Mobile and Cyber Physical Systems - Appunti

Francesco Lorenzoni

February 2024

Contents

Course info

...

Part I

Stefano Chessa

Chapter 1

Internet of Things

The main topics addressed aside from **IoT** itself are how it relates to *Machine Learning* and *Cloud* computing processes, but also *IoT interoperability*, known *Standards*, and the *security* concerns about IoT.

1.1 IoT introduction

Cyber and Physical Systems (CPS) operate in both the Physical and Cyber worlds, thus we can see IoT as an embodiement of CPSs.

In a *smart environment*, smart objects are both physical and cyber, hence they are subject to "physical experiences" such as being placed, moved, damaged and so on.

But actually...
What is a *smart environment*?

The answer actually ain't trivial; a journal on IoT reports:

"smart environments can be defined with a variety of different characteristics based on the applications they serve, their interaction models with humans, the practical system design aspects, as well as the multi-faceted conceptual and algorithmic considerations that would enable them to operate seamlessly and unobtrusively"

1.2 Platforms for IoT

Sensors and actuators are the edge of the cloud. In general the purpose of IoT is to gather and send data, send it somewhere where it gets transformed into information ultimately used to provide some functionality for an end user, or it simply presented to them.

A **Platform for IoT** is essentially a —complex— software hosted on the cloud, which, first of all, **collects** data gathered by IoT devices, but *not* only that:

- ◇ Identification
- ◇ Discovery
- ◇ Device Management
- ◇ Abstraction/virtualization
- ◇ Service composition
  Integrating services of different IoT devices and SW components into a composite service
- ◇ Semantics
- ◇ Data Flow management
  - *sensors* $\longrightarrow$ *applications*
  - *applications* $\longrightarrow$ *sensors*
  - Support for aggregation, processing, analytics

1.3 No-SQL Databases

No-SQL DBs address the problem of the several changes of data formats, sources, cardinality and so on, which happen throughout time.

A common example is **MongoDB**, which stores records in JSON-like objects called *documents*, which are stored in *collections*, the entity corresponding to tables in relational DBs, with the key difference that multiple documents in a single collection may be structured differently.

1.4 IoT Issues

- ◇ Performance
- ◇ Energy Efficiency
- ◇ Security
- ◇ Data analysis/processing
  - – Adaptability/personalization

  The course will cover the basics of signal processing, with mentions to machine learning
- ◇ Communication/brokerage/binding
- ◇ Data representation
- ◇ Interoperability

  Standard discussed will be ZigBee, MQTT, and IEEE 802.15.4 (?)

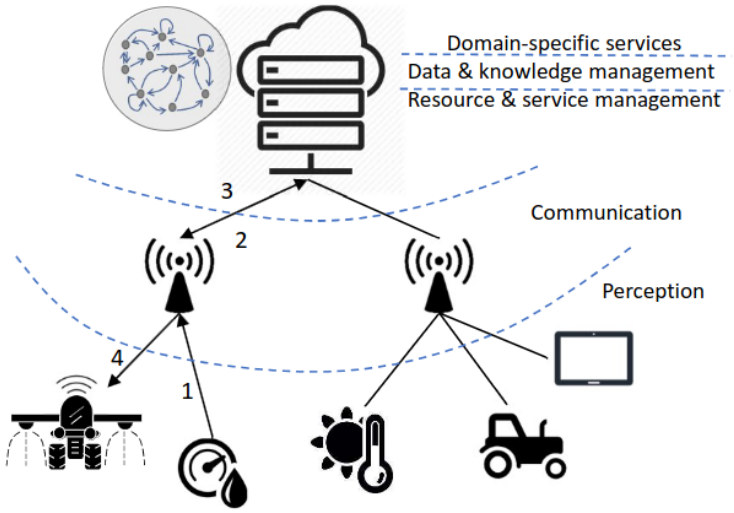


Figure 1.1: Communication outline in IoT
IoT systems are distributed, and servers may be dislocated around the globe, making room for **latency** and **reliability** issues.

To confine the problem displayed in Fig. 1.1 there are proposal to move the ability to make a decision on the data closer to the edge, but this in general isn't trivial.

Key Issues

1. Producing and handling fast-streaming heterogeneous sensed data
2. Make devices context-aware & allow them for continuous adaptation
3. Handle strong computing and energy constraints

1.4.1 Edge and Fog computing

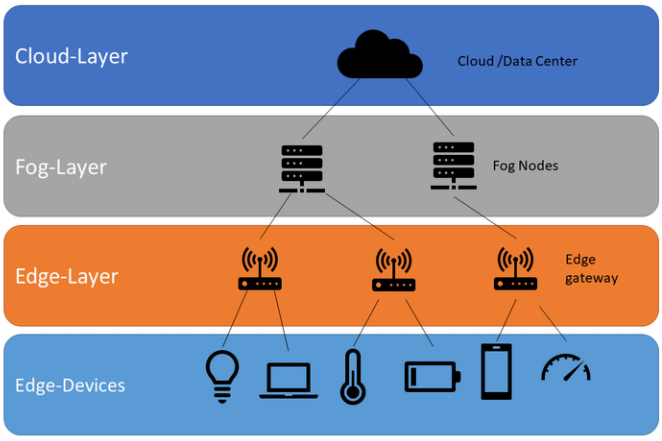


Figure 1.2: Layers scheme

A solution forsees to split the network in 4 layers, allowing for different response times and decisional capabilities.

A gateway on the **edge** interconnects the IoT-enabled devices with the higher-level communication networks, performing protocol translations.

A basic task performed at the fog layer is aggregating and collecting data, and then flushing it to the cloud periodically.

However, some decisions on the aggregated data may be taken at the fog node without querying the cloud, for instance determining where is a nest of tortoises, whether an explosion has occurred (by analyzing data from multiple sensors), and —maybe, one day in a not-so-far future— recognize human language.

prof. Chessa developed an 8 bit controller implementing a model for determining where is a nest of tortoises.
Alexa and *Google Home* currently send audio samples to the cloud for processing, but in the futurue this may be done locally.

1.4.2 Artificial Intelligence

AI splits into **Machine Learning** and **Curated Knowledge**.

ML focuses on mimicking how humans *learn* on new knowledge, while *curated knowledge* focuses on mimicking how humans *reason* on a known set of data.

Machine Learning reveals itself to be particularly useful in aggregatin multiple heterogeneous time-series sensed data about the same environment.

Supervised and Reinforcement learning are more promising than

1.4.3 Blockchain & IoT

A **blockchain** may act as a shared ledger between companies in a supply chain, with IoT devices to track goods and to monitor their quality along the chain, i.e. production stages, shipping and distribution.
With a blockchain each actor along the supply chain can query the ledger to check the —certified— state of the goods.

1.4.4 Interoperability

Vertical Silos

Developing a straight implementation of an IoT solution, starting from physical up to the application layer, is not a problem by itself.
In this way solution you implemented will work only on your devices, making your intervention needed for any change or update; besides, products by other vendors will be incompatible.

Vertical Silos business model leads to **vendor lock-in**s, which basically are service limitations which prevent the users from purchasing and using products from other vendors.

The solution to avoid —or limit— such issues is to introduce **standards**. Standards require common interests and agreements among different manufacturers, they are usually motivated by a reductino of the costs for development of a technology. There must be "*coopetition*" among manufacturers.

There is coopetition usually when a technology becomes **mature**:
- ◇ Big revenues are somewhere else
- ◇ No interest in investing big money in developing the technology

⇒ Without these conditions the standards will most likely fall

For what concerns wireless communication, standards are mainly differentiated by *Range* and *Data Rate*.

However, interoperability may be an issue not strictly related to vertical silos, but also to standards, in case there are *too many*.
The problem of interoperability shifts from low-level to application level.
To solve the problem, **gateways** are introduced, which translate different protocols.

- In type C configuration, how many mappings from one protocol to another (at the same level) the integration gateway should be able to manage?
- What about in type D configuration?

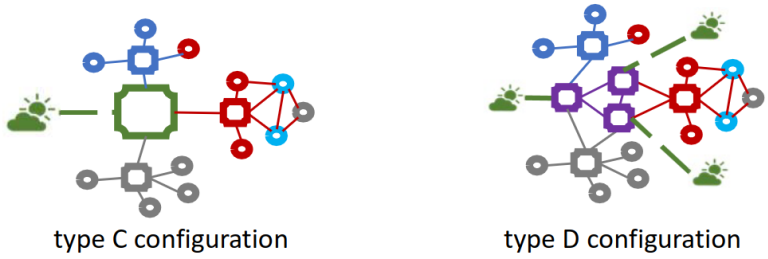


Figure 1.3: Gateway configs

Considering Fig 1.3 and assuming n protocol standards, the gateway in config C must be able to manage a mapping for every possible pair of standards, resulting in $n * n = n^2$ mappings. In configuration D instead evey gateway translates *from* and *to* an **intermediate language** (purple in figure), resulting in a double translation process, but only $2 * n$ mappings, which is much less.

1.5 Security in IoT

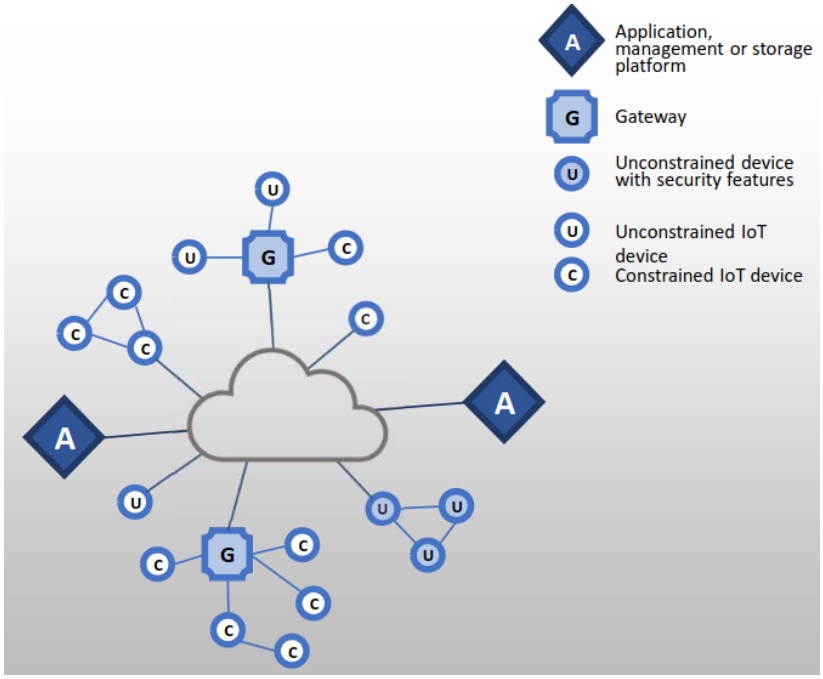


Figure 1.4: Security elements of interest

In an IoT environment there are various elements, each with its characteristics and vulnerabilities.

In general there are many issues concerning **patching vulnerabilities**, which poorly —or not at all— addressed.

- ◇ There is a crisis point with regard to the security of embedded systems, including IoT devices
- ◇ The embedded devices are riddled with vulnerabilities and there is no good way to patch them
- ◇ Chip manufacturers have strong incentives to produce their product as quickly and cheaply as possible
- ◇ The device manufacturers focus is the functionality of the device itself
- ◇ The end user may have no means of patching the system or, if so, little information about when and how to patch
- ◇ The result is that the hundreds of millions of Internet-connected devices in the IoT are vulnerable to attacks
- ◇ This is certainly a problem with sensors, allowing attackers to insert false data into the network

Not so critical for wristbands, but potentially harmful for water quality sensors, even worse for uranium enrichment, or aircraft sensors

- ◇ It is potentially a graver threat with actuators, where the attacker can affect the operation of machinery and other devices

What about **confidentiality**? Is it necessary?
The lecturer provided an example:
Assume that a wristband records the heartbeat without enforcing confidentiality, and assume that such heartbeat indicates a risk of heart disease in the owner. The owner may want to have a life insurance, but if a company had bought the unconfidential data on the black market, and recognized that the owner may suffer from a heart disease. Then the company could rise the price of the insurance for the unconfidential wristband owner.

Aside from these, laws introduce many requirements concerning security, which may be critical to satisfy in an IoT environment. In particular, The `IUT-T` standard Recommendation `Y.2066` includes a list of security requirements for the IoT, which concern the following points, but note that the document does **not** define how to enforce and satisfy such requirements:

- ◇ Communication security
- ◇ Data management security
- ◇ Service provision security
- ◇ Integration of security policies and techniques
- ◇ Mutual authentication and authorization
  It it crucial for the authentication to work both directions, from the gateway to the device, and from the device to the gateway. It is needed because wireless networks are easily trickable by intruders.
- ◇ Security audit

Considering the points mentioned above, we must consider what is the role of **gateways** about security.

Sometimes instead of mutual one, weaker *one-way authentication* may be enforced: either the device authenticates itself to the gateway or the gateway authenticates itself to the device, but not both.

Also the security of the data is not trivial to achieve, especially if constrained devices are used, because they may not be able to enforce task such as encryption or authentication.

This makes **privacy** concers arise especially regarding homes, cars and retail outlets, because with massive IoT, governments and private enterprises are able to collect massive amounts of data about individuals.

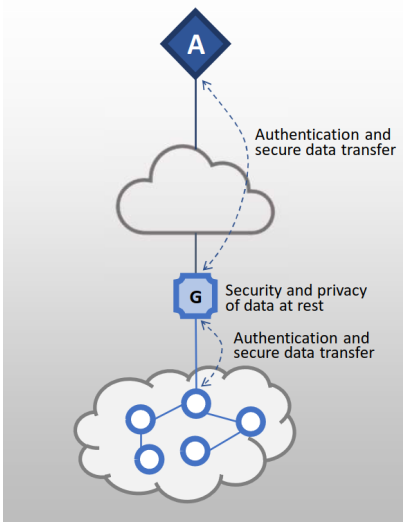


Figure 1.5: Gateways security functions

Chapter 2

MQTT

Things must be connected to the Internet to become *"IoT"* devices, and thus to adopt the internet protocol suite (TCP/IP + application, usually HTTP). However, the Internet stack is thought for *resource-rich* devices, not for IoT ones.
These led the canonical protocol stack to be modified for IoT environments, according to its needs and limitations.

MQTT is a publish-subscribe application protocol, which initially was not designed specifically for IoT. "MQTT" stands for *"Message Queuing Telemetry Transport"*, but "Queing" should not be intended literally as it usually is in the ICT world. MQTT is built upon TCP/IP. TCP isn't the optimal choice for IoT, UDP is generally preferred, but as said before, MQTT was not designed for IoT:

- ◇ Port 1883
- ◇ Port 8883 for using MQTT over SSL
  - – *SSL adds significant overhead!*

Lightweight

- ◇ Small code footprint
- ◇ Low network bandwidth
- ◇ Low packet overhead (guarantees better performances than HTTP)

2.1 Publish-Subscribe recalls

Publish/subscribe is a *loosely coupled*[1] interaction schema, where both publishers and subscribers act as "clients". There is a third party called *event service* (aka **Broker**), which acts as the actual "server" (considering the client-server architecture), and which is known by both publishers and subscribers.
In this paradigm clients are simple, while the complexity resides in the broker.

Publishers, e.g. a sensor, produce events —or any data they wish to share by means of events— and interact only with the broker, while **subscribers** express the interest for an event, and receive an asynchronous notification whenever an event or a pattern of events is generated; also subscribers interact only with the broker.
Publishers and subscribers are **fully decoupled** in *time*, *space* and *synchronization*.

- ◇ Space decoupling:
  - – Publisher and subscriber do not need to know each other and do not share anything
  - – they don't know the IP address and port of each other
  - – they don't know how many peers they have
- ◇ Time decoupling:
  - – Publisher and subscriber do not need to run at the same time.
- ◇ Synchronization decoupling:
  - – Operations on both pub. and sub. are not halted during publish or receiving.

The **Broker**:

- ◇ *Known* to publishers and subscribers
- ◇ *Receives* all incoming messages from the publishers
- ◇ Filters all incoming messages

[1] i.e. peers don't have to share "too much"

- ◇ *Distributes* all messages to the subscribers
- ◇ Manages the requests of *subscription/unsubscription*

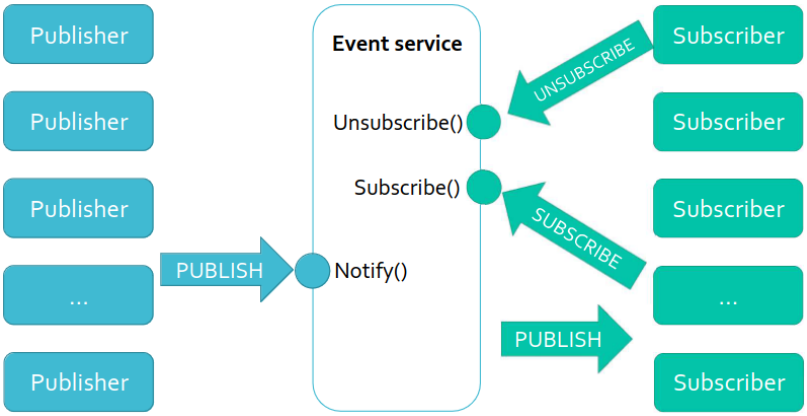


Figure 2.1: Broker management of events

2.1.1 Properties

Due its decoupling **properties**, compared to basic *client-server*, publish-subscribe is considered to be more **scalable**, even if it is implemented using an underlying client-server architecture.

First of all, everything is entirely up to the broker, does not depend on the direct interaction between endpoints. In case of a very large number of devices, the architecture can scale by **parallelizing** the (event-driven) operations on the broker.

Regarding the message filtering performed by the broker, it can happen depending on various fields:

- ◇ **Subject topic**
  - The subject (or topic) is a part of the messages
  - The clients subscribe for a specific topic
  - Typically topics are just strings (possibly organized in a taxonomy)
- ◇ **Content**
  - The clients subscribe for a specific query (e.g. $Temp > 30°$)
  - The broker filters messages based on a specific query
  - Data cannot be encrypted!
- ◇ **Data type**
  - Filtering of events based on both content and structure
  - The type refers to the type/class of the data
  - Tight integration of the middleware and the language (!)

  The second and third approaches require increasing **integration** mechanisms to provide the desired features.

2.2 MQTT and Publish-Subscribe

MQTT provides a specific implementation of the PS paradigm. Since it relies on TCP/IP, Publishers and subscribers need to know the **hostname/ip** and port of the broker *beforehand*.

Thanks to its speed and to being lightweight, in most applications the delivery of messages is mostly in *near-real-time*, but in general this is *not* a guaranteed property.

In MQTT message **filtering** is based only **topics**, which is the most flexible filtering of the ones presented in the previous section.

2.3 Messages

A client connects to a broker by sending a `CONNECT` message. Since such message may be lost, the broker answers with a `CONNECTACK` message, indicating simply whether the connection was accepted, refused, and if there was a previously stored session with the client.

- ◇ Client ID

  - A string that uniquely identifies the client at the broker.
    If empty: the broker assigns a unique `clientID` and does not keep a status for the client.
    In this case *Clean Session* must be TRUE.
    Note also that in version 3.1.1 the servers replies with a CONNECTACK with *no* payload, so the assigned ID is not known to the client.
    This has changed in version 5.0

> **ClientID Uniqueness - Digression**
>
> How can a client know if its Client ID is **unique**?
>
> The answers is not completely addressed by the standard, and the scenario of a new client who wants to connect and have a persistent session is not clearly discussed. ClientIDs may be assigned beforehand, but this is possible only if the admin controls *entirely* the system, it is not possible if the broker is *public*, thus an owner of MQTT clients doesn't know whether there are other clients.
>
> In reality, you can "take your chance", because the ClientID is 23 byte long, so the chance of an overlap between multiple devices is low.
>
> In general, standard specifications tend to omit everything that can be omitted, to avoid posing constraints which are not strictly necessary, by leaving room for personal implementations and needs.

- ◇ Clean Session
  - Set to `FALSE` if the client requests a **persistent session**, allowing for session resuming and better QoS (storing missed messages).
- ◇ Username/Password
  - No encryption, unless security is used at transport layer

optional

- ◇ Will[1] flags [1]
  - If and when the client disconnects ungracefully, the broker will notify the other clients of the disconnection
- ◇ KeepAlive
  - The client commits itself to send a control packet (e.g. a ping message) to the broker within a keep-alive interval expressed in seconds, allowing the broker to detect whether the client is still active (**detect disconnections**)

After `CONNECT` the publishers may send `PUBLISH` messages, which are later forwarded by the broker to the subscribers, and which are structured as follows:

`PUBLISH`

- ◇ `packetId`
  - An integer
  - It is 0 if the QoS level is 0
- ◇ `topicName`
  - a string possibly structured in a hierarchy with "/" as delimiters
  - Example: "home/bedroom/temperature"
- ◇ `qos` 0,1 or 2
- ◇ `payload`
  - The actual message in any form
- ◇ `retainFlag`
  - tells if the message is to be stored by the broker as the last known value for the topic
  - If a subscriber connects later, it will get this message
- ◇ `dupFlag`
  - Indicates that the message is a duplicate of a previous, unacked message
  - Meaningful only if the QoS level is > 0

`SUBSCRIBE`

- ◇ `packetId` an integer
- ◇ `topic_1` a string (see publish messages)
- ◇ `qos_1` 0,1 or 2

[1] This refers to the *Last Will* (Testament), the document with the "wills" of someone dead.

`SUBACK`
- ◇ `packetId` the same of the SUBSCRIBE message
- ◇ `returnCode` one for each topic subscribed

There are also `UNSUBSCRIBE` and `UNSUBACK` messages which have a similar structure but are not described here.

2.4 Topics

TODO

2.5 QoS

The **QoS** is an agreement between the sender and the receiver of a message.

For example, in TCP the QoS includes guaranteed delivery and ordering of messages.

In MQTT the QoS is an agreement between the clients and the broker, and there are three levels:

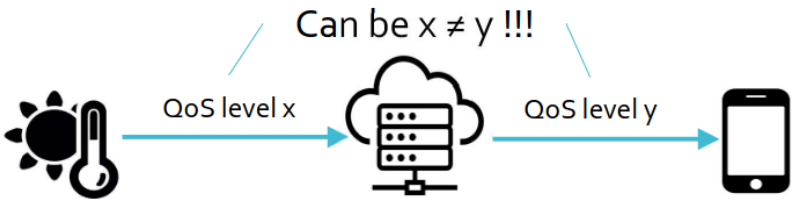
`level 0` **At most once**
- ◇ It is a "best effort" delivery and messages are *not* acknowledged by the receiver

`level 1` **At least once**
- ◇ Messages are numbered and stored by the broker until they are delivered to all subscribers with QoS level 1. Each message is delivered at least once to the subscribers with QoS, but possibly also more.

`level 2` **Exactly once**
- ◇ guarantees that each message is received *exactly once* by the recipient, exploiting a double two-way handshake.

Note that QoS is used both:
- ◇ between publisher and broker
- ◇ between broker and subscriber

But the **QoS** in the two communication **may be different**.



2.5.1 Choosing the right QoS

- ◇ Use QoS level 0 when:
  - The connection is stable and reliable
  - Single message is not that important or get stale with time
  - Messages are updated frequently and old messages become stale
  - Don't need any queuing for offline receivers
- ◇ Use QoS level 1 when:
  - You need all messages and subscribers can handle duplicates
- ◇ Use QoS level 2 when:
  - You need all messages and subscribers cannot handle duplicates
  - Has much higher overhead!!!!

2.6 Persistent Sessions

Persistent sessions keep the state between a client and the broker: if a subscriber disconnects, when it connects again, it does not need to subscribe again the topics.
The session is associated to the clientId defined with the `CONNECT` message, and stores:

- ◇ All **subscriptions**
- ◇ All QoS 1&2 messages that are **not confirmed** yet

- ◇ All QoS 1&2 messages that arrived when the **client was offline**

Note that with `QoS = 0` persistent sessions are useless overhead.

2.7 Retained messages

A publisher has **no guarantee** wether its messages are —or *when*— actually delivered to the subscribers, it can only achieve guarantee on the delivery to the broker.

A **retained message** is a normal message with `retainFlag = True`; the message is stored by the broker, and if a new retained message of the same topic is published, the broker will keep only the last one. When a a client subscribes the topic of the retained message the broker immediately sends the retained message, allowing subscribers to immediately get updated to the "state of the art".

Note that retained messages are kept by the server even if they had already been delivered.

2.8 Last will & testament

Last Will & testament is used to notify other clients about the **ungraceful disconnection** of a client.

The broker stores the **last will message** attached to the `CONNECT` message, but if the client gracefully closes the connection by sending `DISCONNECT`, then the stored *last will message* gets discarded.

Often the Last Will message is used along with retained messages.

2.9 Packet Format

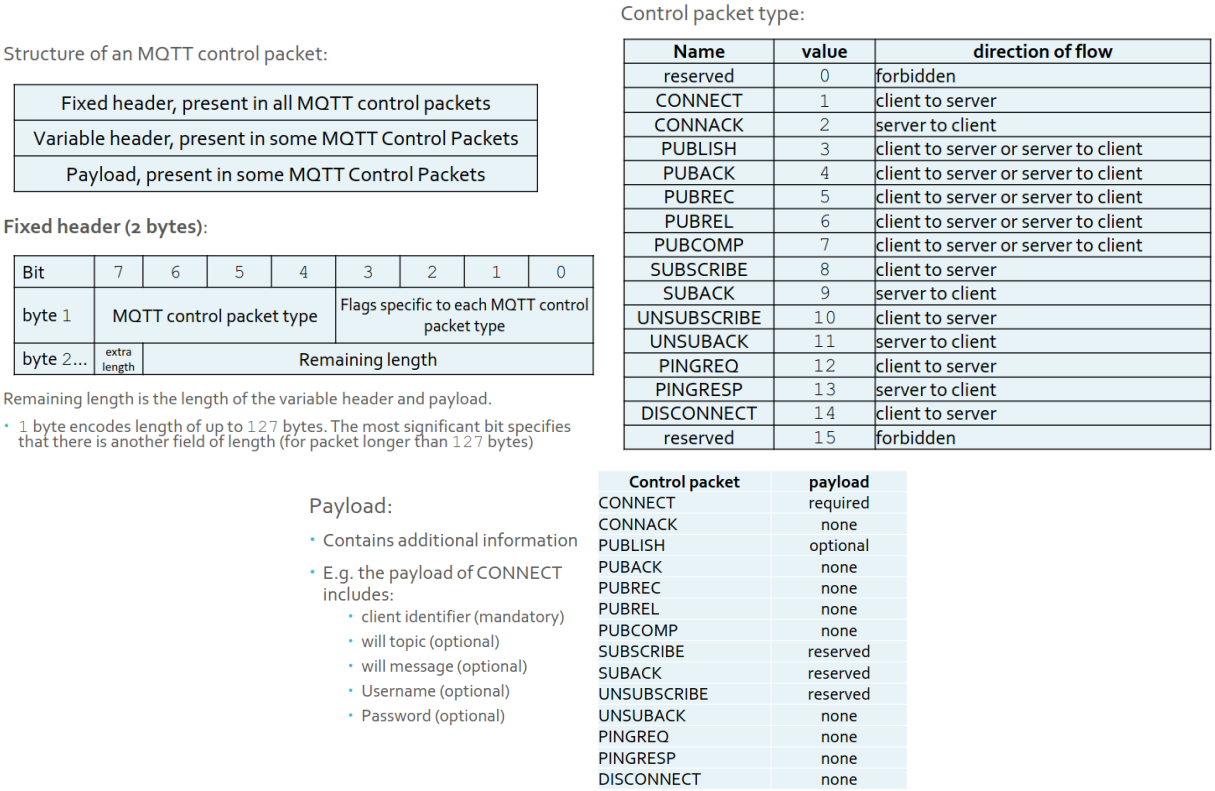


Figure 2.2: MQTT Packet headers

The —not displayed in Fig. 2.2— **Variable header**:

- ◇ Contains the packet identifier (encoded with two bytes)
  - – Only CONNECT and CONNACK control packets do not include this information
  - – The PUBLISH packet contains this information only if `QoS` > 0
- ◇ Contains other information depending on the control packet type

- For example, CONNECT packets include the protocol name and version, plus a number of flags (see CONNECT)

Chapter 3

ZigBee

ZigBee is widely used in various fields from home automation to Mars exploration; it is considered the "cousin" of Bluetooth: they are standardized by the same company and can coexist.

Aside from the application layer, ZigBee defines also a *Network Layer* which perfectly matches and maps to the underlying MAC and Physical Layers, standardized by `IEEE 802.15.4`; ZigBee is built on top of such IEEE standard.

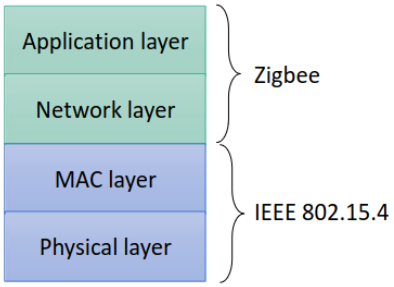


Figure 3.1: ZigBee layers

Key Features

- ◇ Specification of the physical and MAC layers for low-rate Wireless Personal Area Networks (PAN)
- ◇ Infrastructure-less
- ◇ Short range[a]
- ◇ Support for star and peer-to-peer topologies
- ◇ Can coexist with IEEE 802.11 and IEEE 802.15.1 (Bluetooth)
- ◇ Works on licence-free frequency bands

[a] 250m outdoors in ideal conditions

3.1 Architecture

APS provides *transport* services to the ZDO and the Objects in the Application Framework (APOs). It is some kind of Transport layer, similar to TCP but not the same.

APOs are the business logic of the business device, implemented by the user, and in a single device there may be instantiated up to APOs. We may say that for each APO provides a "functionality".

The ZDO is an applicative object that defines and maintaines the device behaviour in a ZigBee network.

An example of this behaviour, is replying to a device discovery message. Such reply is handled by the ZDO

The ZDO is provided by the third parties which are giving you the ZigBee stack. Manufacturers which produce devices compliant with ZigBee, sell them with a ZigBee stack already implemented, allowing for the buyer —e.g. a company which develops ZigBee solutions— to simply

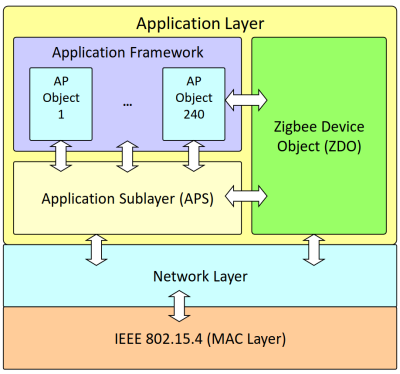


Figure 3.2: Zigbee architecture layers

implement the "functionalities" (i.e. APOs) they want.

3.2 Primitives

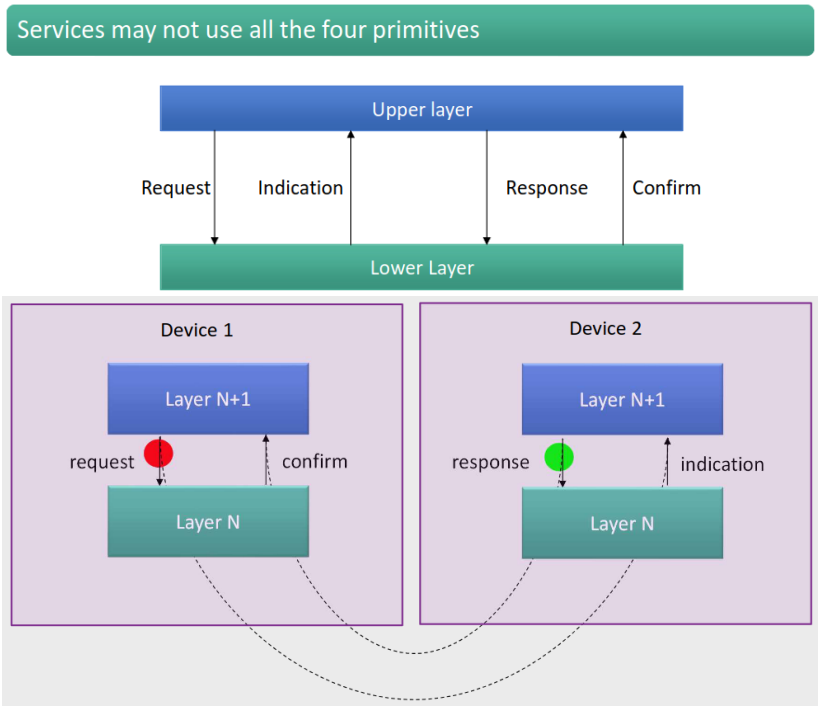


Figure 3.1: Mapping between zigbee primitives

Primitives

1. `Request`
   It is invoked by the upper layer to request for a specific service
2. `Indication`
   Is a sort of *"upcall"*, generated by the lower layer and is directed to the upper layer to notify the occurrence of an event related to a specific service
3. `Response`
   It is invoked by the upper layer to complete a procedure previously initiated by an indication primitive
4. `Confirm`
   It is generated by the lower layer and is directed to the upper layer to convey the results of one or more associated previous service requests.

3.3 Network Layer

The ZigBee network layer provides services for:

1. Data transmission (both unicast and multicast)
2. Network initialization
3. Devices addressing
4. Routes management & routing
5. Management of joins/leaves of devices

In a ZigBee network there are three kinds of devices:

1. **The Network coordinator**
   A FFD[1] that creates and manages the entire network
2. **Routers**
   A FFD with routing capabilities
3. **End-devices**
   Correspond to a RFD[2] or to a FFD acting as simple devices

[1]Full functional Device
[2]Reduced functional device

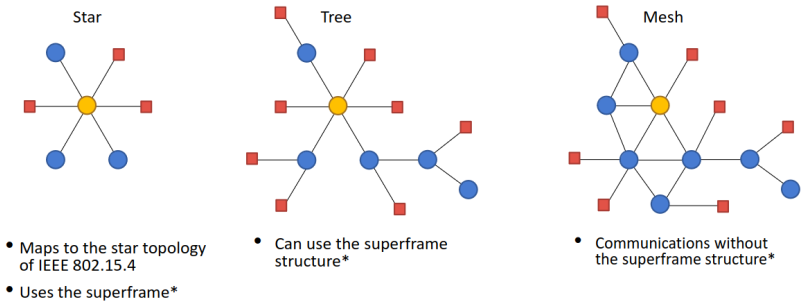


Figure 3.2: ZigBee Network topologies outline

The superframe mentioned above, is a feature used to obtain energy efficiency in ZigBee networks, but we will discuss it later on.

3.3.1 Network formation and joining

Before communicating on a network, a ZigBee device must either:

- ◇ Form a new network ⟶ *ZigBee Coordinator*
- ◇ Join an existing network ⟶ *ZigBee router* or *end-device*

The role of the device is chosen at compile-time

Formation

Network Formation is performed by a coordinator, which uses the MAC layer services to (`SCAN.request`) look for a channel that does not conflict with other existing networks, and then selects a PAN identifier which is not already in use by other PANs.

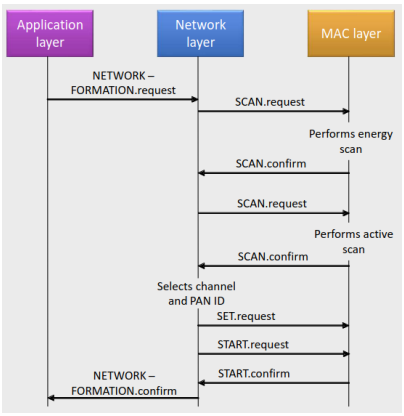


Figure 3.3: Network formation messages

Joining

Joining may happen in two ways, the first is to join through **association**: initiated by a device wishing to join an existing network.

Alternatively a device may perform a **Direct join**: requested by a router or by the coordinator to request a device to join its PAN.

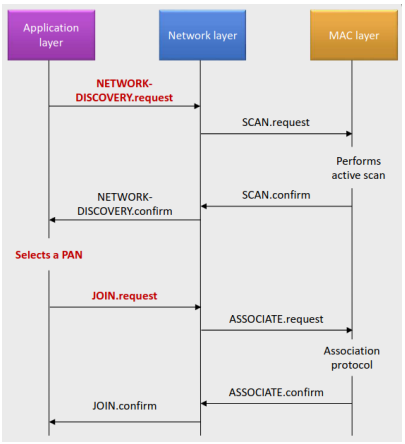


Figure 3.4: Network joining messages

3.4 Application Layer

Up to 240 APOs, each corresponding to an application **Endpoint**, with the Endpoint 0 reserved for the ZDO[3]. Each APO in the network is uniquely identified by its endpoint address and the network address of the hosting device.

3.4.1 APS - Application Support Sublayer

The APS frame uses the concepts of **endpoints**, **cluster IDs**, **profile IDs** and **device IDs**.
It provides:

- ◇ Data service (a light transport layer)
  - – Filtering out packets (non registered endpoints, profiles that do not match)
  - – Generating end-to-end acknowledgments
- ◇ Management:
  - – Local binding table
  - – Local groups table
  - – Local address map

Concepts and related IDs

[3]We could say that the ZDO is an "application object", which would be true, but tailored to specific needs

A **cluster** may be, in the simplest case, a *message*. But this is not necessarily the case.
Informally, a cluster provides access to a service (a functionality) of an application object; Defines both *commands*, which cause actions on a device, and *attributes*, showing the state of a device in a given cluster.

Every cluster has a 16 bit identifier, which according to prof. Chessa is **not** sufficient.

Note that clusters are not related to the physical world interaction, because they must allow reuse.
Each cluster finds a possibly different meaning in each **application profile**. There is a mapping which defines such meanings mappings.

Using this schema, 16 bits become sufficient.

An **application profile** is the specification of the behaviour of a class of applications possibly operating on several ZigBee devices. Each profile is paired with a 16 bit identifier.

Every message sent (or received) is tagged with a profile ID. Different application profiles may co-exist in a single ZigBee network.

ZigBee **Device IDs** range from `0x0000` to `0xFFFF`, and have two purposes:

1. To allow human-readable displays (e.g., an icon related to a device)
2. Allows ZigBee tools to be effective also for humans
   i. a device may implement the on/off cluster, but you don't know whether it is a bulb or a oven . . . you only know you can turn it on or off.
   ii. The device ID tells you what it is, but it does not tell you how to communicate with it, which is given by the IDs of the clusters it implements!

ZigBee discovers services in a network based on profile IDs and cluster IDs, but **not** on device IDs

Cluster Name	Cluster ID
Basic Cluster	0x0000
Power Configuration Cluster	0x0001
Temperature Configuration Cluster	0x0002
Identify Cluster	0x0003
Group Cluster	0x0004
Scenes Cluster	0x0005
OnOff Cluster	0x0006
OnOff Configuration Cluster	0x0007
Level Control Cluster	0x0008
Time Cluster	0x000a
Location Cluster	0x000b

Profile ID	Profile name
0101	Industrial Plant Monitoring
0104	Home Automation
0105	Commercial Building Automation
0107	Telecom Applications
0108	Personal Home & Hospital Care
0109	Advanced Metering Initiative

Figure 3.5: ZigBee General Domain clusters and common Profile IDs

Name	Identifier	Name	Identifier
Range Extender	0x0008	Light Sensor	0x0106
Main Power Outlet	0x0009	Shade	0x0200
On/Off Light	0x0100	Shade Controller	0x0201
Dimmable Light	0x0101	Heating/Cooling Unit	0x0300
On/Off Light Switch	0x0103	Thermostat	0x0301
Dimmer Switch	0x0104	Temperature Sensor	0x0302

Figure 3.6: Device IDs from the *Home Automation* profiles

Back to APS Services

APS Provides:

- ◇ Data service to both the APOs and the ZDO.
- ◇ Binding service to the ZDO
- ◇ Group management services

The APS data service enables the exchange of messages between two or more devices within the network.

- ◇ The data service is defined in terms of the primitives:
- ◇ Request (`send`),
- ◇ Confirm (returns `status` of transmission) and
- ◇ Indication (`receive`).

APS provides also a **message reliability service**, which simply sends multiple times a message until an ACK is received (if it was needed in the first place).

The **group management** provides services to build and maintain groups of APOs, enabling multicast, with each group being identified by a 16-bits address.

MAC addresses in ZigBee contexts are meant to be permanent, even if in recent years FFDs provide functionalities to randomly generate MAC addresses in order to enforce privacy. This in general is not performed on low-end RFD devices.

3.5 Binding

Addresses are indirect, allowing to implicitly specify the destination of messages, which are no longer routed based on a pair $\langle destinationendpoint, destinationnetworkaddress \rangle$ (*direct addressing*), but binding tables and address maps are used instead.

This is one of the key functions of the ZigBee Transport Layer, and is performed by the *APS*.

3.5.1 APS - Address Map

The APS layer contains the address map table, which associates the 16 bit NWK address with the 64 bit IEEE MAC address.
Zigbee end devices (ZED) may change their 16 bit NWK address (e.g. they leave and join again). In that case an announcement is sent on the network and every node updates its internal tables to preserve the bindings.

IEEE Addr	NWK Addr
0x0030D237B0230102	0x0000
0x0030B237B0235CA3	0x0001
0x0031C237b023A291	0x895B

Figure 3.3: Address Map

3.5.2 APS - Binding

We assume that typically the binding is performed by an admin who is —physically— deploying network nodes.

Primitives

- ◇ `BIND.request`
  Creates a new entry in the local binding table taking as input ⟨*source address, source endpoint, cluster identifier, destination address, destination endpoint*⟩ The
- ◇ `UNBIND.request`
  deletes an entry from the local binding table.

binding table associates sources and destinations based on MAC addresses, and is stored in the APS of the ZigBee coordinator (and/or of the routers); it gets updated on explicit request of the ZDO in the routers or in the coordinator, and is usually initialised at the network deployment. In general, it is *static*.

Indirect addressing is implemented exploiting the binding table and the address map:

Src Addr (64 bits)	Src EP	Cluster ID	Dest Addr (16/64 bits)	Addr/Grp	Dest EP
0x3232...	5	0x0006	0x1234...	A	12
0x3232...	6	0x0006	0x796F...	A	240
0x3232...	5	0x0006	0x9999	G	_
0x3232...	5	0x0006	0x5678...	A	44

Figure 3.4: Binding table

◇ matches *source address* ⟨*network addr, endpoint addr*⟩ and the *cluster identifier* into the pair: ⟨*destination endpoint, destin*

3.6 ZDO - ZigBee Device Object

ZDO is a special application attached to endpoint 0 and implements ZigBee End Devices, ZigBee Routers and ZigBee Coordinators.

It is specified by a special profile, the ZigBee Device Profile, which describes the clusters that must be supported by any ZigBee device; it defines also how the ZDO implements the services of discovery and binding and how it manages network and security.

ZDO services

◇ Device and service discovery
◇ Binding management
◇ Network management
◇ Node management

3.6.1 Device and service discovery

The ZigBee Device Profile (ZDP) specifies the device and service discovery mechanisms. **Device discovery** allows a device to obtain the (network or MAC) address of other devices in the network:

◇ **Unicast** ⟶ directed to an individual device
◇ **Broadcast** ⟶ hierarchical implementation based on a tree and subtrees topology: a router returns to its parent its address and the address of all the end devices associated to itself and then the coordinator returns the address of its associated devices

Service discovery exploits queries based on profiles ID, cluster IDs, addresses, or device descriptors. Again may either be unicast or broadcast.

◇ **Unicast** ⟶ if directed to a single end device then the coordinator or the router to which it is connected respond on its behalf
◇ **broadcast** ⟶ The coordinator responds to service discovery queries returning lists of endpoint addresses matching with the query; It exploits a hierarchical implementation: each router collects information from its associated devices and forwards it to its parent

3.6.2 Binding management

The ZDO processes the binding requests received from local or remote EP To *add* or *delete* entries in the APS binding table.

3.6.3 Network and Node Management

◇ **Network management**
  – Implements the protocols of the coordinator, a router or an end device according to the configuration settings established either via a programmed application or at installation.
◇ **Node management**
  – The ZDO serves incoming requests aimed at performing network discovery, retrieving the routing and binding tables of the device and managing joins/leaves of nodes to the network.

3.7 ZigBee Cluster Library

ZCL is a repository for cluster functionalities, a "working library" with regular updates and new functionalities. ZigBee developers are expected to use the ZCL to find relevant cluster functionalities to use for their applications, in order to

- ◇ Avoid re-inventing the wheel
- ◇ Support interoperability
- ◇ Facilitate maintainability

A cluster is a collection of commands and attributes, which define an interface to a specific functionality of a device. Clusters refer to functional domains within the respective profile.
The ZCL forsees a Client-Server model.

- ◇ The device that *stores* the attributes is the *server* of the cluster
- ◇ The device that *manipulates* the attributes is the *client* of the cluster

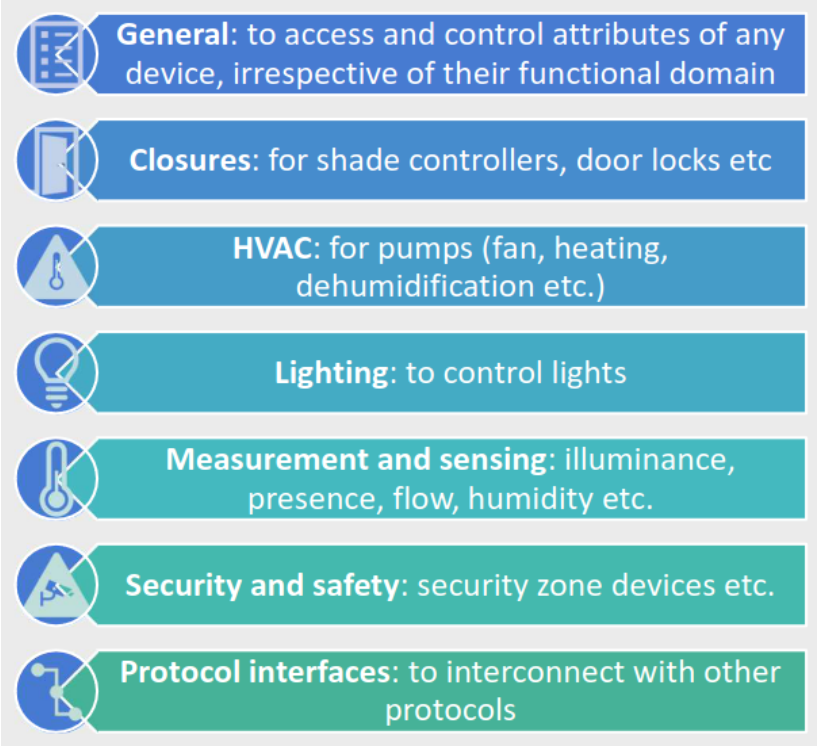


Figure 3.5: Functional domains

TODO integrate some slides

ZCL is built to allow combining simpler clusters into more complex ones, providing a hierarchical approach to define device functionalities.

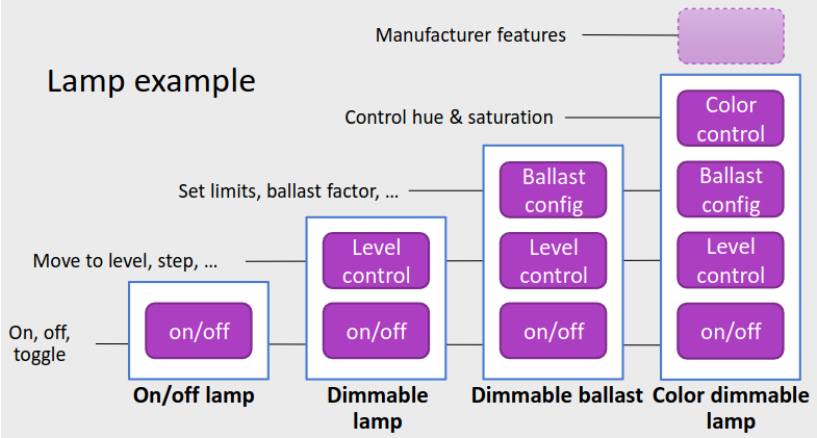


Figure 3.6: ZCL Hierarchical approach

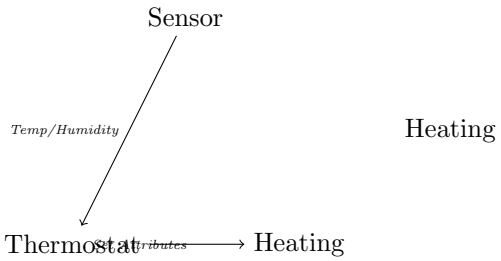


Figure 3.7: Heating System - exercise 2 Schema

Chapter 4

Use Cases

4.1 Industrial IoT in a smart GreenHouse

The topic discussed is a case study of a project funded by the Tucany Region and concluded in 2019 whose aim was to develop a technological **greenhouse** having specific objectives:

- ◇ Monitor drainage waters
- ◇ Control the root zone in terms of the presence of nutrients and waters
- ◇ Compare different cultivation substrates
- ◇ Develop new models based on AI to enable an intelligent control of the greenhouse

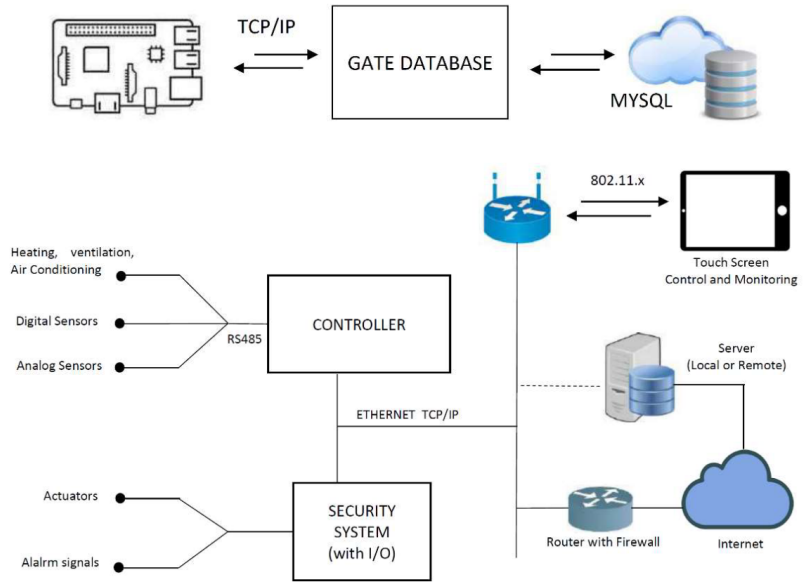


Figure 4.1: Greenhouse network architecture

There are some information concerning the architecture and structure of the greenhouse which are not reported here.

The problem is how to analyze data and integrate it with AI; there are two main aspects:

1. Need a control of the installation/systems that goes beyond the simple "timed logic"
   i. E.g. the fertigation system may need to be controlled depending on the actual needs of the crops...
   ii. ...that however change with the environmental conditions and with their growth
2. Need to forecast the crops growth to optimize the greenhouse (power/water/fertilizers consumption)

The parameters concerning **crops growth** are:

- ◇ Leaf area index(LAI)
- ◇ Dry weight (DW)
- ◇ Evotranspiration (ET)

The idea is to use AI models for growth prediction using a data driven approach. There a period of data acquisition of $35 + 52 + 37$ days (each in a different season) focused on salad.

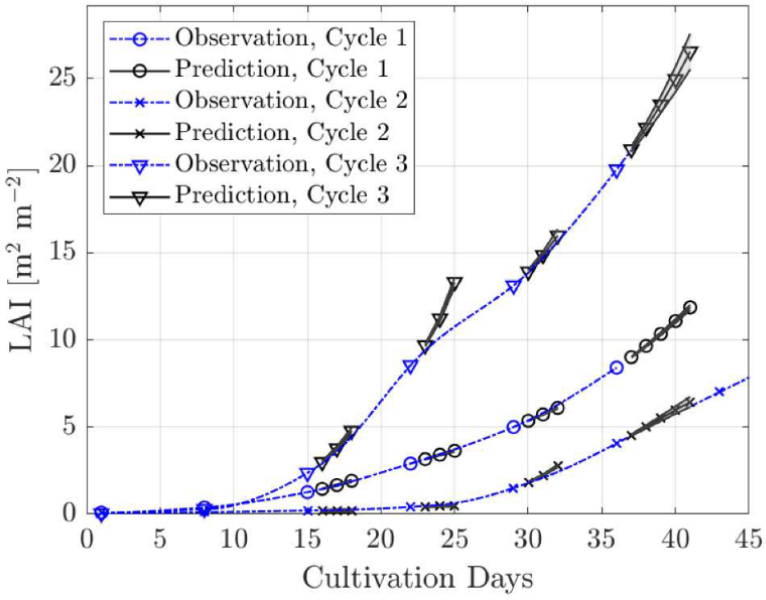


Figure 4.2: Crops growth AI prediction against acquired data

4.2 IoT in Ambient Assisted Living

The first topic discussed is the `DOREMI` project, which stands for *Decrease of cOgnitive decline, malnutRition and sedEntariness by elderly empowerment in lifestyle Management and social Inclusion*

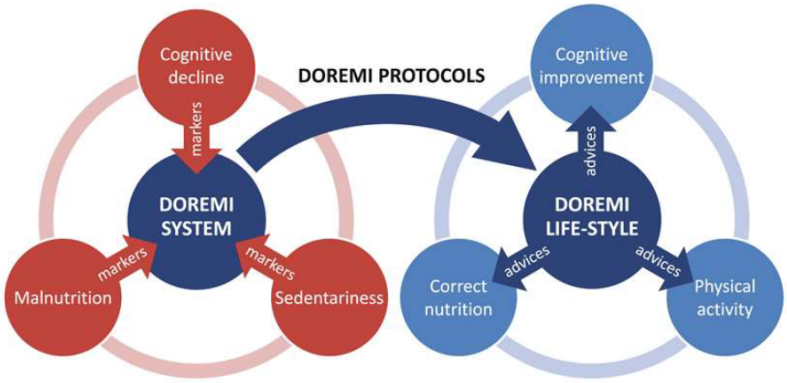


Figure 4.3: DOREMI project

The idea is to gather data from various sources, in order to obtain a profile of an elder.

1. Wristband
   i. assess physical activities, hearth rate, calories expenditure, ...
2. Balance board
   i. assess user balance according to the Berg scale
3. Environmental sensors
   i. Indoor localization
   ii. Socialization assessment
4. App for tablet
   i. Enter dietary data
   ii. Play cognitive, social, exer-games
   iii. Receive feedbacks

KPI type	KPI	Data	Device
Clinical	Vital parameters	Weight	Balance board
		Balance	
		Heart rate	DOREMI wristband
	Physical activity	Wrist acceleration	
		Number of steps	
		Indoor position	
		Sensors activations	Environmental WSN
Social	Number of interactions	GPS	Smartphone

Figure 4.4: DOREMI Sensors and related data

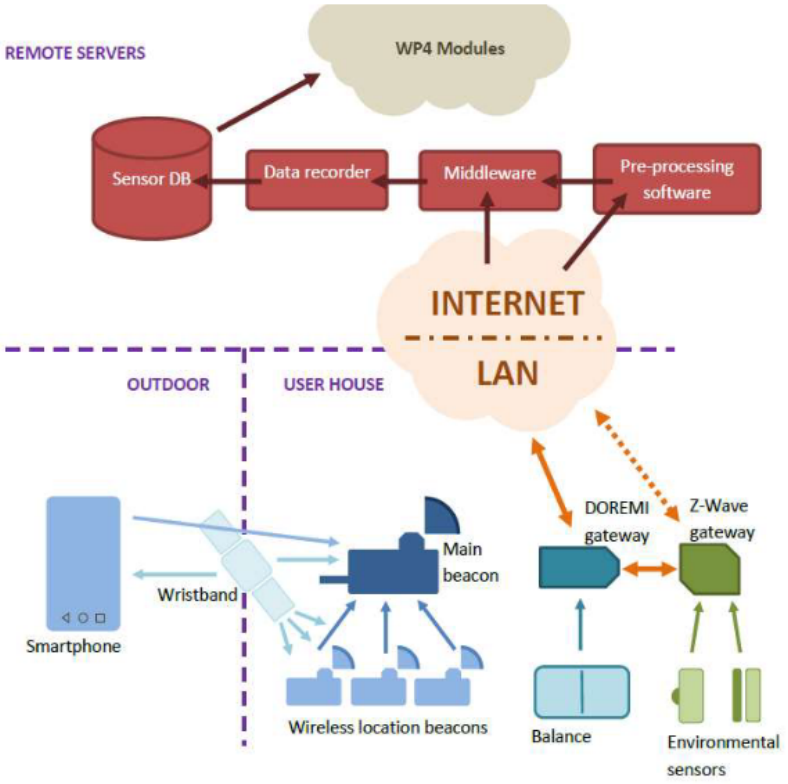


Figure 4.4: Wireless sensor network and Cloud architecture

They used chinese Wii Balance Board replicas to asses data on balance and weight of the elder. Wristbands were needed to measure the Heart Rate, which was later used to establish the calories consumption by applying a Fourier transform to it.

Some sensors instead when activated in a given sequence could indicate a human entering or leaving the house. Overall about 20 devices per user were used. For the "pilots" 400 devices were used in total.

They had to use two distinct databases, because some sensors provided time-series data, while more complex ones, such as the balance board, provided measurements.

32 **pilots** were chosen ranging in 65 to 80 years, half living in Italy and half in the UK, all having

- ◇ Mild to moderate cognitive impairment
- ◇ At risk of malnutrition
- ◇ Low physical activity

The pilots were divided into a *control group* and an *intervention group* made up of 7 and 25 users respectively.

The pilots were forced to follow some rules each day and each, such as wearing the wristband, play the cognitive games, perform suggested exercises etcetera.

4.2.1 Human Factor

When a system is developed for humans it is difficult to predict and handle all possible scenarios. The human factor in this example impact very much, especially the lack of motivation.

Other issues were related to inserting the food intake, picking up people spread around Genoa for social meetings, people not wanting sensors in their houses etc.

Chapter 5

IoT Design Aspects

Each device is usually low power and low cost small and autonomous system, equipped with processor, memory and radio transceiver along with sensing and/or actuating elements, with everything being powered a battery (or something equivalent such as solar cells).

The main limitations in the design of IoT devices are due to the abovementioned components —aside from the sensors/actuators— and is not clear whether technology improvements will overcome such constraints.

5.1 Issues

1. Energy efficiency
   i. sensors are battery-powered or use energy harvesting
   ii. need for HW/SW energy efficient solutions
2. Adaptatability to changing conditions
   i. need for dynamic network management and programming
3. Low-complexity, low overhead protocols
   i. need at any level of the protocol stack due to limitation of nodes' resources
4. Security
   i. at all layers ot the stack
5. Multihop communications
   i. need for protocol stacks and routing protocols
6. Mobility
   i. Need for dynamic routing protocols Data storage and (pre-)processing

5.1.1 Moore's law

"The number of transistors that can be (inexpensively) embedded in a chip grows exponentially"

In other words, *"it doubles every two years"*

Three different —but equally true— interpretations may be given to such law:

1. The performance doubles every two years at the same cost
   Up to now this is true for processors of servers/desktops
2. The chip's size halves every two years at the same cost
   Consequently also the energy consumption is reduced
3. The size and the processing power remain the same but the cost halves every two years

5.2 Battery consumption

In the previous decades the focus was on minimizing the number of bytes sent to optimize network performance. For IoT instead, the focus is to keep the radio off most of the time, since it consumes about 40% of the battery.

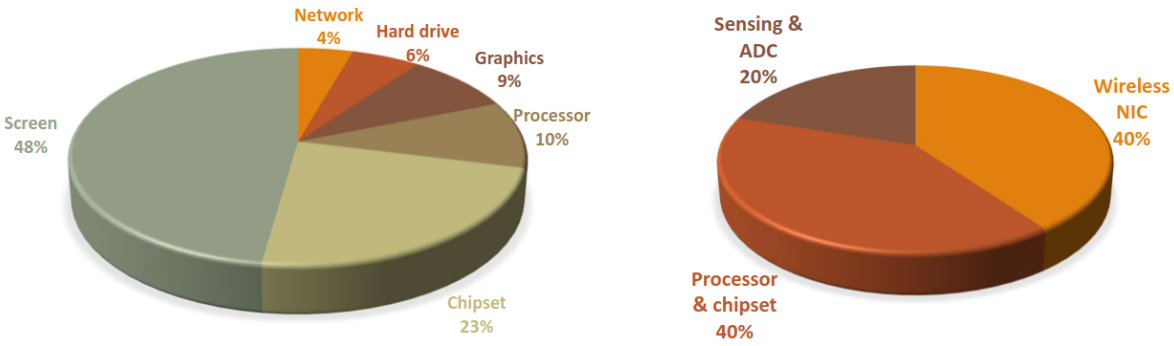


Figure 5.1: Laptop and IoT Sensor battery consumption percentages

5.2.1 Duty Cycle

Also turning off CPU and radio is battery consuming, so it must be done according to some criteria.
Since the activity of an IoT device is (mostly) repetitive, it may define a **duty cycle** by alternating periods of activity to periods of inactivity.

The duty cycle of a system (or a component / device) is defined as the fraction of one period in which the system is active.

100% means always active, while 1% means active only 1% of its period

Note that even if the duty cycle of the whole device may be defined at application level, each component has its own duty cycle, adding some "unpredictability" to the power consumption equation.

```
void loop() {
   // reads the input from analog pin 0:
   turnOn(analogSensor);
   int sensorValue = analogRead(A0);
   turnOff(analogSensor);
   // converts value into a voltage (0-5V):
   float voltage = sensorValue * (5.0 /
       1023.0);
   // transmits voltage over the radio
   turnOn(radioInterface);
   Serial.println(voltage);
   turnOff(radioInterface);
   // waits for next loop
   idle(380);
}
```

Sometimes the specification of an IoT device indicates the power consumption expressed in mA for each component in each state it may be (idle/read/write/etc.). Even if not completely precise, the specification may provide a good approximation of the consumption.
It is also important to note that there should be a record in the specs indicating how much capacity the battery loses over time. Most batteries lose a low percentage (e.g. 3%) capacity each year.

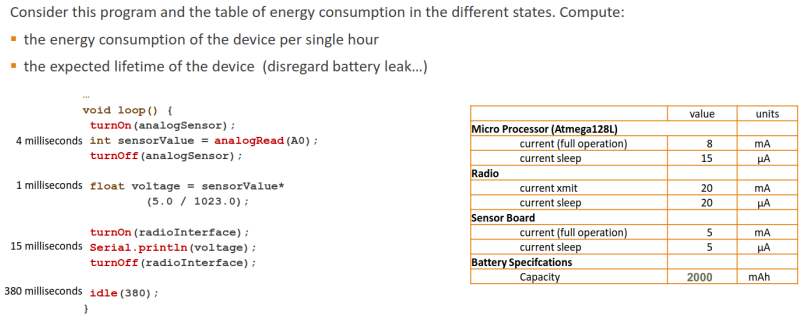


Figure 5.2: Exercise on energy consumption and duty cycling

For what concerns the exercise, these are the calculations

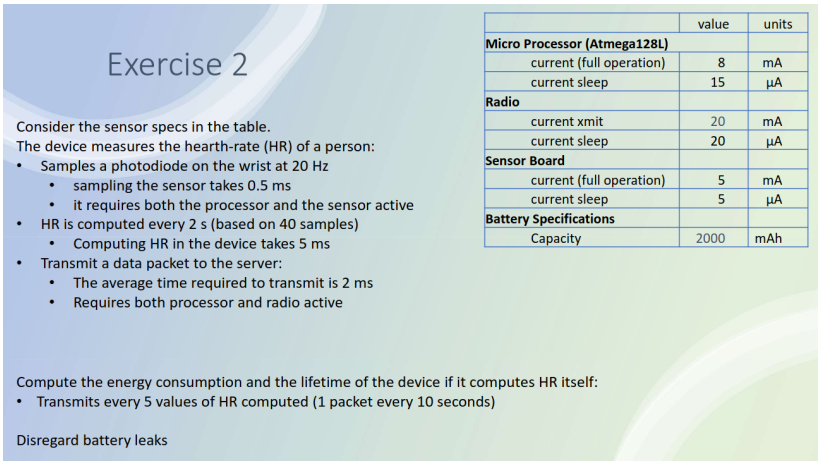
CPU	5	0.05 * 8	0.95 * 0.015	0.41425
Radio	3,75	0.0375 * 20	0.9625 * 0.020	0.9425
Sensor	1	0.01 * 5	0.99 * 0.005	0.05495

Table 5.1: Exercise calculations and solutions

5.2.2 MAC Protocols

In general they are Low-level communication protocols to send/receive packets to/from in-range sensors, but in IoT they also implement strategies for energy efficiency, by synchronize devices and turning off the radio when it is not needed[1].

Exercise 1

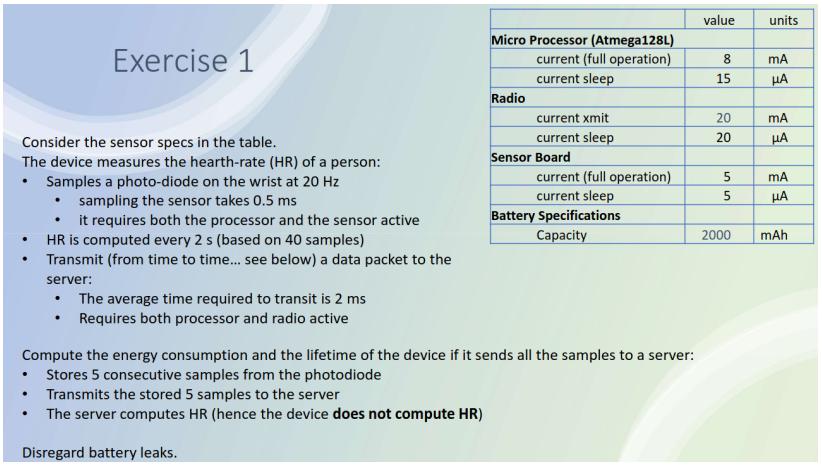


Duty cycle sensor $= 0.01$
Duty cycle radio $= 0.008$
Duty cycle CPU $= (0.5 * 5 + 2)/250ms = 0.018 = 0.01 + 0.008$

Radio energy`ON`	$=20mA * 0.008$	$= 0.16mA$
Radio energy`IDLE`	$=20\mu A * 0.992$	$= 0.01984mA$
Sensor energy`ON`	$=5mA * 0.01$	$= 0.05mA$
Sensor energy`IDLE`	$=5\mu A * 0.99$	$= 0.00495mA$
CPU energy`ON`	$=8mA * 0.018$	$= 0.144mA$
CPU energy`IDLE`	$=15\mu A * 0.982$	$= 0.0147mA$

Total energy per hour $=$
$0.16 + 0.01984 + 0.05 + 0.00495 + 0.144 + 0.0147 = 0.39349mA$
Total energy available $=$ 2000mAh
Expected lifetime $= 2000/0.39349 = 5084.5h = 212d$

Exercise 2



Duty cycle sensor $= 0.01$
Duty cycle radio $= 2/10000ms = 0.0002$

[1] Equivalent to excluding a device from the network

Duty cycle processor $= (0.5 * 200 + 2 + 5 * 5)/10000 = 0.0127$

Radio energy`ON`	$=20mA * 0.0002$	$= 0.004mA$
Radio energy`IDLE`	$=20\mu A * 0.9998$	$= 0.01998mA$
Sensor energy`ON`	$=5mA * 0.01$	$= 0.05mA$
Sensor energy`IDLE`	$=5\mu A * 0.99$	$= 0.00495mA$
CPU energy`ON`	$=8mA * 0.0127$	$= 0.1016mA$
CPU energy`IDLE`	$=15\mu A * 0.9873$	$= 0.0148mA$

Total energy per hour =
$0.004 + 0.01998 + 0.05 + 0.00495 + 0.1016 + 0.0148 = 0.19533mA$
Total energy available = 2000mAh
Expected lifetime $= 2000/0.19533 = 10238.5h = 426d$

The expected lifetime doubled! This is due to the fact that the radio is on for a very short time, and the CPU is on for a longer time but with a lower consumption.

*"**Sending** 1 bit costs 1000 times more than **computing** it" -cit. Anon*

Chapter 6

Case Study - Biologging

Biologging involves the direct observation of animals (and humans). By using sensors, such as accelerometers, GPS, and cameras, biologists can collect data on the behavior and physiology of animals in almost all the aspects of their life.

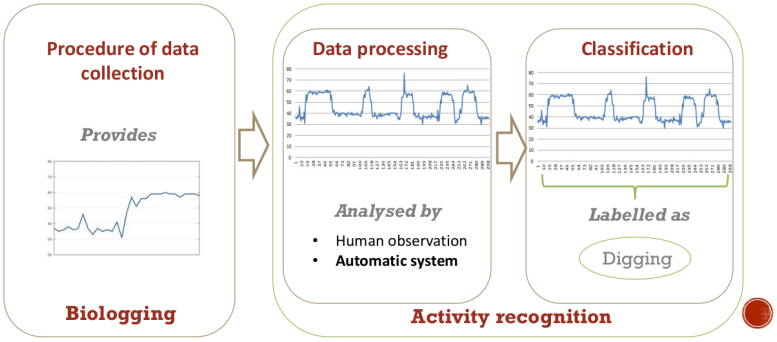


Figure 6.1: Biologging data lifecycle.
Biologging provides raw data which is later processed and classificated.

Strengths

- ◇ Data are punctual observations collected through sensors;
- ◇ Observation in both domestic and wild environments
- ◇ Availability of a huge amount of data (time-series data)
- ◇ Although this requires suitable data analysis algorithms.

Weaknesses

- ◇ Need to attach the device to the subject;
- ◇ Need to physically retrieve the device
- ◇ When the memory fills up or battery drains out
- ◇ Device lifetime in battery- intensive monitoring
- ◇ Often off-line analysis

6.1 Device perspective

The common sense may lead to an approach involving a collector device having a low power logger which produces —huge amounts of— time series data, which then should either send the data to a remote station, or to be physically retrieved to download the time series data.
In case of biologging both options are typically not feasible, because sending data considerably reduces the battery life, and retrieving the device from the back of an elephant is a risky and surely not easy task. ☺

A more *novel approach* consists in **embedding** the automatic classifier on-board, and store and trasmit only the results of classification obtaining both memory and communication efficiency, at the cost of computing power.

6.2 Tortoises case study

"Localizing tortoise nests by neural network" is a study published by Chessa et. al in 2016. The **motivation** for the study was that Protection programs of tortoises aim at retrieving the eggs and bringing them at a protection center, where hatchlings are protected during their most vulnerable period, while the **problem** to overcome was that the identification of tortoise nests in wild is usually very challenging, as tortoises are extremely good in hiding them.

The study aimed to localize the nests of tortoises by using a neural network classifier, which was trained on the data collected by a device attached to the tortoises. The device was equipped with a GPS and an accelerometer, and was able to collect data on the tortoise's movements. The classifier was trained to recognize the patterns of movement associated with the construction of a nest, and was able to accurately predict the location of the nest based on the data collected by the device.

Tortoise@ —the designed classifier's name— is structured as depicted in the Fig 6.2 below.

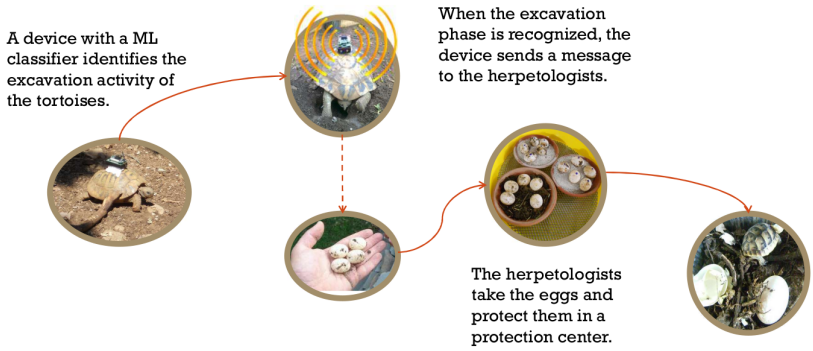


Figure 6.2: Tortoise@ architecture

6.2.1 What about tortoise nests?

- ◇ Tortoises lay eggs in spring (over 4 months)
- ◇ A single tortoise may lay eggs more than once (generally up to three/four times)
- ◇ Nesting happens at daytime, in warm and well-lit places
- ◇ Nesting lasts for more than one hour
- ◇ The tortoises digs their nests with their hind paws

Hence:

1. The device lifetime must be at least 4 months
2. Digging activity indicates a probable nesting: it can be detected with accelerometers, because the tortoise moves its legs in a peculiar way, and initiating a nest takes about 90 seconds. However, not all the digging activities are related to nesting.
3. Use of environmental sensors (light and temperature) to limit the use of accelerometers, because without specific environmental conditions the tortoise will not nest, so it is not necessary to spend energy in that period of time.

6.2.2 Data collection and processing

The dataset was collected at the "Centro di protezione tartarughe meditarrenee" at Massa Marittima in Italy, and was made up of raw time-series from an accelerometer. Such raw data was processed to build a Sequences datasets and a Patterns dataset.

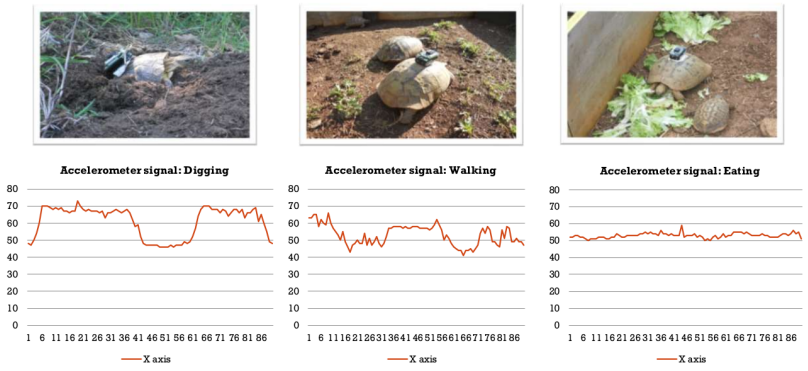


Figure 6.3: Tortoise activities and accelerometer data

The use of SVM and of several models of neural networks

(ESN, IDNN, CNN) are considered. Two approaches may be adopted:

1. **Asynchronous tasks**: classification of single windows. (
   IDNN, CNN, SVM, ESN
)
   The output of the neural network depends on an entire window
2. **Synchronous tasks**: classification in step by step across a stream of data. (
   ESN
)
   The output of the neural network does not need an entire window of samples. It is based on the samples of the last 90 seconds (sufficient to find a pattern)

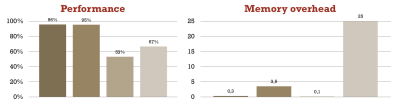


Figure 6.4: Models comparison for *Asynchronous tasks* The hardware used was an Arduino equivalent, with a 16 MHz clock and **4 KB** of RAM, which made some of the displayed model completely unfeasible.

Energy consumption must be clearly taken into account when establishing the sampling frequency, and some assumptions helped achieving efficiency, such as avoiding sampling at night, or when environmental conditions are not suitable for nesting.
Also the data structures required by the models must be taken into account, as they may require a lot of memory, which is a scarce resource in the device.

Only two main data structures were used, one to store the activity recognition machine and another to store acceleration samples.

Cloud vs Local processing

Cloud processing implies:

- ◇ Store (and transmit) GPS position every half an hour (two long, 32 bits each)
- ◇ Store (and transmit) the acceleration data sampled at 1Hz (10 bits each sample)
- ◇ For 4 months continuously: 46 KBytes (GPS) + 13 Mbytes (accelerometer)

Local processing implies instead:

- ◇ **Transmit** GPS position: 32 bits latitude, 32 bits longitude only when detected excavation
  Occurs only a few times in 4 months (not all excavations conclude with nesting)
- ◇ **Storage**:
  - Accelerometer data: at most 3 KBytes (a time window), even less with ESN
  - GPS: only when detects excavation

Chapter 7

MAC Protocols

Duty cycle has a critical role for what concerns devices' networking. Turning the radio off on a device for some time —indirectly— affects also the other devices in the network, as they may not be able to communicate with the sleeping device.
It must be taken in consideration also that in low-powered multi-hop network, hence where only a few nodes have a direct connection to the access point, internal nodes may act as routers, and they may need to be awake to forward packets towards the access point.
In such scenario it is more clear why duty cycle and MAC protocols may disrupt performance and energy efficiency.

MAC protocols for IoT devices have some challenges, such as:

- ◇ **Energy efficiency**: the protocol should be able to save energy, as the devices are battery-powered;
  - Reduce duty cycle
  - Maintain network connectivity
- ◇ Find a **tradeoff** between Energy and Latency & Bandwidth;

Three approaches are possible:

1. **Synchronous MAC protocols** (`S-MAC`): nodes are synchronized and they know when to transmit and when to sleep;
2. **Preamble sampling** (`B-MAC`): nodes wake up periodically to listen to the channel, and if they hear a preamble, they wake up completely to receive the packet;
3. **Polling** (as in 802.15.4): a node is designated as the coordinator and it polls the other nodes to check if they have data to send.

7.1 Synchronization

S-MAC is a protocol which implements only **local** synchronization. It is based on the concept of *periodic listening* and *sleeping*; in fact, nodes alternate synchronized listen and sleep periods, i.e. they are all simultaneously active only for a short time.

Synchronization is achieved by means of periodical (local) broadcasts of SYNC frames containing the wake/sleep schedule of the node. If a node detects adjacent nodes with a predefined schedule, it can synchronize with them, otherwise it chooses its own period, and broadcasts it with SYNC frames. A node may revert its schedule to the one of other nodes in case it is not aligned with them

7.1.1 Issues

- ◇ **Clock drift**: nodes may drift from their schedule, and they need to resynchronize;
  Cheap IoT devices may have a clock drift of 1% per day, and in general are considered to have an unreliable clock.
- ◇ **Hidden terminal problem**: if a node is not synchronized with the others, it may transmit when it should not, causing collisions;
- ◇ **Overhead**: the SYNC frames may cause overhead, and they may not be received by all the nodes.

Adaptive listening is a technique to reduce the overhead of SYNC frames, by making nodes listen for a longer time if

they have not received a SYNC frame for a long time. This leads to slightly more energy consumption, but provides much better network performance, resulting in a good tradeoff.

S-MAC has *never* been used as a standard, it relies on too many assumptions, and it is not suitable for large networks.

7.2 Preamble Sampling

In general we can measure the complexity of a solution to a problem in terms of how many parameters have to be tuned in order to make it work. The more parameters, the more complex is to make it work, and the more likely it is to fail.

B-MAC is extremely simple, and relies on the setup of a single parameter: the **wake-up interval**. A sender sends a packet **whenever** they want, but not *whatever* they want: packets are sent with a *long preamble* —in other words, they start shouting before telling the actual information—, which is a sequence of bits that is sent before the actual packet, and it is used to wake up the receiver.
Nodes activate their radio only for a short time to listen to a preamble, and if they hear one, they wake up completely to receive the subsequent packet.

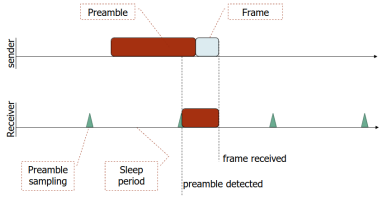


Figure 7.1: BMAC TX-RX schema

The key point is that the **preamble** should be longer than the **wake-up interval** (sleep period).

This ensures that the receiver will not miss any packet preamble

The idea is to waste some energy on the sender node in order to save energy on receiver ones. Even though it appears slightly counterintuitive, it is a good tradeoff, because often there are more receivers than senders.

The receiver has to stay awake —on average— for half the preamble, for the duration of the data transmission, and for the processing of such data.

X-MAC is an improvement of B-MAC, which uses a **short preamble** to wake up the receiver, and then sends a **longer preamble** to synchronize with the receiver. The preamble contains information about who should receive the data, and in case of BoX-MAC contains the data itself. The "empty" preamble is replaced by the repeated data interleaved with an ACK.

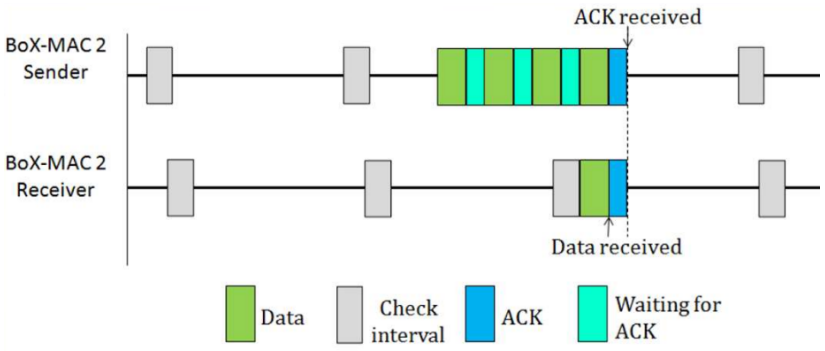


Figure 7.1: BoX-MAC TX-RX schema

7.3 Polling

Polling is a technique used in 802.15.4, where the nodes are organized asymmetrically:

- ◇ one master node issues periodic beacons
- ◇ several slave nodes that can keep the radio off whenever they want

When a slave hears the beacon and sees that there for pending messages for him, it requests them to the master.

Chapter 8

IEEE 802.15.4

IEEE 802.15.4 is a standard specifying **physical** and **MAC** layers for low-rate wireless personal area networks (LR-WPANs).

It is infrastructure-less, short range and its Physical layer may coexist with IEEE 802.11 (Wi-Fi) and IEEE 802.15.1 (Bluetooth).

Physical Layer

The Physical layer may operate in three possible frequency bands:

- 868.3 MHz (Europe)
- 902-928 MHz (Americas) (11 channels, 2 MHz each)
- 2.4 GHz (Worldwide) (16 channels, 5 MHz each)

The Physical Layer works well even in low *Signal-to-noise ratio* (`SNR`) environments, and it is able to work with a very low power consumption. In other words, it is less likely to misread a packet.

For example, the "shouting" of the preamble in B-MAC increases the SNR.

- **Data Service**
  - Transmission/Reception of PHY Protocol Data Unit (PPDU) through the physical medium.
- **Management Service**
  - Radio transceiver activation/deactivation (to implement policies of energy efficiency)
  - Energy Detection - ED
  - Link Quality Indicator - LQI
  - Channel Selection
  - Clear Channel Assessment - CCA
  - PHY-PIB (PHY PAN Information Base) Configuration

MAC Layer

MAC layer provides various services:

- **Data services**
  - transmission and reception of MAC frames (MPDU) across the physical layer.
- **Management services**
  - Synchronization of the communications
  - Channel access
  - Management of guaranteed time slots
  - Association and disassociation of devices.
- **Security services**
  - Data encryption
  - Access control
  - Frame integrity
  - Sequential freshness

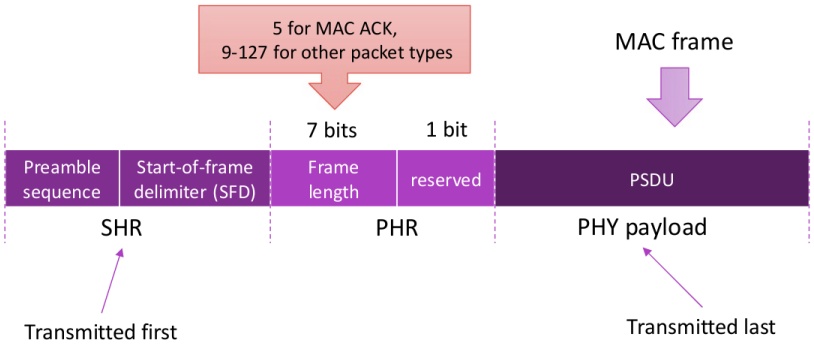


Figure 8.1: Physical frame structure

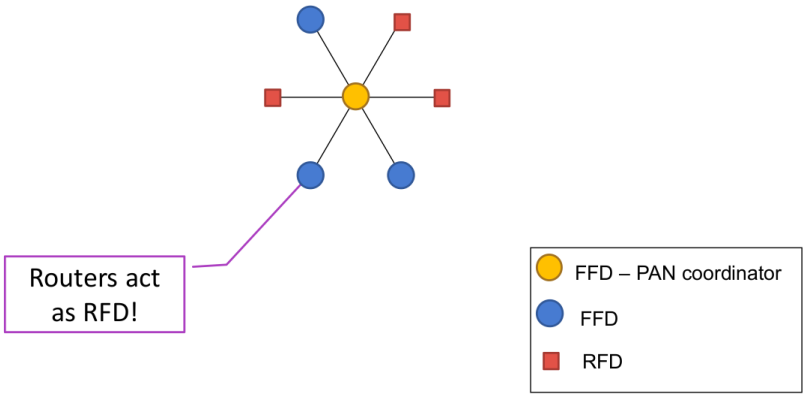
- `SHR` (Synchronization Header): synchronisation with the receiver
- `PHR` (PHY Header): information about the frame length
- `PHY` Payload: the MAC frame

This layer defined two types of devices:

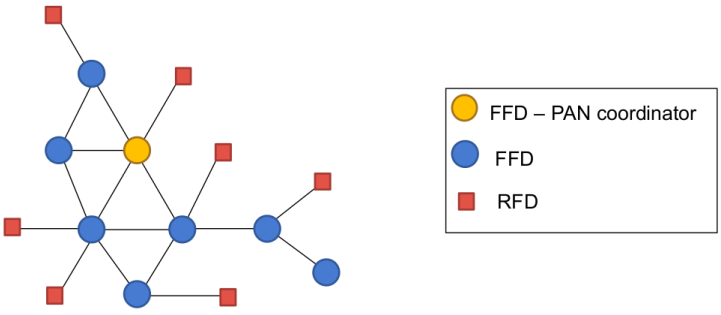
- **Full-function device** (FFD): can act as a coordinator;
- **Reduced-function device** (RFD): cannot act as a coordinator.

At the MAC layer there may be different network topologies:

- **Star**: a central FFD PAN coordinator communicates with all the other nodes, which may be either FFDs or RFDs.



- **Peer-to-peer**: again there is a PAN coordinator, FFDs are routers and RFDs are end devices.



- **Cluster-tree**: a coordinator communicates with other coordinators, which in turn communicate with other nodes.

8.1 Channel Access

There are two options for channel access:

1. **Superframe**: used in star and tree-structured p2p networks, enforces the synchronization of devices.
2. **Non beacon-enabled** (*Without superframe*): this schema is more general and supports communication in arbitrary p2p networks.

8.1.1 Superframe

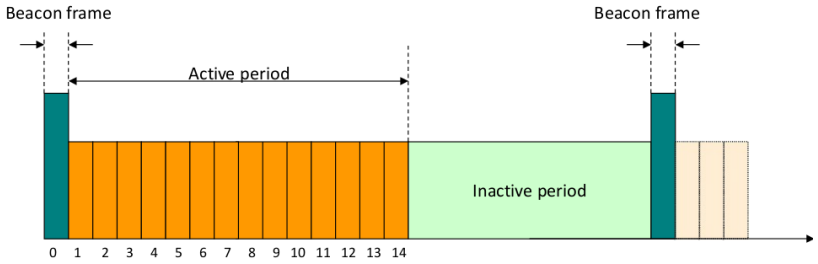


Figure 8.2: Beacons and superframes in 802.15.4

There are equally sized time slots, with the first one being the beacon frame, sent by the PAN coordinator to begin the superframe.

The end devices can communicate with the coordinator in the remaining time slots. The beacons are used to:

- Identify the PAN,
- Synchronize the devices in the PAN,
- Communicate the structure of the superframe.

The superframe comprises an active and an inactive portion. The active portion is made up of (max) 16 time slots and indicates where the communication has to happen. It is divided into two parts:

1. *Contention Access Period* (**CAP**) which comprises up to 15 time slots and the devices here have to "compete" to access the channel.
   - a device shall wait for a random number of slots first.
   - if the slot is busy the device shall wait for another random number of slots before trying again.
   - If the channel is idle, the device can transmit
   - Once the transmission starts, the node keeps the medium until the end of the frame
2. (optional) *Contention Free Period* (**CFP**), which occupies the last time slots of the active period and is meant for low-latency applications or applications with specific data bandwidth.
   It is divided into *Guaranteed Time Slots* (**GTS**)
   - Each GTS assigned by the PAN coordinator to a specific application.
   - The application accesses the GTS without contention.
   - The GTS may comprise more than one time slots.
   - Up to 7 GTS within a single CFP, and each may last more than one time slot.

   What if there are too many GTS requests by application?
   The coordinator answers "picche" ☺

8.1.2 Non beacon-enabled

In this scenario coordinators (PAN coordinator and routers) are always on and ready to receive data from the end-devices, and there are no beacons or slots: communication based on unslotted `CSMA-CA` protocol.

Data transfer from coordinators to end-devices is **poll-based**: the end device periodically wakes up and polls the coordinator for pending messages, and then the coordinator either sends back the pending messages or signals that no message is pending.

Clearly this is possible because the coordinators are expected to be always on.

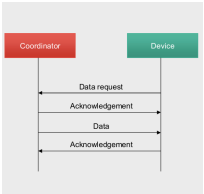
8.1.3 Trasferring data

Data transfer may be of three types, and have different implementations for beacon and non-beacon enabled networks:

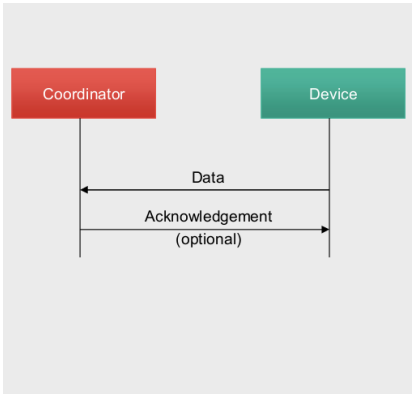
1. End-device to coordinator (or router)
2. Coordinator (or router) to end-device
3. Peer to peer.

Beacon enabled

1. Data transfer from an end device to a coordinator:
   i. The end device first waits for the network beacon to synchronize with the superframe.
   ii. If it owns a GTS it uses it without contention, otherwise it transmits the data frame to the coordinator using the slotted CSMA-CA protocol in one of the frames in the CAP period
   iii. The coordinator may optionally send an acknowledgement



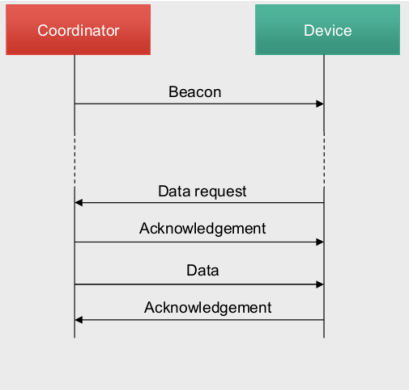
2. Data transfer from a coordinator to an end device:



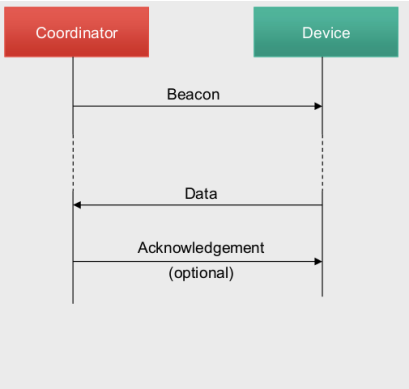
   i. The coordinator stores the message and signals that the message is pending in the beacon;
   ii. The end-device sleeps most of the time and it occasionally listens to the beacon to check for pending messages, and it notices that there is one, it requests the message to the coordinator.
   iii. The coordinator sends the pending message in a successive slot of the CAP
   iv. The device sends a *mandatory* acknowledgment frame in a successive time slot, so that the coordinator may remove the pending message from its list
3. Data transfer in Peer to peer:
   ◇ Between the coordinator (or a router) and an end-device one of the previous schemes is used. But this is not possible between two end-devices, because they cannot communicate directly.
   ◇ Between the coordinator and a router (or between two routers), the sender must first synchronize with the destination beacon and act as an end device.
   The measures to be taken in order to synchronize coordinators are beyond the scope of the IEEE 802.15.4 standard. ☺

Non beacon-enabled

1. Data transfer from an end device to a coordinator:



   - ◇ the end device simply transmits its data frame to the coordinator using unslotted CSMA-CA, assuming that the coordinator is always on.
   - ◇ The coordinator acknowledges the successful reception of the data by transmitting an *optional* acknowledgment frame
2. Data transfer from a coordinator to an end device:



   i. The coordinator stores the message and waits for the device to request for the data.
   ii. A device can poll the coordinator for pending messages by transmitting a Data request (using unslotted CSMA-CA).
   iii. The coordinator send an ack for the request.
   iv. The coordinator transmits the pending message(s) to the devices.
   If none present, an empty message is sent
   v. The device sends an ack and the coordinator discards the sent messages.
3. Data transfer in Peer to peer:
   i. Each device may communicate with every other device in its radio range
   ii. The devices will need to remain always active or to synchronize with each other.
      - ◇ In the first case the device can directly transmit the data.
      - ◇ In the latter case the devices synchronization is beyond the scope of the IEEE 802.15.4 standard (it is left to the upper layers.

8.2 Services - MAC Layer

The MAC layer provides **Data services** by exploiting only three primitives

1. `DATA.request`: invoked by the upper layer to send a message to another device
2. `DATA.confirm`: reports the result of a transmission requested with a previous `DATA.request` primitive to the upper layer
3. `DATA.indication`: corresponds to a receive primitive: it is generated by the MAC layer on receipt of a message from the physical layer to pass the received message to the upper layer

request, confirm and indication primitives.

The MAC layer also provides **Management services**:

- ◇ PAN initialization
- ◇ Detection of existing PANs
- ◇ Devices association/disassociation
- ◇ Other service

Considering the **Associate Protocol** from the end-device perspective: it is invoked by a device willing to associate with a PAN that has already been identified by a preliminary execution of the SCAN service; this procedure thus applies to beacon-enabled networks.
The protocol is fairly simple:

1. The ASSOCIATE.request primitive is sent by the device and has as parameters:
   i. the PAN identifier
   ii. the coordinator address
   iii. the 64-bits extended IEEE address of the device

   Such request message is sent during the CAP using the slotted CSMA-CA protocol.
   Since the device has not yet joined the network, it uses its MAC to send the message.

Chapter 9

Embedded Programming

The learning objectes of this chapter are embedded systems in general, and the Arduino case study.

Embedded systems are systems that are designed to perform a specific task, and they are usually part of a larger system. Hardware and software are often designed together, aka "hardware-software co-design". They are typically based on microcontrollers, optimized for controlling I/O. They are often used in **real-time** systems, where timing is crucial.

Many types of microcontrollers are availble on the market:

- "General purpose", meaning that can be adapted to several embedded applications
- "Application specific integrated circuit"(`ASIC`s) which are very efficient and performative, but tightly bound to a specific task
- "System on a chip"(`SoC`s) which are a combination of a microcontroller and other components, like a radio module.
  The term is very broad, and it can refer to a wide range of devices.

When programming on microcontrollers it must be taken into account the **small memory footprint**, the absence of user interfaces, file systems, OSs. In general is not possible to program and compile code directly, or to control the program execution with a user interface.

Programming Challenges

- Timing correctness
- High reliability
- Often impossible to use debuggers
- Efficient use of memory
- Power management

9.1 Executable and SW organization

When a device starts it executes the `main` program, which first initializes device peripherals and then enters an infinite loop, implementing the functionalities of the device. Such loop naturally defines a duty cycle, typically consisting of:

- Reading from transducers
- Taking a decision
- Controlling actuators
- Optionally communicating with other devices

At low-level, the code interacts with the device hardware by means of commands and interrupts. In conventional OSs insteads commands are available in terms of system calls, which are later handled by the OS.

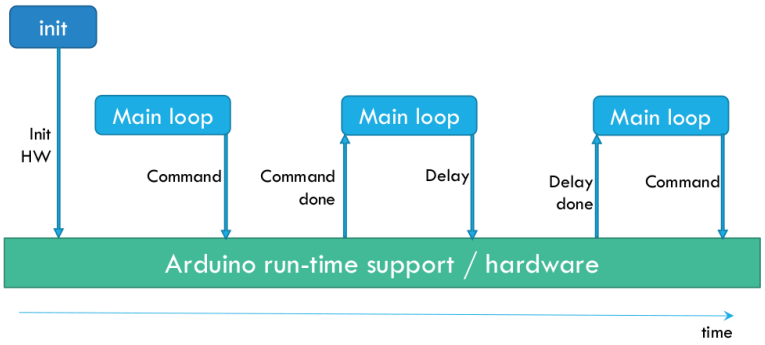


Figure 9.1: Sample flow of execution in Arduino.

9.2 Different Models to implement duty cycles

9.2.1 Arduino Model

The job of Arduino is defined in a single loop function, which clearly may invoke other functions anyway. Such loop is executed repeatedly by a *single* thread which is *never* suspended. In case of an I/O operation, the program waits for the operation to complete, and then continues.

9.2.2 TinyOS Model

TinyOS offers functions (**commands**) to program and activate the hardware, it abstracts interrupts in form of *events*. It defines non-preemptive tasks that are executed sequentially, to manage different activities.

With this approach a task never waits, and can be pre-empted (only) by events. However interrupts should be handled as soon as they arrive by means of an event handler, which should be as short as possible because other simultaneous interrupts would result in serious concurrency problems; this is the case of the `read handler` in the figure 9.2, which instead of processing the data, it yields them to the "upper-level" task by means of a `post` command.

`init` defines a timer which periodically triggers the `timer handler`, whose code may involve the operations to be performed periodically.

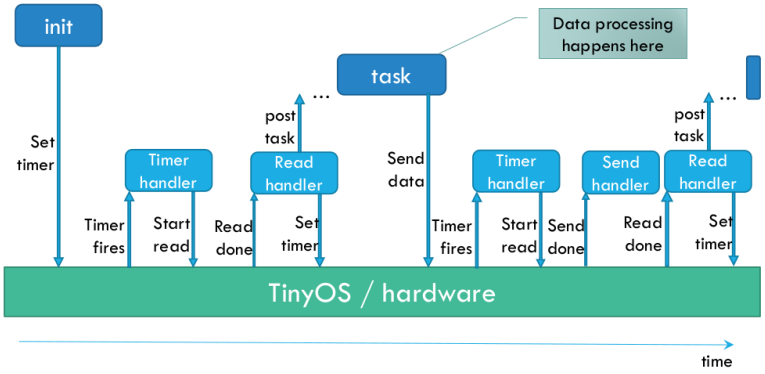


Figure 9.2: Sample flow of execution in TinyOS.

9.3 Arduino

Arduino is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. Its concept comprises the actual Device, the IDE, and the online Forum.

It is now considered part of the IoT world, even though in its original design it was not meant to be connected to the internet, it was more *Internet-of-*"Things".

9.3.1 Interrupts

Despite the main role of Arduino sketches is syncronous reading of sensors, it is possible to handle **interrupts**, allowing for asynchronous access to sensor data and actuators. There are three types of interrupts in Arduino:

- ◇ **External**: a signal outside Arduino coming from a pin
- ◇ **Timer**: internal Arduino timer
- ◇ **Device**: internal signal coming from a device such as ADC, serial line, etc.

Internal interrupts (either Timer or Device) are managed by the run time support of Arduino, external ones instead may be managed with an `attachInterrupt(interrupt#, func-name,mode);` instruction.

There are only two external interrupts available on the Arduino Uno, `INT0` and `INT1`, which are mapped to pins 2 and 3. They can be set to trigger on `RISING`, `FALLING`, `CHANGE`, `HIGH` or `LOW` level. The trigger is interpreted by the hardware, and the interrupt is very fast.

Question on the slides

```
void setup(){
   pinMode(2, INPUT);
   attachInterrupt(0, handler, CHANGE);
   return;
}
void handler(){}
void loop(){
   return;
}
```

9.4 Energy Management

Arduino Sleep Modes to limit power consumption (aka *Arduino Power Save* mode) can be activated by means of additional libraries, for example the `Low power` library. The actual sleep mode mechanisms depend on the version of the device, and of the processor. `LowPower.idle()` is a function from the `Low power` library that puts the device in idle mode, which is the lowest power consumption mode available.

Part II

Federica Paganelli

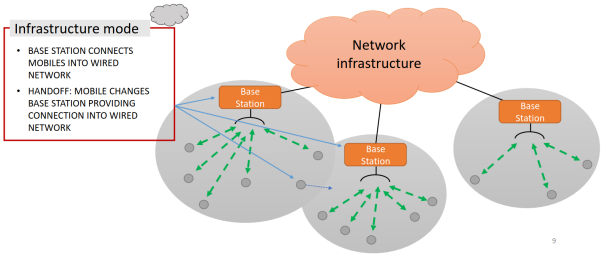
Chapter 10

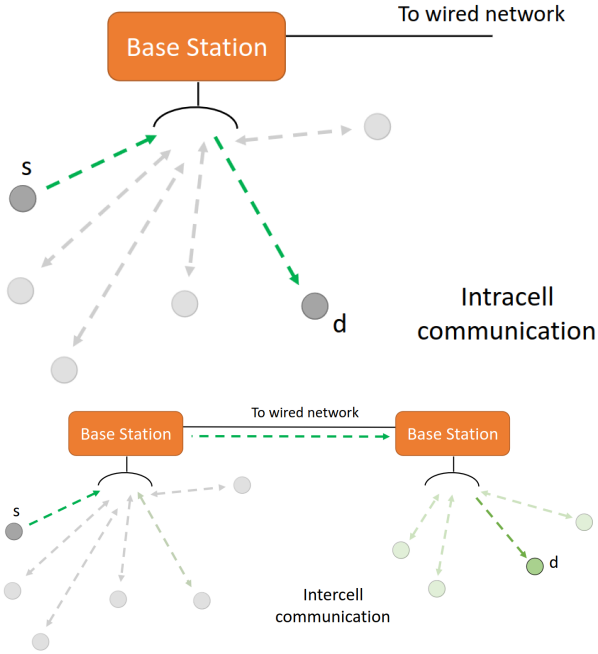
Wireless Networks

Wireless Networks are composed of **hosts**, which are end-system devices that run applications, typically battery powered.

Recall that *wireless* $\neq$ *mobility*

In general Wireless Networks may be based on the interaction *hosts* $\longrightarrow$ *base station* —or access point— or *hosts* $\longrightarrow$ *hosts*. The two resulting functioning modes are called *Infrastructure* and *Ad hoc networking*.





10.1 Link Layer

10.1.1 CSMA/CD

Basic idea of CSMA/CD:

1. When a station has a frame to send it listens to the channel to see if anyone else is transmitting
2. if the channel is busy, the station waits until it becomes idle
3. when channel is idle, the station transmits the frame
4. if a collision occurs the station waits a random amount of time and repeats the procedure.
   Refer to the slides of 21 February for more in depth usage examples

In short: CSMA/CD performs poorly in wireless networks. Firstly because CSMA/CD detects collisions while transmitting, which is ok for wired networks, but not for wireless ones. Secondarily, what truly matters is the interference at the *receiver*, **not** at the *sender*, causing the two problems known as *hidden* and *exposed terminal* problems; to

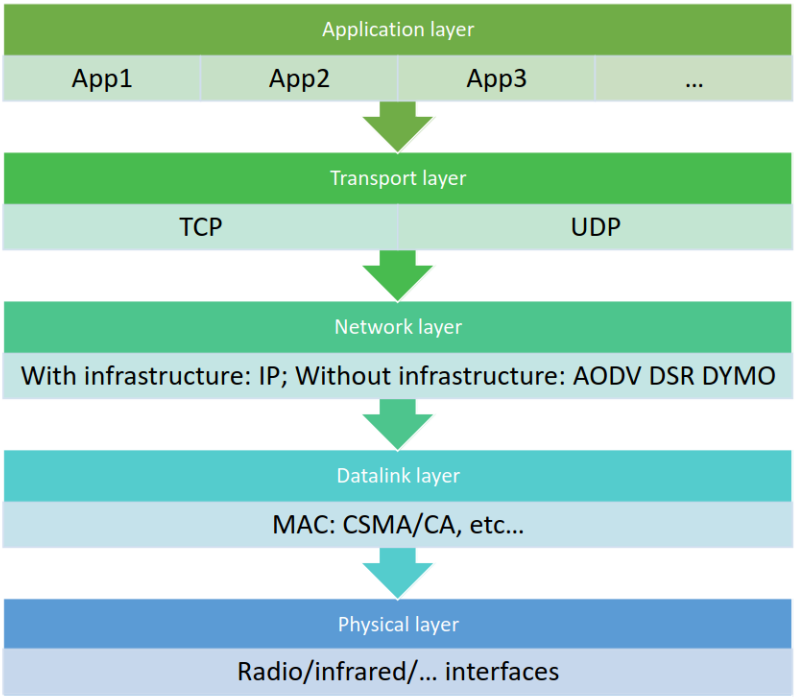


Figure 10.1: Protocol stack

better understand this point, look at the following figure, consider that at the sender, the signal strength of its own transmission (self-signal) would be too strong to detect a collision by another transmitter, making collisions happen at the receiver.

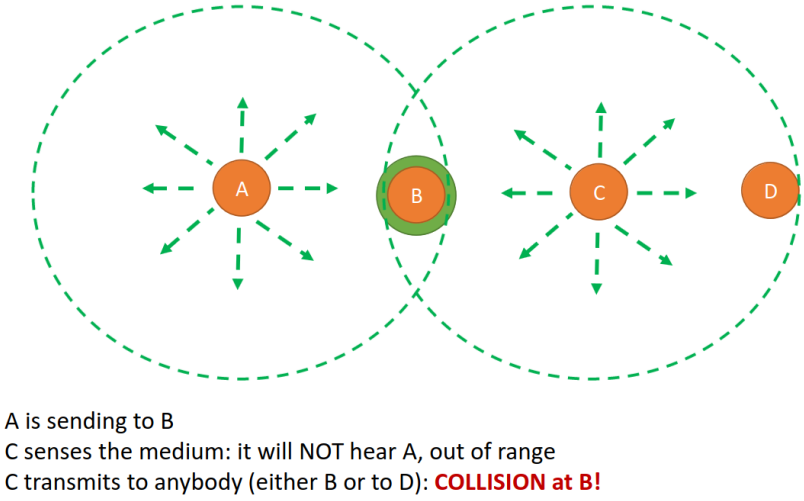


Figure 10.2: **Hidden Terminal** problem
Two or more stations which are out of range of each other transmit simultaneously to a common recipient

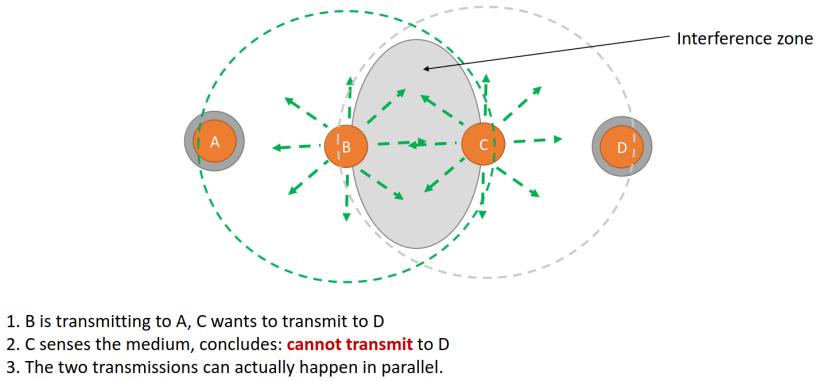


Figure 10.3: **Exposed Terminal** problem
A transmitting station is prevented from sending frames due to interference with another transmitting station

10.1.2 MACA

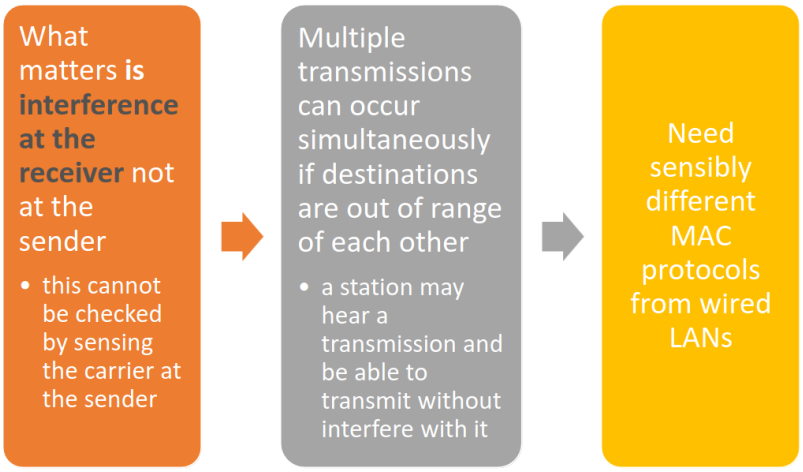


Figure 10.4: MACA Motivations

MACA stands for *Multiple Access with Collision Avoidance*

1. stimulate the receiver into transmitting a short frame first
2. then transmit a (long) data frame
3. stations hearing the short frame refrain from transmitting during the transmission of the subsequent data frame

Basically, a transmitting node sends a *Request to Send* `RTS` and a receiving node answers with *Clear to Send* `CTS`. Other nodes which hear `RTS` or `CTS` must stay silent until the transmission is over.

Further details and examples on how the protocol works are on the slides.

MACAW implements some improvements to MACA:

- ACK frame to acknowledge a successful data frame
- added Carrier Sensing to keep a station from transmitting RTS when a nearby station is also transmitting an RTS to the same destination
- mechanisms to exchange information among stations and recognize temporary congestion problems
  CSMA/CA used in IEEE 802.11 is based on MACAW

Chapter 11

IEEE 802.11

IEEE 802.11 standard	Year	Max data rate	Range	Frequency
802.11b	1999	11 Mbps	30 m	2.4 Ghz
802.11g	2003	54 Mbps	30m	2.4 Ghz
802.11n (WiFi 4)	2009	600	70m	2.4, 5 Ghz
802.11ac (WiFi 5)	2013	3.47Gpbs	70m	5 Ghz
802.11ax (WiFi 6)	2020 (exp.)	14 Gbps	70m	2.4, 5 Ghz
802.11af	2014	35 – 560 Mbps	1 Km	unused TV bands (54-790 MHz)
802.11ah	2017	347Mbps	1 Km	900 Mhz

Figure 11.1: IEEE 802.11 standards

All these standards `use CSMA/CA for multiple access`, and have base-station and ad-hoc network versions

TODO

Chapter 12

Mobile Networks

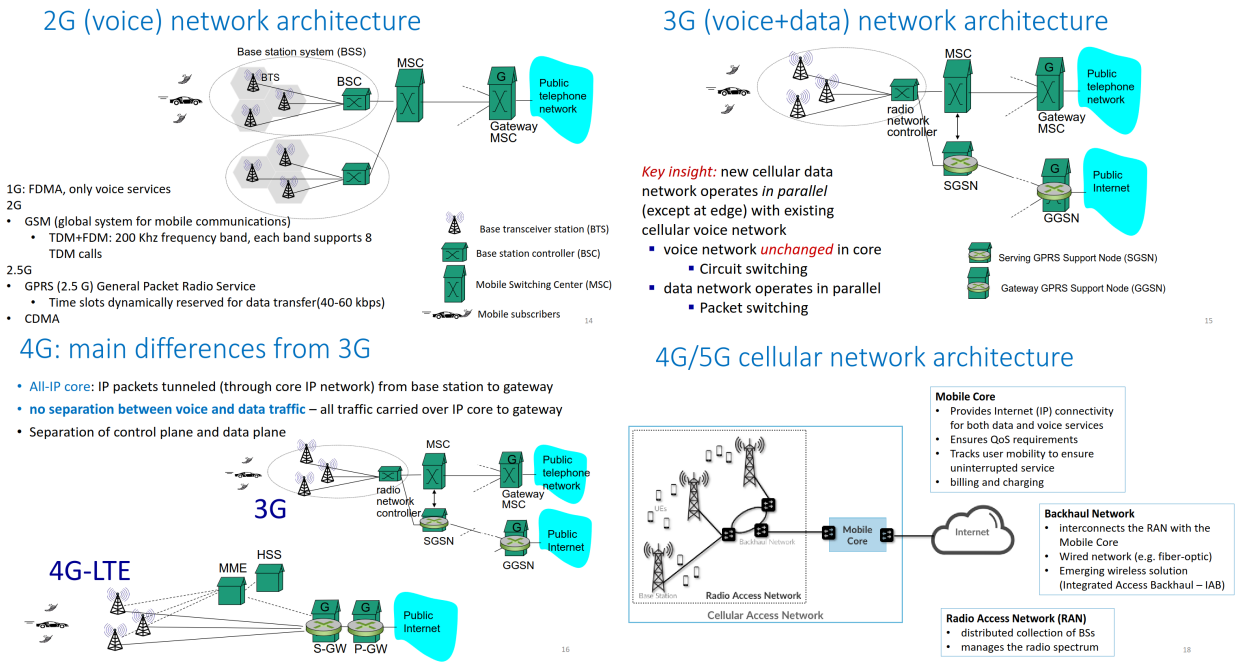


Figure 12.1: Mobile Networks architectures

The key point in **3G** is the introduction of a data service, operating in parallel with voice network, which forced the important modifications to the architecture.

In **4G** also the voice traffic uses *packet switching*, instead of circuit switching.

Control vs Data plane

Control plane includes routing protocols such as BGP and all the processes which handle and determine how data packets should be forwarded.

Data plane instead handles the transport of host/application data, and performs the actually forwarding of packets.

"Think of the control plane as being like the stoplights that operate at the intersections of a city. Meanwhile, the data plane (or the forwarding plane) is more like the cars that drive on the roads, stop at the intersections, and obey the stoplights" Cloudflare Data/Control plane

To handle devices' mobility there must be home network to rely onto.

With respect to the questions posed in the third image in Fig. 12.3, data being send from a device to a mobile one may be routed in three ways.

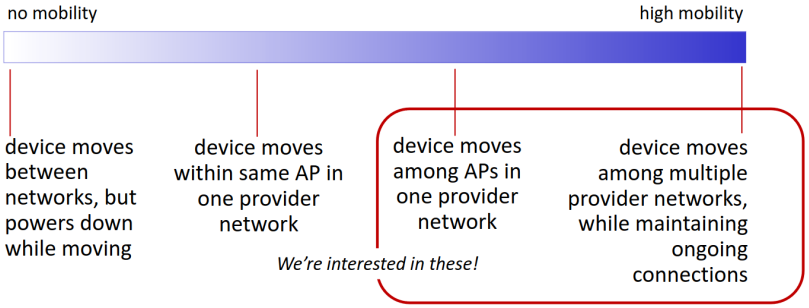


Figure 12.2: Mobility

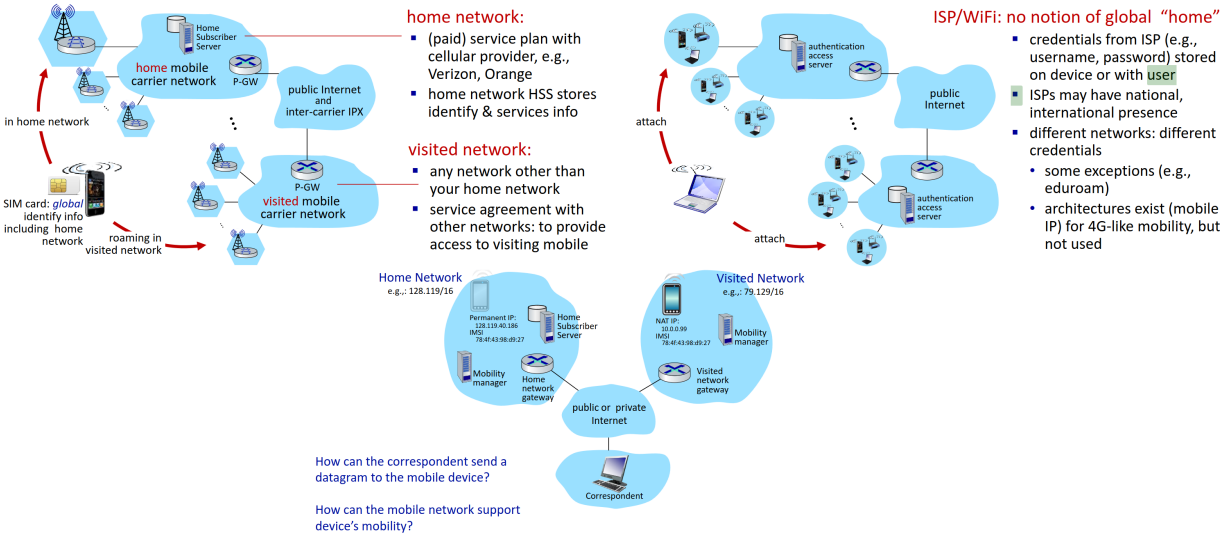


Figure 12.3: Home and visited Networks

1. The first is the canonical routing, using IP addresses and routing tables, but it is not a feasible scenario for billions of devices.
2. The other possibility is to rely on the edge of home and visited networks instead.
   i. **Direct routing**
   Sender gets foreign address of mobile, send directly to mobile
   ii. **Indirect routing**
   Communication from sender to mobile goes through home network, then forwarded to remote mobile

Many things are missing here. Refer to the slides or to chapter 7 of the course book (Kurose) for more information.

Chapter 13

Software Defined Networking

SDN is often referred to as a "radical new idea of networking". It has been made to overcome many requirements and current network technology limitations which arose in the last years:

The demand is increasing
1. Cloud computing
2. Big Data
3. Mobile traffic
4. IoT

Supply also increasing
5. Network technologies capable of absorbing high load
6. Performance of network devices increased
   i. CPU speed
   ii. Buffer capacity and speed
   iii. etc...

Traffic patterns[1] have changed and became more dynamic and complex
This is the most critical aspect

13.1 Traffic Patterns

Modern distributed applications typically access multiple databases and servers that must communicate with each other, creating a lot of *"horizontal"* traffic between servers —in addition to the *"vertical"* client/server one— which was initially neglectable, but now it is not.

Unified communications (UC) services are increasingly used within enterprises. Many communication services such as instant messaging (chat), presence information, voice, mobility, audio, web and video traffic, must be *integrated* into a unique service delivery platform

TODO

13.2 Layering

Layering, as it is applied in TCP/IP stack, implies decomposing delivery into fundamental components, allowing independent but compatible innovation at each layer; it revealed itself to be pretty successful, however, many issues reside in the *Network Control/Management Plane*.

If in computer science usually many clean and elegant abstractions are used, network control instead is strongly related to hardware and verbose protocols, there are *no* principles or abstractions guiding the process, making in general difficult managing traffic flows.

SDN helps providing some abstraction through a centralized view of the network.

[1] Traffic patterns measure how traffic varies in time and space

13.2.1 Network Layer

- ◇ **Forwarding** move packets from router's input to appropriate router output
  Data Plane
- ◇ **Routing** determine route taken by packets from source to destination
  Control Plane

These are the two key tasks. In Fig. a comparison between the tradional and the SDN approaches.

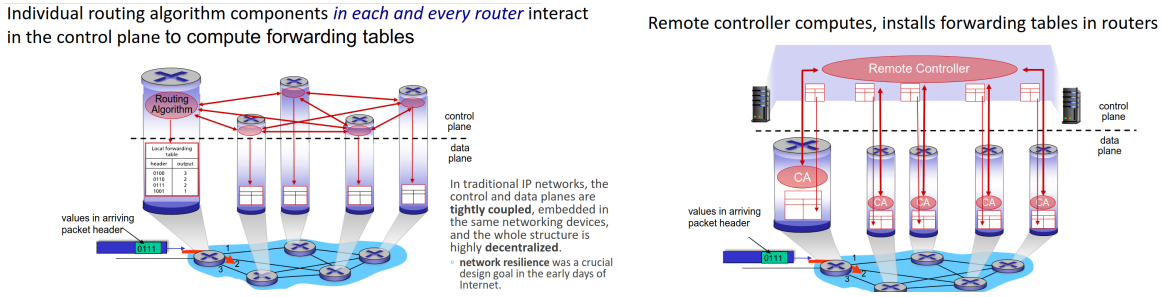


Figure 13.1: Traditional vs SDN

Data-plane switches are fast, simple, commodity switches implementing generalized data-plane forwarding in hardware, and generally provide API for table-based switch control, e.g. `OpenFlow`.

SDN controller (network OS) maintain network state information and interacts with network control applications "above" via northbound API, and with switches using "below" via southbound API. They are implemented as distributed system for performance, scalability, fault-tolerance, robustness.

Network-control apps are "brains" of control, they implement control functions using lower-level services. They are *unbundled*, meaning that they may be provided by a 3^{rd} party different from routing vendor or SDN controller

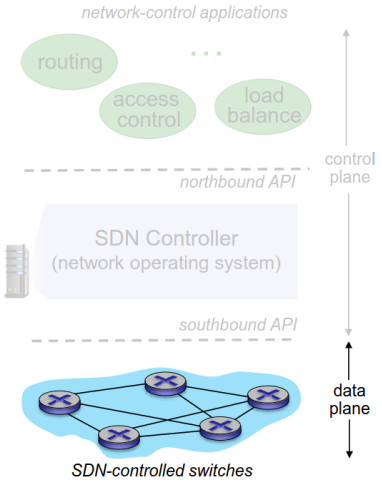


Figure 13.2: SDN Layer Architectures

13.3 Data Plane

Resource or infrastructure layer providing a *Forwarding Abstraction* made up of **forwarding devices** —without embedded software to make autonomous decisions— which perform the transport and processing of data according to decisions made by the SDN control plane.

Inside the routers there is an **OpenFlow flow**[1] **table**.

Some simple forwarding rules are defined:

- ◇ *Match*: Pattern values to be matched in packet header fields
- ◇ *Actions*: Actions to be performed for matched packet: `drop, forward, modify`
- ◇ *Priority*: Used to disambiguate overlapping matching patterns
- ◇ *Counters*: Typically used for `#bytes` and `#packets`

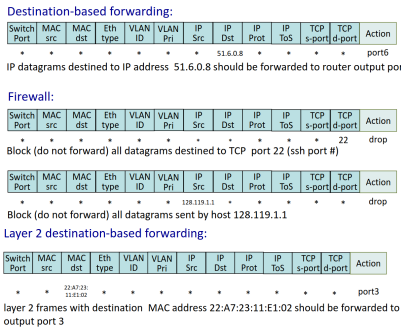


Figure 13.3: OpenFlow rules examples

Router
- *match:* longest destination IP prefix
- *action:* forward out a link

Switch
- *match:* destination MAC address
- *action:* forward or flood

Firewall
- *match*: IP addresses and TCP/UDP port numbers
- *action:* permit or deny

NAT
- *match:* IP address and port
- *action:* rewrite address and port

The abstraction of `match+action` unifies different kinds of devices

13.3.1 OpenFlow to control the Network Device

Data Plane Network devices support, alongside the abovementioned rule-based formwarding capabilities, interaction with the SDN controller for management of forwarding rules, through OpenFlow switch protocol.

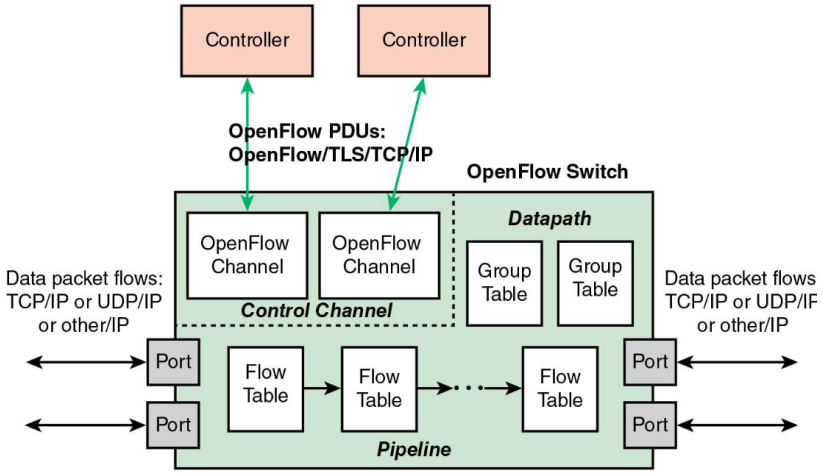


Figure 13.2: OpenFlow switch

Through OpenFlow, the controller can add, update, and delete flow entries in tables, both reactively (in response to packets) and proactively.

A switch is made up of one or multiple pipelined flow tables, possibly allowing for considerable flexibility. Instructions

[1] Defined by *link*, *network* and *transport* layer fields

performed on a packet may explicitly direct it to another flow table (`Goto` instruction) and so on, until it is formwarded to an output port.

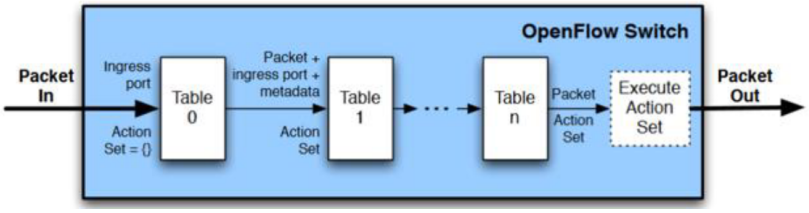


Figure 13.3: Multiple tables inside switch

13.4 Control Plane

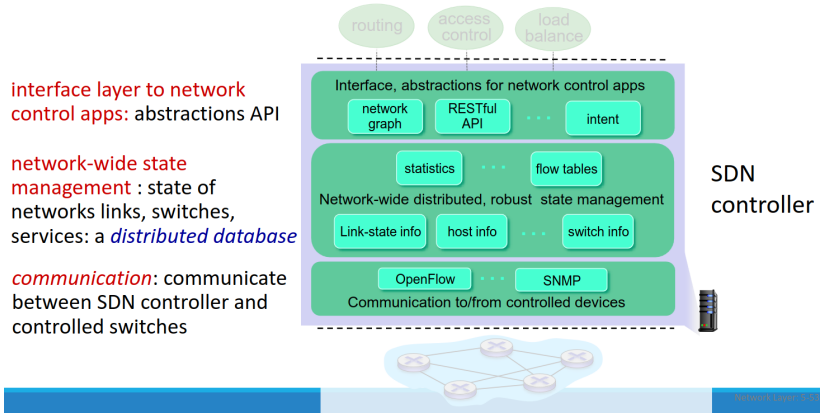


Figure 13.4: Inside an OpenFlow controller

The Openflow protocol operates between acontroller and switches using TCP (optional encryption). There are three classes of OpenFlow messages:

1. Controller-to-switch
   i. *Features*: controller queries switch features, switch replies
   ii. *Configure*: controller queries/sets switch configuration parameters
   iii. *Modify-state (FlowMod)*: add, delete, modify flow entries in the OpenFlow tables
   iv. *Packet-out*: controller can send this packet out of specific switch port
2. Asynchronous (switch to controller)
   i. *Packet-in*: transfer packet (and its control) to controller. See packet-out message from controller
   ii. *Flow-removed*: flow table entry deleted at switch
   iii. *Port status*: inform controller of a change on a port.
3. Symmetric (misc.)

ortunately, network operators don't "program" switches by creating/sending OpenFlow messages directly; they use instead use higher-level abstraction at controller.

13.5 Topology discovery and forwarding in the SDNs

13.5.1 Routing

While traditionally the routing function is *distributed* among the routers in a network, in an SDN controlled network, it makes sense to **centralize** the routing function within the SDN controller, allowing it develop a consistent view of the network state for calculating shortest paths and implementing application-aware routing policies.
Thus, data plane switches are relieved of the processing and storage burden associated with routing, leading to improved performance.

The centralized routing application performes two tasks

- **Link/Topology discovery**
  - The routing function needs to be aware of links between data plane switches

  - The topology discovery in OpenFlow domains currently is not standardized
- ◇ **Topology manager**
  - Maintains the topology information for the network
  - calculates routes in the network (the shortest path between two data plane nodes or between a data plane node and a hos

Every OF switch has initially set the IP address and TCP port of a controller to establish a connection as soon as the device is turned on; it also has preinstalled flow rules to route directly to the controller via a Packet-In message any message of the **Link Layer Discovery Protocol (LLDP)**, which is a neighbor discovery protocol of a single jump, i.e. it advertises its identity and capabilities and receives the same information from the adjacent switches.

LLDP is vendor neutral and works at layer 2, using *Ethernet* as "transport" protocol.

Switches send the LLDP messages (*"frames"*) —periodically at a given interval of time— to discover the underlying topology by request of the controller.

13.6 Google B4 WAN

There are two **backend backbones**, `B2` carrying internet facing traffic, and `B4` moving inter-datacenter traffic; the latter has grown a lot in the recent years.

B4 was needed for various reasons:

- ◇ Not on the public Internet
- ◇ Cost effective network for high volume traffic
- ◇ Bursty/bulk traffic[2]

Flow types traversing backend backbones are:

- ◇ User data copies
- ◇ Remote storage access
- ◇ Large-Scale data push synchronizing state across multiple data centers

[2] *"Burst"*: a group of packets with shorter interpacket gaps than packets arriving before or after that burst

Chapter 14

Network Function Virtualization

$$NFV \neq SDN$$

Even though the definitions were similar a few years ago, they are not the same thing. NFV is about virtualizing network functions, while SDN is about virtualizing the network itself; however, they work nicely together.

14.1 Introduction and context

In typical enterprise networks, there aren't only servers and switches, but also a lot of network functions, such as firewalls, load balancers, intrusion detection systems, etc. These functions are usually implemented in dedicated hardware, which is expensive and hard to manage; besides these components have a strict chaining/ordering, that must be reflected in the network topology.
NFV is about moving these functions to software, so they can be run on commodity hardware.

NFV is one (the?) way to address these challenges by leveraging virtualization technologies. The main idea is **decoupling** physical network equipment from the functions that run on them, with network functions provided as **plain software**.

A Network service can be decomposed into a set of Virtual Network Functions (VNFs). VNFs may then be relocated and instantiated at different network locations without requiring to buy and install new hardware.

One of the key points is to **reduce costs**. Two types of costs are involved:

- **CAPEX** (Capital Expenditure): the cost of buying the hardware. This is reduced by buying general purpose hardware instead of specialized one.
- **OPEX** (Operational Expenditure): the cost of managing the hardware. Acquired by reducing the number of devices to manage.

Performance is also a concern, as the software implementation may be slower than the hardware one. However, the performance of commodity hardware is increasing, and the software can be optimized to run on it.

Benefits

- Improved resource usage efficiency
  - Due to flexible allocation of different NFs on the HW pool
- Elasticity
  - Capacity dedicated to each NF can be dynamically modified (scaling) according to actual load on the network
- Topology reconfiguration
  - Network topology can be dynamically reconfigured to optimize performance

⇓

Higher service availability and resiliency

14.2 Network Service

An end-to-end network service can be defined as a forwarding graph of network functions and end points/terminals; So it can be viewed architecturally as a forwarding graph of Network Functions (NFs) interconnected by the supporting network infrastructure. Nodes running VNFs are called **Points of Presence** (*PoPs*) and are interconnected by logical links, backed by the underlying the network infrastructure physical links.

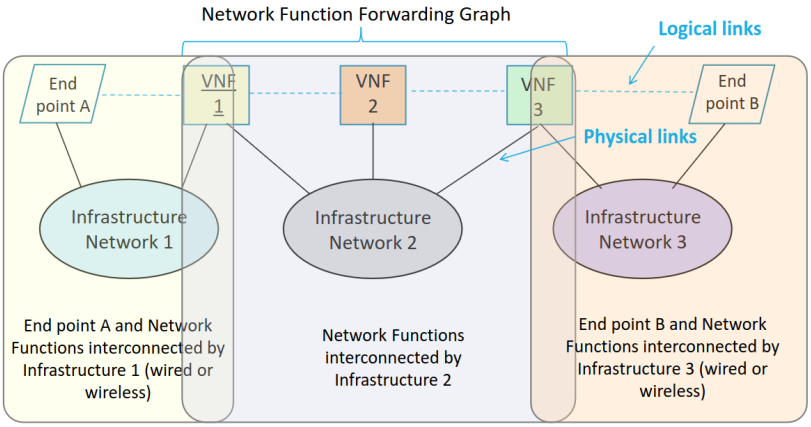


Figure 14.1: VNF Network Service

14.3 NFV Architectural Framework

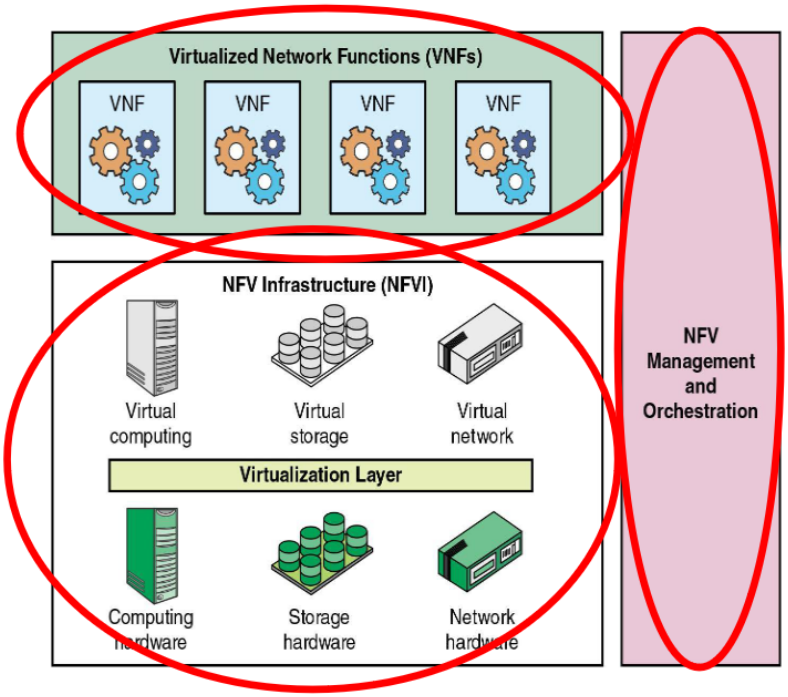


Figure 14.2: NFV Architecture

- **NFV** infrastructure (NFVI)
  Comprises the hardware and software resources that create the environment in which VNFs are deployed.
- **VNF**
  The collection of VNF implemented in software to run on virtual computing, storage, and networking resources.
  NFV management and
- **Orchestration** (NFV-MANO)
  Framework for the management and orchestration of all resources in the NFV environment.

The NFV infrastructure is the typical hardware plus virtual machines managed by a hypervisor. The three domains depicted in Fig. 14.3 are:

- **Compute Domain**

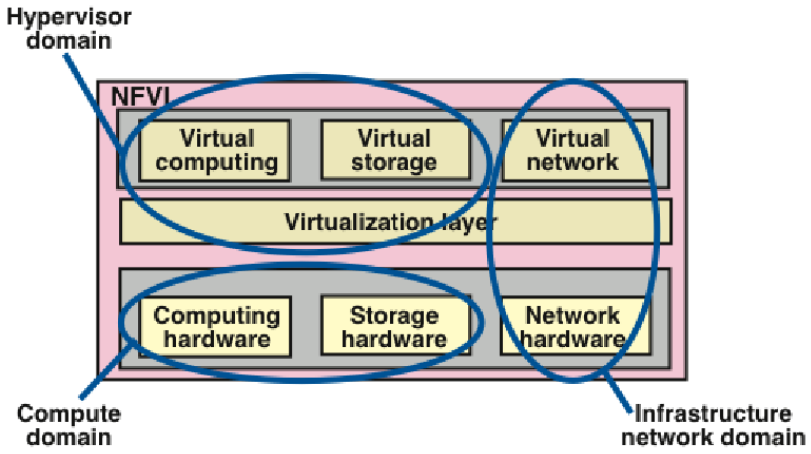


Figure 14.3: NFV Infrastructure

Provides commercial off-the- shelf (COTS) high-volume servers and storage.

- ◇ **Hypervisor Domain**
  Mediates the resources of the compute domain to the VMs (today also Containers) of the software appliances, providing an abstraction of the hardware.
- ◇ **Infrastructure Network Domain**
  Comprises all the generic high volume switches interconnected into a network that can be configured to supply network services

14.3.1 vSwitch - Case study

Virtual Switches provide connectivity between *Virtual Interfaces* (VIFs) and the *Physical Interfaces* (PIFs), and also handle traffic between VIFs located on the same host.
A virtual switch is a software program that emulates a switch (Layer 2 device).

14.3.2 VNF Placement problem

Placing a chain of VNFs is hard, and consists of maximing a utility function, while respecting infrastructure constraints and QoS requirements. The utility function is usually a combination of:

- ◇ Minimization of the overall delay;
- ◇ Minimization of deployments costs,
- ◇ Maximization of remaining bandwidth, etc.

14.4 NFV MANO

- ◇ Oversees the provisioning of the VNFs, and related operations
  - The configuration of the VNFs, and
  - The configuration of the infrastructure the VNFs run on
- ◇ Includes orchestration and lifetime management of physical and/or software resources supporting the infrastructure virtualization and the lifecycle management of VNFs
- ◇ Includes databases used to store information and data models defining deployment and lifecycle properties of functions, services and resources
- ◇ Defines interfaces used for communications between components of the MANO, as well as coordination with traditional network management, such as OSS/BS

VIM - Virtualized Infrastructure Management

VIM comprises the functions that are used to control and manage the interaction of a VNF with computing, storage, and network resources under its authority, as well as their virtualization.

A VIM is responsible for controlling and managing the NFVI compute, storage, and network resources.

A —whole— MANO may orchestrate multiple VIMs.

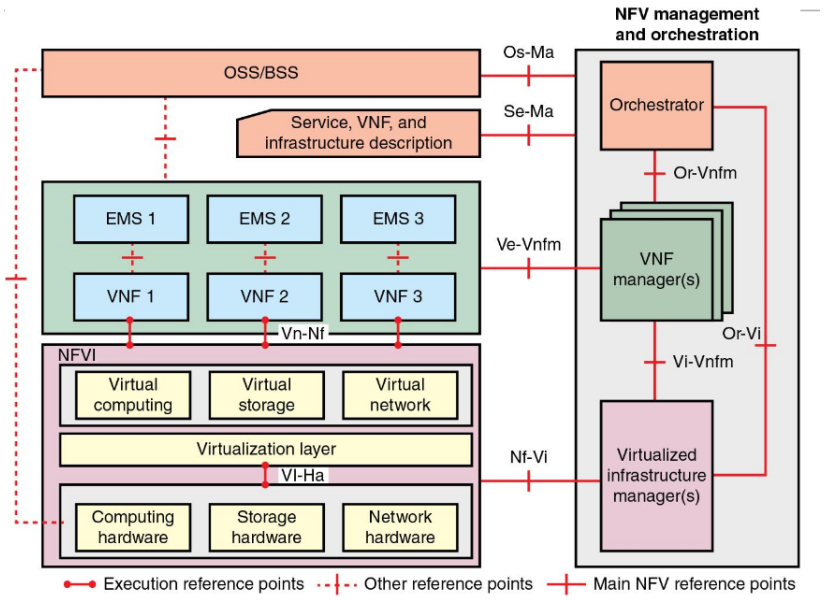


Figure 14.4: NFV Reference - Architectural Framework
Here VNF1,2,3 are VNF instances, EMS1,2,3 are Element Management Systems, and OSS/BSS is the traditional network management,

Virtual Network Function Manager - VNFM

Oversees lifecycle management (e.g. instantiation, update, query, scaling, termination) of VNF instances.

NFV Orchestrator - NFVO

Responsible for installing and configuring new network services (NS) and virtual network function (VNF) packages, NS lifecycle management and global resource management.
Acts also as network service orchestrator, managing/coordinating the creation of an end-to-end service that involves VNFs.

Chapter 15

NFV/SDN Emulation tools

15.1 Mininet

Mininet is a network emulator, it runs a collection of end-hosts, switches, routers, and links on a single Linux kernel. It supports OpenFlow for SDN network emulation.

A Mininet host behaves just like a real machine, e.g can send packets with a given link speed and delay, or may ping or iperf another hosts

`sudo mn` initializes a default topology and launches the CLI
Overriding `build()` of the `mininet.topo.Topo` class to create a custom topology.

- ◇ Topo: the base class for Mininet topologies
  - `addSwitch()`: adds a switch to a Topo and returns the switch name
  - `addHost()`: adds a host to a Topo and returns the host name
  - `addLink()`: adds a bidirectional link to Topo (and returns a link key, but this is not important). Links in Mininet are bidirectional unless noted otherwise.
- ◇ Mininet: main class to create and manage a network
  - `start()`: starts your network
  - `pingAll()`: tests connectivity by trying to have all nodes ping each other
  - `stop()`: stops your network
  - `net.hosts`: all the hosts in a network

  Documentation: http://mininet.org/api/annotated.html

`veth` *pairs* is a full-duplex link between two interfaces in same o separate namespace: packets transmitted from one interface are directly forwarded to the other interface in the pair, so each inteface behaves like an ethernet interface.
`ip link show type veth` or `Mininet> links`

15.2 Signals

Definition 15.1 (Signal) *A signal is a real function of time*

- ◇ Time-continuous and amplitude-continuous
- ◇ Time-continuous and quantized
- ◇ Time-discrete and amplitude-continuous
- ◇ Time-discrete and quantized

15.3 Signal types

Definition 15.2 (Continuous-time signal) *A signal is continuous-time (also known as analog signals) if it is defined for all real numbers. The domain thus $\mathcal{D}$ is the set of real numbers $\mathbb{R}$*
If the codomain $\mathcal{C}$ is the set of real numbers $\mathbb{R}$: continuous amplitude signals.
If the codomain $\mathcal{C}$ is a discrete set (e.g. $\mathbb{Z}$): discrete amplitude signals AKA quantized signals

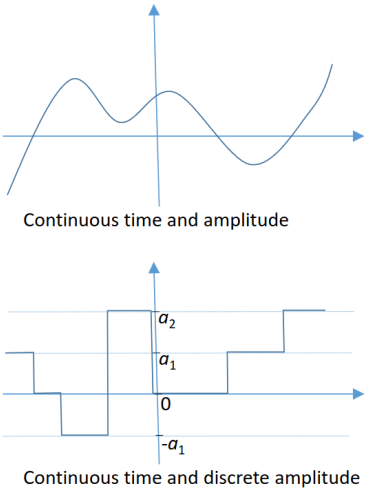


Figure 15.1: Continous signals

Definition 15.3 (Discrete-time signal) *A signal is discrete-time if it is defined only at discrete times.*
The domain $\mathcal{D}$ is the set of integers $\mathbb{Z}_T$, where $\mathbb{Z}_T = \{nT, \forall n \in \mathbb{Z} \text{ and } T \in \mathbb{R}\}$. They are also called discrete signals.
For example, $\mathbb{Z}_2 = \ldots, -4, -2, 0, 2, 4, \ldots$.

A discrete signal is called digital when the codomain is a finite set of symbols. A digital signal is also called a symbolic sequence. A text is an example of a symbolic signal.

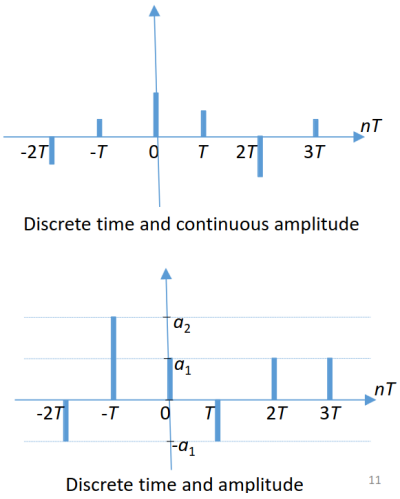


Figure 15.2: Discrete signals

Definition 15.4 (Digital signals) *Consider the alphabet of the 26 symbols* `A=a,b,c,...,x,y,z`*. A symbolic signal is any sequence of such symbols, e.g.:*
`ababfxje...`

Consider now the alphabet of two symbols `B=0,1`*:*

- ◇ *We can represent any symbol in A with a sequence in B*
- ◇ *For example:* `a` $\cong$ `00000`, `b` $\cong$ `00001`*, etc.*
- ◇ *Hence, the digital signal* `ababfxje` *would become:*
  `0000000001000000000100101101110100100100`

Assume a source that samples an analog signal with $f_c$samples/second. Each sample is represented (quantized) with Mbit/sample.
The source has a ***throughput*** *of $f_c \cdot Mbit/second$*

15.4 Transmission of information

Signals may have different nature, depending on the channel, e.g. electromagnetic, sound, optical waves etc. At the source a transducer converts a message into a signal, which propagates through a medium, while at the destination another transducer converts a signal into a message.

For example, the antenna is a transducer for electromagnetic signals

15.4.1 Disturbed

Distortion refers to any change in a signal that alters the basic waveform or the relationship between various frequency components, e.g.:

- ◇ *Attenuation*: a signal composed of contributions in several frequencies is transmitted on one channel, the attenuation can be different according to the frequency
- ◇ *Multipath propagation*: radio signal reflects off objects or ground, arriving at destination at slightly different times

Noise refers instead to an unwanted, random and unpredictable signal that interferes with the communication or measurement of another signal

- ◇ *External noise*: e.g. interference from nearby channels, faulty equipment, etc.
- ◇ *Internal noise*: thermal motion of electrons in conductors

Signal to noise ratio (SNR): is the amount of changes suffered by the messages transmitted through a channel. It is a relative measure of the strength of the received signal (i.e., the information being transmitted) and the background noise in the environment

$$SNR\ in\ \texttt{dB} = 20 * log(signal/noise)$$

15.5 Digital transmission - sampling & quantization

Definition 15.5 (Sampling) *Sampling is the process of converting a continuous signal into a discrete signal. The sampling rate is the number of samples taken per second. The sampling theorem states that a signal can be perfectly reconstructed if it is sampled at a rate higher than twice the highest frequency component of the signal.*

The digital source may be the result of sampling and quantizing an analog source such as speech.

- ◇ At the *transmitter*: sample the analog signal at discrete intervals ("discrete intervals" imply a quantization), by means of an analog to digital converter (ADC) the analog signal is now represented as a sequence of symbols.
- ◇ At *receiver*: the symbols must be converted again into an analog signal by means of a digital to analog converter (DAC)

However, sampling and quantization introduce a further distortion of the analog signal (*quantization noise*), due to:

- ◇ frequency of sampling
- ◇ resolution of the digital symbols, i.e. how many different symbols used to represent a single analog value
  High resolution and high sampling rate *Rightarrow* small quantization error, but also a larger number of symbols to transmit

15.6 Fourier series

Definition 15.6 (Periodic continuous signals) *A continuous signal* $s(t) : \mathbb{R} \to \mathbb{R}$ *is periodic with period* T *if* $s(t) = s(t+T) \quad \forall t \in \mathbb{R}$.
Example: $\sin(nt)$ *and* $\cos(nt)$ *are periodic with period* $T = 2\pi n$ *for all* $n \in \mathbb{Z}$ *Non-periodic signals are called aperiodic.*

Periodic signals can be studied in the period $[0, T]$ *since their behavior remains the same in all its domain of existence. For a periodic signal, it also holds that:* $s(t+T) = s(t+2T) = s(t+nT) \quad \forall t \in \mathbb{R}, n \in \mathbb{Z}$

Definition 15.7 (Periodic extension) *Consider an aperiodic signal* s *with support limited to the interval [a, b) i.e.* $s(t) = 0\ \forall t \notin [a, b)$

The periodic extension s^* *of* s *is defined as:* $s^*(t) = \sum_{n=-\infty}^{\infty} s(t - nT)$ *where* $T = b - a$

In other words, the periodic extension of a signal is the sum of all the copies of the signal shifted by multiples of the period T.

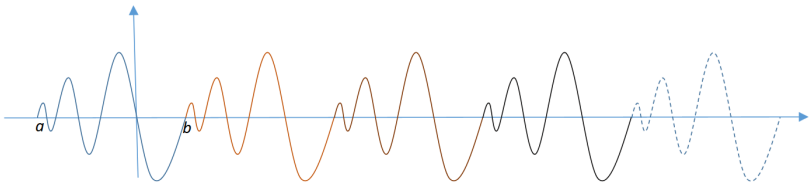
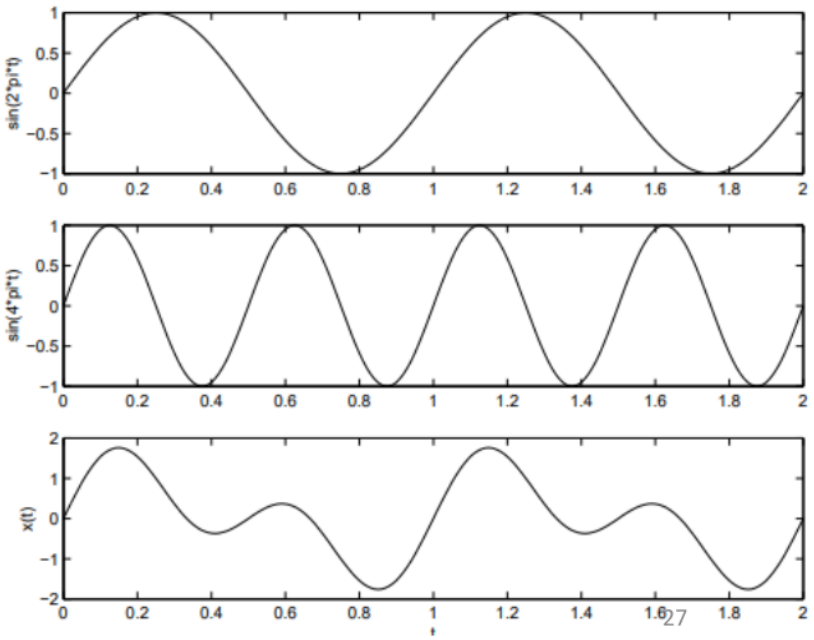


Figure 15.1: Repeating an aperiodic signal results in a periodic one

15.6.1 Combining and decomposing signals

It is possible to combine signals, e.g. by adding them together, to create new signals.



$$x_t = \sin(2\pi t) + \sin(4\pi t)$$

The Fourier series allows to decompose a periodic signal into a sum of sines and cosines, inverting the combining process. More formally the Fourier series decomposes a function (signal) as the sum of an infinite number of continuous functions, oscillating at different frequencies.
This set of continuous functions defines the base of decomposition. The Fourier series has a base represented by a set of functions

$$\phi_n(t), n \in \mathbb{Z}$$

This set of functions must be orthogonal (as in the case of decomposition of vectors in a vector space)

Definition 15.8 (Fourier series) *Given a continuous signal $s(t) : \mathbb{R} \to \mathbb{R}$, periodic in the interval $[-\pi, \pi]$ its Fourier series is defined as:*

$$s(t) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty} (a_n \cos nt + b_n \sin nt)$$

Constant component — Amplitude of the harmonics — Harmonics

where

$$a_0 = \frac{1}{\pi}\int_{-\pi}^{\pi} s(t)\,dt;\quad a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} s(t)\cos nt\,dt;\quad b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} s(t)\sin nt\,dt$$

It is rather complicate to assess the conditions under which an arbitrary $s(t)$ can be developed in a Fourier series, in particular necessary conditions are *not* known; however, there are sufficient conditions:

Theorem 15.1 (Dirichlet) *1. $s(t)$ is periodic*
2. $s(t)$ is piecewise continuous
$\Rightarrow$ the Fourier series of $s(t)$ exists and converges in $\mathbb{R}$

A *piecewise continuous* function is made up of a finite number of continuous pieces on each finite subinterval $[0, T]$. Also the limit of $f(t)$ as t tends to each point of discontinuity is finite.