines can lead to less vulnerabilities in IoT and to the development of secure software applications and services. The establishment of a set of secure development guidelines can contribute to address several IoT security challenges. Example of those guidelines are the check for security vulnerabilities, secure deployment, continuous delivery, ensuring continuity of secure development in cases of integrators etc.

11.4.2 SDLC phases

ENISA strongly encourages **security and privacy by design and by default**. Accordingly, an effective way to minimize vulnerabilities in IoT is the development of secure applications, using the **secure Software Development Life Cycle (sSDLC)** principles and developers experienced in secure coding. As it was mentioned previously, the establishment of a baseline secure development guidelines can contribute to address several IoT security challenges.

In this regard, this section is dedicated to the definition of a set of good practices and guidelines for the various stages of the secure SDLC of IoT solutions. After asking the experts, most of them considered that the SDLC should consist of six phases, as shown in Figure 11-11.

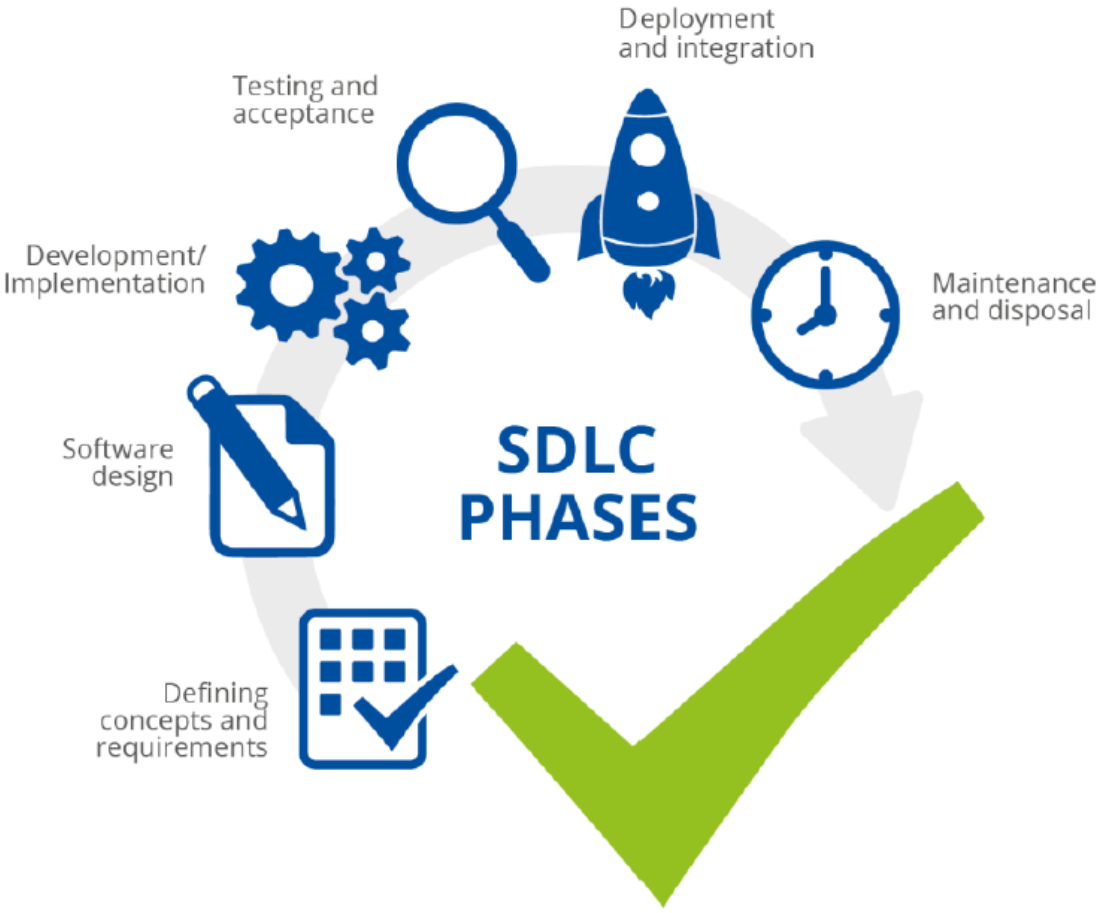


Figure 11-8: SDLC phases [93]

The different phases of SDLC aim to deliver effective and efficient systems based on their design and operational requirements. There are various SDLC models that differ from each other in the manner that security considerations are incorporated. The overall security of the IoT ecosystem is enhanced if security is considered across all phases of IoT SDLC and security measures are applied on the correct assets.

In order to secure the IoT SDLC process, it is important to secure the process across all elements of the IoT ecosystem, meaning the IoT end devices, communications, cloud backend and applications for mobile devices. Furthermore, all types of software of these elements should be evaluated. This includes the end device firmware, IoT services/software

implementations, IoT gateways source code, network protocol implementations, API source code, software running on backend cloud servers etc.

It is evident that the heterogeneity and complexity of the diverse IoT elements and IoT software types amplify the number of cybersecurity issues. Thus, it is increasingly necessary to establish homogenous and good practices for a secure SDLC. Security concerns of the different IoT SDLC phases are discussed in what follows.

11.4.3 Security in SDLC

When integrating security in a process (like the SDLC), one significant factor that is usually overlooked, refers to assessment and evaluation. The current cybersecurity state must be first understood before creating a plan to sustain this state and ameliorate it. In this respect, Security Maturity Models (SMM) are really useful because they guide organisations to recognise their level of security based on their intended requirements. The current level of security, its needs, benefits, and the cost of its support are evaluated, taking into account particular threats to the regulatory and compliance requirements of an organisation's industry, the specific risks existing in an environment, and the organisation's threat profile. Numerous standards function as tools to assess the security of a software project. Examples of the most widely recognised industry standards include the Common Criteria (CC), Capability Maturity Model Integration (CMMI), Building Security in Maturity Model (BSIMM), Security for industrial automation and control systems Part 4-1 - Secure product development lifecycle requirements (IEC 62443-4-1), or Open Software Assurance Maturity Model (OpenSAMM).

All six phases of the IoT SDLC share the same core principle of security. Relevant and applicable controls are required in each phase to assess the security state (e.g. creating security gates and metrics). Security gates must be implemented to confirm that software covers all required security conditions before moving forward to next phases. In each phase, the completion is indicated by severity thresholds that are defined by means of metrics. Utilizing the metrics, vulnerabilities throughout the development process can be analysed, detected and corrected.

Complementary to security, the documentation process is a cross-cutting activity that is often viewed as not necessary during the SDLC process. This is due to multiple reasons, such as the

complexity of the IoT solutions, the number of resources involved in a development process, the module integrations, the IoT interconnectivity, the number of external and internal components, designs, configurations, requirements etc. Nevertheless, a good documentation and a documentation management system, that supports the SDLC process, are crucial to consider it understandable, traceable, and subject to auditing and monitoring.

12 Ethics in IoT Networks and Applications

Author(s): Josephina Antoniou

12.1 General Principles

This chapter discusses the general principles, including general perceptions and a methodical approach to understanding ethics in technology. Following the general principles, the current set of notes focus on IoT Ethics specifically and discuss ethics for developers and users of IoT technology, as well as a limited view into the relation of IoT Ethics and relevant legislation. Finally, a specific case study is explored that investigates the ethical aspects of IoT-based tracking and monitoring applications, discussing quality of experience and ethical concerns. In-class tasks included in the lesson slides are further explained and discussed within this set of notes.

This section introduces general ethical principles and relevant approaches and perceptions to consider in subsequent sections, specifically, for the IoT networks and applications.

12.1.1 General Perceptions of Ethics related to technology and IoT

In 2014, McEwen listed a number of perceptions regarding users' reactions to the Internet of Things, in a publication entitled *Designing the Internet of Things* [94]. Positive perceptions include the fact the Internet of Things is an exciting and new technology, representing progress and opportunity to revisit the way people live, by freeing users from tedious tasks and allowing for more leisure time. Even in the professional realm, the Internet of Things has the potential to create more interesting and rewarding jobs, to replace ones that are becoming obsolete. Perceptions of people, however, did not only focus on IoT's positive potential. Some people argued that this new technology may eliminate jobs, creating unemployment. Moreover, increased automation may result in lazy and unhealthy societies, with people disconnected from tradition. Finally, worries for intrusiveness and lack of privacy were voiced as negative potential results from the deployment of the IoT technology.

Other authors [95] [96] have posed similar concerns on protection of personal data, data ownership, data bias and the linked perception of discrimination potential and the need for justice. The effect of social context on these perceptions is highlighted by the authors of *Engaging with Ethics in Internet of Things* [97], in particular, what is valued within different such contexts. Nevertheless, people are not expected to engage with these social contexts in the same way, and this results in both positive and negative perceptions, as people tend to

engage in one of three ways, according to the article: they are either disengaged, pragmatic or idealistic.

Often perceptions emerge from lack of understanding of something new. The idea of intelligent objects that are interconnected to support different services can be confusing or even intimidating. Therefore, the responsible approach to approaching such new technology is to promote clarity and understanding with regards to the technology's impact on people, life and the environment, and furthermore, promote a responsible and ethical approach to development and use of IoT that such that perceptions are appropriately informed.

12.1.2 Why Ethics?

The idea of Ethics in general, is often viewed as a set of moral principles that can affect humans' quality of life, i.e. what actions are *right* or *wrong* in specific situations. Even though multiple definitions of ethics have been attempted that refer to *quality of character, moral duty or principle, or proper behavior,* the section will undertake a contextualized definition that refers to *Computer and Information Ethics [98].*

Computer and Information Ethics definition refers to a branch of Applied Ethics that studies *ethical problems aggrevated, transformed or created by computer technology*. This computer technology also incorporates the technology studied by this course, namely the Internet of Things. In fact, computer ethics as a specific branch of applied ethics is becoming more and more popular with new conferences, research centres, journals, textbooks, web sites and courses dedicates to it. Additional sub-topics in computer ethics continually emerge as information technology grows. Recently, sub-categories of Computer and Information Ethics, have begun developing focusing on new more specific topics, such as online ethics, "agent" ethics (robots, softbots), the "open source movement", electronic government, ethics and nanotechnology, etc.

The following subsections briefly discuss specific ethical values that often come across in the discussion of Ethics for IoT. In particular, the ethical issues or values we explore include the right to privacy, the principle of informed consent, data security and safety, transparency and trust, information and power asymmetries.

12.1.2.1 The right to privacy

With the Internet of Things, more and more public data about Internet users exists, e.g. through social media, their mobile devices, even their smart spaces. In addition to the collection of vast amounts of data, the existence of massive data storage potential makes this an even more complicated issue, as the data continues existing.

The right to privacy, simply states that each user has the right to decide that their data is not visible to everyone. In addition to user data, as the Internet of Things is about *Things*, data collected from these *Things* may also become available online. As sensor data is so ubiquitous, it inevitably detects more than just the data that you have chosen to make public.

Let's consider an example. Regarding the electricity smart meter, the aggregate data collected from a group of consumers can be very useful for the company (cost benefits) or the environment; but it may still "leak" personal data. One of the ways that this may happen is when it becomes possible for private information about a household to be inferred from such collected data. To address the ethical issue in this situation questions such as "*Who owns such sensor data?"* must be answered. In fact, the issue of data ownership is significant; would the ownership belong to the electricity company, the household owner, the other house residents, or the Internet Service Provider?

Privacy of personal data has been addressed in the European General Data Protection Regulation (GDPR), applicable since May 25th, 2018, in all member states of the EU. The purpose of GDPR is to harmonise data privacy laws across Europe. We will return to the discussion of GDPR, later in the chapter.

12.1.2.2 The principle of informed consent

The Informed Consent is an ethical and legal requirement, usually applied to participants in a specific research study, customers of a specific service, or users of a specific product provided by a company, etc. The informed consent is basically permission by the participant, customer or user to collect and/or manipulate personal data (which may or may not include sensitive data), or permission to join a clinical trial, or permission to be observed, etc.

Given that such permission is required for responsible development and use of technology, and specifically IoT technology, then the consent form should offer necessary information to

transparently and truthfully present the permission request. An example of a consent form should generally include the following components [99].

- Inform the subject, i.e. the participant, customer or user, about: his or her rights while using the IoT technology, the purpose of the study, service or technology product, the procedures to be undergone, the potential risks and/or benefits of participation
- Inform the subject on how the design ofthe study, service or technology product carefully aligns with the requirements of the regional regulatory body, and any appropriate standards related to the responsible and ethical use of the technology.
- In addition to previously discussed rights of being infomed about the process or functionality, the participants, customers or users should also be informed of their right to withdraw from the study, or stop using the service or product, at any time. It should also be clear that withdrawal is their right and it can happen without penalty.

12.1.2.3 Data security and safety

Data security is abroad concept that encompasses more than just safeguarding the actual user data. It is also about safeguarding connected devices and networks, while practicing data security principles in unit devices, e.g. encrypting data. The encryption process safeguards the data as it travels through the network but the participating IoT devices themselves should be used securely. The IoT devices, or the “Things” in the Internet of Things network and infrastructure, are often vulnerable to different kinds of security threats. This is not just because of the devices themselves, but also because of their connectivity in the IoT, which opens them up to a number of vulnerabilities.

Although the issues of security are extensively covered elsewhere in the course, it is important to highlight the ethical issues that arise when security is compromised when it comes to personal and private data and basic safety. The idea of safety is about recognising and mitigating the risks of using the technology. Such risks could include:

- Device Discovery Challenges
- Loss of privacy

- Physical security of devices
- Lack of user awareness for device usage
- Untimely patching and updating

For example, a common IoT device, e.g. a home appliance or a wearable device, can be used by an attacker to infiltrate a larger network, or to obtain sensitive personal information.

12.1.2.4 Transparency and trust

Ideally, data generating users should always be made aware of their rights (by using informed consent). Within the informed consent, the users should acquire information about how the collected data will be used, and who will have access to it. In addition, information about how long the data will be available before it is deleted, should be available.

In fact, for trust and transparency, it is important to explain to the technology users their right to grant or withdraw consent at any time, as well as to request that their data is deleted or can be accessed in its entirety. One of the elements that often makes it uncomfortable for citizens to engage with new technologies is lack of understanding of what the technology actually does, what data it collects and how this data is eventually used, or even whether the data is ever deleted. It is important to create confidence in the usage of new technology and support its potential for beneficial innovation by clearing such doubts for potential users.

For example, transparency and trust are important in dealing with large amounts of IoT data generated as part of a service based on Artificial Intelligence (AI) algorithms, e.g. machine learning. Trust for the actual AI service is crucial but it is based on the trust in the generated data itself (i.e. the IoT generated data). The technology designer and developer need to follow such guidelines that allow the users to eventually have available information that makes the functionality of the technology as transparent as possible. Ignoring this requirement at the design and development stage may result in untrustworthy technology, especially for applications based on algorithmically more complex technologies such as AI.

12.1.2.5 Information and power asymmetries

Asymmetry of Information, and consequently of power between users and providers, is a phenomenon frequently observed in IoT deployment. Oftentimes, it is necessary for the service provider to have more information than the service user (even about the user) in order to be able to use the IoT to support the service itself. This can be with regards to any type of contextual information that is needed for a decision or a recommendation, depending on the IoT service or application.

For example, consider a service that uses location information of customers and vehicles in order to match appropriately the customers to vehicles (e.g. if the customers need a ride or a delivery). In order to complete the transaction, the provider must collect information about the user and match to the information about the vehicles, resulting in information asymmetry. Momentarily, the service provider is in a position of power, as the information that the users and the provider have are not symmetric, i.e. the provider knows more.

In order to ethically approach the need for information asymmetry, the process needs to be transparent, the users must know and agree to the type and amount of information that the service provider will collect. The service provider must use this information only for the purposes of the services and then delete it when it is no longer of use for the specific purpose that the user agreed to. It is important to ensure that the asymmetry is solely allowed for the use within the service and that it is beneficial to the user.

12.1.2.6 Why Ethics?

When we discuss Ethics in IoT, certain applications often come to mind, for example eHealth wearables, congestion control application for transportation, smart metering, smart banking, smart farming, industrial automation applications, and monitoring applications, such as environmental monitoring, or vehicle-tracking applications.

How do IoT types of interactions relate to ethics? One type of interaction that may raise ethical concerns is the interaction between IoT providers and IoT users. Ethical values that must be considered for this interaction is the principle of informed consent, the need for trust within the interaction, and the need to make the terms of use understandable so that any

data collection and data sharing information that may be included in these terms, are clear. IoT interactions included human to human interactions, where the IoT technology is only the medium that supports a specific activity between the two end-users. In such a case the right to privacy and information asymmetries and power asymmetries must be taken into consideration. Such cases include the example of employee tracking by the employer, e.g. for a distribution or delivery company. Finally, interactions of humans to "things" or objects need to be considered. This category of interactions needs to take into account data usage details, issues of security and safety and the overall quality of user experience as this is affected directly or indirectly by the technology itself.

12.1.3 A methodical Approach to Resolution

According to Rushworth M. Kidder [100], there are ethical checkpoints that must be considered during a process such that the decisions taken methodically lead to an overall ethical outcome. Such an approach can be applied to the design, development or of an IoT technology service or application, or an instance of a potential use scenario of the technology.

The approach begins by recognising that there exists a moral or ethical issue in a given scenario. Even when designing a new service, use cases must be considered to be able to identify such potential issues. Once you recognise that there is indeed an issue or more with regards to ethics, the actual ethical value that needs attention must be identified. There needs to be caution when characterising these issues such that there is not too much (hyper-moralist) or too little (cynic) diligence on the issue.

Next, it is important to determine the actor or actors in the specific scenario instance, such that it becomes clear who is involved. The facts of what can happen in the potential scenario must be gathered. Once the actors and the facts are known, then the scenario can be tested for right versus wrong, according to different paradigms. For example, what would be right or wrong according to the law; what would be right or wrong according to the general public; what would be right or wrong, according to a parent or your parent? Several dilemmas can also be used to test the scenario "right versus wrong" approach. For example, we can check the scenario against the "truth versus loyalty" dilemma, or the "individual versus community" dilemma, or the "short-term versus long-term dilemma", or even "justice versus mercy".

To resolve these dilemmas, philosophers have proposed some approaches or principles to follow. These include the Utilitarian principle, the Kantian principle and the "Golden Rule" principle. However, one must not forget that there may be a different way out of a dilemma that is neither of the sides offered by the dilemma itself. Nevertheless, for completion purposes, we next define the three proposed principles for resolution of ethical dilemmas.

The Utilitarian principle, according to the Stanford Encyclopedia of Philosophy[40], states that the morally right action is the action that produces the most good, i.e. the aim from a utilitarian-based decision is to bring about the greatest amount of good for the greater number of people. In fact, Utilitarian principles hold that everyone's happiness counts the same and there is no egoism in decision-making.

The Kantian principle, also from the same source, is based on Kant's *Theory of Judgement,* a deontological theory of ethics, which, in summary, states that an action is good only if based on a principle of duty to the moral law, i.e. that all people should follow regardless of their preferences.

The "Golden Rule" principle, is also known through the phrase "*Do unto others as you would have them do unto you*". Normally, we interpret the "golden rule" as the way in which we are supposed to *act.* However, in practice, its greater role may be psychological, alerting us to everyday self-absorption and the failure to consider our impacts on others. The rule reminds us also that we are peers to others who deserve comparable consideration.

Finally, once a decision is made, revisit it and reflect on it.

12.2 Focusing on IoT Development and Usage

This section focuses on IoT Ethics by considering the perspective of IoT development and of IoT usage; in both cases we investigate ways to alleviate negative impacts on human experience. The following paragraphs that discuss developing and using IoT technology in an ethical manner are informed from the European Horizon 2020 project SHERPA (SHERPA: Shaping the Ethical Dimensions of Smart Information Systems – a European perspective (project-sherpa.eu)).

[40] Stanford Encyclopedia of Philosophy, [online] plato.stanford.edu

When developing or using technology, specific high-level requirements must be considered. The high-level requirements acknowledged for emerging technologies, including IoT technology development and usage, include: human agency, liberty and dignity; technical robustness and safety; privacy and data governance; transparency; diversity, non-discrimination and fairness; individual, societal and environmental wellbeing; accountability.

With regards to developing technology, responsible and ethical development must be based on responsible and ethical development methods, but also responsible and ethical governance frameworks, e.g. policies, that encourage such development. Responsible development methods must include responsible requirement collection, consideration of ethics from the design phase, as well as ensuring that responsible values are considered within the development stage, e.g. transparency. To achieve all the above-mentioned responsibility guidelines, it is important to have responsible and ethical societal structures, e.g. reflected in the educational system, reflected in the business practices, etc.

With regards to using technology, responsible and ethical usage can be achieved once ethics are integrated in governance and management, such that responsible deployment and use of AI is achieved (Responsible IT Management and IT Governance). There is a need for support from other stakeholders and society at large, such as IT suppliers, governmental institutions, educational institutions, professional organisations, clients, etc.

12.2.1 Operational Ethics Requirements for technology developers and users

With regards to the values of Human Agency, Liberty and Dignity, the following requirements should be considered:

- Ensure that stakeholders are informed about how to control the system without being deceived
- Ensure that the system does not indirectly affect autonomy or freedom of stakeholders
- Ensure that the system does not interfere with the stakeholders' ability to make decisions

With regards to Technical Robustness and Safety, the following requirements should be considered:

- Ensure that the system is secure and resilient against attacks
- Ensure that the system is safe in case of failure (safe mode, fallback plan)
- Ensure the accuracy, reliability and reproducibility of the system (proper logging and documentation, audit-ready)

With regards to Privacy and Data Governance, the following requirements should be considered:

- Ensure protection of stakeholders' privacy
- Ensure protection of quality and integrity of data
- Ensure protection of access to the data
- Ensure protection of data rights and ownership

With regards to Transparency, the following requirements should be considered:

- Ensure that the system has a sufficient level of traceability
- Ensure that the system has a sufficient level of explainability
- Ensure that the relevant functions of the system are communicated to stakeholders

With regards to Diversity, Non-Discrimination and Fairness, the following requirements should be considered:

- Ensure the avoidance and reduction of harmful bias
- Ensure fairness and avoidance of discrimination
- Ensure the inclusion and engagement of stakeholders

With regards to Accountability the following requirements should be considered:

- Ensure that the system and its design process is auditable
- Ensure that negative impacts are minimised and reported
- Ensure internal and external governance frameworks
- Ensure human oversight

Finally, with regards to Individual, Societal and Environmental Wellbeing, the following requirements should be considered:

- Ensure a sustainable and environmentally friendly system

- Ensure the protection of democracy and democratic decision-making
- Ensure system evaluation for potential impact on individual well-being (vulnerable groups)
- Ensure the protection of social relationships/cohesion

12.2.2 Law & Ethics and their application to technology development and use

Law is a set of rules produced by the government. The judicial system of the country uses the law in order to provide protection to the public. The aim is to maintain social order, peace and justice for society. The law is compulsory for all citizens and it dictates what a citizen can do and cannot do. Penalties and legal consequences are defined for any law violations.

Ethics move beyond law, as they are based on people's awareness/perception of right and wrong. In fact, ethics can be viewed as a system of moral principles, that outline what is good or bad for individuals and for society. It is not a law, but it can be an agreed code of conduct. Nevertheless, ethics are not compulsory, and may be adopted by citizens. Ethics pretty much suggest how a person should live and interact with other people.

We need to make a special note on Europe's General Data Protection Regulation (GDPR). On 25th of May 2018, the European Union Regulation 2016/679 on data protection (the General Data Protection Regulation), came into effect, replacing the European legislation on data protection (Directive 95/46/EC). One such case study of need to consider GDPR is for the example case of monitoring, e.g. vehicle monitoring, employee monitoring, where monitoring is enabled by IoT technology. Such use of IoT technology may pose ethical concerns even though consent is provided, and hence it is legal under GDPR.

According to GDPR, it is clear that for any employee monitoring to take place (and hence for any personal data to be collected), **consent** must be given by the employee. This type of consent, however, may pose ethical concerns. Even though the seeking of consent between two parties demonstrates mutual **respect**, reinforces **autonomy** and generally assures **fairness**, often this is not the case, since there are at least three ways in which consensual transactions might be invalidated, and they include **fraud**, **exploitation** and **coercion** [101].

12.3 The 'IoT and Ethics' case study

This section is a tutorial that makes use of a case study published in *The Orbit Journal* in 2019 [102], and in particular in the special issue entitled *Case Studies of Ethics and Human Rights in Smart Information Systems*, which visits the ethical implications of an IoT tracking application from a software designer's point of view, through interviews with the software development company. The ORBIT project is funded by the **UK Engineering and Physical Sciences Research Council**. Its purpose is to provide services to promote Responsible Research and Innovation (**RRI**) across the ICT research community. RRI aims to ensure the sustainability, acceptability and desirability of research processes and outputs.

The Case Study took place within a European funded project (acronym: SHERPA), as one of several case studies that investigated the **ethical implications of developing and using emerging technologies**, like the IoT, AI, Big Data, etc. SHERPA stands for 'Shaping the Ethical Dimensions of Smart Information Systems – a European perspective'.

The Case Study tackles the following aspects: Interviews with two software designers of IoT-powered tracking applications; Information about how the company provides tracking software as a service nationally and internationally; discussion and conclusions after consideration of both the design and usage of such software in relation to ethics. In addition to ethical implications, any relevant social, economic and legal implications are also identified.

An overview of the IoT based application is offered. Its purpose is to assist and enhance specific business processes. The company employs the principle of **informed consent,** however, the **terms of use** are often not fully understood (note: **data collection** and **data sharing** are often included in such terms). When IoT and Big Data are used to create a Smart Information System, any data collection, any data manipulation to generate useful information, and any decision making based on generated information, must be considered.

The case study employs relevant literature review. Issues arising from literature review (see published case study for more information) include: effects on employer-employee relationship [103], consideration for employer's motivation [104], and a need for guidelines [105] so that the employers can make sure that the employees understand the circumstances under which the monitoring can take place. In a nutshell, employees who do not perceive

much privacy, tend to view their organisation's policies as less fair, they tend to trust upper management less and overall, demonstrate less commitment to their organisations.

When discussing a technology user, we need to consider the user's Quality of Experience (QoE) in interacting with technology. QoE is the degree of **delight** or **annoyance** of the user of an application or service. It results from the fulfillment of his/her **expectations** with respect to the **utility** and/or **enjoyment** of the application or service in the light of the user's personality and current state. According to the International Telecommunications Union, QoE can be measured in the following ways:

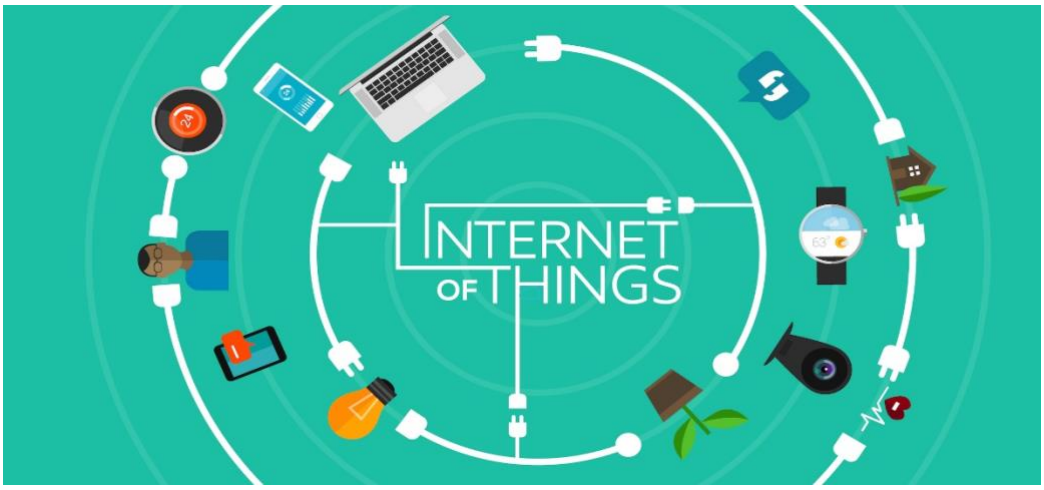
- Either by a subjective QoE assessment, typically based on Mean Opinion Score (MOS) of a service according to user perception,
- or, by an objective QoE assessment, typically involv-ing Quality of Service (QoS) parameters like latency, traffic volume density, reliability and cost etc.

QoE is related to Ethics. Ethics refers to a preferred action or state based on user-centred paradigms, and so does QoE. QoE is more closely related to a Utilitarian approach as it literally evaluates a utility function to quantify the user QoE (using either of the quantification methods mentioned above).

The specific details that are outlined regarding the subscription and billing management application that the case study explores, as an indicative application of IoT-based monitoring is presented as part of exploring the case study and some in class tasks are given for the students. The tasks explore both real-life scenarios that result in ethical dilemmas or some aspects of the day to day practices that may be ethically vulnerable. The Case Study tasks offer the opportunity to students to discuss both sides of these issues with their tutor.

13 Key-Enabling Technologies and Applications in IoT

Author(s): Samiha Falahat
Moath Alsafasfeh
Saud Althunibat



13.1 Introduction

Enabling technologies play main rule in IoT, these technologies explain how to transfer, analyze and process the contextual information between different nodes or machines in IoT network. One of the best scenarios that explain the meaning of enabling technologies in IoT is applying these steps; identify IoT nodes, localize their position, or identify their mobility path. Also, to keep IoT working, we need to reserve sufficient power resources. This can be done by energy management, power optimization, energy harvesting …etc.

IoT Enabling technologies include four divisions:

1. Data sensing technologies (ex: sensors).
2. Techniques that allow items to analyse data.
3. Facilities that enable things to enhance privacy.
4. Technologies help in taking control of actions
5. Technologies that enable power consumptions in IoT.

This chapter will discuss in detail some technologies related to identifications, mobility, positioning, IoT power optimization and energy harvesting.

13.2 Identification

The clear identification of things and people are primary requirement for communicating information with real entities in our environment. [106]. In IoT platforms the Identification process remains a challenge, that's because of heterogeneity between the interconnected platform. [43].

Identification can be divided into three categories as shown in Figure 13-1 Identification in IoT, where each category is explained in detail in next section:

- Object identification which represents physical or virtual objects using different techniques, such as RFID, Barcode, and biometric or vision-based detection.
- Communication identification which identifies the machines and nodes with communication capabilities on the network (ex: IP, OID, etc…)

- Application identification which identifies the service layer application, objects, and logical entities (ex: URI, URL).

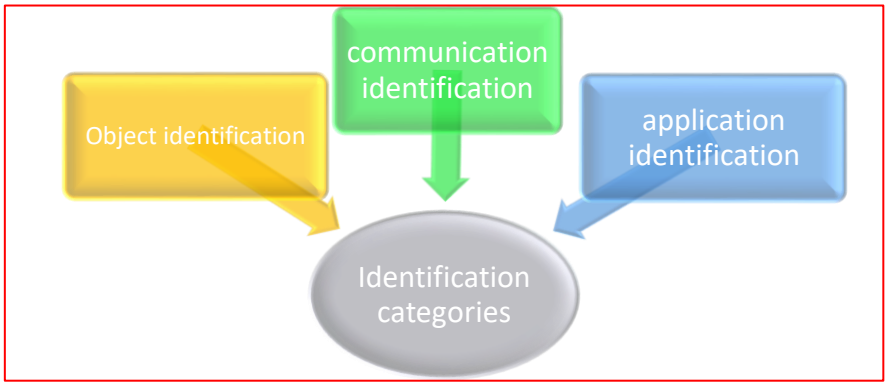


Figure 13-1 Identification in IoT

The main definition of object identification is defining the objects inside IoT network [107]. Currently, there are various efficient sophisticated methods that collect and read the input data then come out with a precise object location. RFID, Biometric, Barcode are featured object identification techniques.

13.2.1 Radio Frequency Identification (RFID):

Figure 13-2 shows the main parts components of an RFID system; reader (includes sender, receiver, and processing unit), Tags, and Antenna.

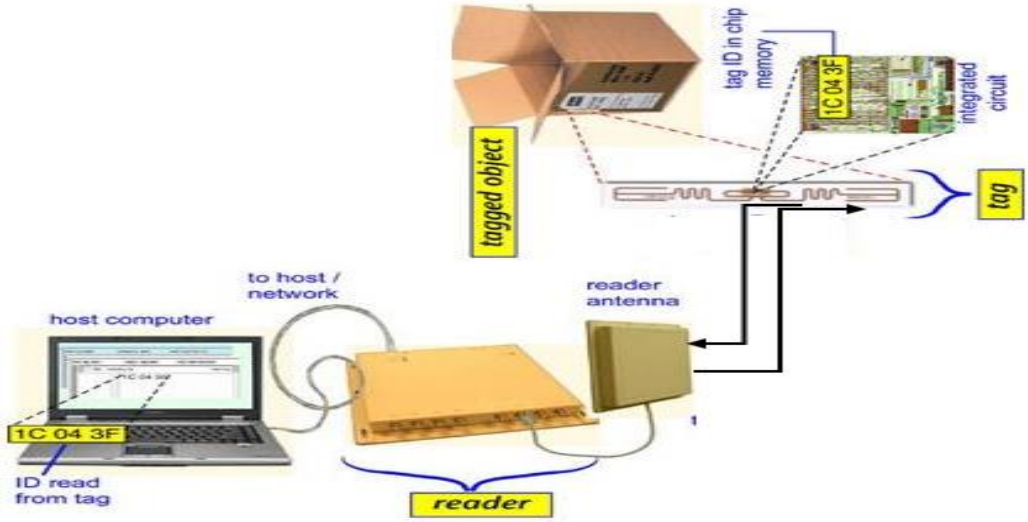


Figure 13-2 RFID system structure

source: https://www.nist.gov/image/14adlprfid-system-overview-graphicjpg

1. Reader: Reader is the intermediate component which connects the host server or computer with a tag through the reader antenna. The RFID reader consist three main components as shown in Figure 13-3. Radio signal generated and transmitted through the upper antenna, while the reader receiver detects the signal through the lower antenna.

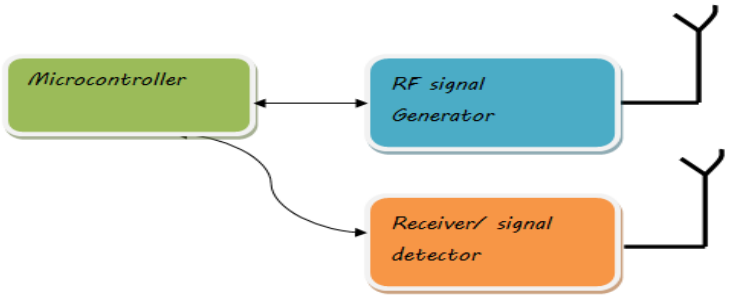


Figure 13-3 Reader Anatomy
source: https://www.youtube.com/watch?v=Ukfpq71BoMo

2. Antenna: The main function of the antenna in RFID is linking the reader with the tag by transfer and receive radio frequency signals between both sides. RFID system has very wide frequency [108].

3. Tag: which is an integrated circuit coupled with an antenna that may come with variant shapes and sizes. There are three classes of tags; passive tags which is smaller and cheaper than active tags that's need for powering from the reader before data transmitting, and the last is semi-passive tags. The tag contains four basic components (the controller, rectifier circuit, transponder, and memory).

4. RFID operating frequencies

There are several different frequencies an RFID system can use, the most common are:

- low frequencies (LF) (frequency<135 KHz)
- Radio High frequencies (HF) (frequency ~=13,56 MHz)
- Ultra-high frequency (UHF) (frequencies almost equal 434 MHz, 869 - 915 MHz and 2.45 GHz);
- Micro-wave (SHF) (frequencies near 2.45 GHz). [38]

13.2.1.1 Energy transmission modes in RFID

The most common transmitted powers are in the hundreds of milliwatts range and are determined by the device context, power block limitations, antenna configurations, and electromagnetic environment. *[38]. Th RFID tag power ranges from µW to mW, depending the circuit structure and internal components.*

The tag power sources classified into:

- Active tag: which has combined power source (ex: battery), due to that, the operational range of active tags is usually much greater than that of passive tags. It can start communication to a reader or other active tags.
- Semi-passive tag: has an embedded internal battery (long reader rang with high cost) and unable to initiate communications to reader.
- Passive tag: has no embedded Internal power source, has less operational range than semi passive tags.

13.2.1.2 Data communications modes in RFID

The frequency of operation defines the data transfer theory. The working principle for low and high frequency operation is inductive coupling, while the working principle for ultra-high frequency operation is electromagnetic coupling.

In inductive coupling operation, the reader produces a field that is used to match the RFID tag antenna. Mutual coupling causes a voltage to be generated in the RFID tag's coil; A portion of this voltage is rectified and used to power the controller and memory modules. as shown in Figure 13-4.

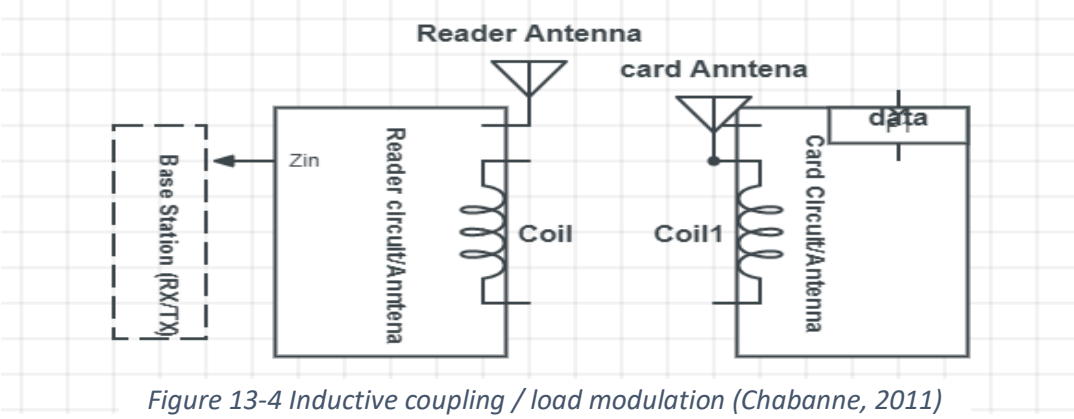


Figure 13-4 Inductive coupling / load modulation (Chabanne, 2011)

In order to send data, when the power is sufficient, by connecting the load of the coil, the current will start to flow through this load, this current will change according to changes in the load impedance. Suppose the load is turned on and off. It seems also the current will alternately turn off and on, causing a voltage to be produced in the RFID reader. The load is turned on and off. The load Turning on and off called load Modulation.

In Ultra-High-Frequency ranges, the effect of coupling is electromagnetic When an RFID obstacle collides with an incident wave released by the reader, it is mirrored back to the reader. [38]. Figure 13-5 explains the working principle of backscattered signal.

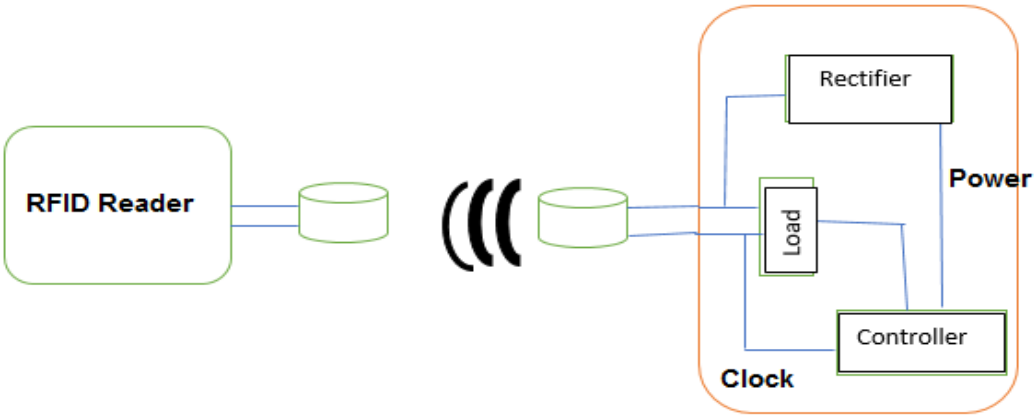


Figure 13-5 Electromagnetic coupling (back scattered signal)

source: http:/www.youtube.com/watch?v=Ukfpq71BoMo

13.2.1.3 Protocols of RFID communications

A communications protocol is the Rules govern the conversation between devices. in RFID, the conversation done between tags and a reader to ensure that data get transferred. RFID has two main communication protocols:

1. Tag-Talk Only (TTO) protocol: the data sent by the tag on regular basis when available.
2. interrogator Talk First protocol: the connection initiated by the reader and when the tag reaches the reader magnetic field, it waits for from the reader to make a request before sending its ID code.

13.2.1.4 RFID Data Coding

The Main three RFID coding types are 1-Binary-voltage level association, 2- signal transition coding, 3-Manchester coding. Figure 13-6 shows these coding operations and other coding techniques such as Miller coding, Bi-phase space codding (FM0), PWN, PPM, PIE. The figure explains the differences between coding techniques mentioned above.

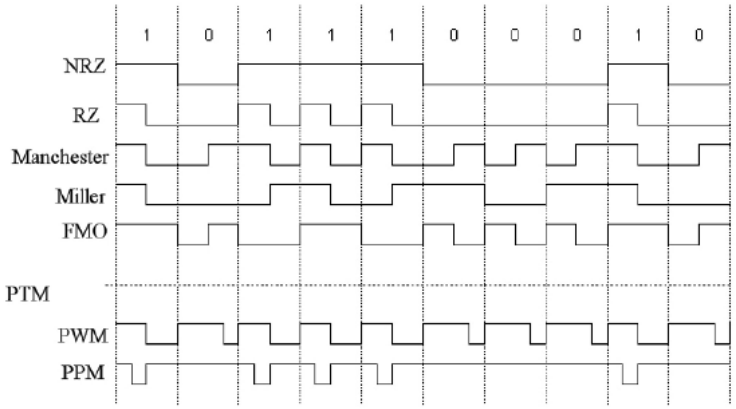


Figure 13-6 coding techniques used in RFID (Chabanne, 2011)

13.2.1.5 Anti-Collision Protocol

The reader should be able to identify tags as quickly as possible. Signals in both directions between the reader and the tag will collide since they communicate over a common wireless channel. As a result, the reader may not be able to identify all tags, or the process of tag recognition may take a long time. An Anti-collision strategy are available in some RFID systems, which solve the issue of tag messages cancelling each other out. [107]. Choosing the Anti-Collision protocol depend on several factors like algorithm performance, bandwidth limitation, implementation costs, noise tolerance, signal integrity, and security. Because of these reasons, the anti-collision method is applied by the majority of RFID systems using the temporal distribution technique. [38].

We will briefly discuss the deterministic anti-collision and probabilistic anti-collision protocols.

13.2.1.6 Deterministic Anti-Collision Protocol

For each RFID tag, the deterministic algorithms attempt to find a unique identification number {UID}. In this scheme, a memory should be implemented because the reader needs to choose from a field that contains a list of tag identifiers. One of the most common

approaches in deterministic protocol is binary tree search algorithm. The principle of work of this algorithm, initially, the reader sends a query to investigate if there are any tag within the magnetic field. The tags involved in field will make a response with respect to a given time. If multiple transponders reply simultaneously, due to their UID's, a collision happened and then it is requiring to detect the collision [38]. The reader sends a request with the number of qualified bits after detecting a collision in a particular bit, which is immediately followed by a bit assigned to 1. (It can be 0 if the reader's designer decides on that value.).

13.2.1.7 Probabilistic Anti-Collision Protocol

In a probabilistic protocol, each tag data packet has a specific transmission time or time slot. A collision occurs when two or more separate tags select the same slot for sending their answer, and all data is lost. The benefit of the probabilistic scheme is that all RFID tags can be detected and recognized with a single instruction, eliminating the need to contact each individual tag.

13.2.1.8 Applications of RFID

RFID emerged quickly with many life aspects, it's used in agriculture, medical, and used widely in industry. Some RFID applications examples are as follows:

1. **In-store traffic patterns:** RFID In-store application benefits are: monetizing high-traffic end caps, monitoring cart or package abandon in-store, eliminating physical pinch areas, etc. Moreover, Employee movements could be tracked using RFID tags in the store. RFID can be used to monitor inventory and equipment movement, providing useful data by observing how equipment moves through the store's physical space.

2. **RFID in Libraries and bookstores**

   Associating books with tags significantly simplifies the control of inputs and outputs in a library. There are various applications.

   – facilitate the procedure of inventory;

   –feature of Anti-theft can be added to tags;

– easy lend and return procedures. Automatic return terminals may also be deployed [38].

13.2.2 Barcode Identification Technique

Barcode is a visual representation of information. The basic Barcode technology advantage is it has low rate of error (Generally, the number is less than one million.).

In detail, the barcode is series of parallel printed lines (bars). These lines may have variable width (mainly two widths available; thin and thick). The computer read bar in form of 0 or 1

13.2.2.1 Barcode work principle

The information found in the bar code can pass from the scanner to the device where the code is detected when the bar code reader or scanner is used to scan the bar code. Figure 13-7 demonstrates the idea of a bar code reader's reading.

The main steps of Barcode working principle can be summarized in the following:

1. The laser beams emitted from the laser diode hit the polygon mirror and scan a bar code.
2. The diffuse reflection light is obtained by the light-receiving portion (photodiode).
3. As seen in the below figure, the defuse reflection assembles an analogue wave
4. Analog to digital process done by the bar code reader.
5. Digital signals are used to recognize narrow/wide bars and narrow/wide spaces.
6. Data decoding done then the output is generated.

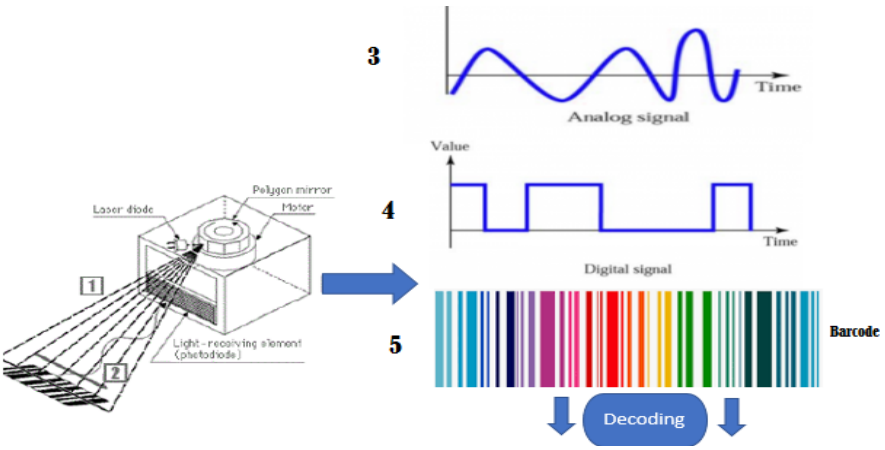


Figure 13-7 laser barcode reader (reading principle) [106]

13.2.2.2 Barcodes types

There are many types of barcodes, they can be classified into two main types:

- One-dimensional barcodes:

Figure 13-8 One- Dimensional Barcodes source
https://www.britannica.com/technology/barcode

- Two Dimensional Barcodes

Two-dimensional (or 2D) barcodes use two-dimensional symbols and shapes to systematically represent data, as shown in *Figure 13-9.*

Figure 13-9 Two-dimensional Barcodes source:
https://www.britannica.com/technology/barcode

13.2.2.3 Barcode Applications and uses

Barcode is commonly used in many sectors; here some examples of barcode uses.

- **Advertising**

Advertisers exploit the barcodes capabilities by using them to reach out to customers in a more interactive way. Simply by downloading and installing an app that can read barcodes on a smartphone, you can learn a lot more about the product being advertised.

- **Tracking food intake**

It's pretty good way to know detailed information about the food Ingredients, calories, product or expiry dates and so on. This is done by scanning the bar code on the food wrap using phone application then the user will be able to read all the food information.

13.2.3 Biometric Identification

Biometric is a technique that help in identify and authenticate a person based on set recognizable data, which are unique and specific to a person [109].

Biometric authentication is the method of determining similarities between data for a person's characteristics and data for a person's biometric "sample."

Biometric identification identity the person or objects.

There are two main categories of biometric; the Physiological features and behavioral features:

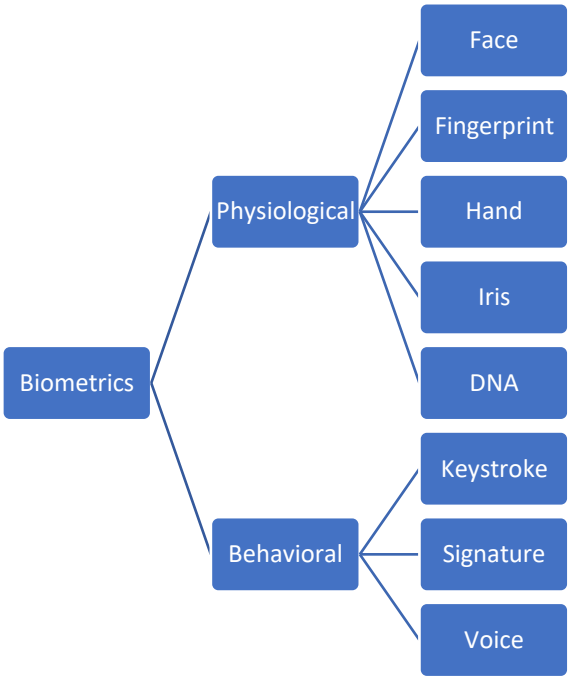


Figure 13-10: biometric physiological/behavioral

Among the various biometric ID techniques, the physiological methods (fingerprint, face, DNA) are steadier than behavioral methods (keystroke, voice print). That's because of physiological features are often immutable except by harmful injury [110].

Examples of Biometric recognition techniques and applications

- **Face recognition**: Is based on a detailed review of the facial features as well as the relative locations of the eyes, nose, and mouth. Face images can now be collected using both traditional video and thermal imaging techniques. Thermal imaging creates facial images using a hot wire formed by blood in the capillaries of the face, while normal video captures images of the face by a camera. [109]
- **Fingerprint Recognition**

  Fingerprints retain their permanency and uniqueness over time. Fingerprints offer more secure and accurate personal identification than passwords and id-cards, according to experiments. Examples such as password protection by implement finger-print devices with computers and smart phones.

13.2.4 Comparison of identification techniques

By comparing between barcodes, biometric, and RFID identification techniques we conclude that the RFID is the most effective and reliable technique. It offers unrivalled benefits in a variety of areas, Transmission distance, reading and writing speed, anti-interference capability, and service life are just a few of the factors to consider. Despite the fact that RFID has a higher manufacturing cost than barcodes and visual codes, it has proven to be an efficient method for constructing IoT systems. [107].

In barcode there is only reading mode, in regards of confidentiality barcode bad choice. has short lifetime, the cost is low and cheap. The biometric also works only in reading mode but it has good confidentiality, long lifetime, and high cost.

13.3 Localization

Localization or positioning term refer to determining the location of an object (ex: Sensors in IoT Network) or human in specific area, or to determine the spatial relationships among different objects.

Localization is important to provide a real physical context to sensor readings, for example, in environmental monitoring, location identification is necessary to sensor reading. Moreover,

the Location information is important for services such as intrusion detection, and surveillance systems [27].

In IoT, for location estimation, localization techniques may be used, which can either externally localize an object or enable an object to decide its own position. The United States' Global Positioning System (GPS) is an example of a global positioning system. [106].

Many localization techniques depend on methods to specify the position of an object, these methods:

1. Geometric calculation such as triangulation (for instance In GSM networks, this is achieved by calculating angles with respect to fixed points or nodes with known positions.).
2. Trilateration (by measuring the distance between neighbouring nodes).
3. Scene analysis determines (called a “footprint”). A Real-time image of the landscape from the corresponding viewing angle or can be stored previously in a table with predetermined values of a point of view [106].

The main challenges of localization process are how to track the mobile object or moving object, and how to handle indoor object position.

13.3.1 Overview of Localization process

The process of localization can be summarized as follow and shown in Figure 13-11.

- Unknown node selection phase: an unknown node must be defined that has at least three reference nodes in its neighbour area, then pick unknown node s, and obtain an ambiguous position for this node.
- Distance estimation phase This involves selecting a reference node, determining its local location, and measuring the distance to the reference node.
- Position computation phase: measure the position of the selected unknown node
- Localization algorithm phase: trigger and activates the algorithm that’s attempts to identify the unknown node location.
- Output phase: the output represents the estimated location (can be in different forms).

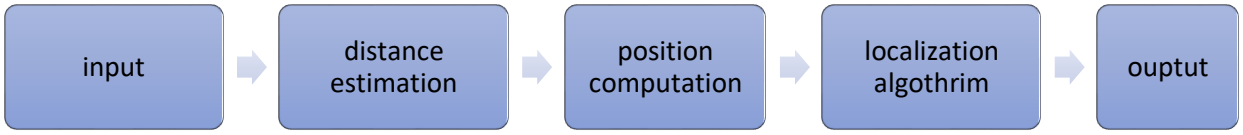


Figure 13-11 Localization process [111]

13.3.2 Localization techniques classification

Techniques for localization can be categorized as centralized, decentralized, or distributed. The classification of localization techniques is shown in Figure 13-12.

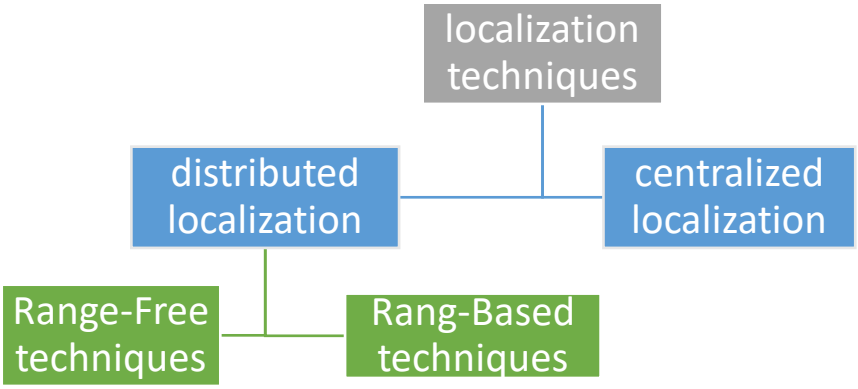


Figure 13-12 localization techniques

In distributed localization, each sensor node calculates the estimated location for itself, and communicate with other sensors in the same area to get their information. There are two types of distributed localization schemes: range-based and range-free.

All measurements are clustered at the fusion Center, where the calculation takes place, in centralized localization. The findings are then sent back to the nodes. Latency, energy usage, and bandwidth consumption are all effects of data transmission in the network.

13.3.3 Positioning systems

Device or nodes positioning and guidance are complicated by the variations between indoor and outdoor environments. Outdoor positioning requires regional or even global coverage, while indoor environments are limited to rooms and houses. [112].

There are many methods and techniques could be applied for both indoor and outdoor localization. These technologies are sometimes classified as network-based or satellite-based systems. Another classification is dependent on whether the positioning solution is performed by the mobile user or the base station, resulting in mobile terminal (user)-centric (such as GPS, A-GPS, E-OTD), network-centric (COO, TOA, TDOA, AOA, RSS, multipath pattern matching), or hybrid solutions (See Figure 13-13).

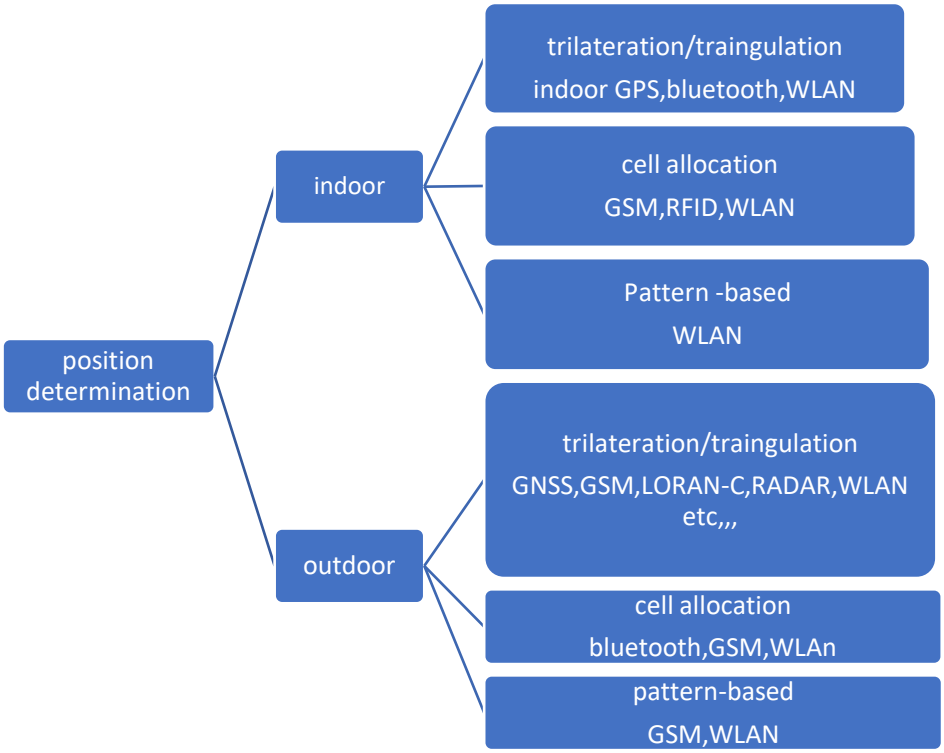


Figure 13-13 localization/positioning methods [27]

13.3.3.1 Outdoor Positioning Systems

Only GNSS systems are able to provide sub meter accuracy in outdoor location. The other alternative are cellular based systems, where the accuracy varies with several parameters and even in best conditions, achieved accuracy lies in the order of meters.

Differential GNSS is developed extension of GNSS that provides higher system precision, it is derived from the principle of positioning error differentiation, thus increasing the accuracy of measurements. The main disadvantage of DGNSS, which is the fact that the rover must be near the base stations in order for the corrections to take effect.

Another method for precise outdoor localization is called Precise Point Positioning (PPP). This method only needs a standalone receiver to operate. PPP relies heavily on post processing, with very precise satellite ephemeris, clock information and dual carrier receivers, the solution is able to deliver sub meter accuracy. The very long convergence and offset of moving rover are main drawback of PPP systems.

13.3.3.2 Indoor Positioning Systems

Ultra Wideband (UWB) is one of the most used radio technologies capable of providing sub meter positioning accuracy in indoor environments. UWB communication is strongly immune to multipath and noise. Since the UWB transmit power lies at the noise floor, the technology is also highly resistant to interference; due to very low transmit power. Many common indoor positioning technologies like Wi - Fi and Bluetooth or RFID still get fairly accurate results in the order of meters, plus the overall price is generally lower than with UWB solution.

13.3.3.3 Factors influence the selection of localization techniques

positioning methods optimal requirements for IoT network are

- **Accuracy**: it defines distance mismatch error rate between the measured distance and exact device location
- **Responsiveness**: In localization, responsiveness can be defined as the speed of location update for specific node.
- **Coverage**: it determines the network coverage area interferences problems.

- **Scalability:** if the system works within in expanded areas, its performance is measured in terms of scalability. Less scalability means less performance.
- **Size and devices costs**

13.3.4 Ranging Techniques

many Ranging methods have been investigated by researcher in previous years. The relative distance is main value to measure, measuring methods like time-of-arrival , time difference of arrival , or propagation model generated from RSSI value used to find the relative distance measurements [113].

Range-based schemes are vulnerable to their surroundings, and obstacles may cause errors.

13.3.4.1 Time of Arrival

Time of Arrival's main concept relies on determining the distance between the transmitter and receiver using measured propagation time of signal, and predefined signal velocity. For example, a radio spectrum propagates at the light speed (around 3 x 10^8 m/s), that is, the signal only requires around 30 ns to travel 10 m. As result, is that radio-based distance measurements require high resolution clock, which increase the cost and complexity of a sensor nodes network [27].

Time of arrival method divided into two main schemes;

1. *One-way* time of arrival method: this way to calculate the difference between signal transmitting time and receiving time, that's based on distance between two nodes.
2. Two-way time method: calculate the Round-trip time (RTT) at the transmitter node

Figure 13-14 shows the two method time measurements

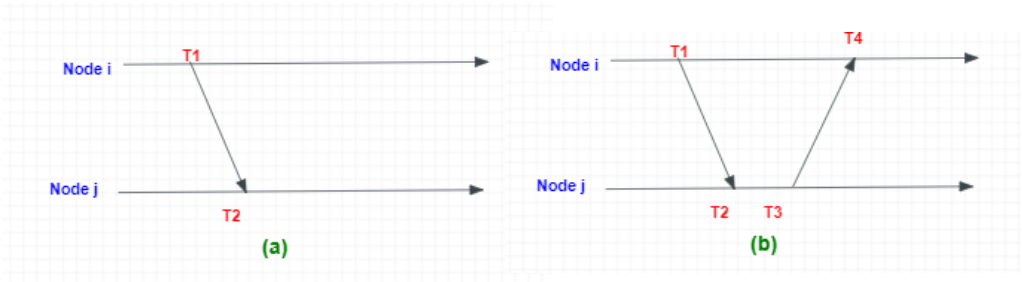


Figure 13-14 (a) one-way scheme. (b) Two-way scheme [24]

13.3.4.2 Time Difference of Arrival (TDoA)

In TDoA approach two different signals with different speed were travels together. the position can be determined at the receiving node (See Figure 13-15). For instance, the first signal is radio signal (t_1 :issued and t_2:received), followed by an acoustic signal either immediately or after a constant time interval [27] . Therefore, the receiving node can calculate the distance by:

$$d_{ij} = (v_1 - v_2)\,(t_4 - t_2 - t_{wait})$$

No synchronization clock needed between the sender and receiver. this approach has extra hardware costs

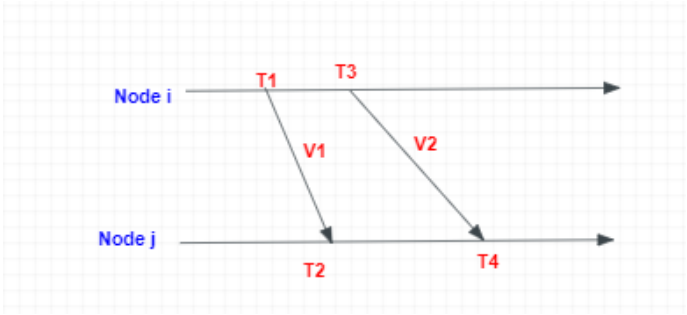


Figure 13-15 TDoA Scheme [24]

13.3.4.3 Propagation Model

the relative distance is calculated by the obtained signal intensity (RSS). When a computer senses a radio signal, it may use the propagation model and RSS to calculate the distance.

13.3.4.4 Angel of Arrival (AoA)

The basic AOA concept is to find the direction of propagated signal, typically, massive array of antennas is needed [27]. There is no need for clock synchronization. however, this technology suffers from a low accuracy because of the performance of the angular estimator.

13.3.5 Range-base Localization

13.3.5.1 Triangulation

Triangulation depends on triangle geometric characteristics to estimate node locations

Figure 13-16 shows how to determine the unknown location of anchor using three angles measurements by use (AoA).

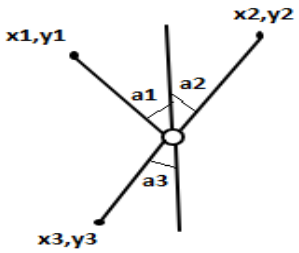


Figure 13-16 triangulation [24]

13.3.5.2 Trilateration

Trilateration is known as the procedure of calculating node's location by measure the distances between itself and a number of anchor nodes with known locations. (See Figure 13-17)

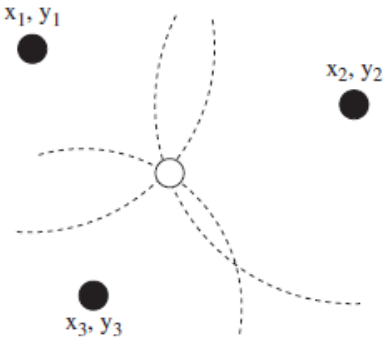


Figure 13-17 trilateration [24]

13.3.5.3 Iterative and Collaborative Multilateration

Multilateration (MLAT) is a navigation technique that uses the difference in distance between two stations at known locations that transmit signals at known times to calculate the distance between them. There are two main multilateration method. **The iterative multilateration** where the nodes in network exchange their estimated location with each other in repeated manner until they are totally localized. The second, the **collaborative multilateration**, in order

to estimate the location of the node, it uses the multi-hop information from neighbour node then attempt to calculate the possible location for both nodes.

13.3.6 Range-Free Localization

In range-free schemes, there is no need for additional hardware needed for distance estimation process; it simple and low cost because of these features range free techniques have attract research attention in recent years.

13.3.6.1 Approximate Point in Triangle (APIT)

In APIT usually, the nodes know their locations which they equipped with high powered transmitters A triangular region is formed by any three points, and a node's may locate within or outside, such a region allows a node to narrow down its possible locations. The Point in Triangulation test, which enables a node to evaluate the set of triangles within which it resides, is the first step in APIT localization. After a node N has received position messages from a set of nodes, it evaluates all possible triangles formed by these nodes [27]. Figure 13-18 represents APIT test scenario.

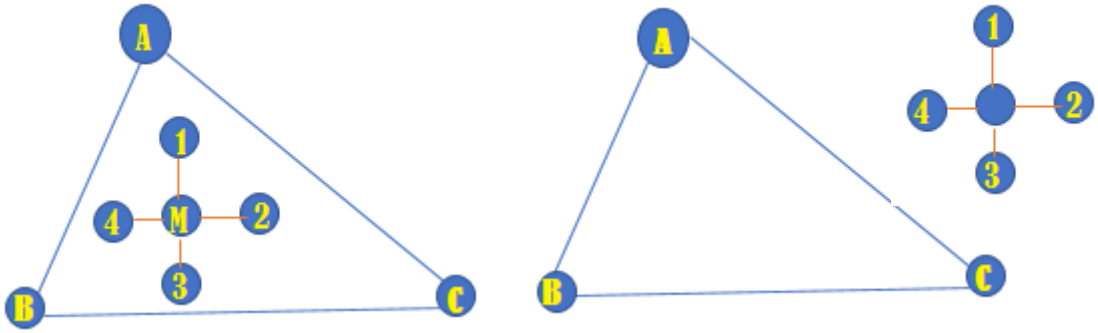


Figure 13-18 APIT test scenario

13.3.6.2 Ad Hoc Positioning System (APS)

In APS, one node that knows its location, broadcast a message includes the location with hop count. every receiving node store the minimum value. After ignoring higher values. The location propagation to all neighbours done using distance vector algorithm. Each node has routing table which exchanged and updated periodically with other nodes. Both nodes in the network, as well as other anchors, are given the shortest distance in hops in the DV-hop

scheme. The following equation can be used to calculate the total single hop distance in anchor (i) [114].

$$c_i = \frac{\sum \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}}{\sum h_i}$$

where anchor j is at position (x_i, y_j) and h_j is the distance in hops from j to i .

13.3.6.3 Fingerprint –based indoor localization techniques

Because of its high accuracy compared to other methods, fingerprinting is a common method of localization. It has a low degree of sophistication, needs no line-of-sight measurements of access points, and has many various applications in the complex indoor environment.

Visual fingerprint-based localization, motion fingerprint-based systems, and signal fingerprint-based methods are all examples of fingerprint-based methods used in indoor localization.

1. Visual fingerprint-based localization

In this scheme the devices can collect their position through image processing and matching procedure. The ability of image processing a feature must combined with these devices. the process is start after taking the photo by a camera which embedded in device, which been analysed by specific computing methodologies, next we need to use AI algorithms to gather the features and learn how to react. Then image processing techniques and algorithm activated to find the matching.

Matching speed is the biggest challenge in visual fingerprint-based localization. In addition, camera operation also consumes too much of the device's battery energy. Moreover, this type of localization method is still not capable of achieving high precision.

2. **Motion fingerprint-based localization**

In this method, the primary concept is to detect user motion data, which is collected using embedded sensors such as an accelerator and a gyroscope. For example, some moving patterns or user paths can be found to match a predefined motion signature. The major

disadvantage of this approach is that the sensors can generate noise, and the occurrence of an error can be accumulated over time.

3. Signal fingerprint-based localization:

Signal fingerprint-based localization is commonly deployed in areas where a large amount of Wi-Fi infrastructure and services are implemented, particularly in the enclosed environment. Some outdoor applications use this scheme as well. These positioning methods typically include phase of offline training and the online fingerprint matching sequence. The main objective of the first stage is to develop a fingerprint database that stores the similarity between the (RSS) patterns from different access points and the locations to be fixed. The location of the device is then determined at the matching stage. A high accurate matching algorithm used to search the fingerprint in database and find the minimum differences with the device need to be located. [113].

13.3.7 Comparison of localization techniques

There are key differences between the several localization techniques, it could be summarized in Table 13-1.

Table 13-1 localization techniques comparison [115]

Techniques	Cost	Accuracy	Energy efficiency	Hardware size
Centralized based	depend	high	less	Depends
Distributed based	depend	low	high	Depends
received signal strength indicator	low	Medium	high	Small
Time of Arrival	high	Medium	less	Large
Time Deference of Arrival	low	high	high	Large, but less complex
Angel Of Arrival	high	low	Medium	Large
Distance vector hop	low	Medium	high	Small

approximate point in triangle	Medium	Medium	high	Medium

Summary

Since position information is considered basic information in many applications, localization is a very important topic in IoT. Many researchers have worked on it, and a number of algorithms and techniques have been proposed. To achieve higher localization accuracy, each technique has specific features and operations.

The most significant factor in determining localization is accuracy. Indeed, the majority of IoT implementations require extreme accuracy. With specialized hardware, Range-based schemes usually produce better precision based on node-to-node distances or angles.

13.4 IoT Power Management

Power consumption is a very critical issue in IoT technology due to several reasons. First, IoT nodes are usually small, and hence, cannot be equipped with heavy/large power resources. Also, IoT can be distributed in harsh environments which make battery charging or replacement is a challenge. As such, not all power supplies are sufficient for IoT applications. They must be highly effective at both low and full load, space-saving, reliable, and, most importantly, affordable, because they will be as popular as the sensors, processors, and actuators that they support.

Power management techniques are used in IoT devices in order to minimize energy waste and make the IoT system much more effective. Since multiple aspects have an effect on battery life, In the area of IoT devices that use batteries for a longer period of time, power management is a major challenge.

The sensor devices normally get their power from a battery source. Figure 13-19 shows a standard IoT power management architecture. Since the rectified input is exchanged with the switching regulator, it can be used in many areas. The output of the sensors, for instance, is sent to the switching converter, and then the output of the DC-to-DC converter is sent to the unit of power management, which is responsible for the IoT system's energy harvesting.

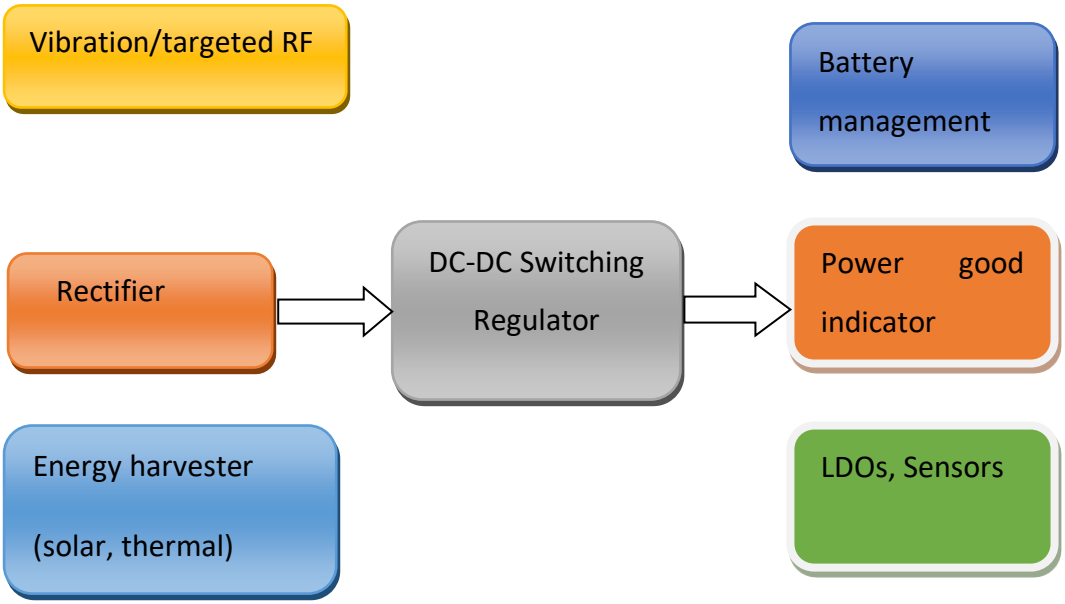


Figure 13-19 architecture of typical power management of IoT node

The following are the most relevant power management elements:

- Voltage regulators: which take input voltage and regulate it into another application. For this purpose, we can use (linear regulator, buck convertor, boost convertor, or buck boost convertor). The regulator needs to step up or down the voltage between the battery and different sub-circuits in the device. Step-up functionality is required by high-voltage devices, while the step-down function helps reduce the power consumed by digital CMOS circuits. This provides longer battery life and enables new features like additional camera.
- AC-DC Controllers which take the AC out from main AC socket and regulate it into a stable voltage.
- The LED Drivers Block which is basically, making from a DC voltage constant current.
- Battery Management block to control the power in batteries, by charging batteries or checking the battery state of charge with battery gauges function.
- USB- C power delivery function block: it is a new technology used in power management.

Figure 13-20 depicts these function blocks with their associated techniques.

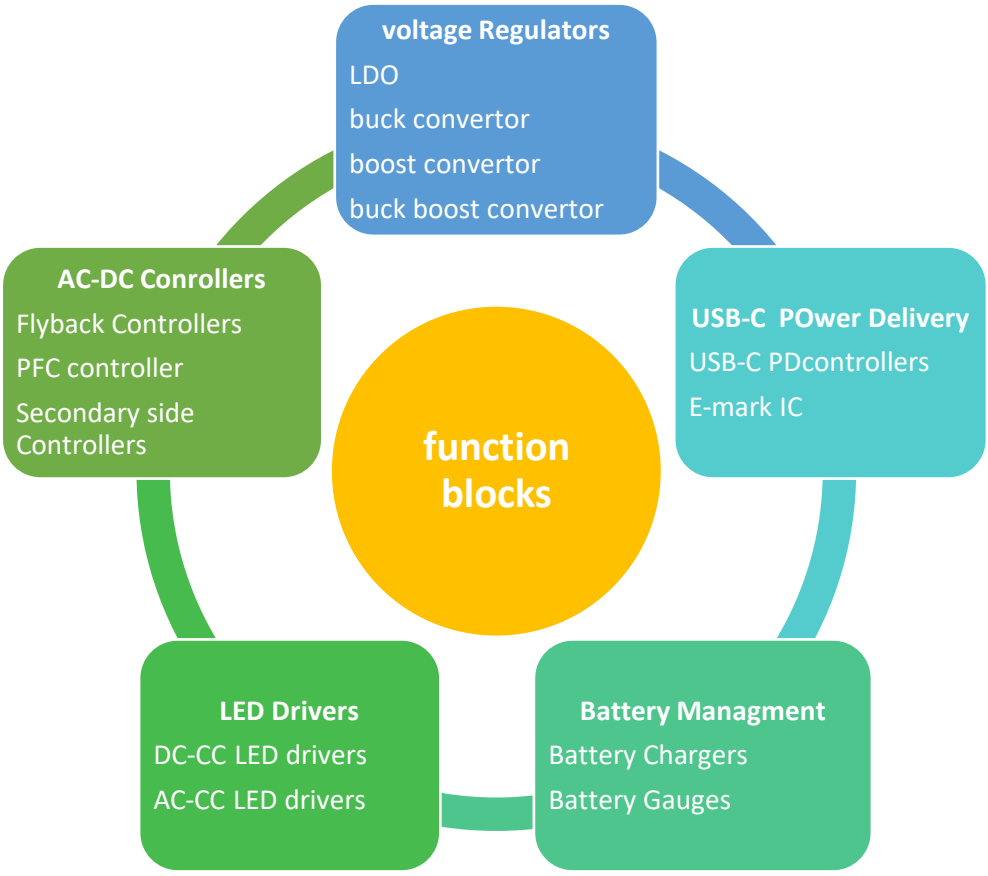


Figure 13-20 Power management Functions blocks source: https://www.youtube.com/watch?v=ut8cC5YXMj4&t=768s

13.4.1 Energy Harvesting

Embedded devices and remote sensors, for example, use batteries to power their electronics. Long-lasting batteries, on the other hand, have a finite lifetime and must be replaced every few years. Since there are hundreds of sensors in remote areas, removing them can be incredibly expensive. “Energy harvesting” is one technology being developed to solve this issue. Energy harvesting is the method of extracting energy from natural sources and using it to power machines. The purpose of energy harvesting is to boost the system lifetime while lowering repairs; this can be done by exploiting some sustainable resources, such as solar, vibration, wind and thermal gradients. Figure 13-21 shows the structure of energy harvesting technology.

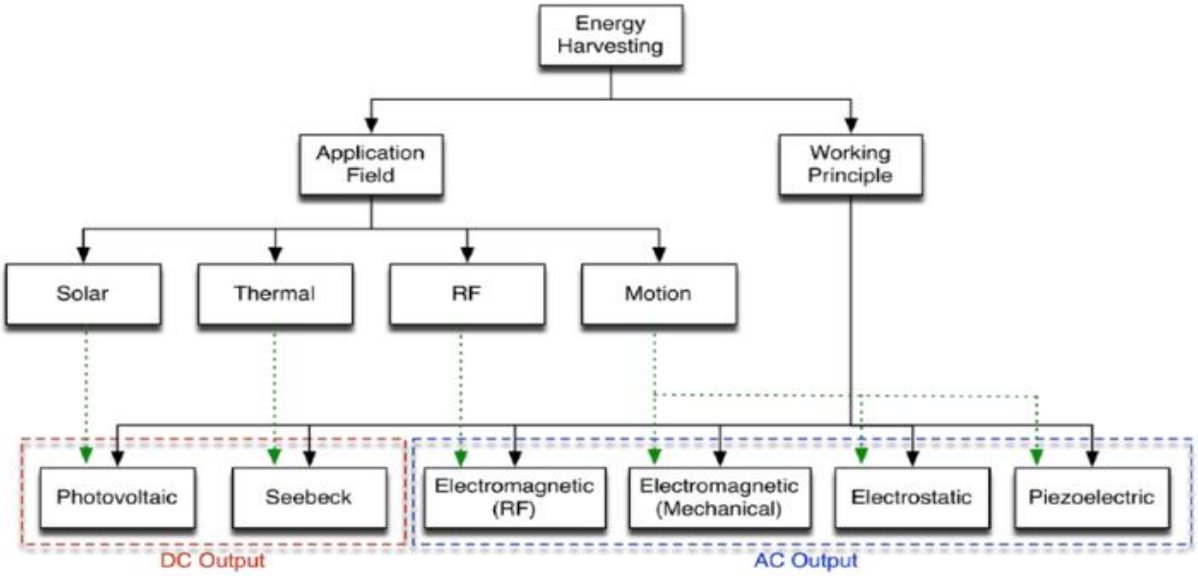


Figure 13-21 Different technologies for energy harvesting

13.4.1.1 Technologies of Energy Harvesting

1. Piezoelectric technology

The main concept based on conversion of oscillatory mechanical energy into AC electrical energy. This energy can be transformed into usable electrical energy, which can be used to power portable electronic devices such as sensors and GPS receivers. Some consumer electronic devices, such as cellular phones, can be powered directly using piezoelectric energy harvesting. [116].

When mechanical force or pressure is applied to such special materials, referred to as piezoelectric materials, the atomic structure of the crystal changes due to the net motions of positive and negative ions with respect to each other, resulting in electrical dipoles or

polarization. Thus, The dielectric crystal transforms into a loaded material. The amount of voltage produced is proportional to how much stress or voltage is applied to the crystal. [117].

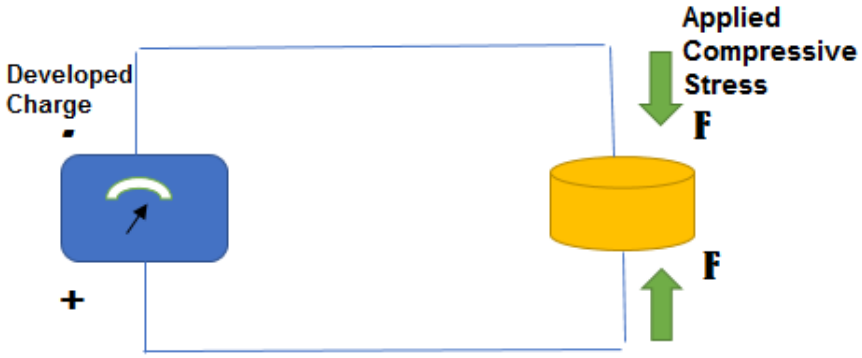


Figure 13-22: piezoelectric direct effect mode source: https://www.elprocus.com/what-is-a-piezoelectric-material-working/

Ionically bound piezoelectric materials have positive and negative ions in pairs known as unit cells. These materials occur in nature as an anisotropic dielectric with a non-centro-symmetric crystal lattice, meaning they have no free electrical charges and the ions have no symmetry centre.

Piezoelectric materials are classified into four types: ceramics, single crystals, polymers and composites. Piezoelectric ceramics are generally used as a piezoelectric material in energy harvesting machinery due to their low cost, good piezoelectric properties and ease of integration into energy harvesting equipment.

Electromagnetic Energy Harvesting

The electromagnetic energy harvester was developed to transform a frequency range of 100-200 Hz input source (such as vibration) into usable electrical energy.

This technology's basic idea is based on the magnetic field power that can be used to charge a battery in a fixed period of time. Faraday's law on electromagnetic induction (Faraday's law address that a current will be induced in a conductor when the magnetic field changed) is the core concept of electromagnetic energy harvester design (see Figure 13-23).

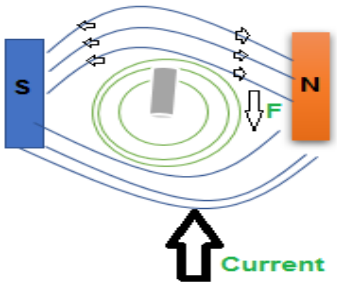


Figure 13-23 Interaction of Magnetic fields and current carrying conductor
source: https://www.electrical4u.com/

- **Linear electromagnetic energy harvester**

The linear electromagnetic energy harvester is based on the resonance principle. In other words, when the external source's frequency meets the harvester's fundamental frequency, a solid magnet is set in motion compared to a stationary coil, producing a time-varying current in the coil according to Faraday's law. [118]. The linear electromagnetic harvester has the potential to play a significant role in the harvesting of vibration energy in a number of sectors and industries.

3- Photovoltaic (Solar) Energy Harvesting

Photovoltaic solar collector transforms solar radiation (called insolation) into direct current (DC) electric power. Depending on the application (grid, off-grid, battery backup), photovoltaic solar solutions usually involve multiple panels linked together (called arrays), electrical disconnects, over-current safety (circuit breakers or fuses), inverters, junction boxes, and other special tools. Figure 13-24 shows the typical PV harvester with single panel.

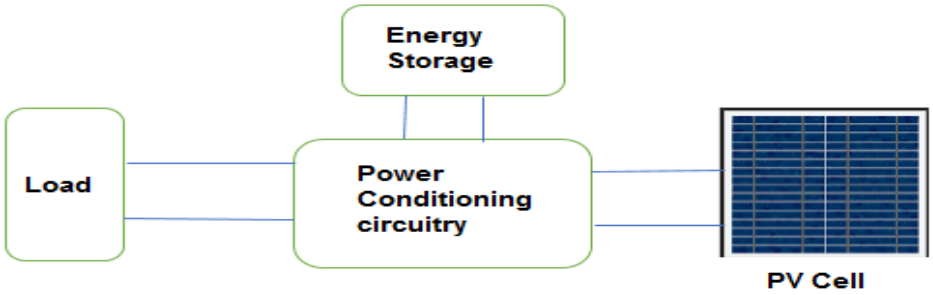


Figure 13-24 PV harvester system (Bizon, 2017)

4- Thermoelectric (Temperature Difference) Energy Harvesting

Thermoelectric energy harvesting based on the operation of the thermoelectric generator. A thermoelectric generator converts heat directly into electricity according to a phenomenon called as the Seebeck effect.

Thermoelectric harvesters provide clean energy for energy harvesting with a variety of benefits: they are no maintenance needed, due to the use of highly durable and lightweight solid-state equipment; they are silent and quiet; and they are highly effective in terms of the atmosphere, since heat is obtained from waste heating systems and transferred into electricity.

5- Electrostatic energy harvesting

A variable capacitor and a power transmission circuit make up an electrostatic energy harvester. as shown in Figure 13-25. The electrostatic effect that occurs when electrical charge is stored between parallel plates of a capacitor. The electrical energy harvesting is accomplished by modifying one of the variable capacitor parameters by changing one of the plates and shifting the other with an external mechanical motion: separation of plate. According to the theory of electrostatics, mechanical movement can be converted into electrical energy.

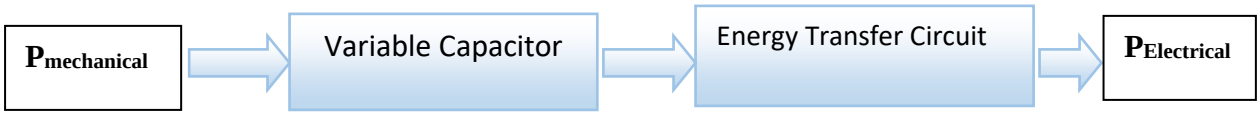


Figure 13-25 electrostatic harvester

Comparison of energy harvesting technologies

A brief comparison between different energy harvesting technologies summarized in Table 13-2.

Table 13-2 energy harvesting technologies comparison [119]

	Piezoelectric	electromagnetic	photovoltaic	Electrostatic
Advantages	-Robust -Easy to use -High output voltage -Large temperature rang	-High output current -Long life -robust	-clean and silent -Small-scale solar plants -flexibility	-High output voltage -Simple integration -Capturing low frequencies
Disadvantages	The conversion properties depend on piezoelectric element	-Low output voltage -Low efficiency -Heaviness	-expensive -variable energy source	-The requirement of a polarization source -Poor mechanical guiding -Complicated power circuit management

13.4.2 Battery technologies for IoT

Choosing batteries for IoT can be complicated, as there are a broad range of application types. Battery requirements for specific class IoT devices can be assisted by interpreting their physical, electrical and functional components. As a result, the physical size of the IoT gadget may confine the physical size of the battery that can be thought of. The operating

environment will determine if the choice of batteries is limited to those with modern car or business temperature assessments. Components and functionality will determine the resilience and strength requirements, as well as whether the essential or battery-powered battery is more eligible for the application.

In principle, batteries can be classified either as non-rechargeable or rechargeable cells, referred to as main and secondary cells, respectively. Batteries can be made from a wide variety of chemical substances, the essential chemistry of primary batteries called (alkaline), examples of primary batteries are: (AA, AAA, C, D-Cell).

Li-ion cylindrical and Li-polymer pouch cells are examples of secondary batteries (both made from lithium). Each chemistry and structural methods has its market in terms of price and performance, with lithium-based primary and secondary cells of different kinds having been widely used in recent years due to its relatively high energy density, higher relative to alkaline cells nominal voltage of 3.0–3.6 Volt in many of the cell types, and usually low self-discharge.

Factors influence batteries selection for IoT Application

When choosing a battery for an IoT unit, there are thousands of combinations of battery shapes, sizes, capacities, cell types, and other parameters must be taken in consideration.

Below, are the parameters must take in consideration to make battery selection.

-The nominal and cut-off voltage of an IoT application the output voltages of various technologies and chemistries vary. We must pick the one that holds the device above the cut-off voltage for the remainder of its existence.

- The environment's temperature: Think about where the IoT device would be implemented to ensure an optimum and consistent supply of power to the object.

- The overall pulse current and frequency, as well as the consumption profile: specify if the IoT platform requires a low or high pulse frequency.

Bibliography

[1] I. Analytics, "State of IoT Q4/2020 & Outlook 2021," [Online]. Available: https://iot-analytics.com/product/state-of-iot-q4-2020-outlook-2021/.

[2] Gartner, "Leading the IoT," 2017. [Online]. Available: https://www.gartner.com/imagesrv/books/iot/iotEbook_digital.pdf. [Accessed 2021].

[3] J. Koistra, "Newzoo's 2018 Global Mobile Market Report: Insights into the World's 3 Billion Smartphone Users," Newzoo, 2018. [Online]. Available: https://newzoo.com/insights/articles/newzoos-2018-global-mobile-market-report-insights-into-the-worlds-3-billion-smartphone-users/. [Accessed February 2021].

[4] ITU, "www.itu.int/internetofthings," International Telecommunicaiton Union, 2005. [Online]. Available: https://www.itu.int/osg/spu/publications/internetofthings/InternetofThings_summary.pdf. [Accessed 21 June 2020].

[5] D. Evans, "The Internet of Things: How the Next Evolution of the Internet in Changing Everything," Cisco Internet Business Solutions Group (IBSG), 2011. [Online]. Available: https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf. [Accessed 21 June 2020].

[6] O. Said and M. Masud, "Towards internet of things: survey and future vision," *International Journal of Computer Networks,* vol. 5, no. 1, pp. pp.1-17, 2013.

[7] W. Miao, T. L., F. L and I. S. a. H. D., "Research of the architecture of Interent of Things," in *3rd International Conference on Advanced Computer Theory and Engineering (ICATE '10)*, Chengdu, China, 2010.

[8] ITU-T, "Overview of the Internet of Things," *Series Y: Global Information Infracstructure, Internet Protocol Aspects and Next-Generation Networks: Next Generation Networks -*

Frameworks and functional Architecture Models, Vols. Recommendation ITU-T Y. 2060, June 2012.

[9] Y. B and H. G., "Supply chain information transmission on RFID and Interent of Things," in *International Colloquium on Computing, Communication Control and Management*, Sanya, China , 2009.

[10] P. S. a. S. R. S., "Internet of Things: Architectures, Protocols and Applications," *Journal of Electrical and Computer Engineering,* vol. 2017, pp. 1-25, 2017.

[11] Z. Yang, Y. Yue, Y. Yang, Y. Peng and X. W. a. W. Liu, "Study and application on the architecture and key technologies for IOT," in *2011 International Conference on Multimedia Technology*, Hangzhou, China , 2011.

[12] "A survey on IoT architectures, protocols, applications, security, privacy, real-world implementation and future trends," in *11th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM 2015)*, Shanghai, China, 2015.

[13] J. Gubbi, R. Buyya, S. Marusic and a. M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Generation Computer Systems,* vol. 29, no. 7, pp. 1645-1660, 2013.

[14] F. Bonomi and N. P. a. Z. J. Milito R., "Fog Computing: A Platform for Internet of Things and Analytics," in *Big Data and Internet of Things: A Roadmap for Smart Environments*, Berlin, Springer, 2014, pp. 169-186.

[15] F. Bonomi, R. Milito, J. Zju and S. Adepalli, "Fog computing and its role in the internet of things," in *MCC workshop on Mobile cloud computing*, 2012.

[16] M. a. H. E.-N. Aazam, "Fog Computing and Smart Gateway Based Communication for Cloud of Things," in *International Conference on Future Internet of Things and Cloud*, Barcelona, 2014.

[17] L. Atzori and A. a. M. G. Iera, "SIoT: Giving a Social Structure to the Internet of Things," *IEEE Communication Letters,* vol. 15, no. 11, pp. 1193-1195, 2011.

[18] ITU-T, "Y.4460 - Architectural Reference Models of Devices for Internet of things applications," *Series Y: Global Information Instrustructure, Internet Protocol Aspects, Next-Generation networks, Internet of things and Smart Cities,* 2019.

[19] D. Z. a. S. Guo, "The Web of Things: A Survey," *Journal of Communications,* vol. 6, no. 6, pp. 424-438, 2011.

[20] "National Scienec Foundation (NSF), Cyber physical systems NSF10515," Arlington, VA, USA, 2013. [Online]. Available: https://www.nsf.gov/pubs/2010/nsf10515/nsf10515.htm.. [Accessed 10 8 2020].

[21] S. Peisert, J. Margulies, D. M. Nicol and H. Khurana, "Designed-in Security for Cyber-Physical Systems," *IEEE Security and Privacy Magazine,* vol. 12, no. 5, pp. 9-12, 2014.

[22] N. David, J. S. Silva and F. Boavida, A Practical Introduction to Human-in-the-Loop Cyber-Physical Systems, Wiley-IEEE Press, 2018.

[23] R. Baheti and H. Gill, "Cyber-physical systems," *The Impact of Control Technology, IEEE,* no. 1st Edition, pp. 161-166, 2011.

[24] H. Song, D. Rawat, S. Jeschke and C. Brecher, Cyber-Physical Systems: Foundations, Principles and Applications, Elsevier Academic Pres, 2016.

[25] Elrharbi, Simon, and Stefan Barbu, "Characteristics of RFID Radio Signals," in *RFID and the Internet of Things*, Wiley, 2013, pp. 7-55.

[26] C. d. M. Cordeiro and D. P. Agrawal, Ad Hoc and Sensor Networks: Theory and Applications, World Scientific , 2006.

[27] Dargie, Waltenegus, and Christian Poellabauer., Fundamentals of wireless senor networks theory and practices, John Wiley & Sons, 2010.

[28] A. A. Jahromi and D. Kndur, "Fundamentals of Cyber-Physical Systems," *Cyber-Physical Systems in the Built Environment ,* pp. 1-13, 2020.

[29] W. Reisig, A Primer in Petri Net Design, Springer-Verlag Berlin Heidelberg, 1992.

[30] R. R. Igorevich, P. Park, J. Choi and D. Min, "iVision based Context-Aware Smart Home system," in *The 1st IEEE Global Conference on Consumer Electronics*, Tokyo, 2012.

[31] K. S. Manoj, Indusrtrial Automation with SCADA: Concepts, Communications and Security, Chennai: Notion Press, 2019.

[32] L. Monostori, B. Kádár, T. Bauernhansl, S. Kondoh , S. Kumara, G. Reinhart, O. Sauer, G. Schuh, W. Sihn and K. Ueda, "Cyber-physical systems in manufacturing," *CIRP Annals - Manufacturing Technology,* vol. 65, no. 2, p. 621–641, 2016.

[33] M. El-Hajj, M. Chamoun, A. Fadlallah and A. Serhrouchni, "Analysis of Cryptographic Algorithms on IoT Hardware Platforms," in *2nd CyberSecurity in Netwroking Conference (CSNET)*, Paris, 2018.

[34] J. Califano, "How to Choose a Microcontroller for IoT," 29 6 2018. [Online]. Available: https://dzone.com/articles/how-to-choose-a-microcontroller-for-iot. [Accessed 24 8 2020].

[35] D. Zeng, S. Guo and Z. Cheng, "The Web of Things: A Survey," *Journal of Communications,* vol. 6, no. 6, pp. 424-438, 2011.

[36] M. Naeem, W. Ejaz, L. Karim, S. H. Ahmad, A. Anpalagan, M. Jo and H. Song, "Distributed Gateway Selection for M2M Communication in Cognitive 5G Networks," *IEEE Network,* vol. 31, no. 6, pp. 94-100, 2017.

[37] Z. Shelby, K. Hartke, C. Bormann and B. Frank, "The Internet Engineering Task Force (IETF)- BCP78 and BCP79," 9 6 2013. [Online]. Available: https://tools.ietf.org/pdf/draft-ietf-core-coap-13.pdf. [Accessed 24 8 2020].

[38] H. Chabanne, RFID and the Internet of Things, Wiley, 2011.

[39] Yida, "Latest Open Tech From Seeed Studio for Emerging IoT, AI and Autonomous Applications on the Edge," 10 2019. [Online]. Available: https://www.seeedstudio.com/blog/2019/09/25/uart-vs-i2c-vs-spi-communication-protocols-and-uses/. [Accessed 25 8 2020].

[40] P. Visconti, G. Giannotta, R. Brama, P. Primiceri, A. Malvasi and A. Centuori, "Features, operation principle and limits of spi and I2C communication protocols for smart objects: A novel spi-based hybrid protocol especially suitable for IoT applications," *International Journal on Smart Sensing and Intelligent Systems,* vol. 10, no. 2, pp. 262-295, 2017.

[41] P. Spasov, Microcontroller Technology: The 68HC11 and 68HC12, Pearson, 2004.

[42] S. Vuppala and H. K. Kumar, "Service Applications - Exploiting the Internet of Things," in *Annual SRII Global Conference, SRII.* , San Jose, 2014.

[43] Aftab, H., Gilani, K., Lee, J., Nkenyereye, L., Jeong, S., & Song, "Analysis of identifiers on IoT platforms," *Digital Communications and Networks,* pp. 1-3, 2019.

[44] B. S. Mohammad, Embedded Memory Design for Muti-Core and Systems on Chip, New York: Springer Science and Business Media, 2014.

[45] M. Barr, "Memory Types," *Embedded Systems Programming,* vol. 14, no. 5, pp. 103-104, 2001.

[46] J. Singh and B. Raj, "SRAM Cells for Embedded Systems," in *Embedded Systems - Theory and Design Methodology*, London, IntechOpen Limited, 2012, pp. 387-406.

[47] D. A. Hodges, Analysis and Design of Digital Integrated Circuits, McGraw-Hill Publishing Company Limited, 2003.

[48] S. Fuji, K. Natori, T. Furuyama, S. Saito, H. Toda and O. Ozawa, "A Low-Power Sub 100ns 256K Bit Dynamic RAM," *IEEE Journal of Solid-State Circuits,* vol. 8, no. 5, pp. 441-446, 1983.

[49] B. Keeth and R. J. Baker, DRAM Circuit Design: A Tutorial, Wiley-IEEE Press, 2000.

[50] R. BEZ, B. CAMERLENGHI, A. MODELLI and A. VISCONTI, "Introduction to Flash Memory," *PROCEEDINGS OF THE IEEE,* vol. 9, no. 4, pp. 489-502, 2003.

[51] A. Sehgal, V. Perelman, S. Kuryla and J. Schönwälder, "Management of Resource Constrained Devices in the Internet of Things," *IEEE Communications Magazine,* vol. 50, no. 12, pp. 144-149, 2012.

[52] C. Bormann, M. Ersue and A. Keranen, "Internet Engineering Task Force," May 2014. [Online]. Available: https://tools.ietf.org/html/rfc7228#section-2.1. [Accessed 15 8 2020].

[53] A. Schmidt and K. Van Laerhoven, "How to Build Smart Appliances?," *IEEE Personal Communications,* vol. 8, no. 4, pp. 66-71, 2001.

[54] M. Ersue, D. Romascanu, J. Schönwälder and A. Sehgal, "Management of Networks with Constrained Devices: Use Cases," Internet Engineering Task Force (IETF) RFC 7548, 2015.

[55] P. Sethi and S. R. Sarangi, "Internet of Things: Architectures, Protocols, and Applications," *Journal of Electrical and Computer Engineering,* vol. 2017, no. Article ID 9324035, 2017.

[56] K. Boeckl, M. Fagan, W. Fisher, N. Lefkovitz, K. N. Megas, E. Nadeau, D. G. O'Rourke, B. Piccarreta and K. Scarfone, "Considerations for Managing Internet of Things (IoT) Cybersecurity and Privacy Risks," National Institute of Standards and Technology, Gaithersburg, 2019.

[57] "Arduino UNO," [Online]. Available: https://create.arduino.cc/projecthub/patchr_io/explorer-uno-pcb-template-design-your-own-de89da.

[58] "Arduino IDE," [Online]. Available: https://www.arduino.cc/en/main/software .

[59] "Contiki," [Online]. Available: https://sourceforge.net/projects/contiki/ .

[60] "Android Things," [Online]. Available: https://github.com/androidthings .

[61] "RIOT," [Online]. Available: https://github.com/RIOT-OS/RIOT .

[62] "Apache Mynewt," [Online]. Available: https://github.com/apache/mynewt-core .

[63] "Huawei LiteOS," [Online]. Available: https://github.com/LiteOS .

[64] "Zephyr," [Online]. Available: https://github.com/zephyrproject-rtos/zephyr .

[65] "Ubuntu Snappy," [Online]. Available: https://ubuntu.com/download/iot/raspberry-pi-2-3-core.

[66] "TinyOS," [Online]. Available: https://github.com/saikatbsk/tinyOS .

[67] "FuchsiaOS," [Online]. Available: https://github.com/FuchsiaOS .

[68] "Samsung TizenRT," [Online]. Available: https://github.com/Samsung/TizenRT.

[69] "Raspberry Pi OS," [Online]. Available: https://www.raspberrypi.org/downloads/raspberry-pi-os/ .

[70] "Amazon FreeRTOS," [Online]. Available: https://github.com/aws/amazon-freertos.

[71] "Embedded Linux," [Online]. Available: https://github.com/fkromer/awesome-embedded-linux .

[72] "mbedOS," [Online]. Available: https://github.com/ARMmbed/mbed-os/releases/tag/mbed-os-5.13.4 .

[73] e. a. Montenegro G., " Transmission of IPv6 Packets over IEEE 802.15.4 Networks," Internet Engineering Task Force, 2007.

[74] I. S. Association, "IEEE 802.15.4-2020 - IEEE Standard for Low-Rate Wireless Networks," [Online]. Available: https://standards.ieee.org/standard/802_15_4-2020.html.

[75] MIPI, "MIPI M-PHY," [Online]. Available: https://mipi.org/specifications/m-phy.

[76] MIPI, "UniPro v.1.40.00 Specification," [Online]. Available: https://members.mipi.org/mipi-adopters/file-fix/Specifications/Board%20Approved/mipi_UniPro_specification_v1-40-00.pdf.

[77] "Mobile PCIe Specification," [Online]. Available: https://pcisig.com/sites/default/files/files/PCI-SIG%20and%20MIPI%20Alliance%20Announce%20Mobile%20PCIe%20(M-PCIe)%20Specification.pdf.

[78] M. Organization, "Modbus Homepage," [Online]. Available: http://www.modbus.org/.

[79] Oracle, "Internet of Things Intelligent Applications," [Online]. Available: https://www.oracle.com/it/internet-of-things/.

[80] Google, “IoT Overview,” [Online]. Available: https://cloud.google.com/solutions/iot-overview .

[81] Google, “Cloud IoT Core Service,” [Online]. Available: https://cloud.google.com/iot-core.

[82] “Energy@Home Project Hompage,” [Online]. Available: http://www.energy-home.it/.

[83] M. D. F. L. Bononi, “Corso di Internet of Things,” [Online]. Available: https://site.unibo.it/iot/en/teaching-1/the-iot-course .

[84] “DB-Engines.com Website,” [Online]. Available: db-engines.com.

[85] “InfluxDB Homepage,” [Online]. Available: https://www.influxdata.com.

[86] D. Francois, “Methodology and standards for data analysis with machine learning tools,” [Online]. Available: https://www.elen.ucl.ac.be/Proceedings/esann/esannpdf/es2008-4.pdf.

[87] Grafana Labs, “Grafana website,” [Online]. Available: https://grafana.com/.

[88] Elasticsearch B.V., “Kibana website,” [Online]. Available: https://www.elastic.co/kibana.

[89] Microsoft, “Power BI Desktop,” [Online]. Available: https://powerbi.microsoft.com/it-it/desktop/.

[90] W. S. a. L. Brow, Computer Security: Principles and Practice, Pearson, 2018.

[91] ENISA, "Baseline Security Recommendations for IoT in the context of CII_FINAL," ENISA, 2017. [Online]. Available: https://www.enisa.europa.eu/publications/baseline-security-recommendations-for-iot. [Accessed December 2020].

[92] ENISA, "ENISA Report - Guidelines for Securing the Internet of Things," 2020. [Online]. Available: https://www.enisa.europa.eu/news/enisa-news/iot-security-enisa-publishes-guidelines-on-securing-the-iot-supply-chain . [Accessed December 2020].

[93] ENISA, "Good practices for security of IoT," 2019. [Online]. Available: https://www.enisa.europa.eu/publications/good-practices-for-security-of-iot-1 . [Accessed December 2020].

[94] A. McEwen, Designing the Internet of things, Chichester: England, 2014, p. 338.

[95] R. Mansell, Imagining the Internet: Communication, Innovation, and Governance, Oxford: Oxford University Press, 2012.

[96] S. Jasanoff, "Virtual, Visible, and actionable: Data assemblages and the sightlines of justice," *Big Data & Society,* vol. 4, no. 2, 2017.

[97] F. P. A. a. N. S. Ustek-Spilda, " Engaging with Ethics in Internet of Things," *Big Data & Society,* vol. 6, no. 2, 2019.

[98] "The Stanford Encyclopedia of Philosophy," [Online]. Available: plato.stanford.edu.

[99] J. M. M. B. e. a. Nijhawan LP, "Informed consent: Issues and challenges.," *J Adv Pharm Technol Res.,* vol. 4, no. 3, pp. 134-140., 2013.

[100] R. M. Kidder, How Good People Make Tough Choices, Harper Collins, 2003.

[101] K. Macnish, The Ethics of Surveillance: an introduction., Routledge: London. , 2018.

[10 A. A. J. Antoniou, “Case Study: The Internet of Things and Ethics,,” *The Orbit Journal Special Issue - Case Studies of Ethics and Human Rights in Smart Inforation Systems,* vol. 2, no. 2, pp. 1-9, 2019.
2]

[10 R. e. a. Chory, “Organizational Surveillance of Computer-Mediated Workplace Communication: Employee Privacy Concerns and Responses.,” *Employee Responsibilities and Rights, ,* vol. 28, no. 1, pp. 23-43, 2016.
3]

[10 U. Hugl, “Workplace surveillance: examining current instruments, limitations and legal background issues.,” *Tourism & Management Studies,* vol. 9, no. 1, pp. 58-63, 2013.
4]

[10 G. Edwards, Employee Monitoring. What are the Legal Issues?.Credit Management, p.49., 2015.
5]

[10 E. Hossain, Internet of things A to Z, New Jersy: Wiley, 2018.
6]

[10 C. Wang, “Object Identification Techniques and the Application in IoT,” *Advances in Media Technology ,* vol. 9, 2013.
7]

[10 Ranasinghe, D., Cole, P, “Far-field tag antenna design methodology,” in *RFID handbook: applications, technology, security, and privacy*, 2008, p. 65–91.
8]

[10 Z. Feng, “Biometric Identification Technology and Development Trend of Physiological Characteristics,” *Journal of Physics: Conference Series,* vol. 1060, pp. 1-6, 2018.
9]

[11 S. H. Lin, “An introduction to face recognition technology,” *Informing Sci. Int. J. an Emerg. Transdiscipl,* vol. 3, pp. 1-7, 2000.
0]

[11 Farooq-i-Azam, Muhammad, and Muhammad Naeem Ayyaz, “Location and Position Estimation in Wireless Sensor Networks,” in *Wireless Sensor Networks: Current Status and Future Trends*, CRC Press, 2016, pp. 179-214.
1]

[11 R. Mautz, “Combination of Indoor and Outdoor Positioning,” in *1st International Conference on Machine Control & Guidance*, 2008.
2]

[113] Du, H., Zhang, C., Ye, Q., Xu, W., Kibenge, P. L., & Yao, K., " A hybrid outdoor localization scheme with high-position accuracy and low-power consumption," *EURASIP Journal on Wireless Communications and Networking,* vol. 1, no. 4, 2018.

[114] Singh, Santar Pal, and S. C. Sharma, "Range free localization techniques in wireless sensor networks: A review," *Procedia Computer Science ,* vol. 57, no. 7, pp. 7-16 , 2015.

[115] Nabil Alrajeh, Maryam Bashir, "Localization Techniques in Wireless Sensor Networks," *International Journal of Distributed Sensor Networks ,* vol. 9, no. 6, 2013.

[116] C. A. Howells, "Piezoelectric energy harvesting," *Energy Conversion and Management ,* vol. 50, no. 7, pp. 1847-1850, 2009.

[117] "piezoelectric material," elporocus, [Online]. Available: https://www.elprocus.com/what-is-a-piezoelectric-material-working.

[118] X. Lui, "An Electromagnetic Energy Harvester for Powering," *Master Thesis,* 2012.

[119] Tang, X., Wang, X., Cattley, R., Gu, F., & Ball, A. D., "Energy Harvesting Technologies for Achieving Self-Powered Wireless Sensor Networks in Machine Condition Monitoring: A Review," *Sensors,* vol. 18, no. 12, 2018.

[120] A. B. Vijay Madisetti, Internet of Things A Hands-On- Approach, VPT, 2014.

[121] Bizon, N., Tabatabaei, N. M., Blaabjerg, F., & Kurt, E., Energy harvesting and energy efficiency, Berlin: Springer, 2017.

[122] Anusuya, M. A., and Shriniwas K. Katti. , "Speech Recognition by Machine: A Review," *arXiv preprint arXiv:1001.2267 ,* 2010.

[123] Khan, Shafiullah, Al-Sakib Khan Pathan, and Nabil Ali Alrajeh, Wireless Sensor Networks: Current Status and Future Trends, CRC press, 2016.

[12 4] "Electronics Tutorials," [Online]. Available: https://www.electronics-tutorials.ws/io/io_1.html.

[12 5] P. Stokes, "4 Stages of IoT architecture explained in simple words," Medium.com, [Online]. Available: https://medium.com/datadriveninvestor/4-stages-of-iot-architecture-explained-in-simple-words-b2ea8b4f777f#:~:text=In%20essence%2C%20IoT%20architecture%20is,a%20sophisticated%20and%20unified%20network..