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 (?)

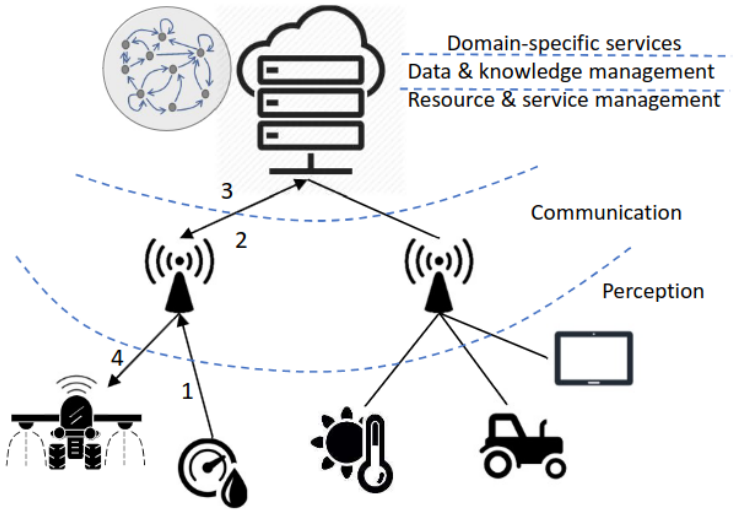


Figure 1.1: Communication outline in IoT

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

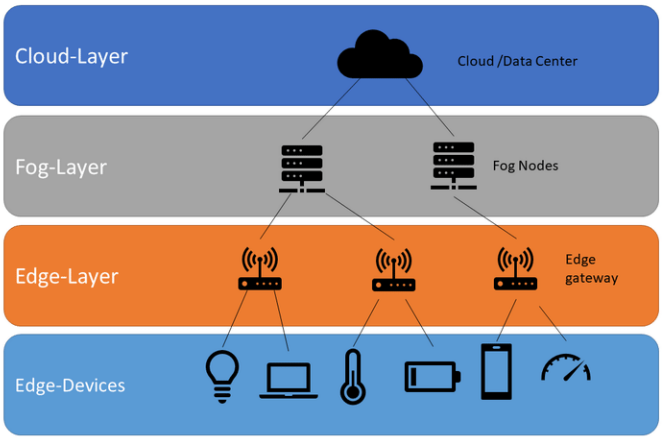


Figure 1.2: Layers scheme

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 reduction 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?

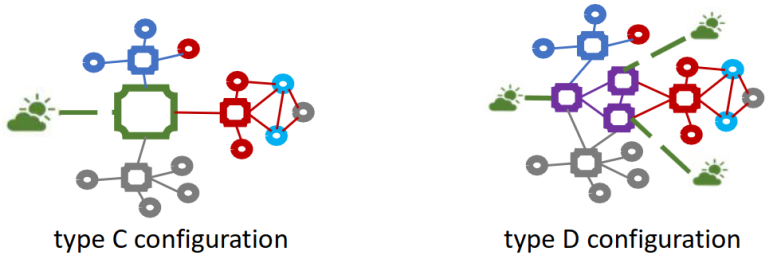


Figure 1.3: Gateway configs

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

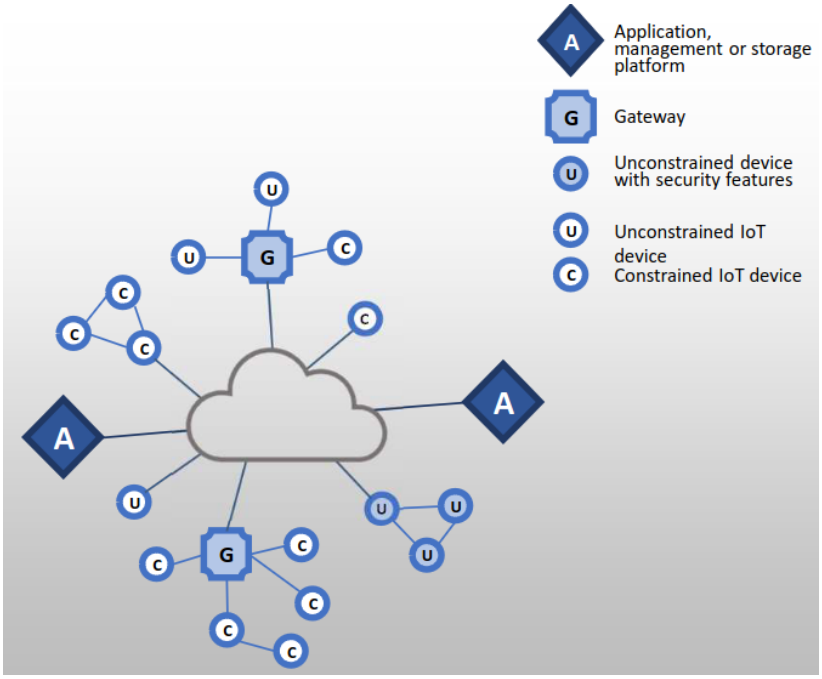


Figure 1.4: Security elements of interest

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.

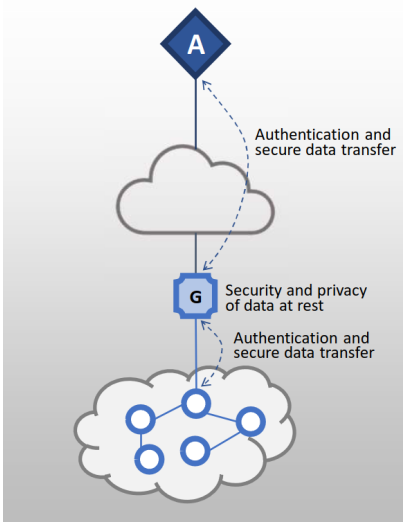


Figure 1.5: Gateways security functions

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*

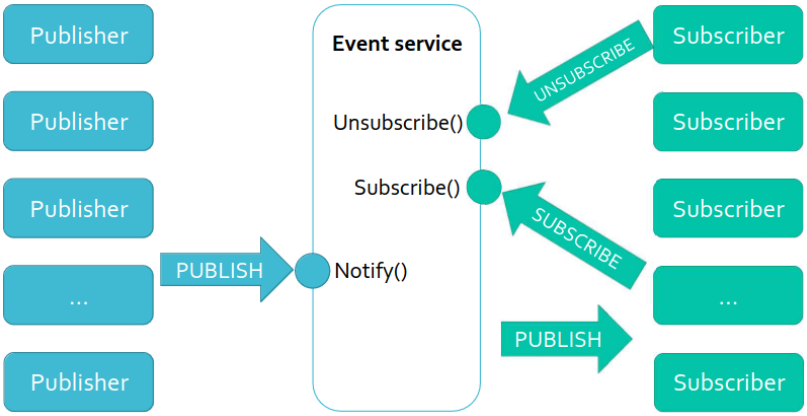


Figure 2.1: Broker management of events

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 with QoS 1,2 not 0!).
- ◇ 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

Topics are strings that are organized into a hierarchy of *"topic levels"*, separated by a *"/"* character.
Example: `home/firstfloor/bedroom/presence`

Wildcard may be used in the topic levels:

- ◇ `home/firstfloor/+/presence`
  Selects all presence sensors in all rooms of the first floor
- ◇ `home/firstfloor/#`
  Selects all sensors in the first floor
- ◇ Topics that begins with a "$" are reserved for internal statistics of MQTT; There is no standardization on such topics, and cannot be published by clients

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:

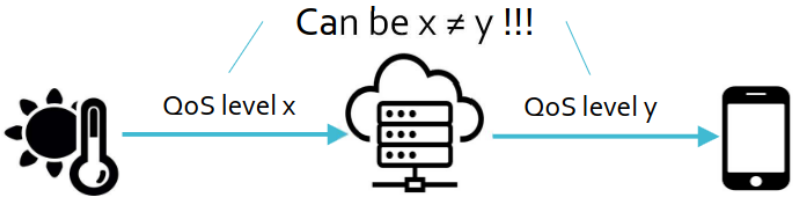
`level 0` **At most once**
- ◇ It is a "best effort" delivery and messages are *not* acknowledged by the receiver

`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

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)

Structure of an MQTT control packet:

Fixed header, present in all MQTT control packets
Variable header, present in some MQTT Control Packets
Payload, present in some MQTT Control Packets

Fixed header (2 bytes):

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT control packet type				Flags specific to each MQTT control packet type			
byte 2...	extra length	Remaining length						

Remaining length is the length of the variable header and payload.

- 1 byte encodes length of up to 127 bytes. The most significant bit specifies that there is another field of length (for packet longer than 127 bytes)

Control packet type:

Name	value	direction of flow
reserved	0	forbidden
CONNECT	1	client to server
CONNACK	2	server to client
PUBLISH	3	client to server or server to client
PUBACK	4	client to server or server to client
PUBREC	5	client to server or server to client
PUBREL	6	client to server or server to client
PUBCOMP	7	client to server or server to client
SUBSCRIBE	8	client to server
SUBACK	9	server to client
UNSUBSCRIBE	10	client to server
UNSUBACK	11	server to client
PINGREQ	12	client to server
PINGRESP	13	server to client
DISCONNECT	14	client to server
reserved	15	forbidden

Payload:

- Contains additional information
- E.g. the payload of CONNECT includes:
  - client identifier (mandatory)
  - will topic (optional)
  - will message (optional)
  - Username (optional)
  - Password (optional)

Control packet	payload
CONNECT	required
CONNACK	none
PUBLISH	optional
PUBACK	none
PUBREC	none
PUBREL	none
PUBCOMP	none
SUBSCRIBE	reserved
SUBACK	reserved
UNSUBSCRIBE	reserved
UNSUBACK	none
PINGREQ	none
PINGRESP	none
DISCONNECT	none

Figure 2.2: MQTT Packet headers

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.

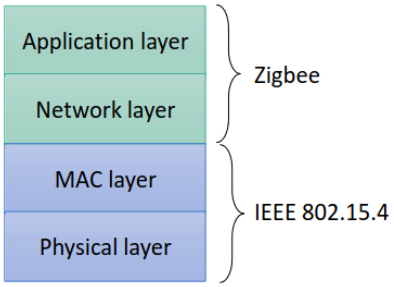


Figure 3.1: ZigBee layers

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

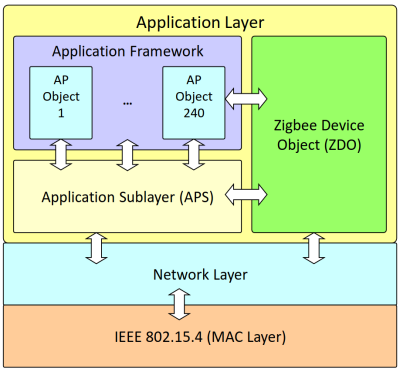


Figure 3.2: Zigbee architecture layers

3.2 Primitives

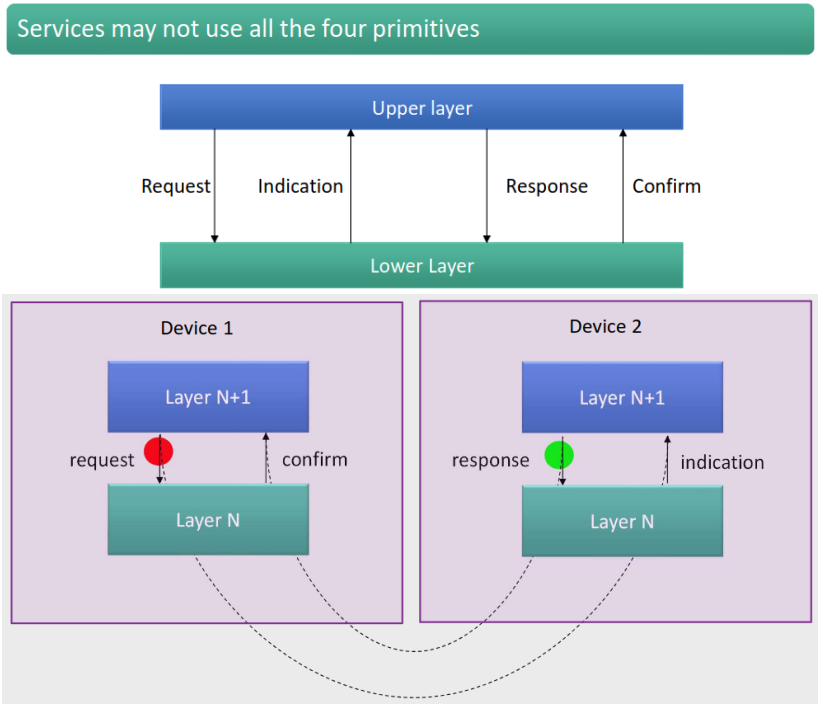


Figure 3.1: Mapping between zigbee primitives

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

3.3.1 Network formation and joining

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

- ◇ Form a new network ⟶ *ZigBee Coordinator*

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

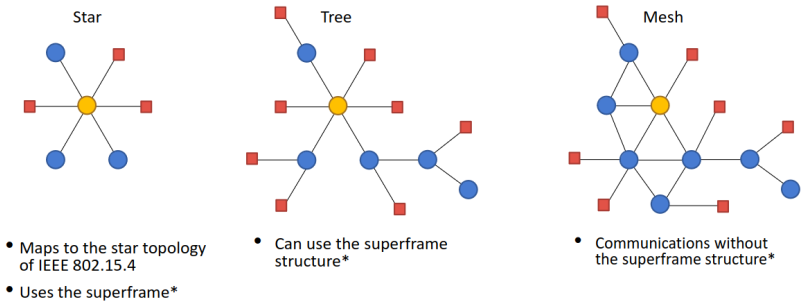


Figure 3.2: ZigBee Network topologies outline

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

◇ Join an existing network ⟶ *ZigBee router* or *end-device*

The role of the device is chosen at compile-time

3.3.1.1 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.

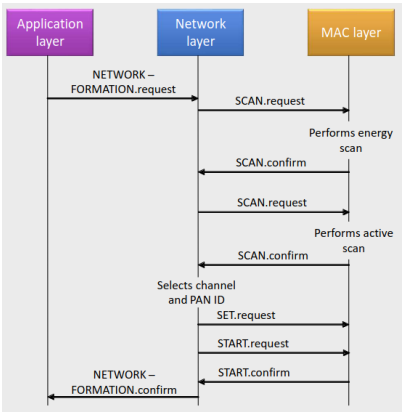


Figure 3.3: Network formation messages

3.3.1.2 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.

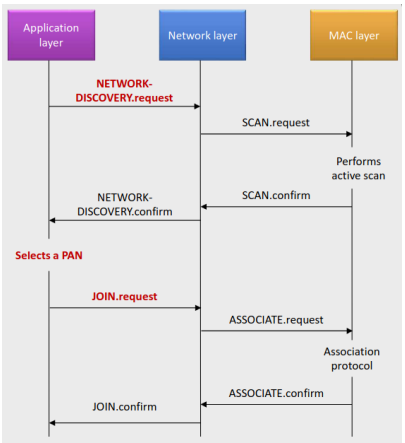


Figure 3.4: Network joining messages

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

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

3.4.1.1 Concepts and related IDs

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 an 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

3.4.1.2 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 $\langle$*destination endpoint*, *destination network address*$\rangle$ (*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 $\langle$*source address, source endpoint, cluster identifier, destination address, destination endpoint*$\rangle$
- ◇ `UNBIND.request`
  deletes an entry from the local binding table.

The 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:

- ◇ matches *source address* $\langle$*network addr, endpoint addr*$\rangle$ and the *cluster identifier* into the pair: $\langle$*destination endpoint, destin*

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

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

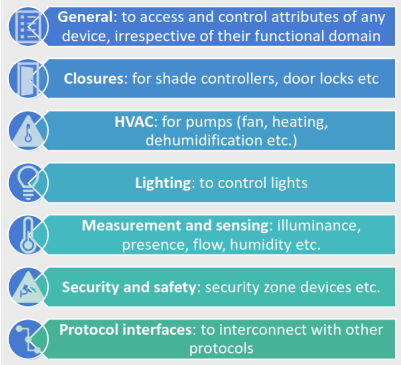


Figure 3.5: Functional domains

Commands are in a format specified by the ZCL, and are used to manipulate the attributes or for dynamic attribute reporting of a cluster; more specifically:

- ◇ Read/write
- ◇ Configure reporting
- ◇ Discover attributes

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

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

ATTRIBUTES – BASIC DEVICE INFO. CLUSTER

Identifier	Name	Data Type	Range	Access	Default	Mandatory / Optional
0x0000	ZCLVersion	uint8	0x00 – 0xff	Read Only	0x02	M
0x0001	ApplicationVersion	uint8	0x00 – 0xff	Read Only	0x00	O
0x0002	StackVersion	uint8	0x00 – 0xff	Read Only	0x00	O
0x0003	HWVersion	uint8	0x00 – 0xff	Read Only	0x00	O
0x0004	ManufacturerName	string	0 – 32 bytes	Read Only	Empty string	O
0x0005	ModelIdentifier	string	0 – 32 bytes	Read Only	Empty string	O
0x0006	DateCode	string	0 – 16 bytes	Read Only	Empty string	O
0x0007	PowerSource	enum8	0x00 – 0xff	Read Only	0x00	M

Figure 3.5: ZigBee basic device cluster attributes

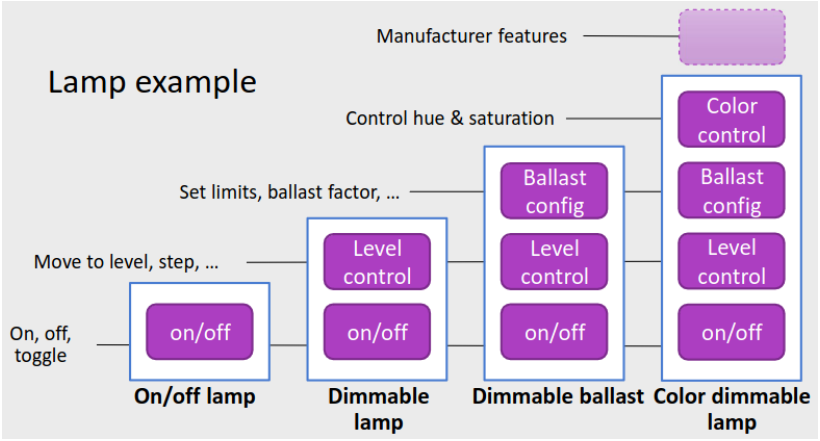


Figure 3.6: ZCL Hierarchical approach, and combining clusters to exploit existing functionalities

3.7.1 Indoor Localization

- ◇ Remote positioning
  A mobile device sends radio beacons to a network of anchors which estimate its positioning.
- ◇ Self positioning
  A network of fixed anchors sends radio beacons to a mobile device which estimates its own position.

3.7.2 Exploiting ZigBee and ZCL for real applications

Manufacturers of ZigBee-compliant HW platform usually provide the ZigBee stack (with also the ZDO) and the ZigBee Cluster libraries. Starting from this, the manufacturer of a ZigBee solution builds its devices, possibly adding the necessary hardware (e.g. transducers), and implementing the rest of the SW and configuration

- ◇ For each device decide its role in the network (coordinator, router,end-device) and configure it accordingly (either by SW or by tools)
- ◇ For each device: implement the APO.
  - An APO implements the "businness logic" of the device
  - If it is compliant with the ZCL it will also be interoperable with devices of other manufacturers

3.8 Security

Security services provided for ZigBee include methods for:

- ◇ key establishment,
- ◇ key transport,
- ◇ frame protection,
- ◇ device management.

These services form the building blocks for implementing security policies within a ZigBee device.

3.8.1 Keys

The problem of security is that some important assumptions, aside from that security protocols are correctly implemented and that keys are securely stored, have to be made regarding the **keying material**; There must be trust in its secure initialization and installation, as well as in its secure processing and storage of keying material.
However, due to the low cost of ZigBee devices, one cannot generally assume the availability of *tamper resistant hardware*. Hence, physical access to a device may yield access to secret keying material and other privileged information, as well as access to the security software and hardware.

Another problem is that APOs in the same device are not logically separated and lower layers are entirely accessible to the application layer; hence different APO in the same device must trust each other.

3.8.1.1 Key Establishment

- ◇ 128-bit **Network** key (shared by all devices):
  - acquired either by key-transport or pre-installation
  - two types of network keys: standard, and high-security
- ◇ 128-bit **Link** keys (shared by two devices):
  - acquired either by key-transport, key-establishment, or pre-installation
  - If key establishment is used, it is based on a pre-existing master key
  - two types of link keys: global or unique
  - A default global key serves to connect to the network trust center
- ◇ 128-bit **Master** key:
  - used only to implement key establishment protocols
  - acquired either by key-transport or pre-installation plus other keys to implement the secure transport of key material
  - derived from the link key with one-way functions

3.8.2 Trust Center

The Trust Center is the device trusted by devices within a network, there shall be exactly one, and exactly one has to be recognized by all members of the network. It distributes keys (key transport) for the purpose of network and end-to-end application configuration management.

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

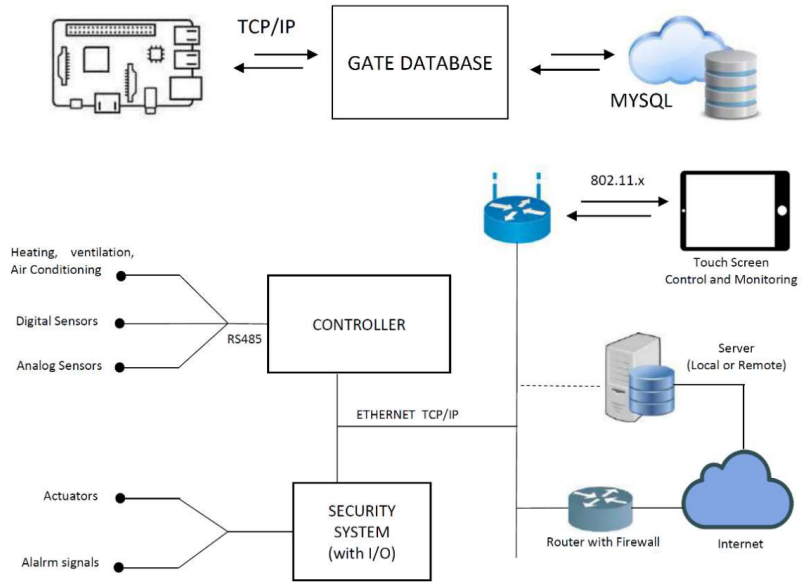


Figure 4.1: Greenhouse network architecture

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.

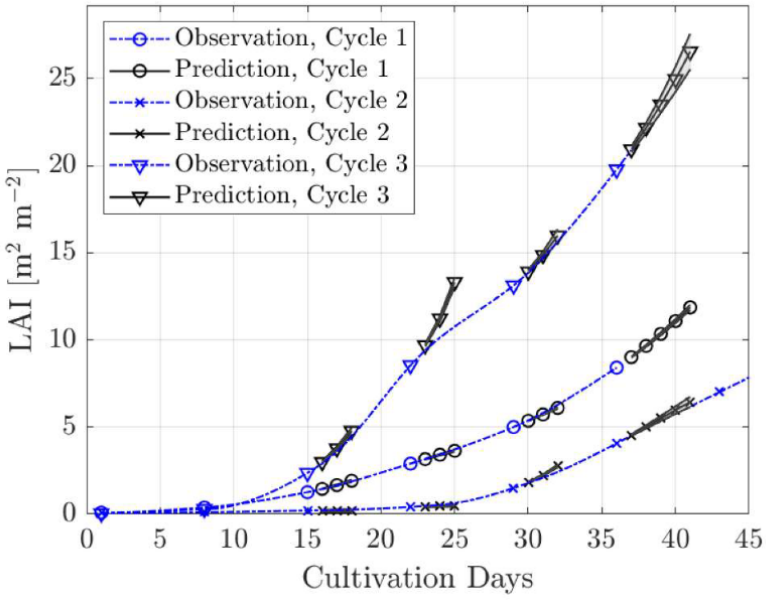


Figure 4.2: Crops growth AI prediction against acquired data

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*

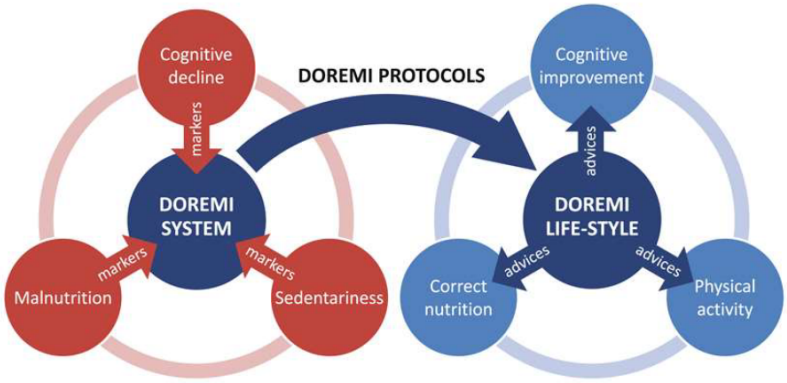


Figure 4.3: DOREMI project

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

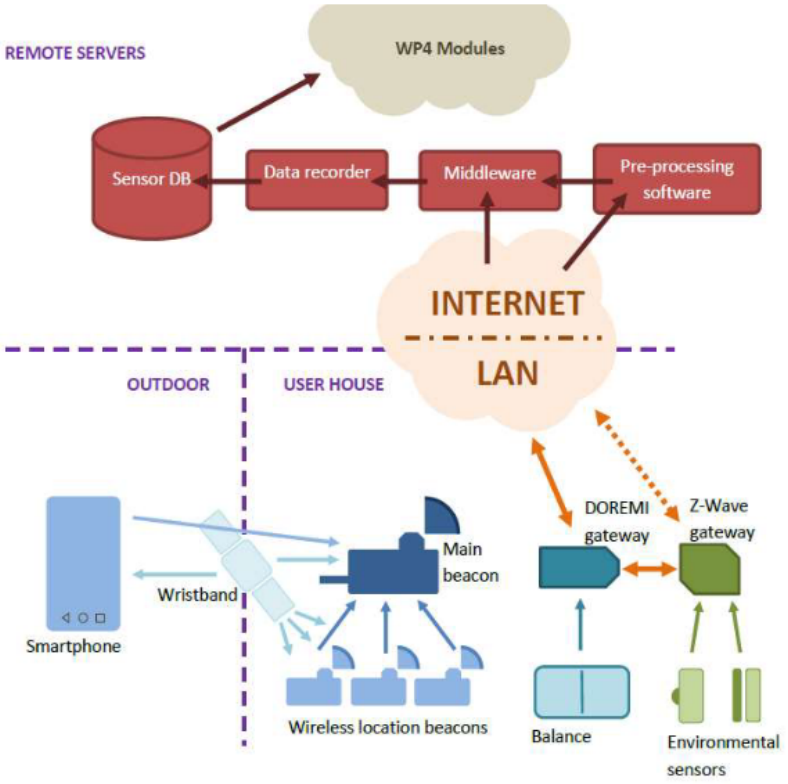


Figure 4.4: Wireless sensor network and Cloud architecture

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.

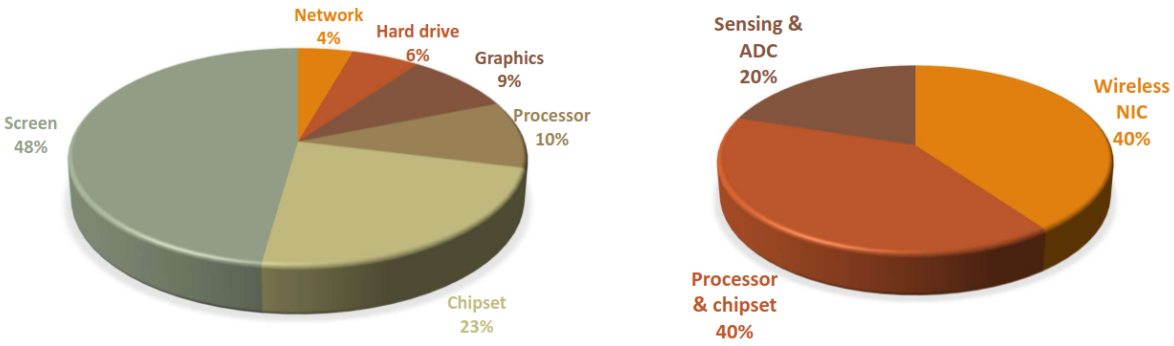


Figure 5.1: Laptop and IoT Sensor battery consumption percentages

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.

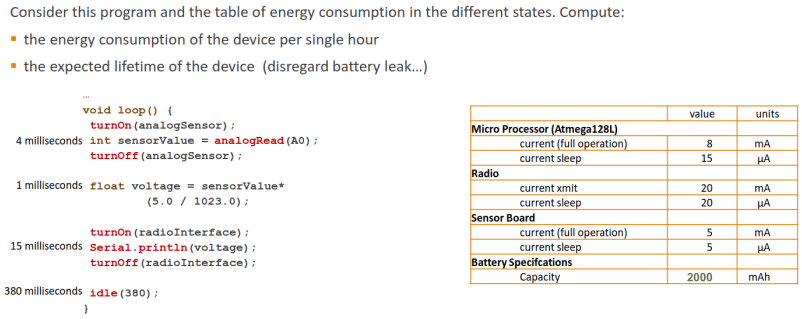
The duty cycle of a system (or a component / device) is defined as the fraction of one period in which the system is active.

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.


Consider this program and the table of energy consumption in the different states. Compute:

- the energy consumption of the device per single hour
- the expected lifetime of the device (disregard battery leak...)

```
...
void loop() {
  turnOn(analogSensor);
4 milliseconds int sensorValue = analogRead(A0);
  turnOff(analogSensor);

1 milliseconds float voltage = sensorValue*
          (5.0 / 1023.0);

  turnOn(radioInterface);
15 milliseconds Serial.println(voltage);
  turnOff(radioInterface);

380 milliseconds idle(380);
}
```

	value	units
Micro Processor (Atmega128L)		
current (full operation)	8	mA
current sleep	15	µA
Radio		
current xmit	20	mA
current sleep	20	µA
Sensor Board		
current (full operation)	5	mA
current sleep	5	µA
Battery Specifcations		
Capacity	2000	mAh

Figure 5.2: Exercise on energy consumption and duty cycling

For what concerns the exercise, these are the calculations

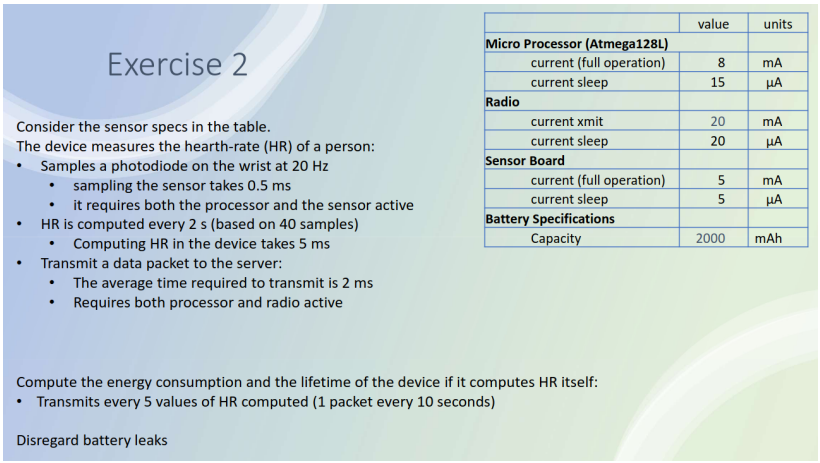
CPU	5	0.05 * 8	0.95 * 0.015	0.41425
Radio	3,75	0.0375 * 20	0.9625 * 0.020	0.9425
Sensor	1	0.01 * 5	0.99 * 0.005	0.05495

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

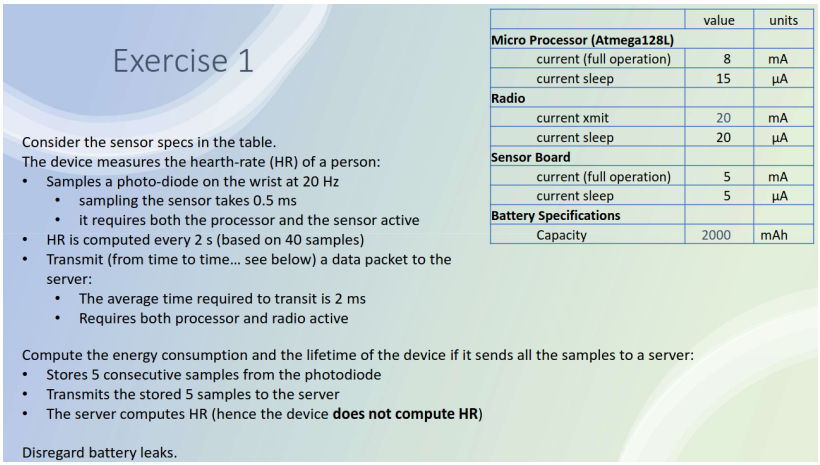


Duty cycle sensor $= 0.01$
Duty cycle radio $= 0.008$
Duty cycle CPU $= (0.5 * 5 + 2)/250ms = 0.018 = 0.01 + 0.008$

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.

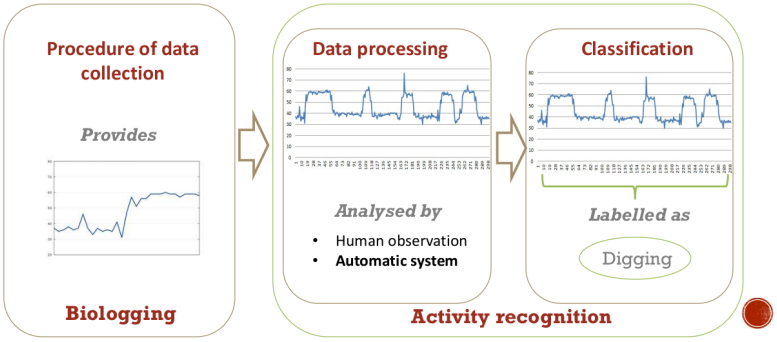


Figure 6.1: Biologging data lifecycle.
Biologging provides raw data which is later processed and classificated.

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.

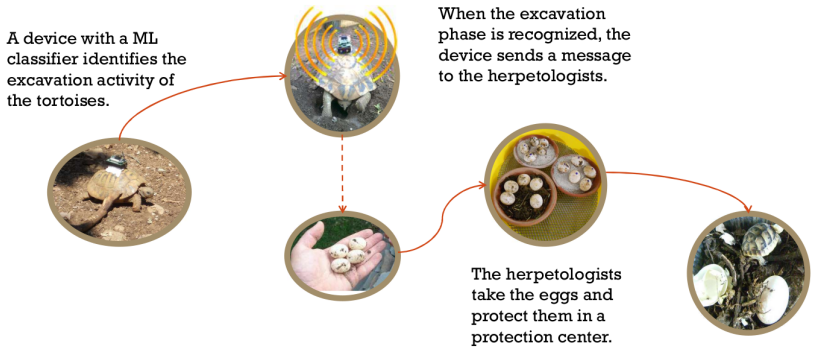


Figure 6.2: Tortoise@ architecture

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.

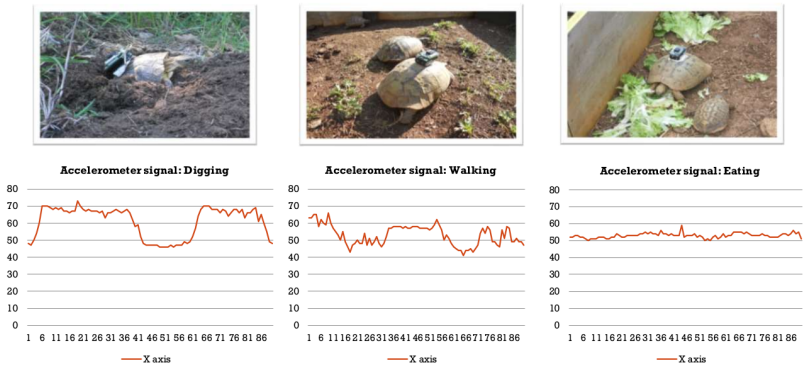


Figure 6.3: Tortoise activities and accelerometer data

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)

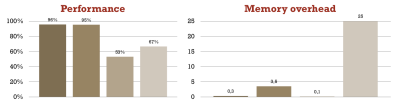


Figure 6.4: Models comparison for *Asynchronous tasks* The hardware used was an Arduino equivalent, with a 16 MHz clock and **4 KB** of RAM, which made some of the displayed model completely unfeasible.

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.

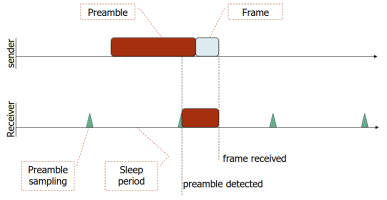


Figure 7.1: BMAC TX-RX schema

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.

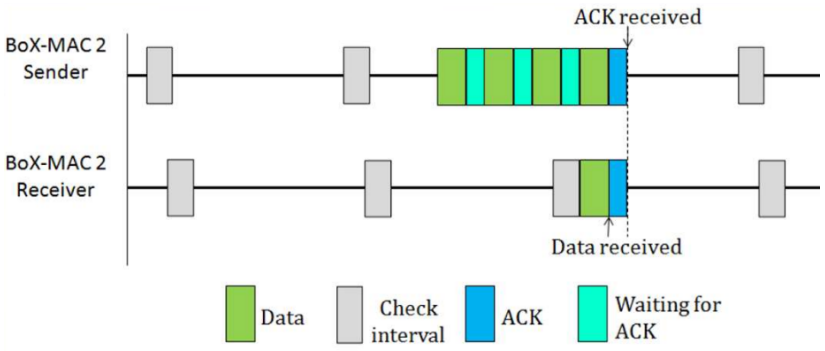


Figure 7.1: BoX-MAC TX-RX schema

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

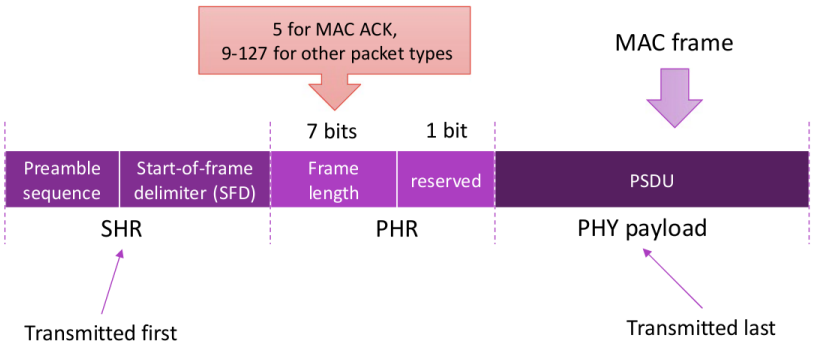


Figure 8.1: Physical frame structure

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

MAC Layer

MAC layer provides various services:

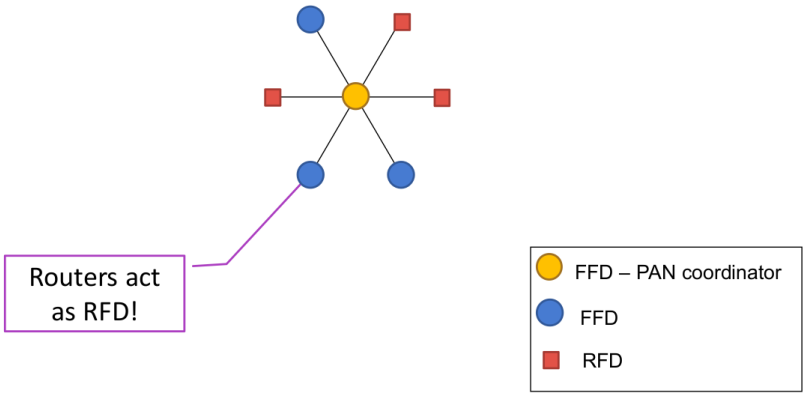
- **Data services**
  - transmission and reception of MAC frames (MPDU) across the physical layer.
- **Management services**
  - Synchronization of the communications
  - Channel access
  - Management of guaranteed time slots
  - Association and disassociation of devices.
- **Security services**
  -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

This layer defined two types of devices:

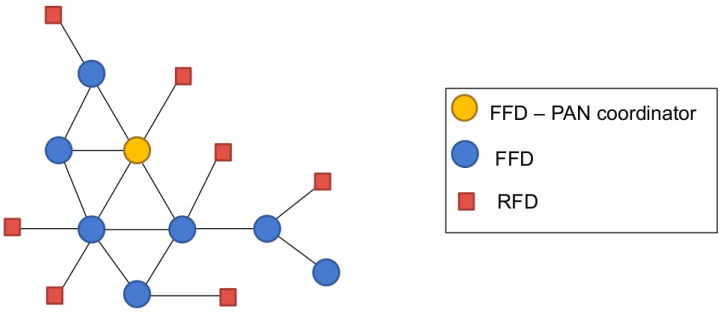
- **Full-function device** (FFD): can act as a coordinator;
- **Reduced-function device** (RFD): cannot act as a coordinator.

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.



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

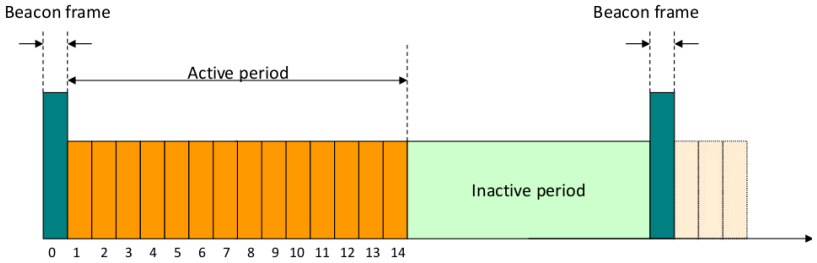


Figure 8.2: Beacons and superframes in 802.15.4

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.

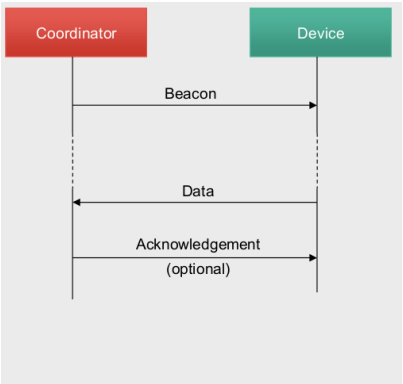
8.1.3 Trasferring data

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.*

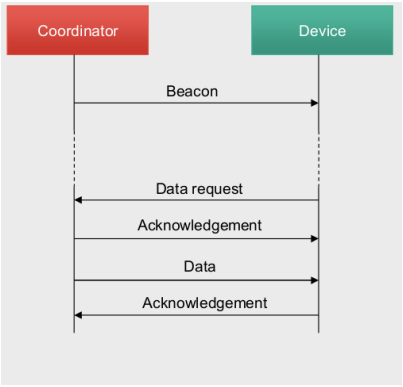
8.1.3.1 Beacon enabled

1. *Data transfer from an end device ⟶ 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 ⟶ end device*:

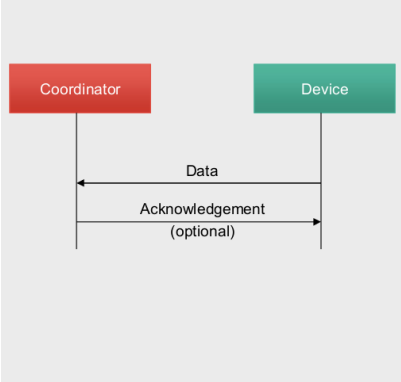


   i. The coordinator stores the message and signals that the message is pending in the beacon;
   ii. The end-device sleeps most of the time and it occasionally listens to the beacon to check for pending messages, and it notices that there is one, it requests the message to the coordinator.
   iii. The 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. ☺

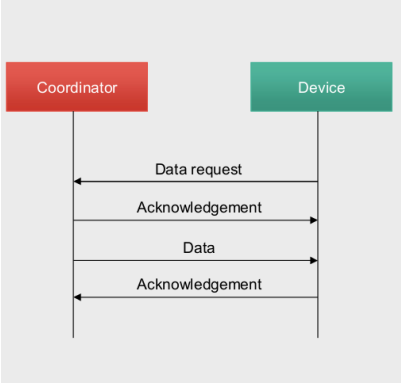
8.1.3.2 Non beacon-enabled

1. *Data transfer from an end device ⟶ 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 ⟶ 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.
   Bad!
      - ◇ 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

8.2.1 Associating

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:

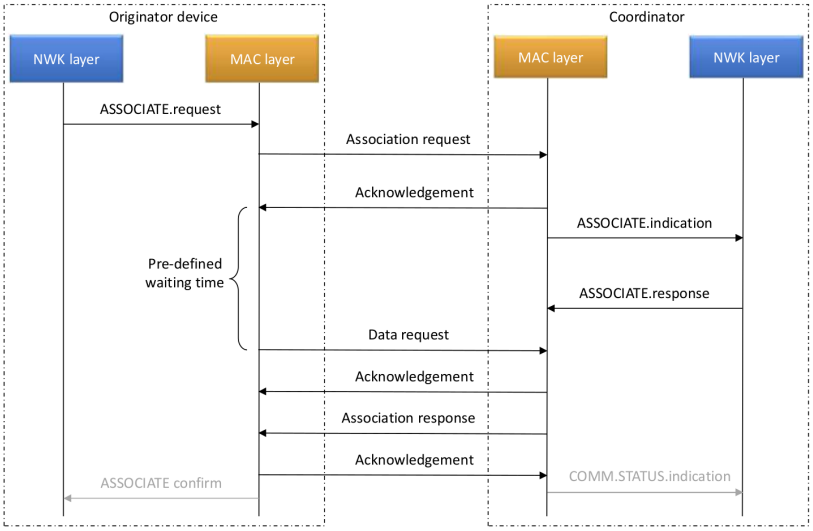


Figure 8.3: Associate protocol in 802.15.4

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.
2. The coordinator answers with an `ACK`, which however doesn't mean that the device has joined the network.
3. The coordinator forwards the association request to the network layer, which in case of acceptance sends an `ASSOCIATE.response` primitive to the MAC layer, indicating both the MAC of the end-device and its "associated" *16-bits address*.
4. After some predefined time, the end device requests the address, followed by an exchange of ACKs
5. Lastly the end-device MAC layer sends an `ASSOCIATE.confirm` primitive to the upper layer.

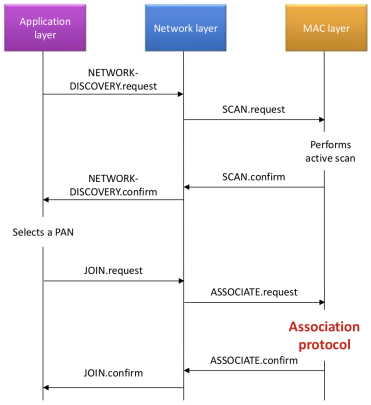


Figure 8.4: The associate protocol takes place in the last part of the ZigBee network join process

Security services

The MAC layer provides some security features:

- ◇ Access control through ACLs
- ◇ Symmetric Data encryption using keys provided by upper layers
- ◇ Frame integrity using the same keys of encryption
- ◇ Sequential freshness of frames

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 using an `init` procedure 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.

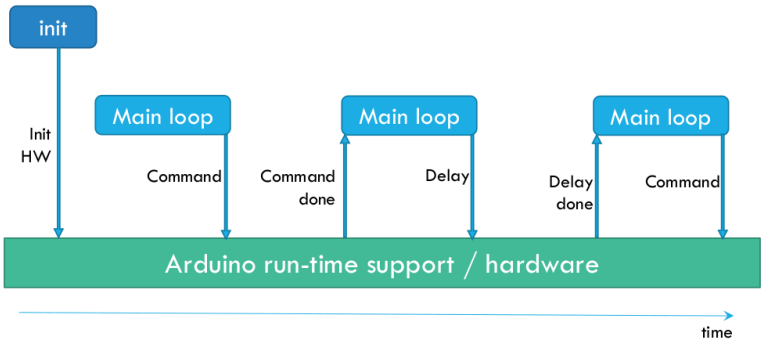


Figure 9.1: Sample flow of execution in Arduino.

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.

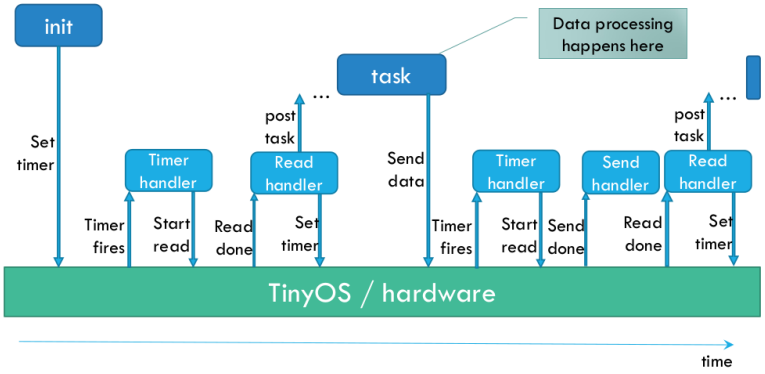


Figure 9.2: Sample flow of execution in TinyOS.

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.

Chapter 10

Energy Harvesting in IoT

In general, finite battery capacity calls for a tradeoff between performance and lifetime of IoT devices, which is a major challenge in the IoT domain. Energy harvesting is a promising solution to this challenge, as it can provide a continuous and sustainable energy source for IoT devices. In this chapter, we provide an overview of energy harvesting in IoT, including the energy sources, energy harvesting techniques, and the challenges and opportunities in this area.

10.1 Introduction and definitions

Energy harvesting converts energy from one form to another. If the harvested energy source is large and periodically/continuously available, a device can last "forever". This approach also allows to tune the device parameters for a better performance, since it removes the constraint of lifetime.

Note that the energy production varies with time and on environmental conditions; this is an aspect typically out of the control of the designer.

Definition 10.1 (Energy Source) *Source of energy to be harvested*
Sun, wind, etc...

Definition 10.2 (Harvesting source) *Any available harvesting technology, (solar cells, wind turbines, piezo-electric harvesters etc...) that extracts energy from the environment.*

Definition 10.3 (Load) *Consumption of energy in a device due to its activities*

- ◇ *A device has many subsystems (processor/memory, radio, storage, transducers/ADC), each with its own states and consumption*
- ◇ *The load varies over time: depends on the specific activities the device is executing at a time*

Definition 10.4 (Harvesting System) *A system that supports a variable load from a variable energy-harvesting source, also when the instantaneous power supply levels from the harvesting source do not match the load.*

10.1.1 Takeaway

In theory, there are two main approaches:

1. **adapt the load** to the actual energy supply
2. use an **energy buffer**
   i.e. a rechargeable battery or a supercapacitor

However, in practice neither of these two approaches may be sufficient, since the load cannot be reduced arbirarily, and the buffer is not ideal (no infinite storage and eventually energy leaks).

10.2 Harvesting Architectures

10.2.1 Harvest-use

In this architecture energy is harvested just-in-time for use.
The power output has to be above the minimum operating point (else the device turns OFF), but note that this might leads abrupt variations to cause the device to oscillate between ON and OFF states.

The problem is that a device may suffer from "Alzheimer's disease", i.e. it forgets what it was doing before turning OFF, since the content of the RAM is lost.

This is an issue currently being addressed by researchers.

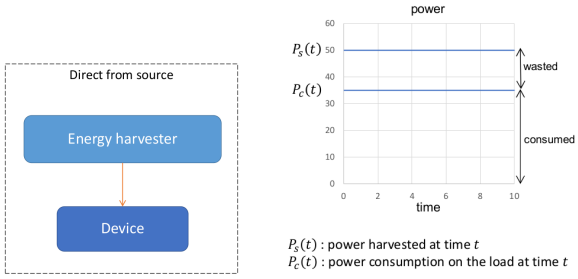


Figure 10.1: Harvest use schema

The energy harvested, but not required by the device functioning is **wasted**.

10.2.2 Harvest-store-use

Energy is harvested whenever possible and stored for future use.
Residual energy is stored and it is used later when either there are no harvesting opportunities or the device tasks require more energy.

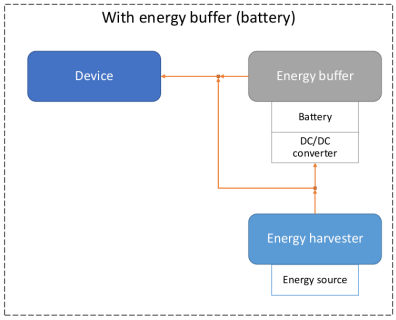


Figure 10.2: Harvest-store-use schema

We assume for the moment that the energy buffer is ideal, i.e. it can store an infinite amount of energy, it does not leak, and has a charging efficiency $\eta = 1$.
Under such assumptions the device can operate at any time if, for every time interval $(0, T]$

$P_s(t)$	*power harvested at time t*
$P_c(t)$	*power consumed on the load at time t*
B_0	*initial energy in the buffer*

$$\int_0^T P_c(t)dt \leq \int_0^T P_s(t)dt + B_0 \quad \forall T \in (0, \infty)$$

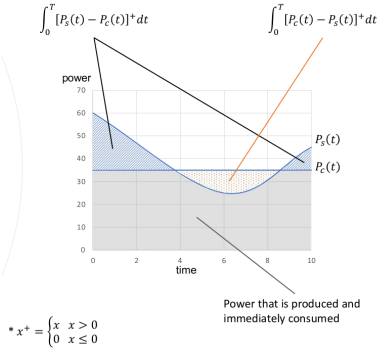


Figure 10.3: Harvest-store-use modeling schema

10.2.3 Harvest-store-use with a non-ideal buffer

We need to introduce other symbols

B_{max}	*maximum battery capacity (energy buffer size)*
B_t	*battery charge at time t*
$P_{leak}(t)$	*be the leakage power of the battery at time t*
$\eta < 1$	*be the charging efficiency of the battery*
$P_s(t)$	*power harvested at time t*
$P_c(t)$	*load at time t*
B_0	*initial charge of the buffer*

$$x^+ = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

x^+ *is a "rectifier function"*

$$B_T = B_0 + \eta \int_0^T [P_s(t) - P_c(t)]^+ dt - \int_0^T [P_c(t) - P_s(t)]^+ dt - \int_0^T P_{leak}(t) dt \quad \forall T \in (0, \infty]$$

10.2.4 Exercise

Question

Consider an harvest-store-use device with a non-ideal energy buffer:

- In the interval $[0, 10sec]$ the energy production is constant and produces $P_s(t) = 80\ mA$
- In the interval $[0, 4sec]$ the load is $P_c(t) = 150\ mA$
- In the interval $[4sec, 10sec]$ the load is $P_c(t) = 20\ mA$

Furthermore:

- The charging efficiency is $\eta = 95\%$
- The battery charge at time 0 is $400\ mAh$
- The energy leak of the battery is negligible

Compute the battery charge at times $4sec$ and $10sec$

$$\begin{aligned} &\textit{Conversion from } mAs \textit{ to } mAh 3600mAs = 1mAh \\ &400mAh + \cancel{0.95 \times (80 - 150 = -70) \times 4} - 4 \times (150 - 80 = 70)mAs \\ &400mAh - 280mAs/3600 = 399.92mAh \\ &\underline{\textit{Battery charge at time 4 sec}} = 399.92mAh \\ &399.92mAh + 0.95 \times (80 - 20 = 60) \times 6/3600mAh - \cancel{6 \times (20 - 60 = -40)} \\ &\underline{\textit{Battery charge at time 10 sec}} = 400.02mAh \end{aligned} \tag{10.1}$$

10.3 Energy sources classification

- ◇ **Fully controllable** energy sources can provide harvestable energy whenever required.
  Example self-power flashlights: the user may shake to generate energy whenever needed
- ◇ **Partially controllable** energy sources may be influenced by the system design, but the result may not be deterministic
  e.g. RF energy source installed in a room and RFID's may extract energy from it. However, the energy produced at a node depends on RF propagation in the room that cannot be fully controlled
- ◇ **Non-controllable** energy sources cannot be activated on demand. In these cases the energy must be harvested whenever available.
  e.g. wind, sun, etc...
  - **Predictable** energy sources are those for which there exist reliable models that forecast the energy availability.
    e.g. Sun cannot be controlled but it can be predicted (to some extent). Day/night production, summer/winter, weather forecasts, ...
  - **Unpredictable** energy sources are those for which there not exist reliable models that forecast the energy availability
    e.g. vibration due to sources for which there are no models available (earthquakes etc...)

Fun fact: the efficiency of the plants' photosynthesis is around $10 - 15\%$

Energy source	Characteristics	Amount of energy available	Harvesting technology	Conversion efficiency	Energy harvested
Solar	Ambient, uncontrollable, predictable	100 mW/cm²	Solar cells	15%	15 mW/cm²
Wind	Ambient, uncontrollable, predictable		Anemometer		1200 mWh/day
Vibrations (industrial)	Controllable		Piezoelectric, induction		100 µW/cm²
Motion (human)	Controllable	19 mW – 67 W	Piezoelectric	7.5% – 11%	2.1 mW – 5 W
Thermal (human)	Uncontrollable, predictable	20 mW/cm²	Thermocouple		30 µW/cm²
Thermal (industrial)	Controllable	100 mW/cm²	Thermocouple		1-10 mW/cm²
Radio frequency (base station)	Controllable	0.3 µW/cm²			0.1 µW/cm²
radioactivity		60 mW/cm³	Radioactive decay		

Figure 10.1: Energy sources summary

10.4 Harvesting sources

10.4.1 Radio Frequency (RF) energy

When electromagnetic radio frequency (RF) field passes through an antenna coil, an AC voltage is generated across the coil.

A passive RF tag powers itself by using RF energy transmitted to it; active RF tags instead have their own battery.

RFIDs are used to identify, locate and track people, assets and animals: An RFID reader queries an RFID tag by sending RF signal, and the RFID tag is entirely powered by the energy harvested by the antenna coil. This energy is just sufficient to send back a reply.

10.4.2 Piezo-electric energy

Use mechanical force to deform a piezo-electric material, which results in an electric potential difference.

- ◇ Piezo-electric *films*: `PVDF` (PolyVinylidene Fluoride)
- ◇ Piezo-electric *ceramic*: `PZT` (Lead Zirconate Titanate)
  PVDF is more flexible than PZT

10.4.3 Wind turbines

There are some wind turbines for IoT applications, which are:

- ◇ Small size
- ◇ Low height
- ◇ Operative range with weak winds

When reading articles about energy harvesting, it is common to encounter diagrams like the one in Fig. 10.2. Different Load resistances provide different efficiency, meaning that the energy produced does not depend only on the wind speed, but also on the load resistance, and does not decrease *linearly*.

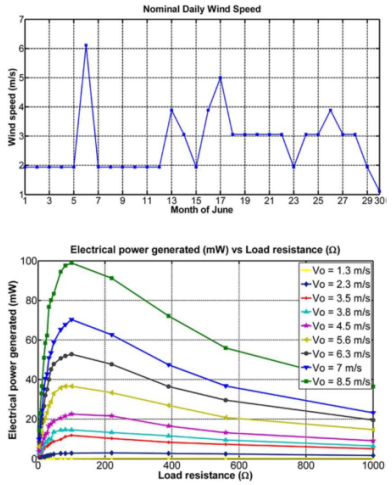
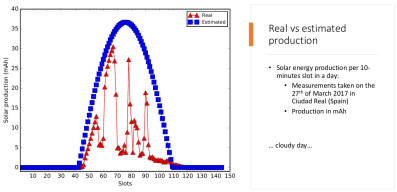


Figure 10.2: Wind turbines energy production diagram

10.4.4 Solar energy

The energy production of a panel depends on several factors:

- ◇ **Solar radiation**
  Depends on the angle drawn by the Sun with, respect to the vertical axis of the Earth surface,that depends on the geographical position of the panel, and on the day in the year; besides it also depends on the weather conditions.
- ◇ **Size of the panel**
- ◇ **Charging efficiency**



The typical countermeasure to such variations is to *oversize* the harvesting subsystem and battery capacity to match with the worst conditions.

10.4.5 Considerations

Energy harvesting does not mean that a device is always operational!

If the device is harvesting solar and wind energy, the residual battery may not last for a whole windless night, so the device may turn off during the night and back on in the morning.

10.5 Storage technologies

Battery type	Nominal voltage (V)	Capacity (mAh)	Weight energy density (Wh/kg)	Power density (W/kg)	Efficiency (%)	Self discharge (%/month)	Memory effect?	Charging method	Recharge cycles
SLA	6	1300	26	180	70-92	20	no	trickle	500-800
NiCd	1.2	1100	42	150	70-90	10	yes	trickle	1500
NiMh	1.2	2500	100	250-1000	66	20	no	trickle	1000
Li-ion	3.7	740	165	1800	99.9	<10	no	pulse	1200

Figure 10.2: Comparison amongst different battery cell types

There are for main battery cell types, listed below and compared in Fig. 10.2:

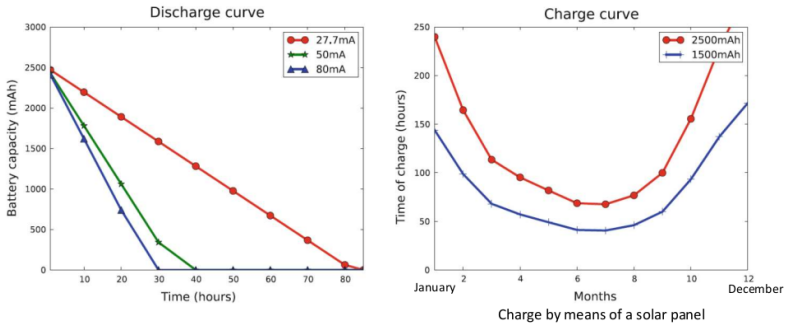
1. Sealed Lead Acid (`SLA`)
2. Nickel Cadmium (`NiCd`)
3. Nickel Metal Hydride (`NiMH`)
4. Lithium Ion (`Li-ion`)

Supercapacitors are a new highly efficient technology, but with high discharge rate and low power density. They can be used to buffer energy if energy source is jittery: when no-load, allows to charge at the same rate of discharge to keep the capacitor at full charge level (*trickle charging*).

Super capacitors	
Weight energy density (Wh/Kg)	5
Efficiency (%)	97 – 98
Self discharge	5.9% per day
Memory effect	no
Charging method	trickle
Recharge cycle	infinite

Figure 10.3: Supercapacitors specs

10.6 Measuring discharge



In order to decide which storage to adopt, it is crucial to measure its discharge, to properly choose a solution which

fits the requirements.

We may measure the battery discharge using mathematical theorems and equations:
assume —an *ideal energy buffer* and that— we have a method for measuring the battery charge $E_b(t)$ at time t, and that the power consumption $p_c(t)$ in a time frame $[t_1, t_2]$ is known, then the energy E_e produced in the same time frame is given by the following equation

$$E_e = \left[\int_{t_1}^{t_2} p_c(t)dt + E_b(t_2) - E_b(t_1)\right]^+$$

A better way is to use electronic circuits to measure the current flowing out of the harvesting source and its voltage. Using a d-bits ADC and assuming minimum and maximum voltage values (v_{min} and v_{max}, corresponding to x_{min} and x_{max}, the "d-bits" values read by the ADC) the battery charge may be obtained with this equation

$$B = B_{min} + \frac{B_{max} - B_{min}}{x_{max} - x_{min}}(x - x_{min})$$

10.7 Modulating the load

An IoT device has several components, each characterized by a power consumption (cost) per unit of time, in fact components can be turned on/off by the processor when not in use, and even the processor can be put in low-power mode.

Load modulation means reducing the amount of work to reduce the energy consumption.

For example, a device may vary the sampling frequency of its sensor:

- ◇ High sampling frequency implies high energy cost (but also high utility)
- ◇ Low sampling frequency implies low energy cost and low utility

Earlier on the goal was to maximize the device lifetime, but nowdays the standard approach is to obtain **energy neutrality**,which is the ability of a device to keep the desired performance level "forever".

10.8 Energy neutrality

1. **Energy-Neutral Operation**:
   How to operate such that, in any give time frame, the energy used is always less than the energy harvested?
2. **Maximum Performance:**
   While ensuring energy-neutral operation, what is the maximum performance level that can be supported in a given harvesting environment?

The key elements are

- The amount of harvested energy from (each) source
- The current charge of the battery
- The power consumption of the device (the load), which is modulated to ensure energy neutrality
- The future energy production estimation
  - By means of weather forecasts or other methods
  - Subject to errors
  - Usually under pessimistic assumptions: errors maximized and expected production minimized

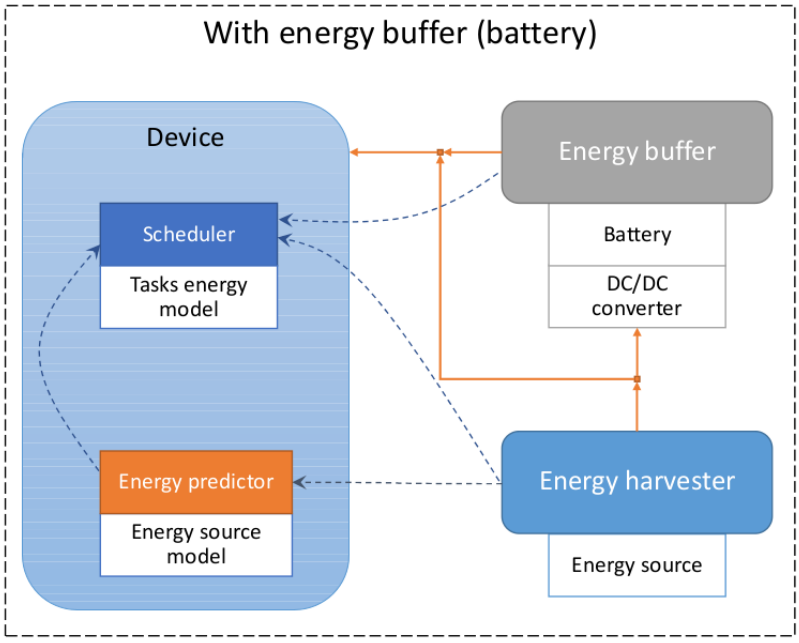


Figure 10.3: Revisited harvest-store-use diagram
Energy is stored for "future use" and is used later when either there are no harvesting opportunities, or the sensor tasks require more energy

10.8.1 Kansal's algorithm for Power Management

Kansal considers the case of *predictable* but *uncontrollable* sources.

The objectives are three:
1. keep the system **energy neutral**
2. ensure the system **never fails** due to energy depletion
3. maximize the node's **performance**

and the approach to do so is to take current end expected battery charge into account and dynamically tune the device's performance (and thus the load), so that the device *neither* operates below minimum performance levels nor switches OFF before the next recharge cycle.

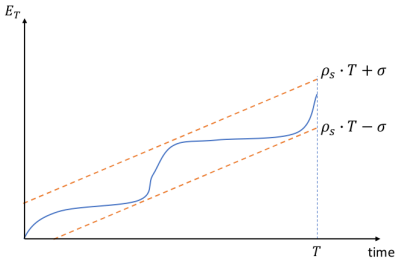


Figure 10.4: Kansal 1^{st} condition: *energy production*

$$E_T = \int_T P_S(t)dt$$

E_T is the amount of energy produced in a $[0, T]$ interval. By looking at the graph, we note/assume that for some $\rho_S, \sigma \in \mathbb{R}$

$$\rho_S T - \sigma < E_T \leq \rho_S T + \sigma$$

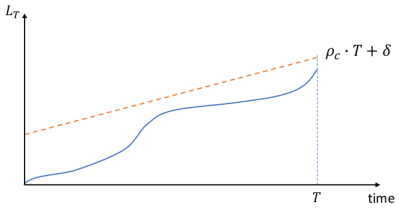


Figure 10.5: Kansal 2^{nd} condition: *load*

$$L_T = \int_T P_C(t)dt$$

L_T is the amount of energy consumed in a $[0, T]$ interval. for some $\rho_S, \phi \in \mathbb{R}$

$$0 < L_T \leq \rho_C T + \Sigma$$

Kansal's theorem

if the previous assumptions on E_T and L_T *hold*, and the energy buffer is characterized by charging efficiency η and leakage ρ_{leak}, then a sufficient condition for energy neutrality of the system is:

$$\begin{cases} \eta\rho_s \geq \rho_c + \rho_{leak} \\ B_0 \geq \eta\sigma + \delta \\ B_{max} \geq B_0 \end{cases}$$

- The trend of production (considering the charging efficiency) should exceed the trend of load and of leak together
- The initial battery charge should be sufficient in the worst case
- The required initial battery charge should be admissible (less than the battery capacity)

Kansal's model of utility and duty cycle enables a modulation of the load on the energy harvesting device. Consider a system designed to detect the presence of intruders in a room.

- ◇ The minimum and the maximum duty cycles imply a range of possible sampling frequencies
- ◇ The higher the frequency the better the ability to detect intruders and the higher is thus the utility
- ◇ The lower the frequency the lower is the ability to detect the intruders and thus the utility
- ◇ Below dc_{min} the detection ability is impaired
- ◇ Above dc_{max} it is useless to sample (the intruder is not that fast...)

This leads to an *optimization problem* where:

- ◇ The *production varies* over time (uncontrolled), but it is *predictable* (by assumption)
- ◇ The *load varies* over time according to the duty cycle (controlled)
- ◇ The battery *charge varies* over time accordingly

  The battery charge variation may be computed, using the —controlled— load and the forecasted energy production

Since the energy source is *predictable* we have to find the load so that the *system is energy* ***neutral*** and the overall ***utility*** *is maximized*

Kansal's approach to assign a duty cycle at each time slot (assuming time is slotted), does not require the knowledge of the future production[1], but just an *estimation* based on previous observations, an it builds the cycle over three components:

1. forecasts future power production based on past power production, using a `EWMA` (Exponentially Weighted Moving Average) filter[2]

$$\tilde{p}_S^{j+1}(i) = \alpha \cdot \tilde{p}_S^j(i) + (1 - \alpha) \cdot \tilde{p}_S^j(i) \tag{10.2}$$

[1] However it may be available in some cases, such as weather forecasting

[2] i.e. assumes the future production to be similar to the past day

2. uses a simple polynomial-time algorithm (suitable for low-power processors) to solve the optimization problem
3. performs a reoptimization (dynamic adaptation) if the actual power production deviates significantly from the expected one

> k number of time slots
> $B(i)$ battery charge at the beginning of slot i
> $B(k+1)$ battery level at the end of slot k (i.e. at the end of the day)
>
> Based on the expected power production $\tilde{p}_S(i)$ at each slot $i \in [1, k]$, the algorithm assigns a duty cycle $dc(i)$ and hence a utility $u(i)$to each slot i, while requiring that $B(k+1) \geq B(1)$ i.e. that the system must be *energy neutral*.

What about the number of time slots k? There is not a fixed number and there is not a method to determine the best one; three considerations may be done in regards to it:

1. Too small implies a simpler model, but higher risk of error. Too large implies a possibly (not necessarily)more accurate model, but higher computational cost
2. If time slots are too small (k is large) the signal output may present too many oscillations
   Consider a device which every minute changes the sample rate: this may lead to a lot of oscillations in the signal output, resulting in more complicate data analysis
3. If time slots are too small, you may not be able to properly feed the estimation model, e.g. 1-minute slotting may not be significant to estimate the production of a solar panel

10.9 Task-based model for Energy Neutrality

Usually, an application for an IoT device implements 4 steps in a cycle, each one with a potentially different implementation:

1. Sensing
2. Storing
3. Processing
4. Transmitting

Different implementations may imply a different behaviour of the server (either in the gateway or in the cloud). Even though the system functionality does not change, the implementation may change indeed.

- ◇ Example 1:
  If the application process all data and transmits only the results, the server may just store the received data.
- ◇ Example 2:
  If the application just transmits all the sensed data, then the server has to process and then store the data.

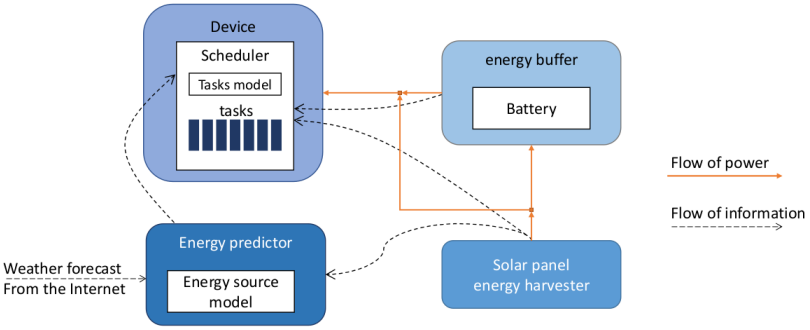


Figure 10.6: Task based model architecture

We call **task** an implementation of the application in the IoT device. We may achieve *load modulation* by scheduling the tasks in execution; clearly, the server needs to know which task is running to behave accordingly.

To each task we may associate a *power consumption* and a *utility*, allowing a dedicated *task scheduler* component to properly choose the task to run in each **time slot**.

Kansal's time slotting goes nicely along with this architecture

The scheduler must solve an **optimization problem** by means of linear programming, which is to maximize the utility of the tasks assigned to the slots:

$$max \sum_{i=1}^{k} u(i) \sum_{j=0}^{n} x_{i,j} \cdot u_j \tag{10.3}$$

Given:

$$\sum_{j=1}^{n} x_{i,j} = 1 \forall i \in [1,k] \qquad \textit{exactly one task per slot} \tag{10.4}$$

$$B(1) \leq B(k+1) \qquad \textit{energy neutrality} \tag{10.5}$$

$$B_{min} \leq B(i) \forall i \in [1,k] \qquad \textit{Battery may never go below the minimum} \tag{10.6}$$

$$B(i+1) = min\left\{B_{max}, B(i) + \eta \cdot p_S^+(i) - p_C^-(i) \forall i \in [1,k]\right\} \qquad \begin{array}{r} i^{th} \textit{ slot charge depends on initial charge,} \\ \textit{production, and consumption in the slot} \end{array} \tag{10.7}$$

The described problem may be reduced to the famous *knapsack problem*, which –sadly— is **NP-hard**, thus hardly solvable for most commodity PCs, let alone for IoT devices.

The knapsack problem is solvable by means of a *dynamic programming* approach, exploiting is a *polynomial-time* algorithm.
Assuming that the system state is defined by (i, b), where

- i is the time slot up to which the system is optimized
- b is the battery charge at the beginning of the slot

Then the optimal utility $opt(i, b)$ can be computed exploiting a backware recursive rule:

$$opt(k,b) = max_{j=1,\ldots,n}\left\{u_j : b + \eta \cdot p_S^+(k) - p_C^-(k) \geq B(1)\right\} \tag{10.8}$$

$$opt(i,b) = max_{j=1,\ldots,n}\left\{u_j : opt\left(i+1, B^j(i+1)\right) : B^j(i+1) \geq B_{min}\right\} \tag{10.9}$$

- Where in $opt(k, b)$ results from assuming task j is assigned to slot k
- Where $B(i+1)$ is the residual battery at the end of slot i if task T is assigned to that slot

Chapter 11

Wireless sensors networks

Traditional sensor monitoring was implemented with basic sensors made up only by transducers connected by a wire to a centralized device, such as Arduino. In WSNs instead, the sensors are connected wirelessly to a centralized device, which is usually a computer or a microcontroller, and they present many characteristics:

- ◇ Intelligent
- ◇ Wireless
- ◇ Autonomous
- ◇ Capable of building a Network

11.1 Deploying a WSN

Sensor deployed in a "Sensing Field" form a network made up —one or more— sink nodes and a set of sensor nodes. Each sensor produces a stream of data which may be pre-processed by the sensor itself before being eventually sent to a sink node. Sinks may not always be available, so the sensors may act as "loggers" and store data for future retrieval; since energy efficiency is a key factor, some **data aggregation** may be performed on the network to improve the efficiency.

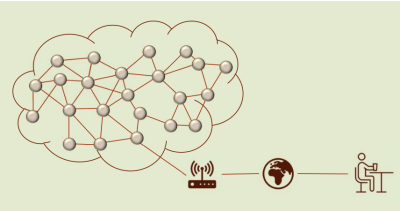


Figure 11.1: WSN architecture

11.1.1 WSNs Strenghts

- ◇ Network deployment is easy:
  - – no need for cables
  - – self-configurable (no centralized control)
- ◇ Sensors are cheap:
  - – The number of sensors can scale up
  - – redundant sensors to enforce fault tolerance
- ◇ Sensors can be mobile
  - – if wearable or deployed on mobile objects
- ◇ Sensors can be programmable on the fly
  - – to adapt to changing conditions
  - – to implement new sensing tasks

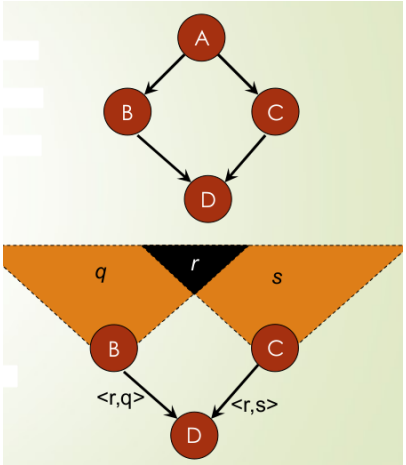


Figure 11.2: Implosion and Overlap issues, described in Sec. 11.2.1

11.2 WSN Characteristics

WSNs first of all may be data-centric or node-centric. This refers especially to routing algorithms, since tradinational ones are too resource-consuming for WSNs. Most WSN are data-centric, meaning that the data is the most important thing, and the network is built around it.

11.2.1 Implosion and Overlap

The **"implosion"** problem happens due to flooding-based dissemination:

- ◇ node A starts by flooding its data to all of its neighbors, including B,C connected to node D.
- ◇ Two copies of the data eventually arrive at the aggregation node D.
- ◇ The system wastes resources in one unnecessary send and receive.

The **"overlap"** problem happens when two nodes send the same data to the same node, which is a waste of resources. This may happen when to sensor cover the same overlapping geographical region, (e.g. two cameras covering the same area, or two termometers in the same room) in Figure 11.2, such region is the *"r"* marked.

11.2.2 Directed Diffusion

This is an approach designed and presented by **Deborah Estrin** and her team in 2000 at Mobicom.

- ◇ Data is **named** using attribute-value pairs.
- ◇ The sink disseminates a sensing task in the network as an interest for named data.
- ◇ The dissemination of interests sets up gradients.
- ◇ gradients "draw" events (i.e. data matching the interest)
- ◇ Data matching the interest flows towards the sink
  Along the gradients, following multiple paths
- ◇ The sink *reinforces* one or some of these paths.

Interest are named by a sequence of attribute-value pairs that describe the task.

```
type        = four-legged animal       // detect animal location
interval    = 20 ms                    // send back events every 20 ms
duration    = 10 seconds               // .. for the next 10 seconds
region      = [-100, 100, 200, 400]    // from sensors within rectangle
```

The data sent in response to the interest is also named using a similar naming scheme.

```
type        = four-legged animal       // type of animal seen
instance    = elephant                 // instance of this type
location    = [125, 220]               // node location
intensity   = 0.6                      // signal amplitude
confidence  = 0.85                     // confidence in the match
timestamp   = 01:20:40                 // event generation time
```

An interest is associated to a duty cycle (?), which is the time interval between two consecutive transmissions of the same interest. *"Refreshes"* (i.e. broadcasting again the same interest) are necessary because dissemination of interests is not reliable. The first broadcast of an interest instead is called *"Exploratory"*.

Nodes **cache** the received interests, allowing for ones differing only for sampling rate to be aggregated; interests in cache expire when the duration time expires.
Each interest in cache has a **gradient**, which expresses:

- ◇ a direction (the node from which the interest was received), used to route data back to the sink
- ◇ a data rate

11.3 Direct Diffusion Finite State Machine

When a sensor detects an event matching with an interest in cache:

1. Starts sampling the event at the largest sampling rate of the corresponding gradients
2. Sends sampled events through the gradients associated to the interests in cache
   - These gradients correspond to the neighbors interested in the event

Neighbors forward the data only if a corresponding interest (with a gradient) is in their cache, and if the data has not been forwarded before (i.e. redundant copies are dropped).

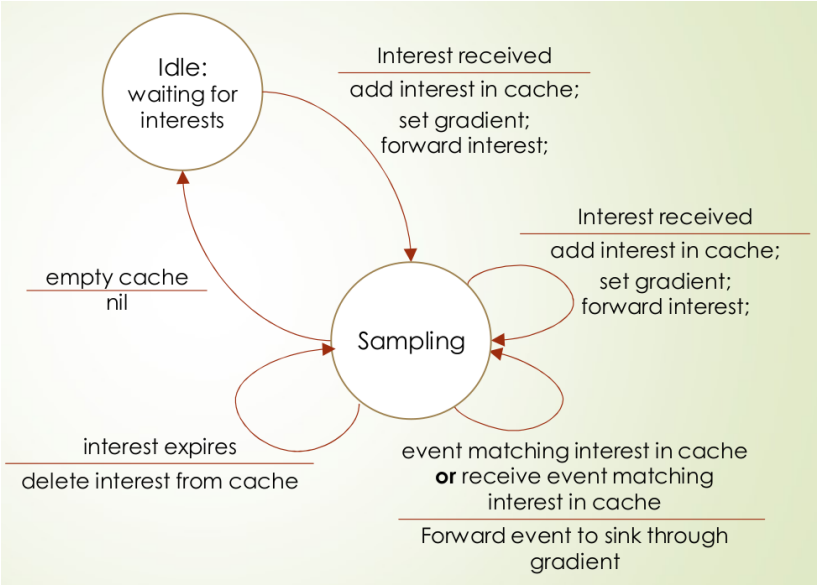


Figure 11.1: Direct Diffusion Finite State Machine

Once it has received data matching an exploratory interest from sensor n, the sink reinforces n to improve the quality (higher sampling rate) of received data

- ◇ Exploratory interests usually have a low sampling rate
- ◇ Reinforces of interests specify an interest with *higher sampling rate*
- ◇ In turn, the reinforce propagates along the path from which the events are received up to sensor n

The *overlap* problem instead (multiple sensors covering the same area) is not addressed by this approach, and may still occur(?).
Reliability is *not* guaranteed, since a message may be lost, however, it's not a big deal, there will soon be another one ☺.

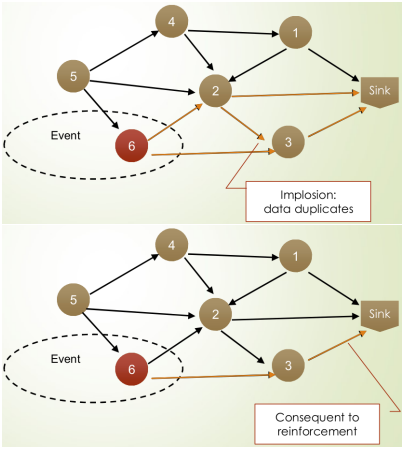


Figure 11.2: Since the gradients —routes to the sink— may be multiple, *implosion* may occur. The sink, once it has received data from a sensor n, may reinforce one of the paths to avoid this.

Assumptions

Note that Direct Diffusion works under the following key assumptions:

- ◇ One sink (with `id 0` and denoted n_0)
- ◇ the sink has a double radio interface: one to the WSN and one to the internet
- ◇ Each node has a unique `id` (denoted n_i)
- ◇ The sink initializes and maintains the routing tree
- ◇ Unicast messages from sensors to the sink
- ◇ Only the sink broadcasts to the entire network
- ◇ Sink has 2 antennas: one for the WSN and one for the internet

11.4 Topologies

Trees provide scalability but sink nodes (two corners of the figure) have larger power consumption, just as nodes close to them, which act as **bottlenecks**. Note that the position in the tree resembles the phyisical position of the nodes in the field, which is often not arbitrary, but is dictated by various factors instead.

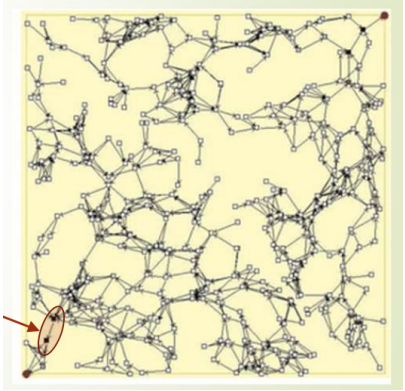


Figure 11.3: WSN Tree topology issue

There some major drawbacks for WSN in tree topologies:

1. Sink must always be connected to the network (which in turn is not autonomous)
2. Does not exploit —even if little— processing and storage capabilities of the sensors
3. Detection of composite events requires communication among sensors
4. Bottlenecks and load unbalance

On the other side, without trees we need more complex routing algorithms, which are more resource-consuming. That's where GPSR comes in.

11.5 GPSR

The desire was to provide internet-like routing, but without the resource overhead of traditional routing algorithms, in order to overcome the drawbacks of WSN applied to tree topologies. This problem was addressed in 2003 by **Brad Karp** and **H. T. Kung** with the **Greedy Perimeter Stateless Routing** algorithm, which exploits the GPS information of the nodes to route the packets.

The protocol works under three assumptions:

- The network is **connected**
  Source knows the coordinate of the destination.
  This is kind of odd, since nodes only have a local view of the network, not of the whole network. So... *how does the source know the destination's coordinates?*
- The network is **planar**
  Nodes are deployed in a 2D plane, and the network is planar if no two nodes overlap.
- The network is **localized**
  Nodes are aware of their own position and of the position of the neighbors

The protocol completely removes the need for route discovery and consequent route caches and routing tables.

11.5.1 Greedy and Perimeter Modes

GSPR comprises two modes:

- **Greedy forwarding**
  The packet is forwarded to the neighbor that is closest to the destination.
  More formally, The forwarding node x select as next hop a neighbor y such that y is closer to Destination D than x and that y is the closes to D among all the neighbors of x.
  Greepy forwarding fails if it encounters a *void* region.
- **Perimeter forwarding**
  The packet is forwarded to the neighbor that moves it along the perimeter of the void region.
  It is based on the `LHL` (Left Hand on the Left) rule, or the `RHR` (Right Hand on the Right) rule; i.e. When arriving from y to x selects the first counterclockwise edge from (x, y).
  In other words, the packet traverses the interior of a closed polygonal region (face) in clockwise —or counterclockwise— order.

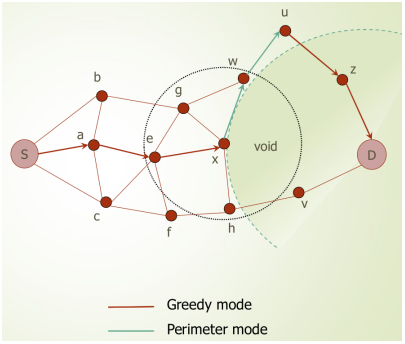


Figure 11.4: GPSR Greedy and Perimeter forwarding

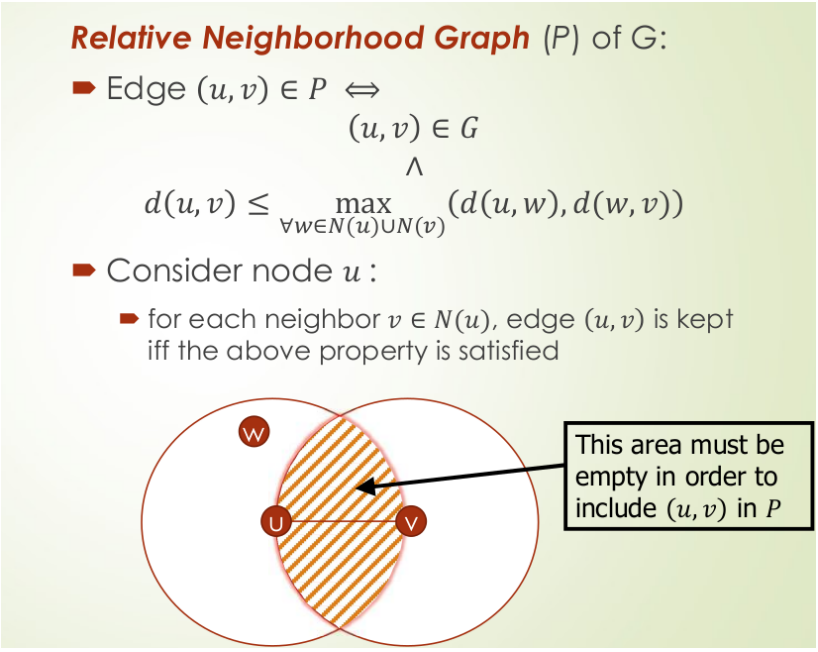
Switching from the two modes

The switch from Greedy to Perimeter mode is triggered when the Greedy mode fails, i.e. when the node finds a void region, so no nodes closer to D are reachable.
The switch back to Greedy mode cannot be triggered as the node exits the void region, since this may lead to fallback loops. The switch is instead performed when a node closer to D than $\tilde{x}$ —the node from which the perimeter mode started— is found.

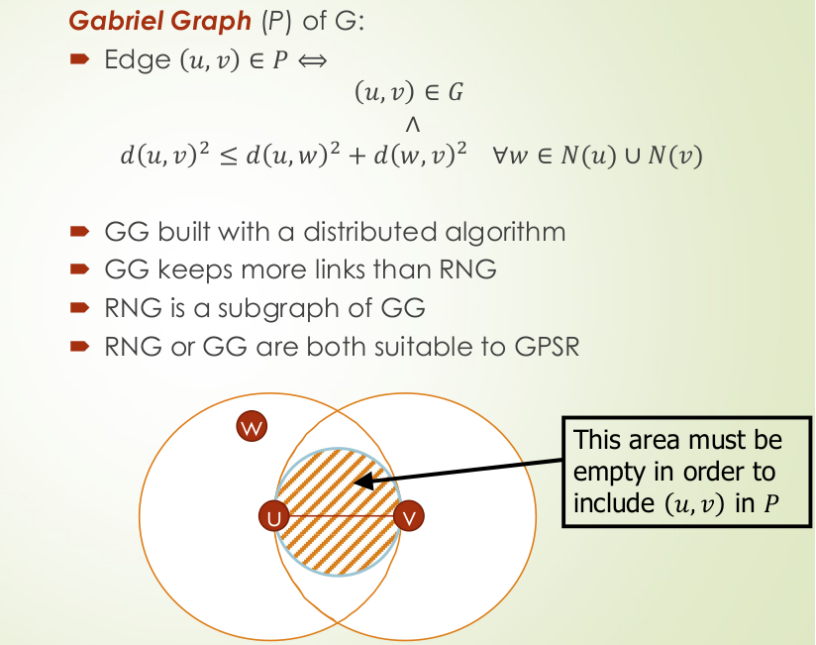
11.5.2 More on Perimeter mode

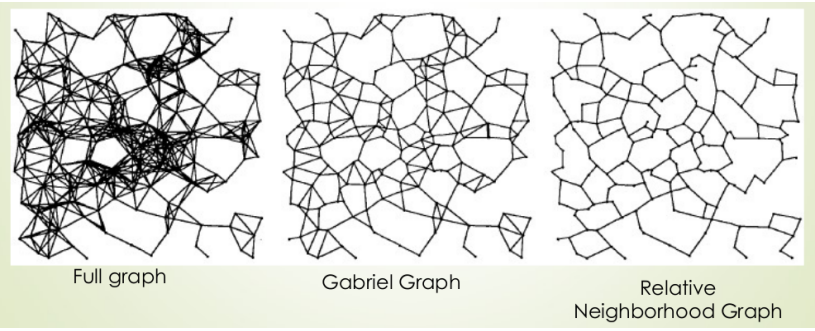
RHR applied to a non-planar graph may lead to a degenerate tour that does not trace the boundary of a closed polygon. For this reason, given the non-planar graph G, the algorithm constructs a planar —sub—graph P of G, and applies RHR to P; P may either be the

- *Relative Neighborhood Graph* of G



- *Gabriel Graph* of G





Greedy mode exploits the whole graph G, while Perimeter mode exploits only the links in the planar graph P.

Mobility

The problem with GPSR is **mobility**, mostly due to the need for a planar graph for perimeter mode to work correctly. It needs a freshly planarized graph, using "stale" planarized graph may result in performance degradation; On the other side, planarizing the graph at each topology change is not good either, since it may lead to link instability, because nodes may move within a node's transmission range, which may continuously change the selection of links by GG or RNG.

To solve the problem, a proactive approach is used, where nodes periodically communicate their position to their neighbors (beaconing), and beacons are used to keep the neighbor's list. When the changes are excessive, planarization is performed.

There are three possible configurations for GPSR responsible for possible loops. The most problematic is the umbrella-shaped one. Since the configuration may be complex and made up of multiple nodes, it is not easy to detect, and doing would required the whole view of the network.

11.5.3 GPSR obstacle fail

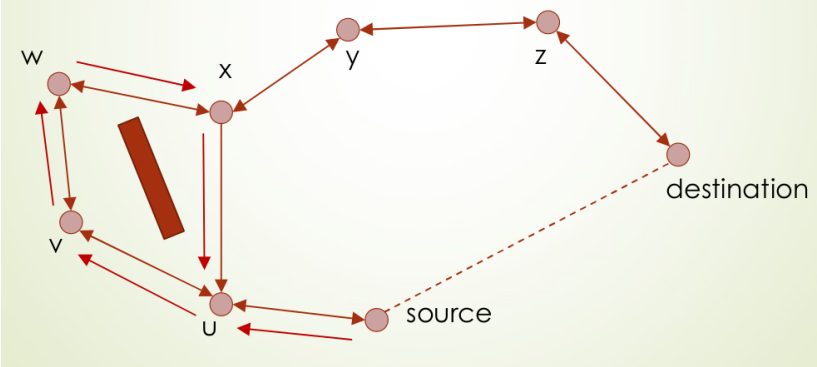


Figure 11.2: Planarization loop with obstacle

In case of an obstacle, the planarization may lead to a loop, as shown in Figure 11.2. TODO explain

Chapter 12

Indoor Localization

We usually take the localization for granted, but it is a very complex problem, especially in indoor environments.

For outdoor is much easier, since we can use GPS.

Relative location, may be turned in **absolute location** if the reference points absolute location is known, or if multiple relative readings and fusion strategy are available.

GNSS (Global Navigation Satellite System) is the most common outdoor localization system, and includes:

- GPS (USA)
- GLONASS (Russia)
- Galileo (EU)
- Beidou (China)

Indoor Localization Systems

Currently this is an open research issue, typically solved by means of ad hoc solutions.

It is incredibly various, in terms of:

- **Signal types**
- **Signal metrics**
- **Metric processing methods**

12.1 Signal Types

- Infrared
- UWB
- Ultrasound
- WiFi

12.2 Signal Metrics

There are three main ranging approaches which exploit different signal metrics:

- **Time** based
- **Angle** based
  These two are the most common, but work only if transmitters and measuring units are in LOS (Line of Sight)
- Received Signal **Indicators** based

12.2.1 ToA - Time of Arrival

The distance between a measuring unit and a mobile target is directly proportional to the propagation time.
Mathematically speaking, being p the propagation speed, t the time of transmission and t' the time of reception, the

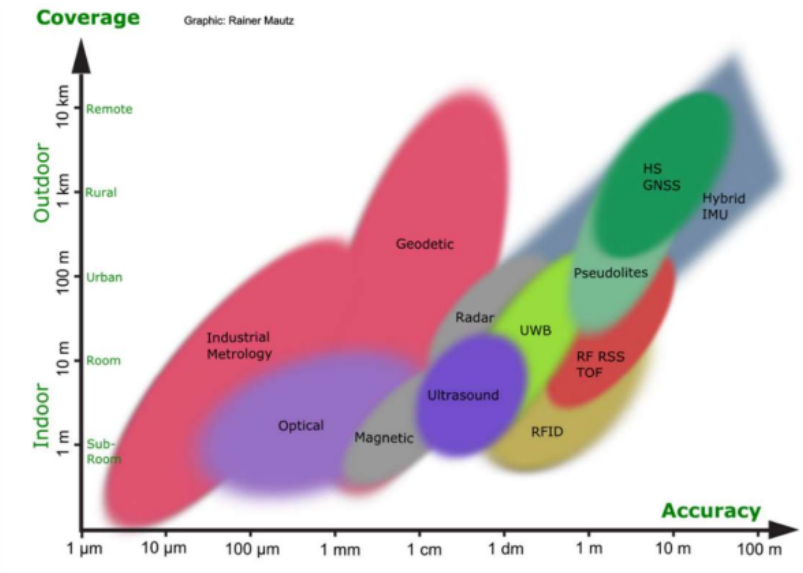


Figure 12.1: Signal types graph

distance d is:

$$d = (t' - t) \cdot p$$

This method requires a tight synchronization of transmitter and receiver, and that the signal encodes the transmission time t.
Besides, to compute the position in a plane (2D), at least 3 anchors —located in different positions— are needed.

In some applications, ToA is implemented by two types of signals, an Acoustic signal and a Radio one. The radio, since it is order of magnitude faster than the acoustic signal, is used to synchronize the measuring units, while the acoustic signal is used to measure the distance.

12.2.2 TDoA - Time Difference of Arrival

The distance between a measuring unit and a mobile target is directly proportional to the difference in propagation time between two signals.

The hyperbolic location theory is: The hyperbola is the set of points at a constant range-difference $c \cdot \Delta t$ from two foci (?). Each sensor pair gives a hyperbola on which the emitter lies. Location estimation is the intersection of all hyperbolas.

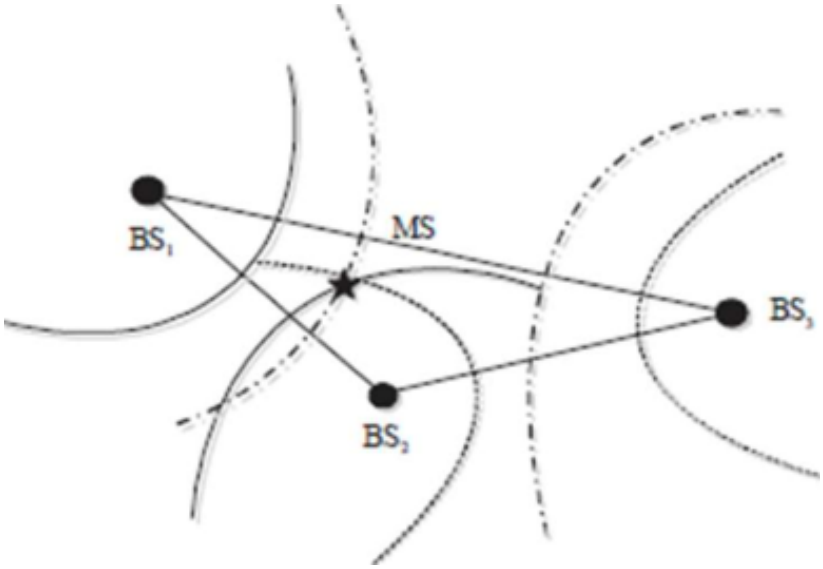


Figure 12.2: TDoA example

12.2.3 RTOF - Round Trip Time of Flight

The transmitter and the measuring unit are the same. The device to be localized is only a transponder, i.e. receives the signal and sends it back.

$$distance = c \cdot (t_1 - t_2)/2$$

This requires much less synchronization compared to ToA, but the signal must be able to be received and sent back.

12.2.4 AoA - Angle of Arrival

There are some antennas which allow to measure the angle of arrival of a signal, and this can be used to estimate the position of the emitter.
2D localization requires at least 2 antennas, while 3D localization requires at least 3 antennas.

AoA is more expensive and usually not available in sensors. Besides, it is very sensitive to the environment, multipath and reflection may affect the measurement.

12.3 Signal Processing

Processing methods may be **Range-free** or **Range-based**.

12.3.1 Range-free

One of the most common range-free methods is the **Centroid**. The idea, is to do not use any ranging at all, but simply deploy enough anchors periodically broadcasting their location. The algorithm is simple and based on listening to anchors, but their positioning is crucial.

DV-Hop is a range-free localization algorithm, which uses the distance between nodes to estimate the position of the nodes.

- ◇ Anchors:
  - Flood network with known positions
  - Flood network with average hop distance
- ◇ Nodes:
  - Count *#hops* to anchors
  - Multiply with avg hop distance

In APIT (Approximate Point In Triangle test) anchors divide the environment into triangular regions: a node's presence inside or outside of these triangular regions allows to narrow the area in which it can potentially reside.

12.3.2 Range-based

Multilateration is a range-based method, which uses the distance between the measuring unit and the anchors to estimate the position of the measuring unit.

The node collects the beacons and estimate its distance to each beacon.

In MIN-MAX a node creates an association between each anchor position and the strength of the RSS from that anchor. By inverting the nominal distance-power loss law, the node may estimate his distance from each beacon

TODO

12.3.3 TODO

filippo.palumbo@isti.cnr.it

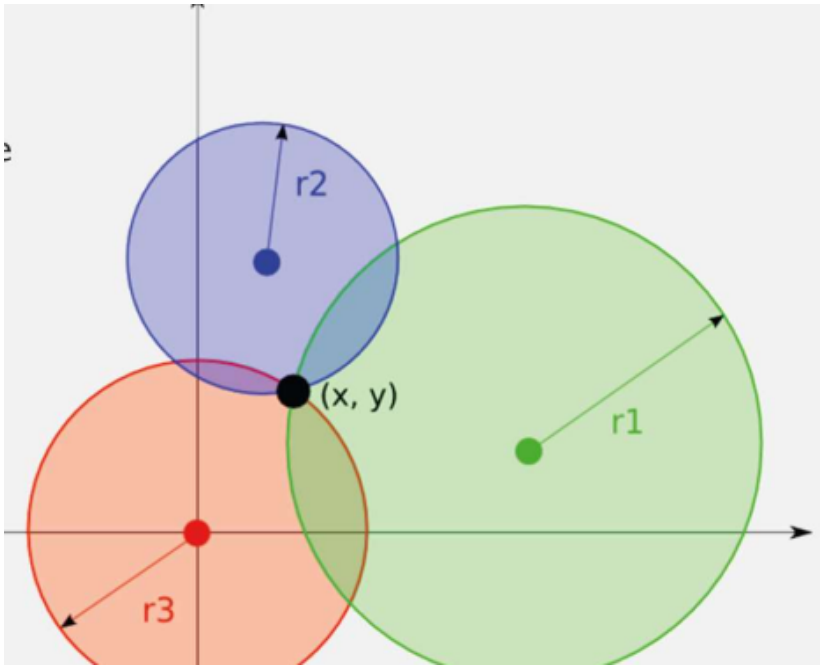


Figure 12.3: Multilateration example

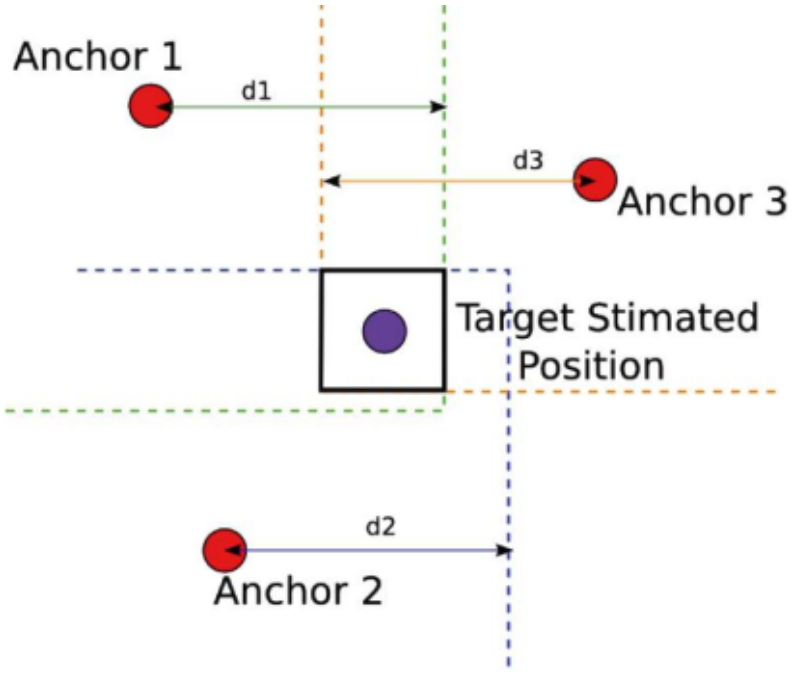


Figure 12.4: MinMax algorithm in the plane

Chapter 13

ZigBee - WSN Case Study

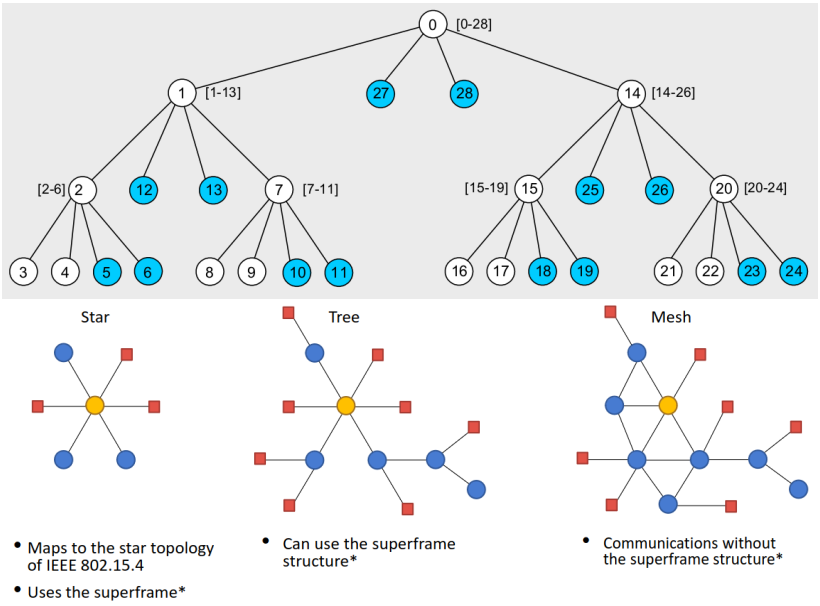


Figure 13.1: ZigBee tree topology example with assigned address ranges. Comparison between star/tree/mesh topologies.

The tree topology is used to assign the short network addresses, the ZigBee coordinator is statically configured with:

- ◇ The maximum number of routers (Rm) each router may have as children
- ◇ The maximum number of end-devices (Dm) that each router may have as children
- ◇ The maximum depth of the tree (Lm).

Each router is assigned with a range of addresses

- ◇ To assign addresses to its children
- ◇ Computed based on Rm, Dm, and Lm.

Devices join as high up the tree as possible, minimizing the number of hops. Although the addresses are assigned according to a tree structure the actual topology can be a mesh.

13.1 Routing

Tree routing is used to route packets between devices that are directly connected. It exploits beaconing to maintain the tree structure.

Mesh routing routing is used to route packets between devices that are not directly connected. When no entry for the destination is in the routing table, *route discovery* is performed. Mesh does *not* allow beaconing.

Tree and mesh topologies may coexist in the same network.

Chapter 14

Bluetooth

Bluetooth is a wireless technology standard for exchanging data over short distances (using short-wavelength UHF radio waves in the ISM band from 2.4 to 2.485 GHz) from fixed and mobile devices, and building personal area networks (PANs). It was originally conceived as a wireless alternative to RS-232 data cables. Bluetooth is managed by the Bluetooth Special Interest Group (SIG), which has more than 35,000 member companies in the areas of telecommunication, computing, networking, and consumer electronics. The IEEE standardized Bluetooth as IEEE 802.15.1, but no longer maintains the standard. The Bluetooth SIG oversees development of the specification, manages the qualification program, and protects the trademarks. A manufacturer must meet Bluetooth SIG standards to market it as a Bluetooth device. A network of patents apply to the technology, which are licensed to individual qualifying devices.

14.1 Architecture

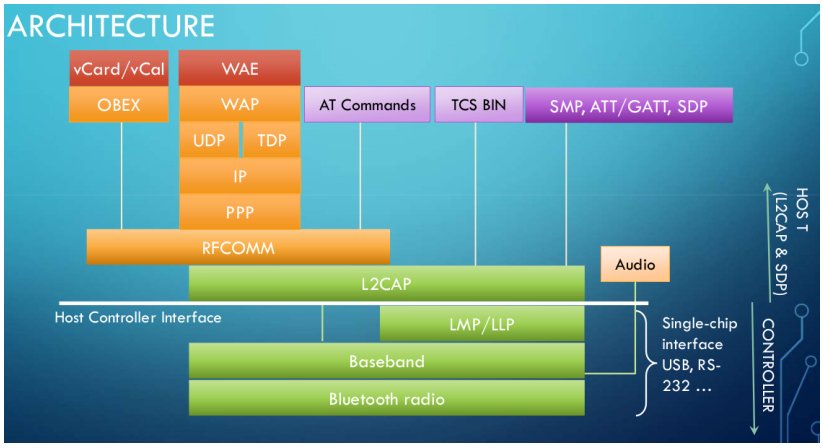


Figure 14.1: Architecture of Bluetooth.

The bluetooth core system is composed of one or more **controllers** and a **host**. The controller is a microcontroller that executes the physical layer and the link layer, while the host is a microcontroller that executes the higher layers of the protocol stack. The host communicates with the controller through a serial interface.
The host comprises all layer below the HCI (Host Controller Interface) shown in Fig. 14.1 , while the controller comprises the HCI and the layers below.

TODO check this

The core protocols are various:

- *Radio*: defines the specification of the radio frequencies.
- *Baseband*: defines the low level procedures of the PHY layer
- *LMP* (link management protocol - for `BR/EDR`) / *LLP* (link layer protocol - for `BLE`): setup and control of the link.
- *HCI* (Host Controller Interface): interface between hardware and software.
- *L2CAP* (Logical Link Control and Adaptation): higher-level protocol multiplexing, packet segmentation and

reassembly, and the conveying of quality of service information.
  - Provides an interface to higher level protocols and applications to transmit and receive data packets, up to 64 kilobytes in length.
  - Implements flow control and retransmission modes.
- ◇ *SDP*: Service Discovery Protocol

14.1.1 BR/EDR vs BLE

BR/EDR (Basic Rate/Enhanced Data Rate), used for streaming audio and other data. It is a point-to-point connection, and can connect up to 7 devices.
It is connection-oriented, meaning that the a link between two devices is kept even if no data is flowing; There are however *sniff modes* which allow devices to sleep and thus to save energy.
Peak transmit current is about $25mA$, which is low compared to other wireless technologies, but still not enough low for very low-power devices or coin cell batteries.

BLE (Bluetooth Low Energy) is a different protocol, designed for ultra low power consumption. It is connectionless, and can connect up to 40 devices.
It exploits short packets to reduce TX peak current and RX time, and less channels to improve discovery and connection time, ultimately resulting in overall lower power consumption, with $20mA$ as peak transmit current, and $6\mu A$ as average current.

BLE operates at 2.4GHz and uses 40 channels with FDMA (Frequency Division Multiple Access), with 2MHz spacing. IChannels 37,38,39 are used for connection, discovery and broadcast, while the others for data. The maximum data rate is $1Mbps$, however this protocol is meant for smaller packets, it is not optimized for file transferring.
Each channel is divided into time units (events), either

- ◇ *Advertising* events
- ◇ *Connection* events

14.2 Network topologies

Two topologies are possible, *Piconet* and *Scatternet*. The Scatternet is a network of Piconets, where a device can be part of multiple Piconets, and thus can act as a bridge between them.

The master is unique in each piconet and synchronizes it, besides it also controls the access to the channel of all slave stations.
In BE/EDR up to 7 slaves can be connected to a master, while in BLE there is no limit, but a device may be in only one piconet.

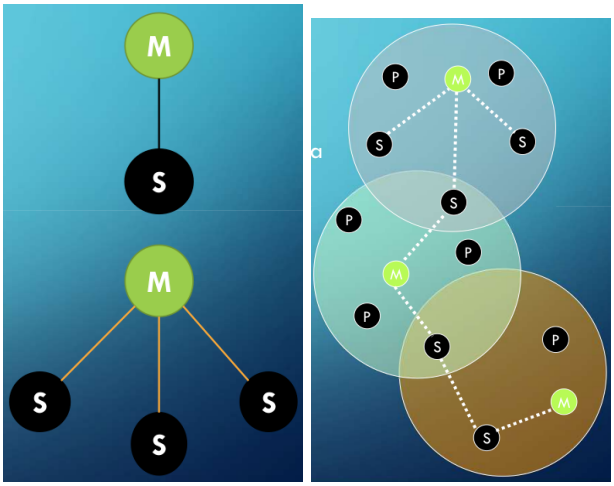


Figure 14.5: Devices sharing the same channel form a *Piconet*, with one Master and —possibly— multiple slaves. Multiple Piconets can be connected to form a *Scatternet*.

14.3 Baseband layer

14.4 Phyisical Layer

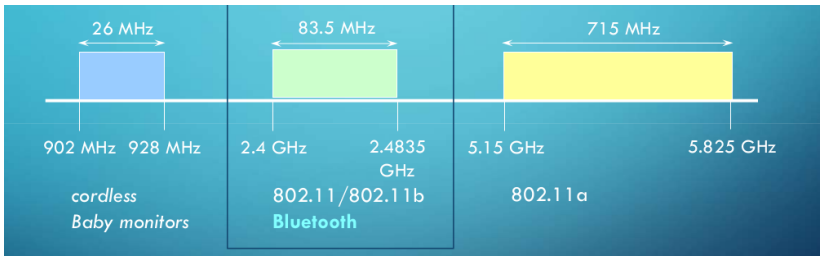


Figure 14.2: Operating bluetooth frequencies

14.5 Substates

The BLE protocol defines many substates, which are used to manage the power consumption of the device. The device can switch between these states depending on the activity it is performing.

Conn Establishment

- ◇ Page
- ◇ Page scan
- ◇ Page response
- ◇ Inquiry
- ◇ Inquiry scan
- ◇ Inquiry response

Conn state

- ◇ Active mode
- ◇ Sniff mode
- ◇ Hold mode
- ◇ Park

Part II

Federica Paganelli

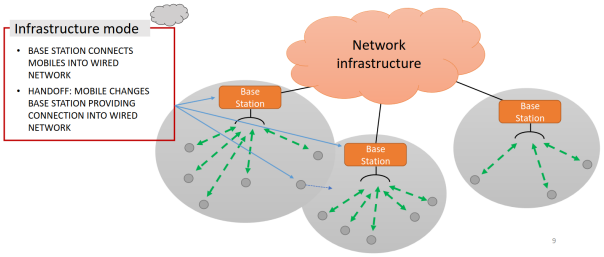
Chapter 15

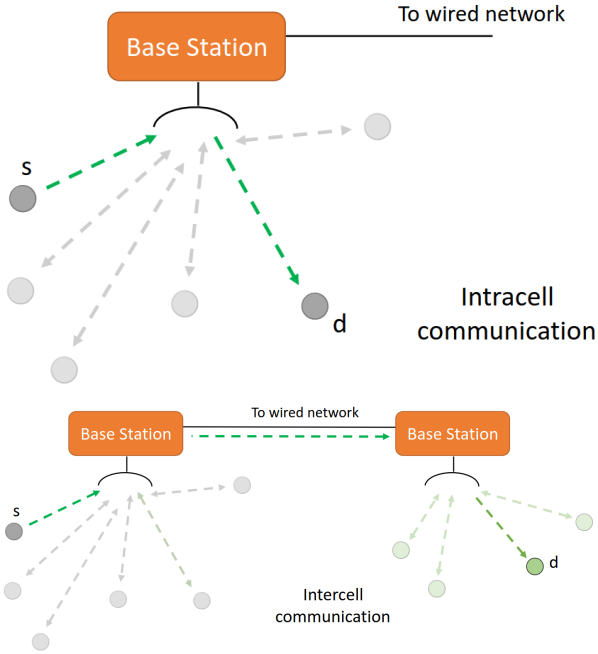
Wireless Networks

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*.





15.1 Link Layer

15.1.1 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

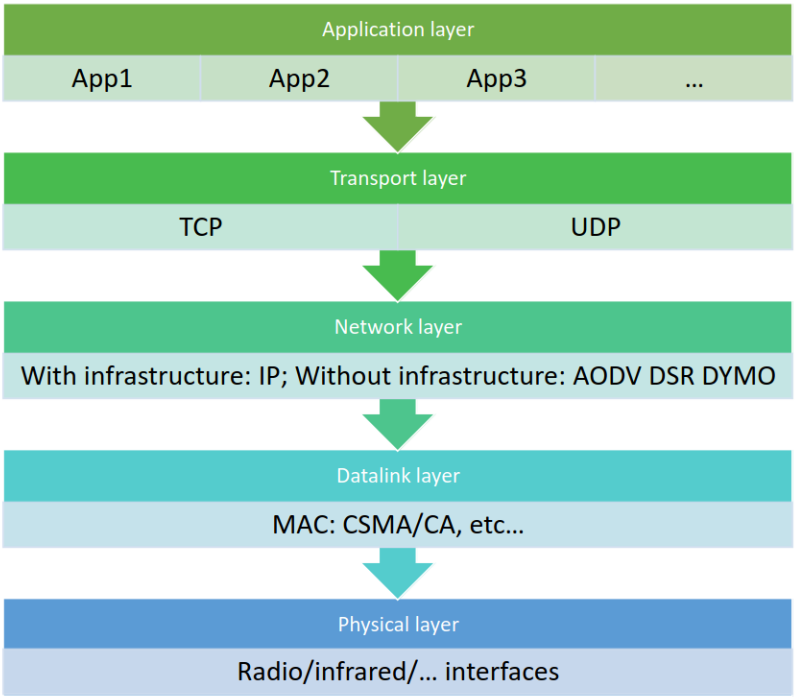


Figure 15.1: Protocol stack

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.

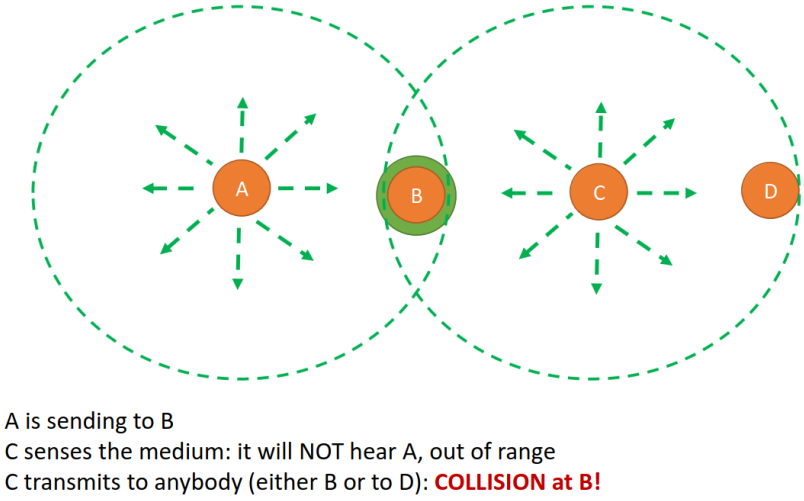


Figure 15.2: **Hidden Terminal** problem
Two or more stations which are out of range of each other transmit simultaneously to a common recipient

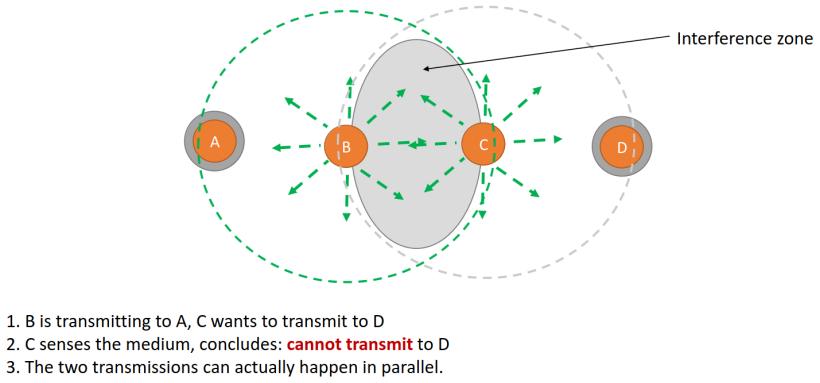


Figure 15.3: **Exposed Terminal** problem
A transmitting station is prevented from sending frames due to interference with another transmitting station

15.1.2 MACA

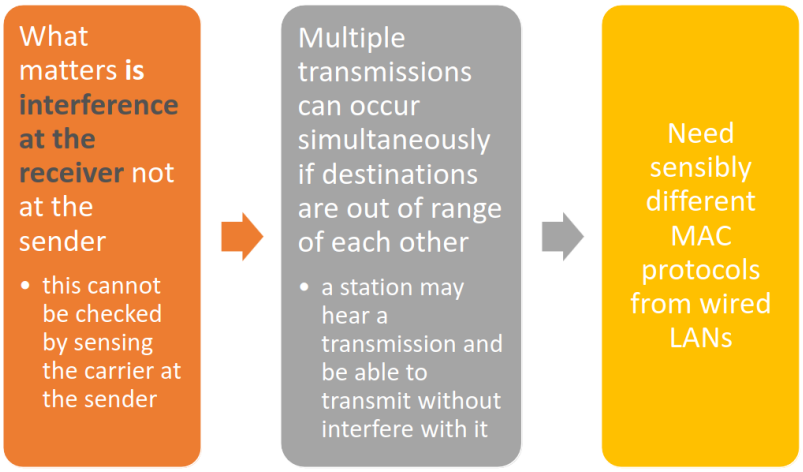


Figure 15.4: MACA Motivations

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 `CTS` must stay silent until the transmission is over. Hearing `RTS` instead does not imply to stay silent, i.e. hearing a `RTS` does not forbid to transmit stuff to other nodes.

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 16

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 16.1: IEEE 802.11 standards

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

IEEE 802.11 standards refer to the *Physical* Layer. In case of an AP (Access Point), every station must channel its communication through the AP to talk to any other. Otherwise, in an *Ad-Hoc* network, stations can communicate directly with each other.

TODO

TODO non ci sono domande su questa cose nel pdf

Chapter 17

Mobile Networks

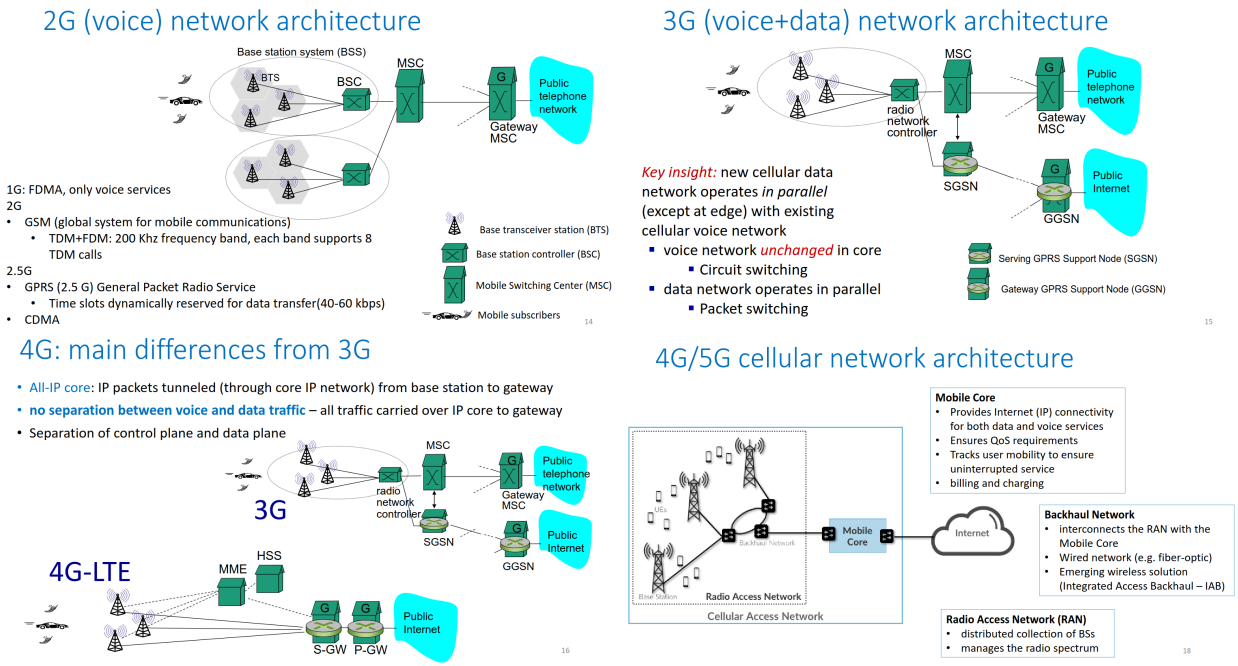


Figure 17.1: Mobile Networks architectures

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.

17.1 4G Focus

- ◇ Mobile End devices
- ◇ Base station
  - – Creates device-specific IP tunnels from the mobile device to gateways
  - – At the edge of carrier's network
- ◇ Home Subscriber Server (HSS)
  - – A database that contains all subscriber-related information
  - – Stores info about mobile devices for which the HSS's network is their "home network"
  - – Works with MME in device authentication
- ◇ Mobility Management Entity (MME)
  - – Device authentication (device-to-network, network- to-device) coordinated with mobile home network HSS
  - – Mobile device management:
    - ○ Device handover between cells
    - ○ Tracking/paging device location
  - – Path (tunneling) setup from mobile device to P-GW

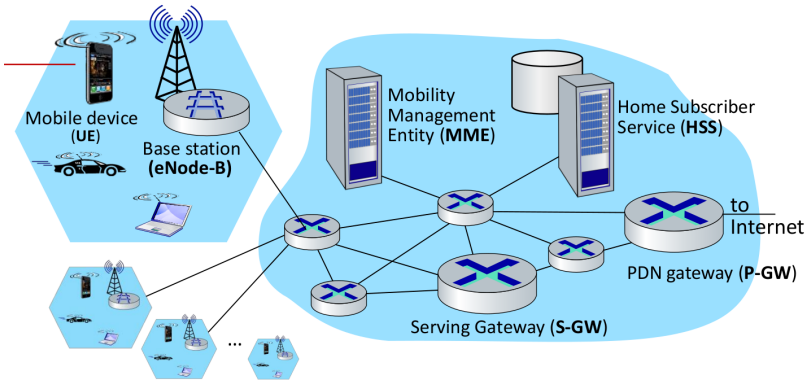


Figure 17.2: 4G architecture key elements

- ◇ Serving Gateway (SGW)
  - Forwards data packets between to and from RAN (Radio Access Network)
  - Responsible for inter-enB handover, and provides mobility between 4G and other networks such as between 2G/3G and PG-W
- ◇ Packet Data Network Gateway (PGW)
  - A gateway to mobile cellular network
  - Looks like any other internet router and provides NAT services

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

17.2 Mobility

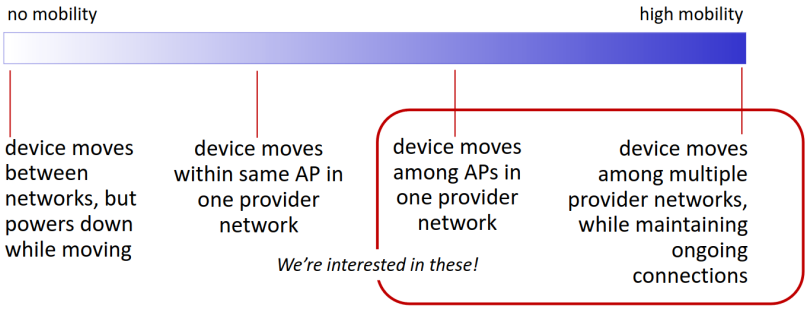


Figure 17.3: Mobility

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

With respect to the questions posed in the third image in Fig. 17.4, data being send from a device to a mobile one may be routed in three ways.

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

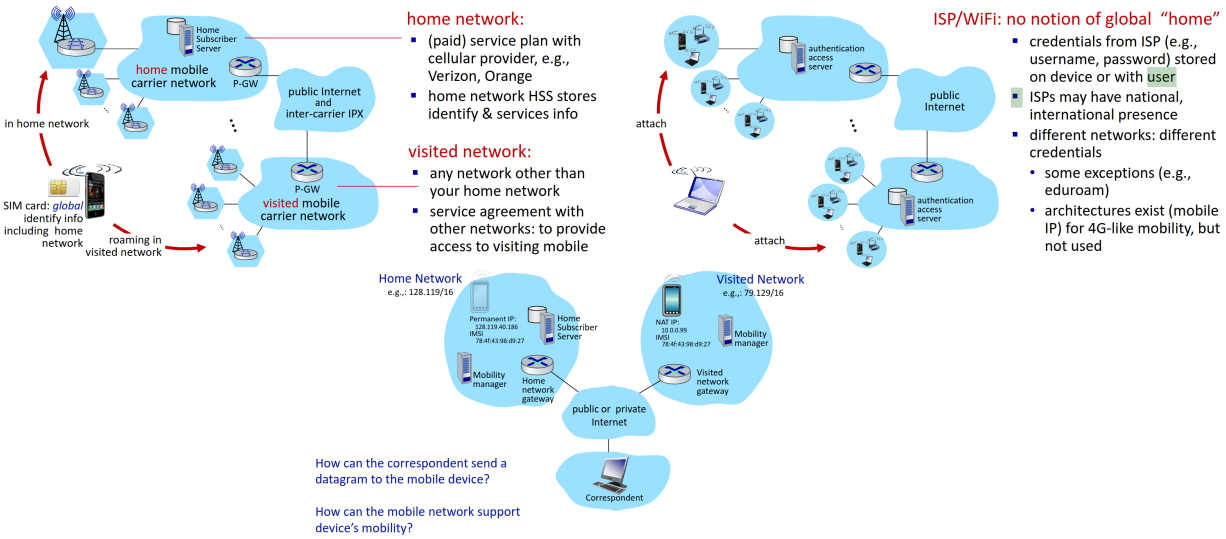
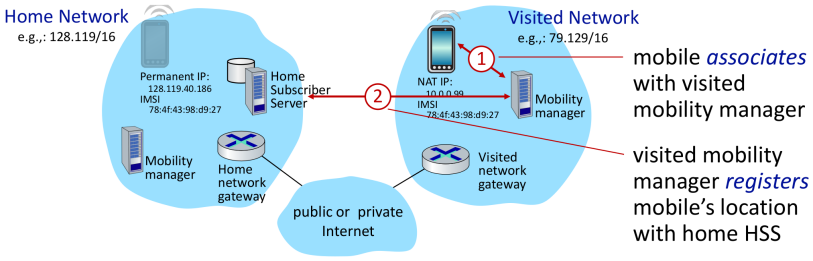


Figure 17.4: Home and visited Networks

ii. **Indirect routing**
Communication from sender to mobile goes through home network, then forwarded to remote mobile

We will focus on the second possibility, which is the most common in mobile networks. First of all the home network needs to know the current location of the mobile device, so first it associates with the Mobility Management Entity (MME) of the visited network to authenticate the device, which in turn informs the Home Subscriber Server (HSS) of the device's location.



17.2.1 Indirect Routing

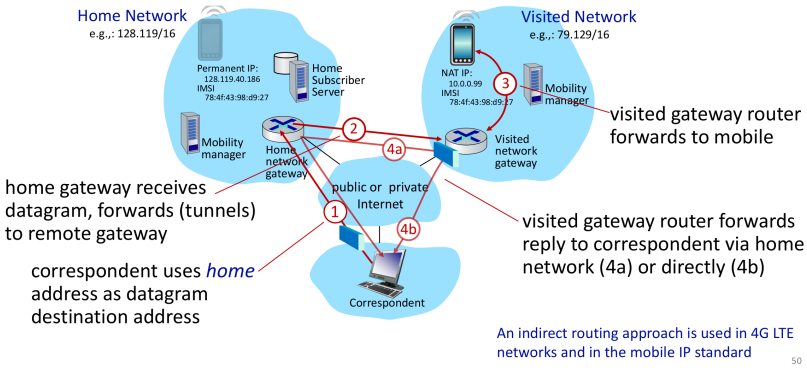


Figure 17.5: Indirect routing schema

Once the mobile's location is known, the schema is fairly simple: the sender sends the packet to the home network, which forwards it to the visited network, which in turn delivers it to the mobile device, lastly, the packet will be sent back to the sender, either directly or by using the same path.

To implement such solution, the requirements are:

- ◇ Protocol to associate and disassociate a mobile to a visited network
- ◇ A protocol to Register the mobile's location in the HSS

- ◇ Tunnelling protocol to forward packets between the home and visited networks
  - The home gateway performs encapsulation and forwarding of the correspondent's original datagram
  - On the receiving side, the gateway router performs decapsulation, NAT translation, and forwarding of the original datagram to the mobile device

Indirect Routing is inefficient when mobile and sender are in the same network, but allows the mobility of the device to be **transparent** to the sender: ongoing connections (e.g. TCP) are not interrupted and can be maintained.

17.2.2 Direct Routing

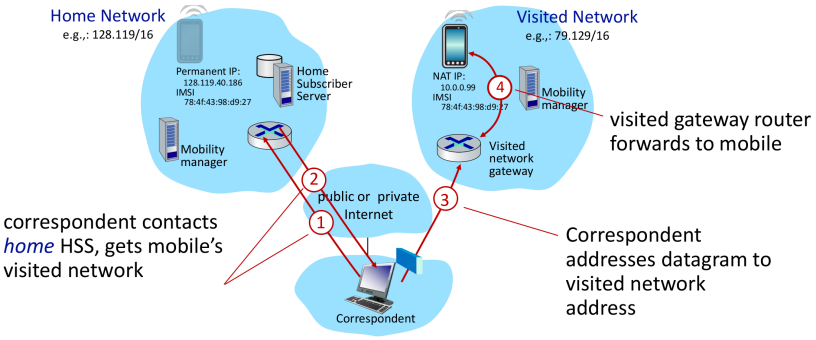


Figure 17.6: Direct routing schema

As displayed in Fig. 17.6, the main difference here is that the sender sends the packet directly to the mobile device, after receiving the foreign address from the home network.

This approach overcomes the inefficiency of the indirect routing, but it requires the sender to know the mobile's location, which may change multiple times, requiring the sender to ask the home network for the new location.

17.2.3 Key-tasks in 4G Mobility

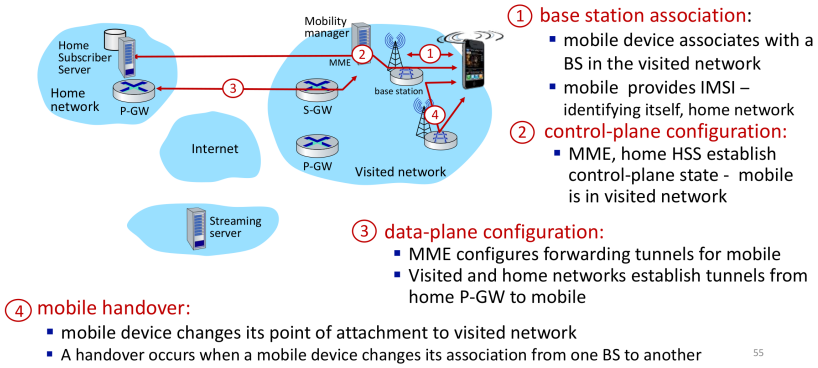


Figure 17.7: Mobility tasks

17.3 Handover in the same cellular network

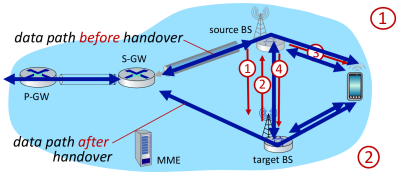


Figure 17.8: Handover pt. 1

1. Current BS decides to perform handover and selects a *target BS*, to whom sends a `handover request`.
   - ◇ May happen due to a change in the signal strength, cell overloading or whatsoever
   - ◇ Note that it's the BS that decides to perform the handover, not the mobile device
   - ◇ *On which criteria is the target BS selected?*
2. Target BS preallocates radio time slots and resources for the mobile device and sends a `handover ACK` to the current BS, appending info for the mobile device.
3. Source BS notifies the device of the new BS using the information received from the target BS.
   From the mobile device's perspective, the handover is not transparent, but it gets completed with this

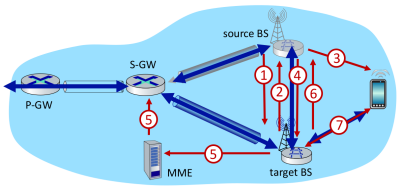


Figure 17.9: Handover pt. 2

third step.

4. Source BS stops sending datagrams to mobile, and forwards incoming ones to the target BS.

5. Target BS informs the MME that it is the new serving BS for the mobile device, which in turn informs the SGW to update the path to the mobile device.
6. Now target BS sends an `ACK` to the source BS, which can release resources and stop forwarding packets.
   - ◇ Shouldn't the target BS wait for an ACK by the MME?
7. Datagrams now flow through the new channel from target BS to SGW.

Chapter 18

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

18.1 Motivations and 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

Increasing use of cloud shifted traffic relegated in enterprise networks to remote clouds, causing unpredictable loads on enterprise routers.

Server virtualization, now a common practice, besides increasing the number of virtual machines (VMs) on a single physical server and corresponding communications, brings east-west traffic due to migration and remote storage access.

Summing up, up to 70% of traffic in datacenters is now East West, and traditional three-tier *access-aggregation-core* network architecture no longer fits the bandwidth traffic requirements.

18.2 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

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

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.

18.2.1 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
  Traditionally it was implemented *per-router*, with SDN instead is *logically centralized*

These are the two key tasks. In Fig. 18.1 a comparison between the tradional and the SDN approaches. In Internet

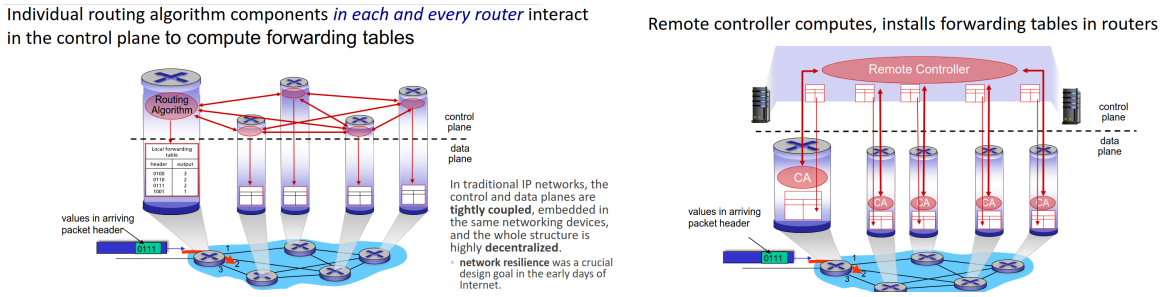


Figure 18.1: Traditional vs SDN

early days, **resiliency** was a key issue, so the network was designed to be decentralized and distributed, with data and control tightly coupled and embedded in the same —routers— networking devices.
However, this design does not allow to arbitrarily choose the traffic paths with ease: the only way to do so is to change the link weights, and waiting for a recomputation of the routing tables, which is somewhat far from being considered "control".

18.3 SDN Architecture

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 (`OpenFlow`). 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

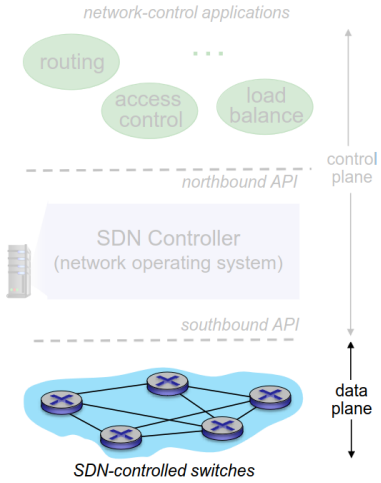


Figure 18.2: SDN Layer Architectures

18.3.1 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**. The pattern implemented is called *"match+action"* and is used to define the rules for packet processing. For each packet bits are analysed and compared to the rules in the flow table, when a match is found, the corresponding action is taken.

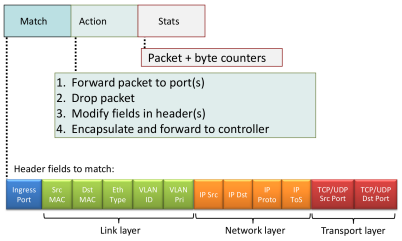


Figure 18.3: OpenFlow table

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`

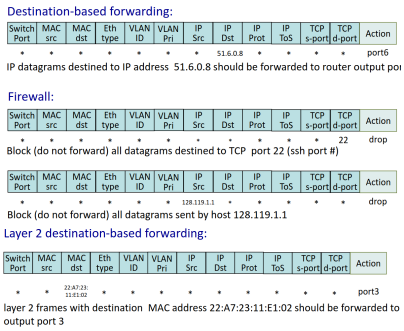


Figure 18.4: OpenFlow rules examples

to enable the

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

18.3.2 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.

Through OpenFlow, the controller can add, update, and delete flowentries 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 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. A packet may also be redirected to controller, enabling it to define a new flow for that and similar packets, or decide to drop the packet

18.4 Control Plane

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

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

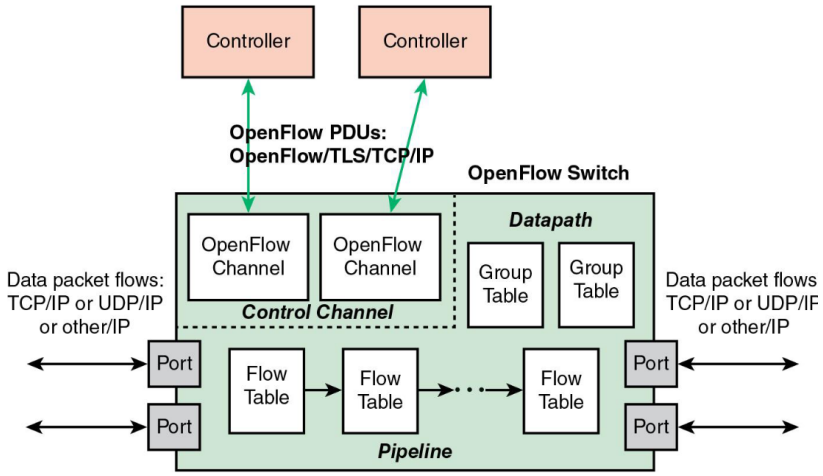


Figure 18.2: OpenFlow switch

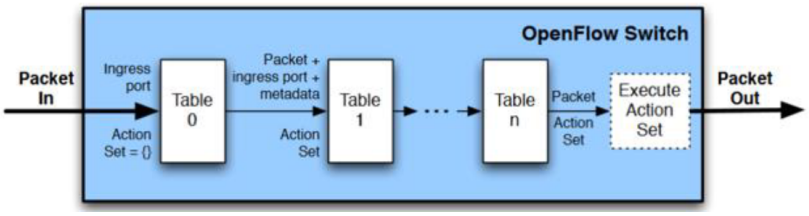


Figure 18.3: Multiple tables inside switch

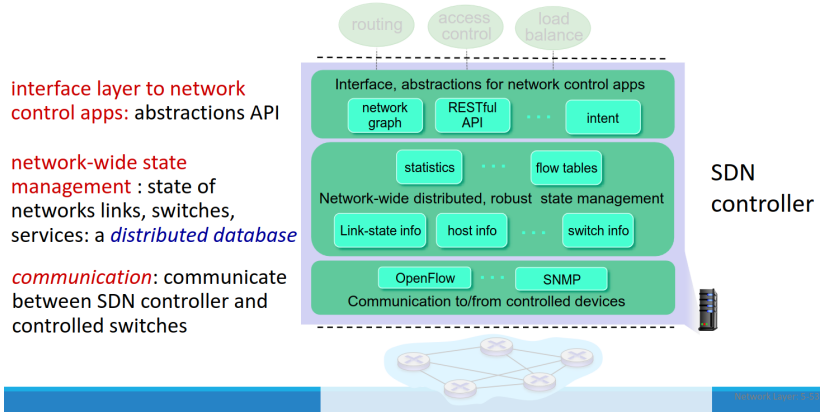
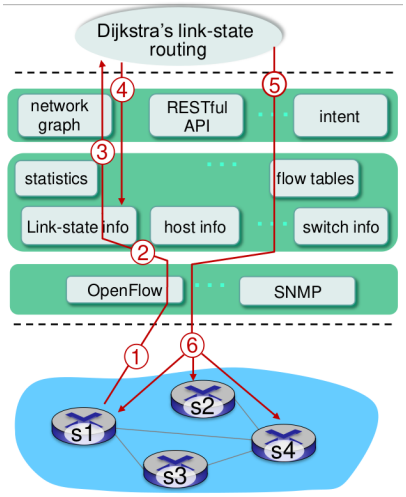


Figure 18.4: Inside an OpenFlow controller

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.)

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

18.4.1 Link Failure scenario



1. S1 detects link failure and uses OpenFlow port status message to notify controller.
2. SDN controller receives OpenFlow message, and updates link status info.
3. Dijkstra's routing algorithm gets called in this case
4. The algorithm accesses enetwork graph info, link state info in controller, and computes new routes.
5. Link state routing app interacts with flow-table-computation component in SDN controller, which computes the new flow tables for switches.
6. The controller sends OpenFlow messages to install new tables in switches that need updating.

Figure 18.5: S1 loses the link (not displayed) with S2

18.5 Topology discovery and forwarding in the SDNs

18.5.1 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 host)

18.5.2 Topology Discovery

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.

18.5.2.1 LLDP message content

- ◇ `Chassis ID`: identifier of switch sending the packet
- ◇ `Port ID`: identifier of the port through which the LLDP packet is sent
- ◇ `TTL`: time in seconds during which the information received in the LLDP packet is going to be valid.
- ◇ End of `LLDPDU`: indicates the end of the payload in the LLDP frame

18.5.2.2 Discovery process

1. The controller generates a Packet-Out message per active port on each switch discovered on the network and encapsulates a LLDP packet inside each generated message.
2. When an OF switch receives a LLDP message sent by the controller, it forwards the message by the appropriate Port ID included in the message to adjacent switches
3. Upon receiving the messages by a port that is not the controller port, the adjacent switches encapsulate the packet within a Packet-In message addressed to the controller. Metadata is included in the message such as Switch ID, Port ID where the LLDP packet is received, among others
4. When the packet comes back to the controller from an adjacent switch, the controller extracts this information, and then it knows a link exists between those switches
   Switch ID and Port ID in the LLDP message and Switch ID and Port ID in metadata

This process is repeated for each switch.

18.6 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 19

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.

19.1 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** (`VNF`s). 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. Reduced 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

19.2 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.

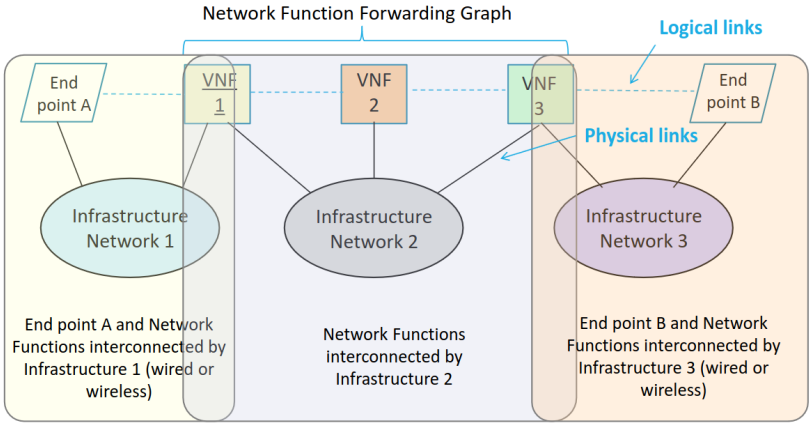


Figure 19.1: VNF Network Service

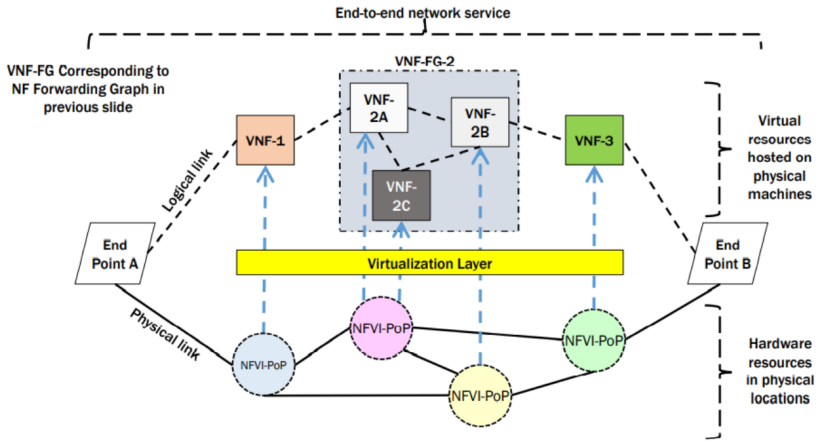


Figure 19.2: VNF more complete example

As displayed in figure a single PoP may run multiple VNFs. Logical links between VNFs may be managed using SDN, and must be backed by physical links in the underlying network infrastructure.

19.3 NFV Architectural Framework

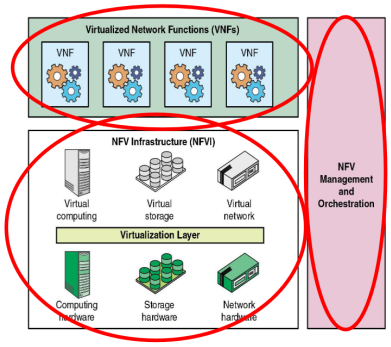


Figure 19.3: NFV Architecture

- **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. 19.4 are:

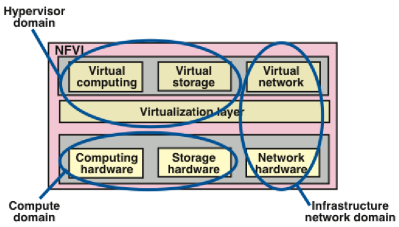


Figure 19.4: NFV Infrastructure

- **Compute Domain**
  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

19.3.1 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).

19.3.2 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.

19.4 NFV MANO

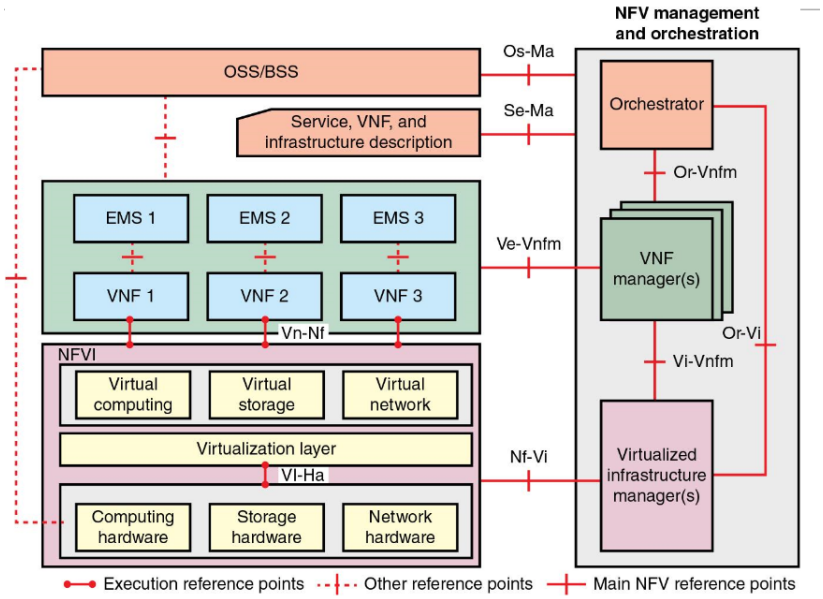


Figure 19.5: NFV Reference - Architectural Framework
Here VNF1,2,3 are VNF instances, EMS1,2,3 are Element Management Systems, and OSS/BSS is the traditional network management,

- 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.

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 20

NFV/SDN Emulation tools

20.1 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`

Chapter 21

Signals

Definition 21.1 (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

21.1 Signal types

Definition 21.2 (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

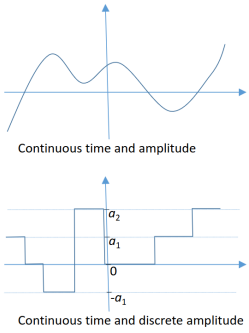


Figure 21.1: Continous signals

Definition 21.3 (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.

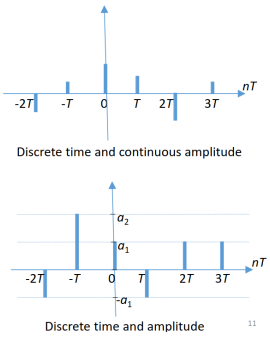


Figure 21.2: Discrete signals

Definition 21.4 (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:*

00000000010000000001001011011101001001 00

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 $f_c \cdot Mbit/second$*

21.2 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

21.2.1 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

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

21.3 Digital transmission - sampling & quantization

Definition 21.5 (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

21.4 Periodicity and Fourier series

Definition 21.6 (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 21.7 (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.

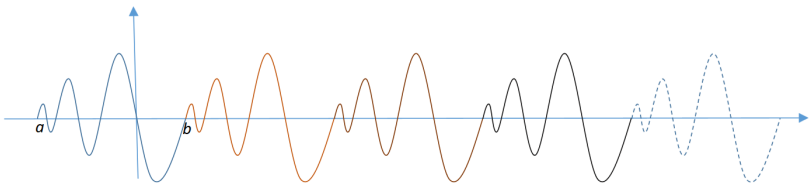
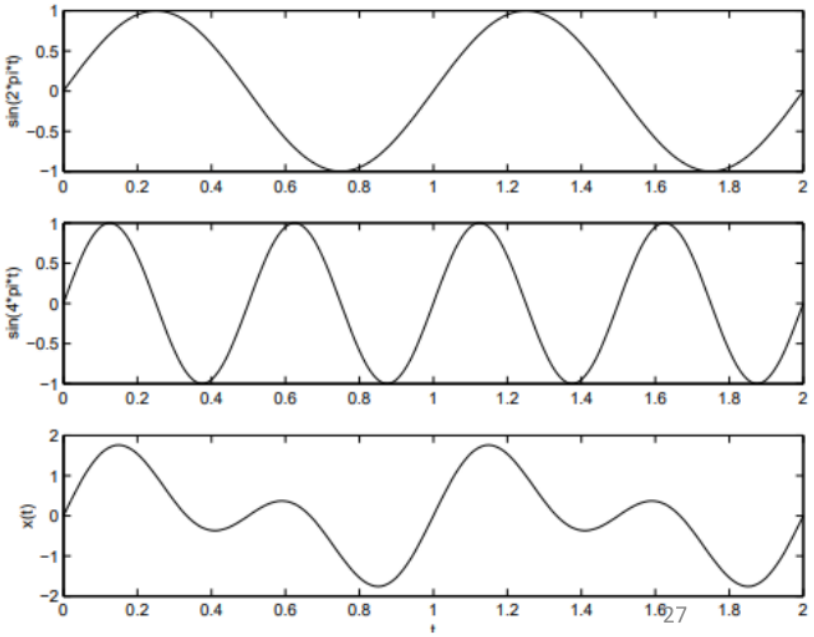


Figure 21.1: Repeating an aperiodic signal results in a periodic one

21.4.1 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 21.8 (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 21.1 (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.

21.5 Energy and Power signals

Definition 21.9 (Energy) *The **energy** E of a signal $s(t)$ is given by:*

$$E(T) \triangleq \int_{-\frac{T}{2}}^{+\frac{T}{2}} |s(t)|^2 dt \tag{21.1}$$

The signal has *finite energy* (is an **energy signal**) if the limit of $E(T)$ as T tends to infinity is finite:

$$0 < \lim_{T\to\infty} E(T) < +\infty \tag{21.2}$$

$$0 < E_s = \int_{-\infty}^{+\infty} |s(t)|^2 dt < +\infty \tag{21.3}$$

Even infinite signals may have finite energy. In the physical world, all signals have finite energy.

Definition 21.10 (Power) *The **power** P_f of a signal $s(t)$ is given by:*

$$P_f(T) \triangleq \frac{1}{T}\int_{-\frac{T}{2}}^{+\frac{T}{2}} |s(t)|^2 dt = \frac{E_s(T)}{T} \tag{21.4}$$

The signal has *finite power* (is a **power signal**) if the limit of $P(T)$ as T tends to infinity is finite:

$$0 < P_s = \lim_{T\to\infty} \frac{1}{T}\int_{-\frac{T}{2}}^{+\frac{T}{2}} |s(t)|^2 dt < +\infty \tag{21.5}$$

Periodic signals have some peculiarities in terms of power and energy:

- They have infinite energy
  - They are *power signals*, not energy signals
- Their average power equals the average power computed in a single period

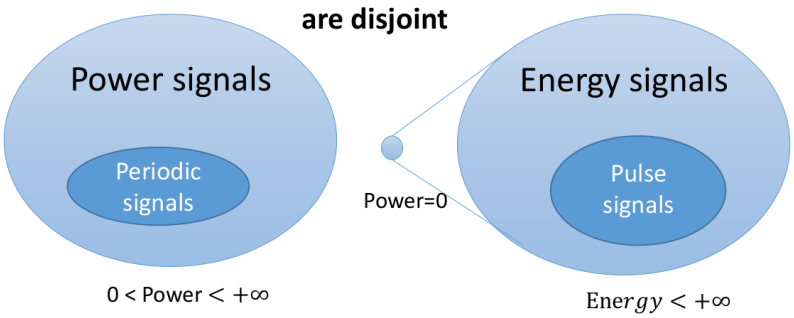


Figure 21.2: Power and energy signals classes

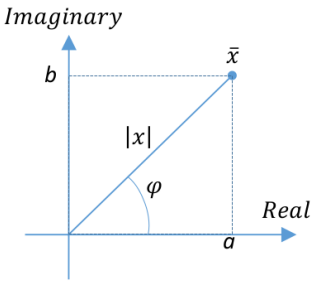
1. A signal may be an energy signal, a power signal, or none
2. A signal cannot be both an energy signal and a power signal

21.6 Fourier Transform

21.6.1 Preliminaries

Definition 21.11 (Complex numbers) *A complex number $\bar{x}$ is a number of the form $x = a + jb$, where a and b are real numbers and j is the imaginary unit, the square root of -1.*

It may be represented in the imaginary plane as a vector with modulus $|x| = \sqrt{a^2 + b^2}$ and phase Φ



- $\bar{x} = a + jb$
- $|x| = \sqrt{a^2 + b^2}$;
- $\varphi = \tan^{-1}\frac{b}{a}$
- $a = |x| \cos\varphi$
- $b = |x| \sin\varphi$

and also:

- $\bar{x} = |x| \cos\varphi + j|x| \sin\varphi$

Figure 21.3: Complex number in the imaginary plane

Definition 21.12 (Euler's exponential) *The exponential $e^{j\theta}$ is a complex number that can be represented in the imaginary plane as a vector of modulus 1 and phase θ. It is defined as:*

$$e^{j\theta} = \cos(\theta) + j\sin(\theta) \tag{21.6}$$

From which we can derive Euler's formulas:

$$\cos(\theta) = \frac{e^{j\theta} + e^{-j\theta}}{2} \tag{21.7}$$

$$\sin(\theta) = \frac{e^{j\theta} - e^{-j\theta}}{2j} \tag{21.8}$$

A complex number may also be written as $x = |x|e^{j\Phi}$, where $|x|$ is the modulus and Φ is the phase. The conjugate of $\bar{x}$ may also be written as $\bar{x} = |x|e^{-j\Phi}$

The Fourier series is usually expressed in the complex number domain, to represent signals $s(t) : \mathbb{R} \to \mathbb{C}$

The **base of Fourier** is the set of functions:

$$e^{j2\pi nFt} \quad \forall n \in \mathbb{Z} \tag{21.9}$$

Considering also the definition of the exponential e^{x+jy}, we get the last equation:

$$e^{x+jy} = e^x e^{jy} = e^x(\cos(y) + j\sin(y)) \tag{21.10}$$

$$e^{j2\pi nFt} = \cos(2\pi nFt) + j\sin(2\pi nFt) \tag{21.11}$$

Definition 21.13 (Fourier series with exponential base) *Given a periodic signal $s(t)$ with frequency $F = 1/T$, we can represent it as a linear combination*

$$s(t) = \sum_{n=-\infty}^{+\infty} S_n e^{j2\pi nFt} \tag{21.12}$$

Where S_n is the —Fourier— series:

$$S_n = \frac{1}{T}\int_0^T s(t)e^{-j2\pi nFt}dt \tag{21.13}$$

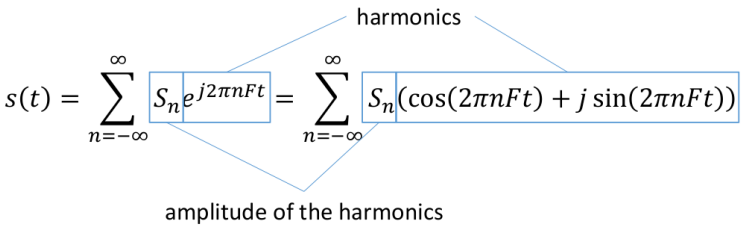


Figure 21.4: Fourier Geometrical interpretation

Definition 21.14 (Signal Spectrum) *The **spectrum** of a signal is the ordered set of its Fourier coefficients S_n from $n = -\infty$ to $n = +\infty$.*

The spectrum may be seen as a discrete signal

Definition 21.15 (Fourier Transform) *The transition from the continuous signal $s(t)$ to its spectrum i.e. to its discrete signal S_n is called the **Fourier Transform**, denoted in the following ways*

$$s(t) \overset{FT}{\Longleftrightarrow} S_n \tag{21.14}$$

$$S_n = \mathcal{F}(s(t)) \qquad \textit{transform} \tag{21.15}$$

$$s(t) = \mathcal{F}^{-1}(S_n) \qquad \textit{inverse transform} \tag{21.16}$$

S_n may be also represented using the Euler's exponential, with the first term being the *amplitude* of the harmonics, and the e the *phase* of harmonics:

$$S_n = |S_n|e^{j\theta_n} \tag{21.17}$$

21.7 From FT to Discrete FT

Non ho capito molto di questa cosa

In the field of digital signal processing, signals and spectrum are processed only in sampled form.

- ◇ We have a signal s known only at a finite number instants separated by sample times (i.e. a finite sequence of data)
- ◇ We want to compute a finite number of samples of the Fourier Transform so that...
- ◇ Discrete-time data sets are converted into a finite discrete-frequency representation

Discrete time signals are typically obtained from continuous signals with a domain restriction from $\mathbb{R}$ into $\mathbb{Z}(T)$. This operation, called **sampling**, is stated by the simple relationship $sc(nT) = s(nT), nT \in \mathbb{Z}(T)$ Where $s(t), t \in \mathbb{R}$, is the reference continuous signal, while $s(nT)$ is the corresponding sampled discrete signal.

Definition 21.16 (Discrete signal) *A discrete signal is a complex function of a discrete variable:*

$$f(t) = \mathbb{Z}(T) \to \mathcal{C}$$

Where $\mathbb{Z}(T)$ is the subset of $\mathbb{Z}$ made up only of $T > 0$ multiples $\cdots -T, 0, T, 2T$

TODO altre equazioni complicate

Part III

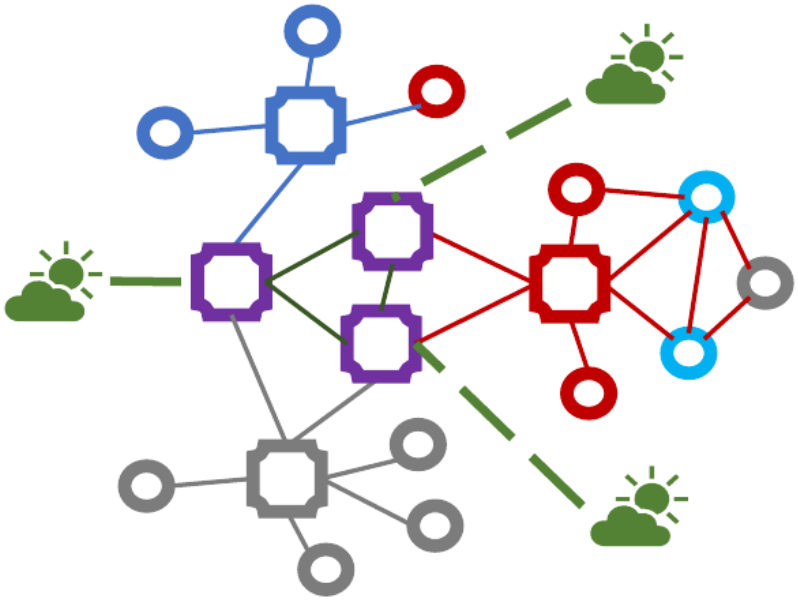
Exam Questions

Francesco Lorenzoni
Emiliano Sescu

Chapter 22

Risposte orale Chessa

22.1 Interoperabilità di reti IoT



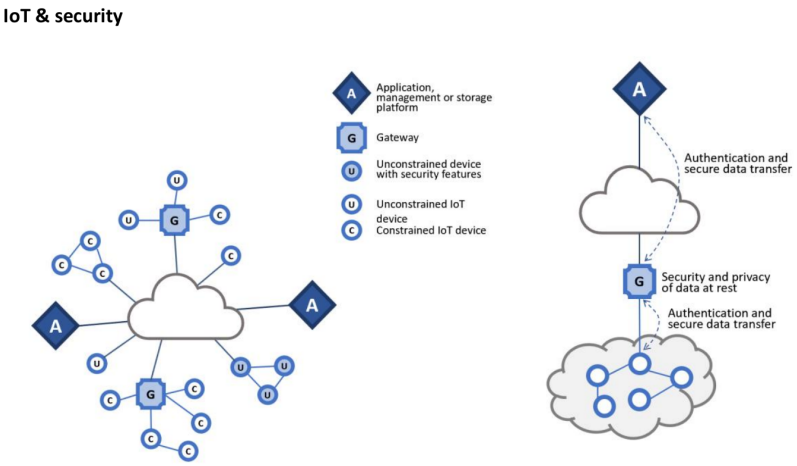
Un problema comune nelle reti di device IoT sono i *Vertical Silos*, business model adottato da diversi vendor, che porta le soluzioni IoT ad avere vari **vendor lock-in**, fra cui la limitatezza dei protocolli utilizzabili dai dispositivi prodotti.
Il problema è parzialmente risolto dagli **standard** di comunicazione, che tuttavia possono comunque essere "troppi" in una rete ampia, creando la necessità di tradurre da uno all'altro per permettere l'interoperabilità fra dispositivi di vendor diversi.
Primo argine al problema sono i **service gateway** (azzurro, rosso e grigio in figura), che permettono agli end-device di comunicare con internet (o altri gateway come in figura) e ad internet di comunicare con essi attraverso un endpoint unificato.
I dispositivi sono partizionati *non* in base al vendor, bensì al protocollo di comunicazione che utilizzano.

Quella rappresentata in figura è una rete formata da **integration gateway** distribuiti che eseguono un mapping dai protocolli usati da end-device e service gateway ("non-viola" in figura) a un linguaggio intermedio utilizzato per le comunicazioni fra di essi, e poi nuovamente mappato in un altro protocollo. Questo permette di avere $2n$ mapping piuttosto che $n \times n$.

Nella figura sembra che utilizzino lo stesso protocollo "verde" per comunicare con internet, ma non è necessario.

All'interno di reti composte da device appartenenti a vendor (costruttori) diversi, che utilizzano protocolli diversi, è necessario l'utilizzo di un integration gateway per permettere l'interoperabilità tra i vari device. Mentre tradurre la serializzazione di un oggetto può essere vista come una funzione bigettiva/invertibile, per, ad esempio, richieste e risposte la traduzione da e verso il protocollo intermedio può richiedere operazioni differenti, da cui la necessità di $2n$ mapping, n *protocolli* $\longrightarrow$ *lang intermedio* e viceversa.

22.2 Sicurezza nei sistemi IoT



Il tema della sicurezza in ambito IoT è un tema molto importante, in quanto per loro natura molte volte i dispositivi IoT sono sprovvisti di feature di sicurezza. Questo problema deriva dal fatto che i costruttori di questi device danno priorità al time-to-market e al minimizzare il costo di produzione, a scapito della sicurezza: le performance sono affette negativamente da essa, così come la durata della batteria. Spesso il sistema operativo installato è "leggero" e quindi privo di quelle funzionalità di security integrate all'interno di un sistema operativo full-fledged; inoltre raramente vengono realizzate patch di sicurezza per i dispositivi IoT, e anche in tal caso è difficile applicarle.

Per questo motivo sono stati standardizzati dall'ITU-T nel documento `Y.2066` dei *requisiti di sicurezza* che riguardano diversi ambiti, tra i quali: comunicazioni, mutua autenticazione e autorizzazione, gestione dei dati, fornitura dei servizi, integrazione di politiche e tecniche di sicurezza e infine di security audit. Tuttavia, non viene specificato come implementare soluzioni che soddisfino tali requirements.

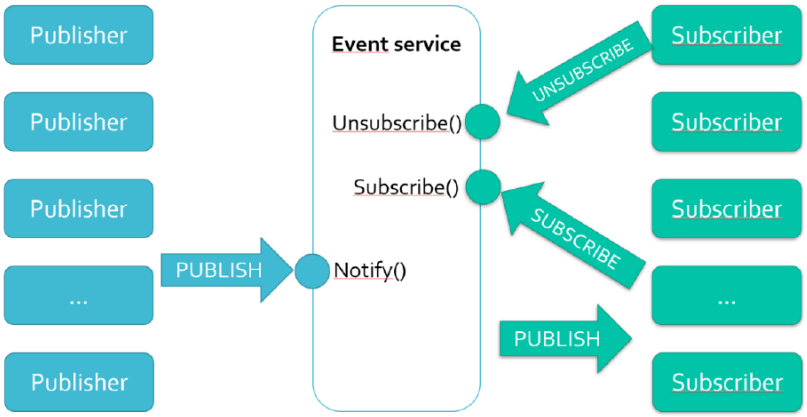
Il focus dell'immagine è quello di mostrare quanto sia difficile garantire la sicurezza di una rete IoT, soprattutto in prossimità dei punti di accesso alla rete, questo dovuto anche al fatto che i dispositivi connessi sono di diversi tipi, esistono infatti dispositivi *constrained* con limitate capacità computazionali e sprovvisti di ogni funzionalità di sicurezza, dispositivi *uncostrained* con un'alta capacità computazionale ma senza funzionalità di sicurezza e altri dispositivi *uncostrained* che però implementano funzionalità di sicurezza.
L'autenticazione mutua o one-way è fornita dai gateway. La sicurezza dei dati che si pone su tre livelli diversi:

1. i dati salvati nei gateway e sui dispositivi;
2. i dati trasferiti tra gateway e dispositivi;
3. i dati trasferiti tra gateway e applicazioni.

Tali requisiti di sicurezza diventano però difficili da soddisfare quando parliamo di constrained device, i quali raramente possono gestire encryption e autenticazione, portando a problemi di privacy sui dati, soprattutto riguardo i dispositivi nelle case. È il caso dei dispositivi "c" in figura esposti direttamente a internet.

Possibile esempio climatizzazione, luci, frigo ecc... Oppure wristband e heart rate

22.3 Publish/subscribe e MQTT



MQTT è un protocollo di trasporto di messaggi affidabile e leggero basato sul modello publish/subscribe, un modello di comunicazione asincrona basato su eventi in cui sono coinvolte tre tipologie di entità:

- ◇ il message broker, o event service: funziona come un server che gestisce lo scambio dei messaggi basato su topic;
- ◇ i publisher: funzionano come client che pubblicano messaggi sul message broker marcandoli con un determinato topic;
- ◇ i subscriber: funzionano come client e si sottoscrivono ad un determinato topic sul message broker.

I client devono conoscere l'IP/hostname del broker *beforehand*, essendo MQTT basato su TCP/IP. I client sono identificati dal broker attraverso un *Client ID* che deve essere univoco.

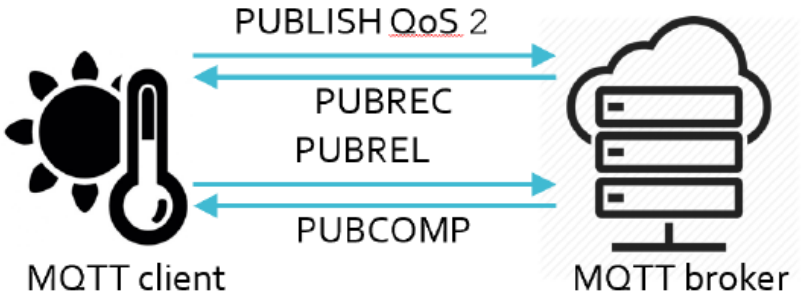
Quando un publisher pubblica un nuovo messaggio su un topic il message broker invia una notifica a tutti i subscriber che sono iscritti a quel topic, se ne esistono. Ogni subscriber è libero di disiscriversi da un determinato topic in ogni momento.

I subscriber potrebbero anche essere offline al momento del publish, ma comunque ricevere successivamente il messaggio (se e solo se i messaggi hanno QoS 1 e 2) in caso abbiano richiesto una sessione persistente; i messaggi possono anche essere *retained*, e in tal caso vengono inoltrati non appena un client si sottoscrive al topic di appartenenza.

Questo modello permette la non dipendenza tra spazio, tempo e sincronizzazione facendo si che tutti gli agenti possano lavorare in modo indipendente senza dover attendere il lavoro di altri. La scalabilità è inoltre migliore rispetto a un generico client/server, in quanto basta aggiungere più broker che funzionano in paralleo.

Il broker può filtrare i messaggi basandosi su tipo, contenuto e *topic*.

22.4 MQTT esempio di interazione e QoS



Nell'immagine viene mostrato uno scambio di messaggi tra un client e il broker MQTT (event service) in cui nello specifico si nota il QoS 2 del pacchetto inviato. MQTT infatti offre una politica di QoS che risulta essere un accordo tra client e broker con 3 diversi livelli:

- ◇ Livello 0 (at most once): consegna best-effort senza garanzie, dove non abbiamo riscontri da parte del broker/server, il quale non salva i messaggi (neanche in caso di sessione persistente);
- ◇ Livello 1 (at least once): consegna garantita almeno una volta al destinatario, dove il broker conferma la ricezione del messaggio tramite una risposta `PUBACK` (il messaggio può anche essere recapitato più volte); il broker invia il messaggio ai subscriber subito. In caso `PUBACK` sia perso, il client invierà di nuovo il messaggio, e il broker potrà scartare il duplicato (non per forza), ma qualora avesse scartato il riferimento al messaggio (e.g. refresh cache), il messaggio duplicato verrà processato nuovamente e inviato ai subscriber.
- ◇ Livello 2 (exactly once): garantisce la consegna esattamente una volta attraverso doppio two-way handshake, dove il broker notifica prima la ricezione del messaggio tramite un messaggio `PUBREC`, il client elimina il messaggio e risponde con `PUBREL` (*"Release"*), il broker scarta il riferimento al messaggio e solo adesso lo invia ai subscriber, infine termina lo scambio con `PUBCOMP` (*"Complete"*).

Solo `PUBREC` non risulta essere sufficiente, dato che se perso, il client manderà di nuovo un messaggio, quindi il broker deve salvare un riferimento del messaggio, che verrà scartato quando otterrà `PUBREL`.

22.5 MQTT

1. *Explain how messages are filtered by the broker and what are the topics in MQTT:* I messaggi vengono filtrati dal broker basandosi su tre diverse caratteristiche: il topic del soggetto (cioè l'argomento di interesse che tipicamente è una stringa), il contenuto (cioè una specifica query che controlla uno o più dati) oppure sul tipo (cioè controllando sia il contenuto che la struttura riferendosi ad una classe/tipo di dati).

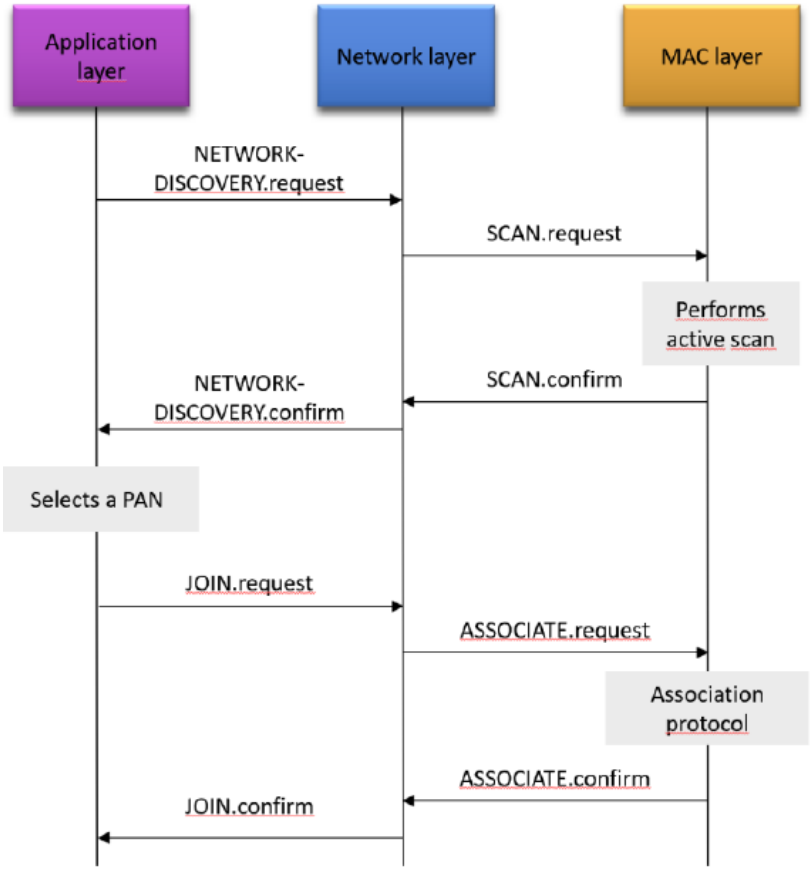
I topic sono stringhe organizzate in una gerarchia (ognuna separata da "/") che permettono di identificare un argomento di interesse. I subscriber possono utilizzare le wildcards per specificare un gruppo di topics:

- ◇ `home/firstfloor/+/presence`
  Seleziona tutti i `presence` sensors in tutte le stanze di `firstfloor`. Il + è una wildcard che seleziona un solo livello della gerarchia.
- ◇ `home/firstfloor/#`
  Seleziona tutti i sensori del primo piano. Il # è una wildcard che seleziona tutti i rimanenti livelli della gerarchia, deve essere l'ultimo carattere del topic.
- ◇ I topic che iniziano con il carattere "`$`" sono riservati per statistiche interne di MQTT; Questi topic non sono standardizzati, e non possono essere pubblicati dai client.

Notare che isciversi a `/home/firstfloor`, **non** significa ricevere notifiche per i messaggi relativi a tutti i subtopic come `/home/firstfloor/presence`, è necessario usare l'opportuna wildcard multilivello #

2. *Explain the persistent sessions in MQTT:* Una sessione persistente mantiene lo stato tra il client e il broker, e.g. se un subscriber si disconnette, quando si connette nuovamente non ha bisogno di iscriversi ancora una volta ai topic a cui era sottoscritto. Le sessioni sono associate al *clientId*. È necessario richiedere la sessione persistente al momento della connessione, settando a `FALSE` il flag `Clean Session`.
   Il broker memorizza tutte le iscrizioni, tutti i messaggi con QoS 1 e 2 non ancora confermati e che arrivano quando il client è offline.
   Si ricorda che il `Client ID` deve essere univoco per evitare complicazioni.
3. *Explain the retained messages in MQTT*: I retained message sono messaggi utilizzati dai publisher che vogliono tenere aggiornati i nuovi subscriber sullo stato delle comunicazioni. Questi messaggi sono semplici messaggi con il *retainFlag* settato a **true**, ciò permette al broker di salvare e poi recapitare il messaggio ad un nuovo client che si sottoscrive a quel determinato topic, anche se il messaggio ha QoS 0. Solitamente questa tipologia di messaggi viene utilizzata per aggiornamenti non frequenti dello stato, come ad esempio lo stato di apertura/chiusura di una porta.
4. *Explain the Lastwill&testament and the Keep Alive mechanism in MQTT:* Sono due meccanismi utilizzati in MQTT, e fanno riferimento agli omonimi campi nel messaggio `CONNECT`:
   i. **LastWill&Testament** è una feature che viene impostata a tempo di connessione da un client per notificare gli altri client in caso di una sua disconnessione improvvisa. Quando il broker riscontrerà una disconnessione improvvisa del client invierà il messaggio indicato a tutti gli iscritti del topic specificato quando la feature è stata impostata.
   ii. **KeepAlive**: assicura che un client rimanga attivo, per farlo il client invia periodicamente un messaggio `PINGREQ` (seguito da `PINGRESP`) al broker prima che l'intervallo definito nel campo. Se questo non avviene in tempo il broker termina la connessione ed invia il messaggio *LastWill&Testament* se presente.

22.6 Zigbee I



Un device ZigBee può connettersi a una PAN in seguito a una richiesta specifica del Coordinator: in tal caso si tratta di **Direct Join**. La figura invece illustra la procedura di **join through association** avviata da device ZigBee con un Coordinator di una PAN selezionata a seguito di una scansione preliminare.
I messaggi in sequenza sono:

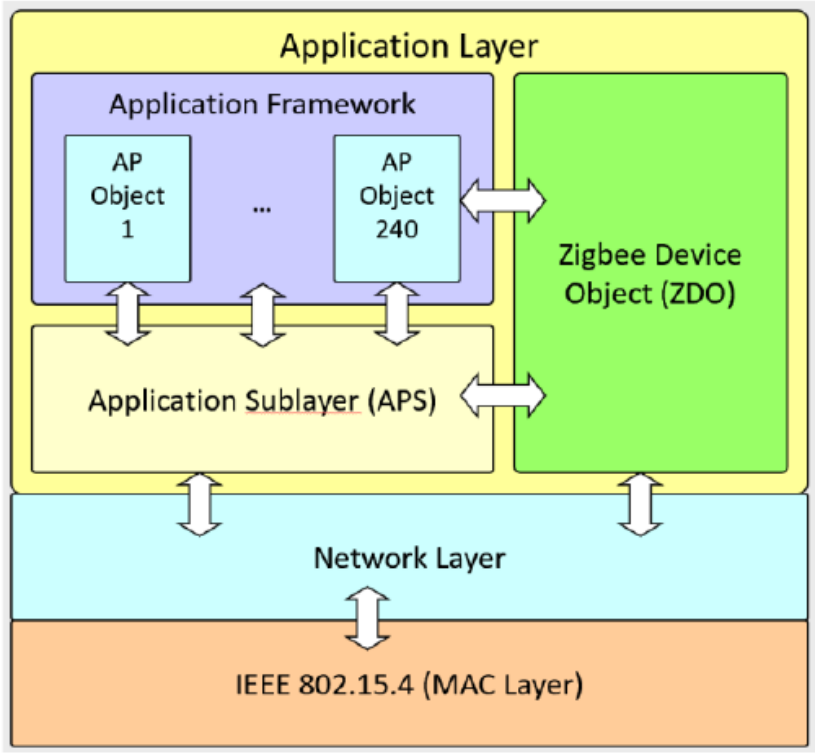
1. La prima richiesta inviata è una `NETWORK-DISCOVERY.request`, che il livello applicativo manda al sottostante livello di rete per identificare le PAN disponibili;
2. Il livello di rete inoltra tale richiesta al livello MAC —inviando una `SCAN.request`— che avvia una active scan che restituisce al livello di rete una serie di PAN ID, con informazioni sui relativi router/coordinator, attraverso una `SCAN.confirm`;
3. Il livello di rete propaga a livello applicazione il set di PAN ID con `NETWORK-DISCOVERY.confirm`, permettendo la selezione della PAN a cui unirsi;
4. La `JOIN.request`, che viene inoltrata al network layer, contiene dei parametri come l'identificatore della PAN scelta, un flag che indica il tipo di dispositivo che si unisce (router o end-device);
5. A livello rete viene selezionato il nodo "genitore" corrispondente alla PAN scelta;
6. Questo porta poi all'invio di `ASSOCIATE.request` verso il livello MAC che avvia il protocollo di associazione in cui si ottiene l'indirizzo breve a 16-bit.
7. L'indirizzo di livello rete NWK lungo 16 bit (e.g. `0x0FF3`)[1] viene restituito a livello rete da `ASSOCIATE.confirm` e poi propagato a livello applicazione da `JOIN.confirm`.

22.7 Zigbee II

Nella figura viene rappresentato il livello applicazione di ZigBee a sua volta diviso in 3 sotto-livelli:

- **Application Framework**: contiene fino a 240 *Application Object (APO)*, ovvero applicazioni ZigBee definite e implementate dall'utente. Ogni APO è identificato in modo univoco all'interno della rete e i più semplici possono essere interrogati utilizzando il *Key Value Pair (KVP) data service*. Gli APO più complessi al contrario, possono gestire al loro interno anche uno stato più complicato; di conseguenza necessitano l'interazione con il *Message data service*. Ciascun APO rappresenta un componente singolo del device (e.g., lampadina di un dispositivo o interruttore);

[1] Max address is `0xFFFF`



- ◇ **ZigBee Device Object (ZDO)**: è un'applicazione speciale situata sull'endpoint 0 e gestisce il comportamento di un device in una rete ZigBee.
  Dunque implementa le funzionalità di end-device, router o coordinator. Implementa comportamenti come rispondere a una device discovery.
  Un profilo speciale, lo *ZigBee Device Profile*, descrive i cluster che devono essere supportati da ogni device ZigBee, che sono implementati dallo ZDO. In particolare sono implementati i servizi di (device e service) discovery e binding, e come gestisce network e sicurezza.
  - device e service discovery;
  - binding management;
  - network management;
  - node management
- ◇ **Application Support Sublayer (APS)**: è una sorta di livello di trasporto, simile a TCP, ma offre funzionalità diverse. È responsabile principalmente di fornire dati e gestire i servizi agli APO e ZDO. Inoltre, definisce:
  - gli endpoint: identificatori univoci degli APO che vanno da 1 a 240;
  - i cluster: protocolli che definiscono un set di messaggi e comandi che i device possono usare per comunicare, ogni cluster è identificato in modo univoco tramite un cluster ID;
  - profile ID e device ID.

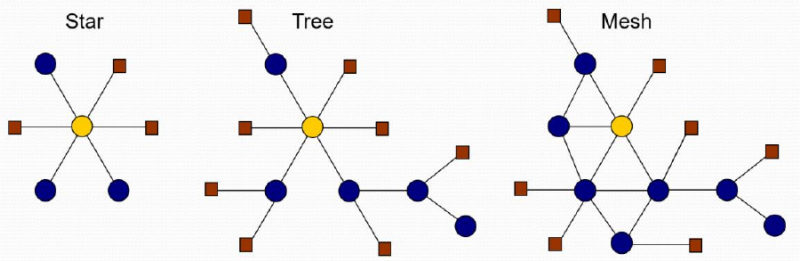
  Uno dei servizi fondamentali dell'APS è l'APS binding che permette di interconnettere l'endpoint di un nodo con uno o più endpoint di un altro nodo. Inoltre, il binding è unidirezionale e può essere gestito dallo ZDO di un coordinatore o di un router.
  Nell'APS risiedono l'**Address Map** e la **Binding Table**.

Il **Network Layer** fornisce gestisce l'addressing, il routing e funzionalità di sicurezza (No SSL, si utilizza AES-128 o nell'APS o nel NWK); infatti gli indirizzi NWK servono per l'identificazione dei device (e APOs). Fornisce un livello di astrazione rispetto al **MAC layer**, che opera più a basso livello astraendo dal livello fisico e occupandosi di (ri)trasmissione/ricezione dei frame, scan delle reti disponibili, unione a una rete, ecc.

22.8 Topologie ZigBee

Possiamo distinguere tre tipo di topologie di rete:

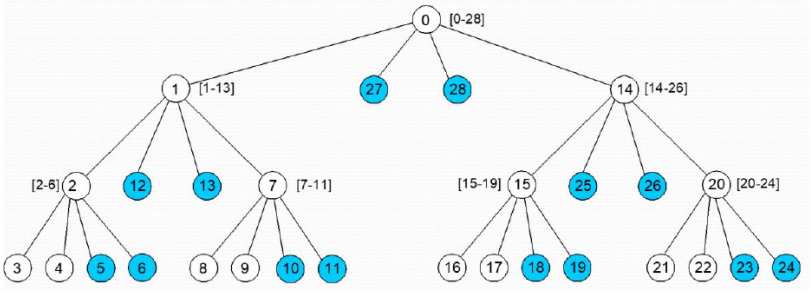
- ◇ **Topologia a stella**: il nodo centrale è sempre un coordinatore, mentre gli altri nodi possono essere indifferentemente router o end-device che svolgono lo stesso ruolo. Per la comunicazione utilizzano il *superframe*;
- ◇ **Topologia ad albero**: sfrutta i router per ampliare la copertura della rete. Il nodo radice sarà sempre un network coordinator, i router possono assumere il ruolo di nodi intermedi attraverso i quali avvengono le comunicazioni oppure possono essere nodi foglia, mentre gli end-device possono soltanto essere nodi foglia. Inoltre, vengono supportate le funzionalità della rete multi-hop;



Superframe facoltativo

◇ **Topologia mesh**: Essenzialmente è una rete p2p multi-hop, con path multipli che portano alla stessa destinazione. Anche se possibile utilizzarlo, la comunicazione è possibile anche tra nodi anche senza affidarsi all'infrastruttura superframe, sfruttando channel sharing e meccanismi di collision avoidance.

22.9 ZigBee Address Tree



Topologia basata su alberi viene sfruttata dal livello rete di ZigBee per assegnare l'indirizzo di rete breve basato su una politica definita dal coordinatore ZigBee. I parametri utilizzati sono:

◇ Il **massimo numero di router** che ogni router ha come figlio: ***Rm***;
◇ Il **massimo numero di dispositivi terminali** che ogni router ha come figlio: ***Dm***;
◇ La **massima lunghezza dell'albero**: ***Lm***.

Guardando l'immagine si ha una rete ad albero con Rm=2, Dm=2 e Lm=3. Il coordinatore ha indice 0 e il suo figlio sinistro 1. Il numero massimo di nodi in questo esempio è 28 perché è la soglia data dai tre parametri prima descritti. Questo limita il numero di nuovi dispositivi che possono unirsi alla rete, quindi un nuovo dispositivo non può entrare nella rete se viene raggiunto il limite. L'assegnazione del range viene dato seguendo un ordine DFS.

Se il nodo 3 vuole comunicare con il nodo 25, deve inviare il pacchetto al nodo 2, ed essendo 25 fuori dal suo range esso deve mandare il pacchetto al nodo 1 (nodo padre) e per lo stesso motivo al nodo 0. Il pacchetto viene poi inviato al nodo 14 che ha nel suo range 25. Queste comunicazioni avvengono per maggioranza tra router, tranne tre, quella che porta da 14 a 25 poiché è un dispositivo terminale, ma anche quelle da 1 a 0 e da 0 a 14, visto che 0 è la root ed è un network coordinator.

22.10 Tabella di binding ZigBee

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

Tabella di binding gestita dal Application Support Sublayer (APS). L'operazione di binding permette ad un end-point (APO) di un device di connettersi ad uno o più end-point su altri nodi senza conoscerne il NWK address né tantomeno l'endpoint di destinazione: il binding è unidirezionale e può essere configurato solo dal ZDO del coordinator o di un router.
Il binding quindi fornisce un modo di non specificare l'indirizzo di destinazione di un messaggio (**Indirect Addressing**), opposto al più canonico metodo del **direct addressing** che prevede che la sorgente indichi la tupla

⟨`destination address`, `destination endpoint`⟩ e che potrebbe essere non applicabile per dispositivi molto semplici. In questo modo, l'invio dei messaggi presso i destinatari è totalmente delegato a router e coordinator, dove risiede la binding table, tipicamente inizializzata a tempo di configurazione della rete, e aggiornata solo su esplicita richiesta dello ZDO di coordinator e router.

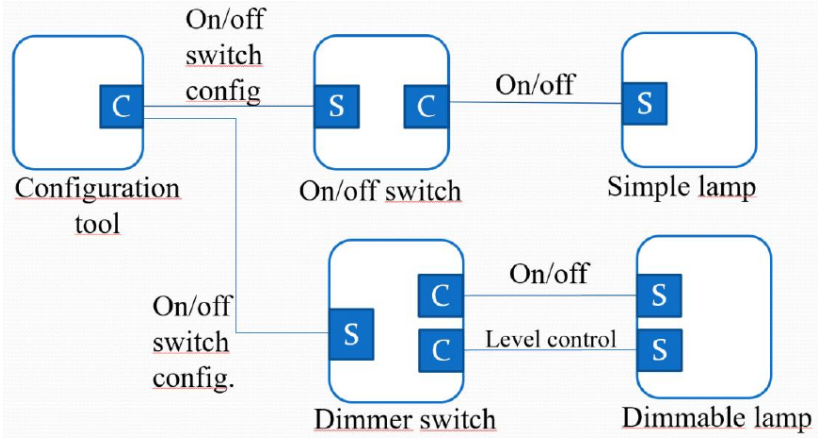
Le due primitive principali utilizzate per il binding sono BIND e UNBIND che servono rispettivamente per creare una nuova entry nella tabella di binding locale e per eliminarla.
La entry mantiene l'indirizzo sorgente (MAC a 64 bit), l'endpoint sorgente, l'identificatore del cluster, l'indirizzo a 64 bit di destinazione (o indirizzo 16 bit multicast[2]) e l'endpoint di destinazione. Questa tabella viene salvata in modo persistente nell'APS del coordinatore di ZigBee e/o di un altro router: viene caricata su esplicita richiesta del ZDO.

Un messaggio viene inoltrato alla/e destinazione/i trovando le entry che nella binding table matchano la tupla ⟨`source address`, `source endpoint`, `clusterID`⟩ del messaggio in arrivo. In caso di multipli match, avverrà una comunicazione multicast e il messaggio sarà inoltrato alle multiple destinazioni.

In figura per la tupla $\langle 0x3232..., 5, 0x0006\rangle$ abbiamo più entry corrispondenti a più destinazioni.

Se non ci sono match, il messaggio viene scartato.

22.11 Diagramma di configurazione e interconnessione delle luci



Esempio di utilizzo della ZigBee Cluster Library (ZCL) che contiene le specifiche e i framework per i cluster implementati dalle applicazioni degli utenti. Per cluster si intende collezione di comandi e attributi relativi a un *dominio funzionale* (che definiscono un'interfaccia di una specifica funzionalità). Tali comandi sono definiti seguendo un modello client-server, dove il dispositivo che salva gli attributi mantenendo uno stato interno è un server mentre i dispositivi che manipolano gli attributi sono client.
La ZCL definisce anche dei messaggi per:

- ◇ Leggere/scrivere attributi
- ◇ Configurare un report e leggere una risposta in formato report
- ◇ Scoprire ID e tipi degli attributi supportati da un server

Nella figura si parla di cluster nel dominio funzionale *Lighting* in cui si vuole configurare un insieme di APO con uno switch ON/OFF connesso con una "simple lamp" e un "dimmer switch" connesso ad una "dimmable lamp", che sono i cluster. Il configuration tool crea l'associazione tra i due APO funzionando come un client che imposta la configurazione del dispositivo di switch in modo che quando la lampada sia accesa o spenta questo possa essere visto nel dispositivo corrispondente. Il configuration tool imposta una tabella di routing nel coordinator conoscendo l'indirizzo dei dispositivi e dei relativi APO.

22.12 Duty Cycle I

Il codice in figura mostra un esempio di un programma Arduino, in cui ogni componente viene accesso, utilizzato e poi successivamente subito spento per cercare di ottimizzare il consumo energetico del device su cui questo codice è installato. Per aumentare il lifetime è necessario ottimizzare il duty cycle di ogni componente, ovvero il rapporto tra il tempo di attività e e il periodo[3]. Il duty cycle complessivo di un dispositivo è pari alla somma dei duty cycle di ogni singolo componente, assumendo che i cicli di operatività siano "disgiunti" e alcune semplificazioni sull'effettivo

[2] NON il NWK address a 16bit!!!
[3] valore arbitrario maggiore o uguale del tempo di attività, onde evitare complicazioni nei calcoli

```
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);
}
```

	Value	units
Micro Processor (Atmega128L)		
current (full operation)	8	mA
current sleep	15	µA
Radio		
current in receive	19,7	mA
current xmit	17,4	mA
current sleep	20	µA
Logger (storage in the flash memory)		
write	15	mA
read	4	mA
sleep	2	µA
Sensor Board		
current (full operation)	5	mA
current sleep	5	µA
Battery Specifcations		
Capacity Loss/Yr	3	%

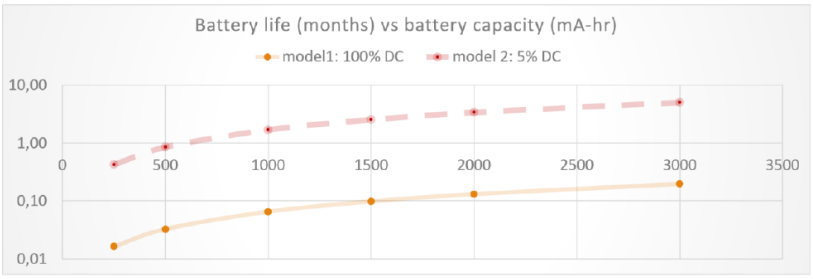
istante di ON/OFF. Ad esempio, *Duty cycle = 1%* implica che il device sia accesso solo per il 1% del tempo totale, e spento per il restante 99%.

Negli esercizi si considera il tempo richiesto per ciascuna operazione di CPU/Radio/Sensore/ecc. (in figura non riportato), e si rapporta al periodo complessivo (in figura il periodo **non** è 380!).

Per calcolare i consumi energetici, si fa riferimento alla tabella dei consumi tipicamente fornita per ogni dispositivo IoT. Data la capacità della batteria equipaggiata, è possibile calcolare l'expected lifetime, moltiplicando il duty cycle per il la corrente consumata di ciascun componente, sommando i risultati, e dividendo la capacità della batteria per tale numero.

Duty cycle CPU $= 1\% = 0.01$
$0.01 \cdot 8mA + 0.99 \cdot 0.015mA = 0.095mA$
$2000mAh/0.095mA = 21052ore = 2.4anni$

22.13 Duty Cycle II



In questo diagramma si ha il confronto tra due modelli con duty cycle differenti:

- ◇ Il primo modello ha un DC pari al 100%, ovvero con ogni componente del device sempre in funzione;
- ◇ Il secondo modello ha un DC pari al 5%, ovvero con un tempo di inattività delle sue componenti pari al 95%.

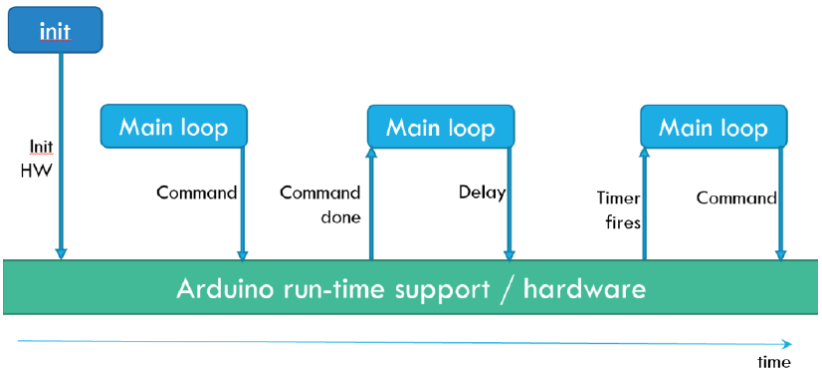
Come è possibile osservare dal grafico, il secondo modello ha un tempo di vita della batteria maggiore rispetto al primo a parità di capacità della batteria, questo perché il tempo di utilizzo limitato incrementa sicuramente il tempo di vita di un sensore, ma d'altro canto richiede che le attività delle varie componenti del device siano schedulate in modo efficiente, per rispettare le performance attese.

È necessario fare il tuning dei parametri del duty cycle affinché non solo il dispositivo funzioni efficientemente, ma che tale duty cycle gli permetta di fare sampling in modo produttivo e di funzionare correttamente. (e.g. Un sensore della temperatura per una stanza in una casa, non può fare sampling ogni due giorni)

22.14 Embedded programming I

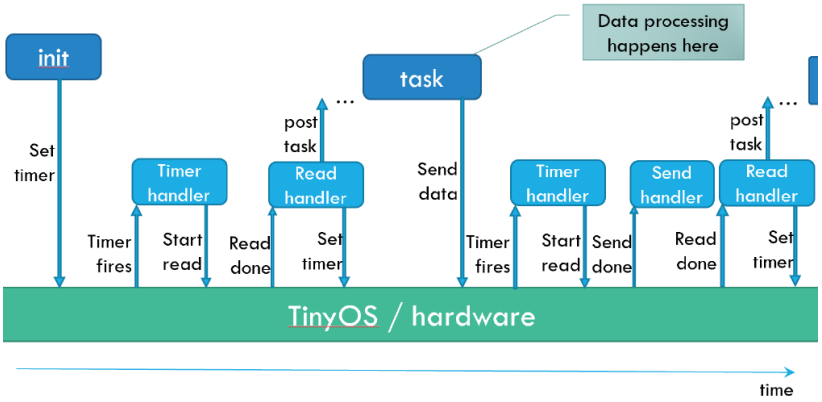
Arduino è single threaded e il suo flow di esecuzione è una singola loop function che può invocare altre funzioni. Non c'è la sospensione del thread, dunque operazioni di I/O fanno attendere il main thread finché non sono completate. Gli step dell'esecuzione sono i seguenti:

1. Il device invoca la funzione `init` che permette l'inizializzazione del dispositivo stesso e abilita la comunicazione con i componenti hardware;
2. Viene invocato il main `loop()` rappresentante il ciclo delle operazioni da svolgere periodicamente.
3. Nell'esempio viene inviato un comando a una componente hardware; quando è stato eseguito l'Arduino RTS invia un segnale di terminazione del comando e restituisce il controllo al main loop;



4. Se il carico di lavoro attivo è stato compiuto si può invocare in modo esplicito la funzione `delay` che invia un comando ad un timer sospendendo la CPU finché il timer riattiva il `main loop` con un segnale (`timer fires`).

22.15 Embedded programming II



In figura viene mostrato come TinyOS gestisce il software di un device, progettato per gestire principalmente eventi asincroni attraverso 3 concetti fondamentali:

- ◇ *Comandi*: utilizzati per programmare e comunicare con l'hardware.
- ◇ *Eventi*: astraggono interruzioni hardware come una sorta di upcall.
- ◇ *Task*: definiti per gestire attività differenti e indipendenti. Quando un task è terminato e non ce ne sono altri da eseguire allora la CPU passa in `idle mode`;

1. La funzione `init` dopo aver inizializzato il device setta un timer e un handler da eseguire allo scadere.
2. `timer handler` —definita dall'utente oppure fornita dal RTS— può, ad esempio, richiedere una lettura da un sensore;
3. Terminata la lettura, un `read handler` gestirà questo evento schedulando un nuovo task (ad alto livello) che potrà eseguire operazioni più complesse sui dati.
4. Terminate le operazioni sui dati, questi saranno inviati all'hardware e verrà settato nuovamente un timer, che farà ripartire il ciclo degli eventi come successivamente alla funzione di init.
5. TinyOS gestisce l'esecuzione di un singolo handler alla volta alla volta e infatti dovrebbero essere brevi: più interruzioni simultanee potrebbero causare seri problemi di concorrenza.Tuttavia è possibile che un'operazione hardware in corso (come l'ultima lettura in figura) possa essere interrotta da un'altra "upcall".

I comandi invocati dagli handler non devono riguardare il componente HW che ha triggerato l'handler, onde evitare stati inconsistenti.

Con questo modello una task "non aspetta mai" che venga eseguita lettura o scrittura hardware, e può essere pre-empted solo da degli eventi; non sarà lei a richiedere esplicitamente di andare in idle, come nel caso di Arduino con il comando `delay`.

22.16 Interruzioni Arduino

Nel codice si può osservare il meccanismo delle interruzioni di Arduino che, come in TinyOS, permette di gestire eventi asincroni. Le interruzioni possono essere di tre tipi: *esterne*, *timer* o di *dispositivo*. Esse vengono gestite

```
/**
 * test interrupt
 *
 */

volatile int greenLed=7;
volatile int count=0;

void setup()
{
  Serial.begin(9600);
  pinMode(greenLed, OUTPUT);

  digitalWrite(greenLed, LOW);

  attachInterrupt(0,
      interruptSwitchGreen,RISING);
  // 0 because it is pin 2
}
```

```
void loop() {
  count++;
  delay(1000);
  // interrupts are received
  // also within delay!

  Serial.print("waiting:");
  Serial.println(count);

  if ( count == 10 )
  {
    count = 0;
    digitalWrite(greenLed, LOW);
    Serial.println("now off");
  }
}

void interruptSwitchGreen()
{
  digitalWrite(greenLed, HIGH);
  count=0;
  Serial.print("now on");
}
```

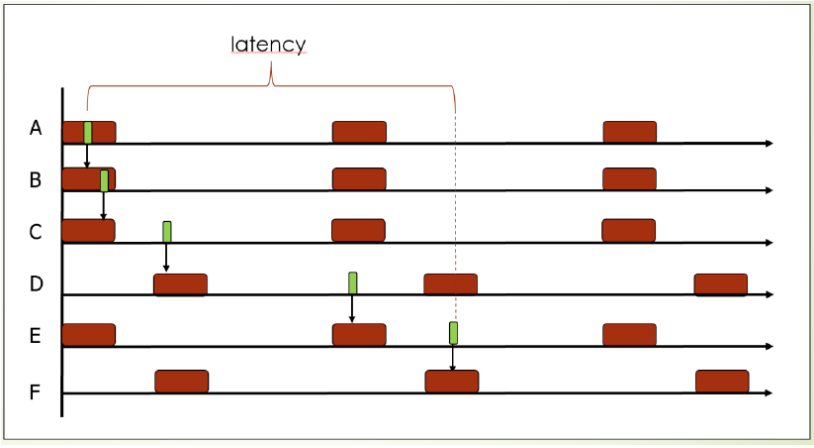
dal RTS. In questo caso osserviamo come si ha una funzione di `loop` principale in cui c'è un'attesa fissa che viene seguita da una scrittura digitale. All'occorrenza di un'interruzione *esterna* è possibile come in questo caso definire un comportamento predefinito attraverso la funzione `attachInterrupt(interrupt\#, funct-name, mode)` nella funzione di `setup()`. Le modalità sono le seguenti:

- **RISING**: l'interruzione avviene quando il pin passa da un valore LOW ad un valore HIGH;
- **FALLING**: l'interruzione avviene quando il pin passa da un valore HIGH ad un valore LOW;
- **CHANGE**: l'interruzione avviene quando il pin cambia stato;
- **LOW**: l'interruzione avviene quando il pin ha uno stato basso. Non necessariamente un cambio di stato, se rimane LOW l'interruzione avviene di nuovo;
- **HIGH**: l'interruzione avviene quando il pin ha uno stato alto. Non necessariamente un cambio di stato, se rimane HIGH l'interruzione avviene di nuovo.

Nella figura, `greenlLed = 7` definisce il pin 7 come il pin di output per il led verde, mentre `attachInterrupt(0, interruptSwitchGreen, CHANGE)` definisce che l'interruzione avverrà sul pin 2 (corrispondente all'interruzione 0) e che la funzione `interruptSwitchGreen` verrà eseguita quando il pin passerà da `LOW` ad `HIGH` (il cambio di stato è `RISING`). `interruptSwitchGreen` è una funzione che cambia lo stato del led verde. `Serial.begin(9600)` setta il data rate di trasmissione a 9600bit/s

Saper spiegare il resto del codice a voce

22.17 SMAC



Lo schema in figura rappresenta lo scambio di messaggi tramite l'utilizzo del protocollo SMAC, un protocollo MAC (*Medium Access Control*) per le reti wireless multihop. SMAC è utilizzato in 802.11, mentre in 802.15.4 si utilizza il **polling** (beacon). In questo protocollo i nodi utilizzano una sincronizzazione locale per poter comunicare con i propri vicini, ottenuta tramite lo scambio di messaggi `SYNC` che permettono di sincronizzare i duty cycle —della radio— dei dispositivi in modo che si accendano nello stesso momento.

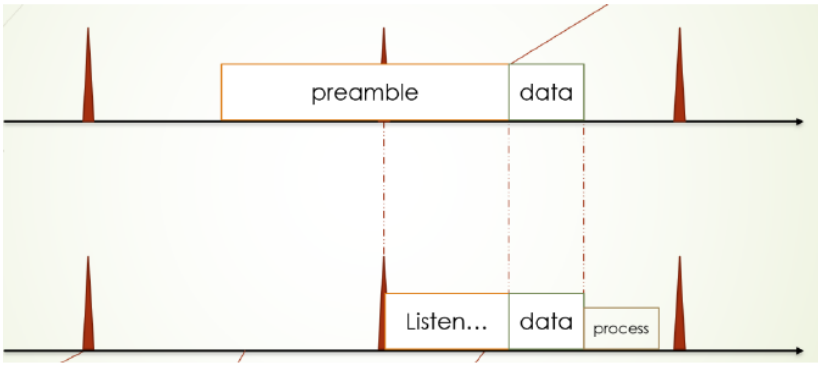
Qualora un nodo voglia inviare a un ricevente che non è sincronizzato con il suo stesso ciclo, dovrà comunicare nel time slot del nodo ricevente, eventualmente svegliandosi fuori dal suo slot predefinito.

Visto che le comunicazioni avvengono nel timeframe del ricevente è possibile che si vengano a creare delle collisioni, per questo motivo prima di inviare un messaggio viene effettuato un controllo *Carrier Sense* per comprendere se il canale sia occupato o meno, causando ritardi nella trasmissione. Se vengono ricevuti o un `RTS` (Request To Send) o un `CTS` (Clear To Send) la trasmissione viene lasciata attiva perché questa sia terminata.

Questo protocollo ha due problemi principali, il primo, la **latenza**, dovuta alla difficoltà nel sincronizzare molteplici nodi distanti (resa accettabile da cluster di nodi vicini che invece riescono a sincronizzarsi), mentre il secondo è la qualità degli **orologi interni** dei device (spesso cheap), che possono portare a oscillazioni nella misurazione del tempo e dunque a desincronizzazioni.

Inoltre il CSMA/CA usato ha i problemi di *hidden/exposed terminal*.

22.18 BMAC



Lo schema in figura rappresenta lo scambio di dati attraverso il protocollo BMAC. Il protocollo BMAC è un protocollo *medium access control* che riduce la complessità del protocollo SMAC, in quanto richiede la configurazione di un unico parametro, stabilito out-of-band.

In questo protocollo il mittente può inviare un messaggio che contiene un lungo preambolo (preamble), in qualsiasi momento. Al contrario il ricevente attiva la radio periodicamente controllando se c'è un preamble che deve essere catturato (preamble sampling) basandosi sulla modalità *Low-Power Listening (LPL)*.
Questi preamble dovranno avere una lunghezza maggiore rispetto all'intervallo che avviene tra un campionamento ed un altro e un punto a sfavore risiede nel fatto che il ricevente deve attendere la fine del preamble prima di ricevere i dati.

Viene da chiedersi se un nodo prima di inviare un preambolo debba provare ad ascoltare per capire se il canale non sia già occupato, o se invece sia legato al spike rosso indicante il wake-up

Il protocollo permette ai receiver di utilizzare meno energia, a scapito dei consumi maggiori del sender; dunque BMAC trova un'utile applicazione in scenari dove ci sono più receiver e pochi sender.

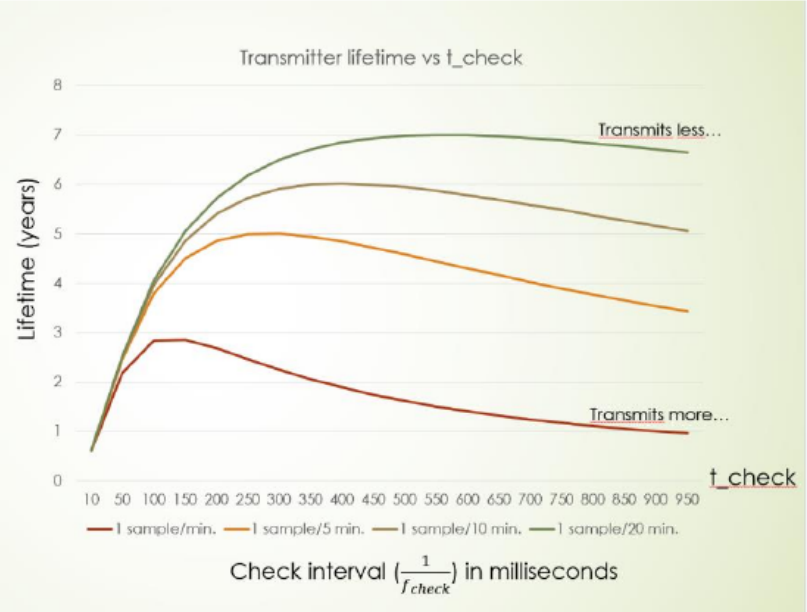
22.19 Device Lifetime e BMAC

Il diagramma in figura mostra l'andamento del tempo di vita di un **trasmettitore** mettendo in rapporto la frequenza di invio dei sample, il tempo fra i wake-up (t_check), e gli anni di vita di un dispositivo. Si può osservare come ad una frequenza di campionamento maggiore (per esempio 1 sample al minuto - linea rossa) il tempo di vita sia molto breve, addirittura inferiore ai 3 anni, mentre con un campionamento meno frequente (per esempio 1 sample ogni 20 minuti - linea verde), il tempo di vita aumenta sensibilmente raggiungendo un massimo che supera sfiora 3 volte il tempo di vita usando il campionamento con la linea rossa.

Ciò che è interessante è vedere come indipendentemente dalla frequenza dei sample, il trend sia comunque decrescente oltre un certo t_{check}, il tempo fra un risveglio e l'altro, indicante anche la lunghezza del preambolo da mandare.

Tali valori sono ottenuti attraverso le seguenti formule che formalizzano il tempo di vita di un trasmettitore:

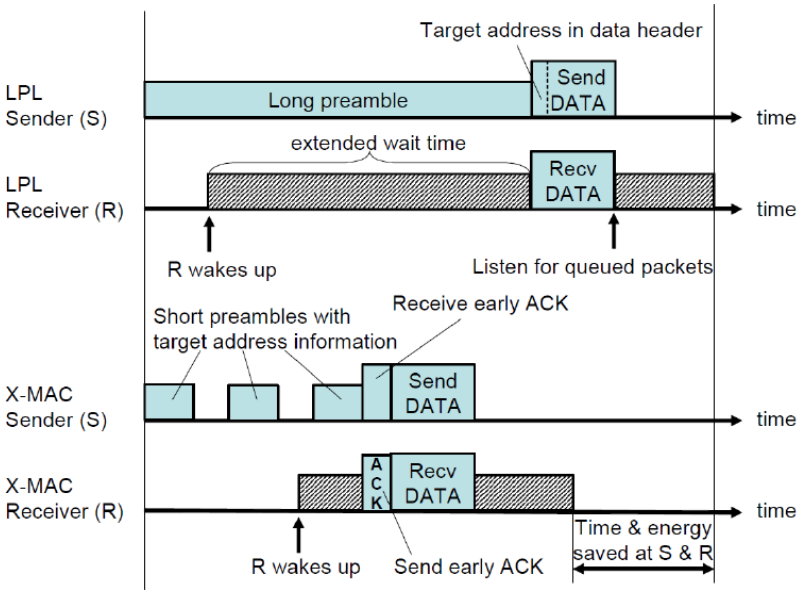
- $DC_{tx} = f_{data} \cdot (t_{preamble} + t_{data})$



- dove abbiamo f_{data} frequenza di invio dei dati, $t_{preamble}$ l'intervallo di invio del preamble, t_{data} l'intervallo di invio dei dati.
- $DC_{check} = f_{check} \cdot t_{check}$
- dove abbiamo f_{check} la frequenza di sampling, t_{check} l'intervallo di sample frequency
- $ET(t) = t \cdot (p_{tx}DC_{tx} + p_{tx}DC_{check} + p_{sleep} * (1 - DC_{tx} - DC_{check}))$
- dove p_{tx}, p_{sleep} rappresentano rispettivamente il consumo energetico durante la fase di trasmissione e il consumo energetico durante la fase di riposo

TODO verifica formule se sono scritte bene

22.20 XMAC/BMAC



Il protocollo X-MAC è un'evoluzione del protocollo B-MAC che mira a ridurre l'impiego di lunghi preamble vuoti.

Per fare ciò il protocollo permette al ricevente di fermare l'invio del preamble inviando un ACK:
quando un receiver trova il suo ID all'interno di uno dei preamble brevi inviati da un sender allora questo invierà un early ACK al sender che sospenderà l'invio dei preamble e invierà i dati al ricevente che è pronto a riceverli.

Il ricevente dopo la ricezione dei dati non spegnerà immediatamente la componente radio per permettere ad altri eventuali nodi di inviargli dati nel caso in cui abbiano trovato il canale precedentemente occupato.

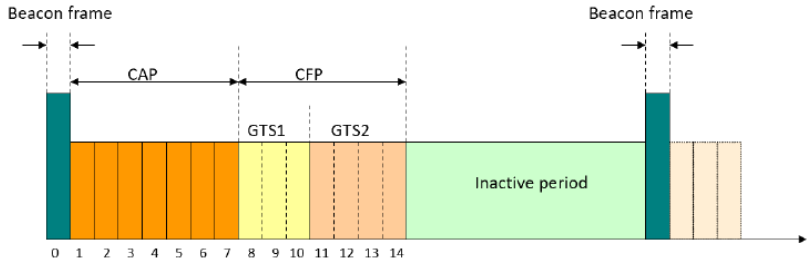
Nell'immagine si può vedere come in XMAC nella fase di trasmissione invia un preambolo breve contenente informazioni sull'indirizzo dei target, quando il ricevente si sveglia e cattura il preambolo breve replica con un ACK che

permette al mittente di iniziare l'invio dei preamboli brevi e di iniziare ad inviare i frame con i dati.

BoX-MAC è un'ulteriore sviluppo di XMAC, in cui si sostituisce il preambolo con l'invio ripetuto dei dati (insieme alle informazioni sui destinatari).

È adatto per applicazioni in cui i dati sono piccoli e la latenza è critica.

22.21 IEEE 802.15.4 superframe



L'immagine mostra il servizio di channel access per il livello MAC del protocollo IEEE 802.04.15. Abbiamo due tipi di accesso a livello MAC:

1. con superframe structure;
2. senza superframe structure;

Nello specifico l'immagine mostra un channel access con superframe structure, utilizzato solitamente nelle topologia a stella o nelle topologie peer-to-peer, ma con struttura ad albero. In caso di reti *non-beacon enable*, la comunicazione avviene senza superframe structure, con un protocollo unslotted `CSMA-CA`; il coordinator è sempre pronto a ricevere dati da un end-device mentre la trasmissione da end-device a coordinator è poll-based: l'end-device si sveglia periodicamente e interroga il coordinator per i messaggi in attesa.

Un superframe è composto da due parti, una attiva e l'altra inattiva. Tutte le comunicazioni avvengono durante il periodo attivo, mentre in quello inattivo i nodi possono andare in idle.

La parte attiva, in cui è possibile comunicare è composta a sua volta da 15 time frame della stessa dimensione, di cui il primo di questi è detto beacon frame ed è inviato dal PAN coordinator. Il beacon frame serve per sincronizzare i dispositivi collegati, per identificare il PAN e descrivere la struttura del superframe.

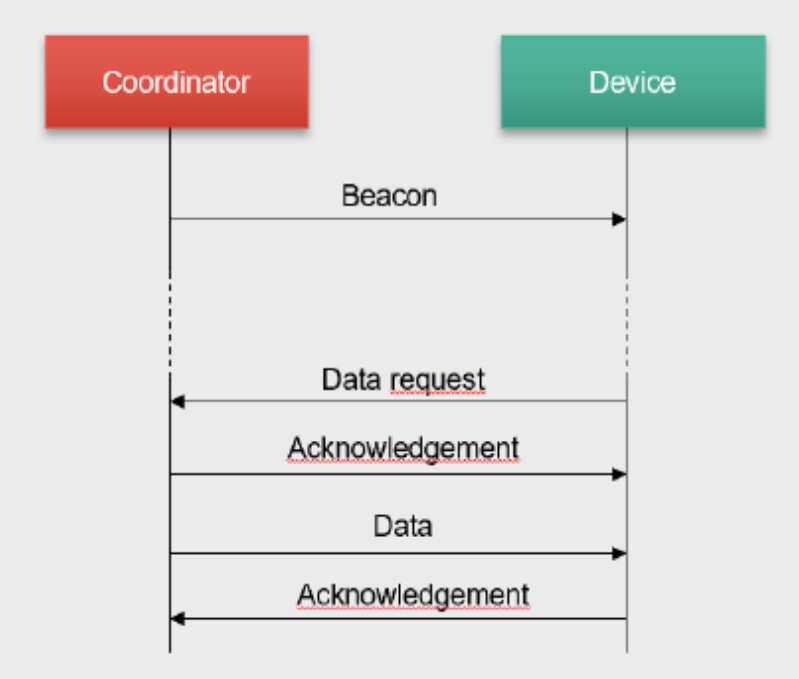
La comunicazione vera e propria tra l'end device e il coordinatore (non con i router) avviene nella restante parte del periodi di attività, che è divisa in due parti:

- **Contention Access Period (CAP)**: il dispositivo compete per accedere al canale utilizzando il protocollo standard slotted CSMA/CA. Un dispositivo che vuole trasmettere i frame dati prima deve aspettare il beacon frame e dopo randomicamente seleziona uno slot temporale per la sua trasmissione: se seleziona uno slot occupato da un'altra comunicazione allora proverà a selezionarne un altro, altrimenti attenderà il successivo superframe. Una volta iniziata la trasmissione, il sender tiene il canale occupato fino alla fine del frame.
  Il CAP period deve essere composto da almeno uno slot, per permettere a nuovi device di unirsi alla rete
- **Contention Free Period (CFP)**: opzionale e utilizzato per applicazioni a bassa latenza o che richiedono uno specifico bandwidth. Il coordinatore PAN può assegnare porzioni di questa parte chiamati **GTS (Guaranteed Time Slot)** a specifiche applicazioni.
  In caso ci siano più richieste di GTS di quante ne siano disponibili, il coordinatore PAN può assegnare i GTS in base a priorità o in base a un algoritmo di scheduling, altrimenti risponde ai richiedenti "picche" ☺.

22.22 Trasferimento dei dati IEEE 802.15.4

Il sequence diagram nell'immagine mostra il trasferimento di dati da un *coordinator* $\longrightarrow$ *device* in una rete **beacon enabled**.
Esistono 3 diversi modelli di trasferimento dati: end-device al coordinatore, coordinatore all'end-device, peer-to-peer.

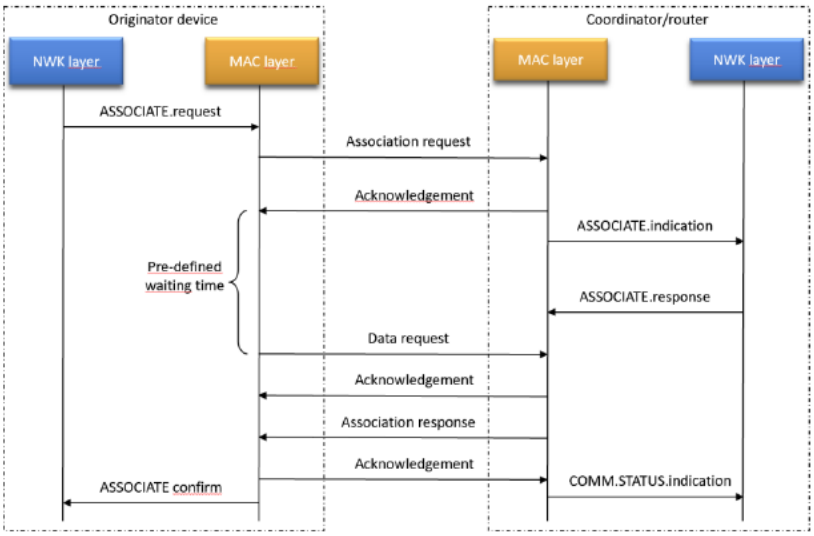
- coordinator a end device:
  il coordinator indica nel network beacon che esiste un messaggio per l'end device.
  Gli end device restano per la maggior parte del tempo in sleep mode, ma periodicamente si svegliano per controllare messaggi pending, e se ce ne sono li richiedono usando uno slot nel CAP.
  Il coordinator invia il pending message utilizzando anch'esso un protocollo CSMA-CA con slot all'interno del



CAP. Una volta ricevuto il frame, il dispositivo può riconoscere la ricezione trasmettendo un ACK in uno slot temporale successivo;

- ◇ end device a coordinator: il dispositivo attende il beacon di rete per sincronizzarsi con il superframe. Dopodiché, se possiede un GTS lo usa direttamente, altrimenti trasmette il data frame al coordinator utilizzando protocollo standard CSMA-CA in uno dei frame del CAP (Contention Access Period). Il coordinator riconosce la ricezione con successo trasmettendo un ACK in uno slot temporale successivo;
- ◇ peer-to-peer: nel caso uno dei due dispositivi coinvolti nella comunicazione sia un end device allora si utilizza uno dei due protocolli sopra. Al contrario se entrambi i device sono coordinator e ognuno possiede il proprio network beacon, allora il mittente del messaggio deve sincronizzarsi prima con il network beacon del ricevente e comportarsi poi come un end device.

22.23 IEEE 802.15.4



Il sequence diagram in figura rappresenta una richiesta di servizio ASSOCIATE, che viene invocato da un dispositivo che desidera associarsi con un PAN di cui ha l'identificativo da una invocazione preliminare del servizio SCAN.

1. La primitiva `ASSOCIATE.request` prende come parametro l'identificatore di una PAN, l'indirizzo del coordinatore e l'indirizzo IEEE esteso a 64 bit del dispositivo stesso.
   La primitiva invia un messaggio di richiesta di associazione al coordinatore e, poiché la procedura di associazione è pensata per le reti abilitate al beacon, il messaggio di richiesta di associazione viene inviato durante il periodo CAP utilizzando il protocollo CSMA-CA.
2. Il coordinatore invia un messaggio di ACK in cui riconosce la ricezione del messaggio ma non per questo

l'associazione alla PAN è avvenuta con successo.
3. Il coordinatore, attraverso la primitiva `ASSOCIATE.indication`, passa la richiesta al livello di rete dove viene effettuata la decisione sull'associazione: in caso questa sia accettata viene selezionato un indirizzo a 16 bit che il dispositivo utilizzerà in seguito al posto di quello IEEE da 64 bit.
4. Il livello rete ritorna con la primitiva `ASSOCIATE.response` al livello MAC che prende come parametro l'indirizzo a 64 bit del dispositivo, il nuovo indirizzo a 16 bit e lo stato della richiesta.
   Il coordinatore genera così un pending message con la risposta dell'associazione.
5. Dopo un periodo definito di attesa il dispositivo fa una *Data request* a cui corrisponde un ACK del coordinatore e a seguire il messaggio contenente la risposta dell'associazione, corrisposto da un ACK inviato dal device al coordinator
   Deve essere il device a contattare il coordinator perché non conosce ancora il suo NWK address e il coordinator dovrebbe inserire tale address per inviargli un messaggio tramite il beacon.
6. Lato dispositivo quindi viene invocata la primitiva `ASSOCIATE.confirm` verso il livello rete mentre lato coordinatore viene emesso `COMM-STATUS.indication` per informare il livello superiore che il protocollo di associazione si è concluso (con successo o con un codice di errore).

22.24 Energy harvesting

22.24.1 Explain the difference between an harvest-use and an harvest-store-use architecture

Nell'architettura **Harvest-Use** si ha un modello di harvesting diretto in cui non viene fornito nessun buffer energetico per questo l'energia accumulata viene immediatamente consumata. Questo comporta due principali svantaggi:
- ◇ Spreco di energia quando quella prodotta è minore di quella consumata;
- ◇ Spegnimento del dispositivo quando viene prodotta meno energia di quella consumata.

Tale modello può funzionare in caso solo di fonti di energia —molto— prevedibili, o di carico di lavoro statico o facilmente adattabile.

Nel caso dell'architettura **Harvest-Store-Use** viene introdotto un buffer che permette la raccolta dell'energia in eccesso per usi futuri. Semplificazioni del modello potrebbero non tenere conto dell'efficienza di carica e degli energy leaks. In base al materiale della batteria (`SLA`, `NiCD`, `NiMH`, `Li-on`, o supercapacitors) è possibile stimare alcuni dei comportamenti della batteria nel tempo.

22.24.2 Explain the classification of sources in terms of controllability and predictability

1. **Controllabile** fornisce una raccolta di energia quando è richiesta
   Self-powered flashlight, girando una manovella la luce si accende.
2. **Parzialmente controllabile** produce energia in modo limitato dall'ambiente
   Sorgente di energa RF (energia a radiofrequenza) può fornire energia a RFID, ma non si possono controllare l'intensità di propagazione nella stanza (non possiamo garantire con precisione come questa si distribuirà nello spazio)
3. **Non controllabili** non possono essere attivate su richiesta
   i. **prevedibili**: hanno dei modelli di utilizzo che ne descrivono la disponibilità
      Sole
   ii. **imprevedibili**: indovina.
      Terremoti

Le sorgenti più interessanti e maggiormente oggetto di studio sono quelle *non* controllabili e prevedibili, in quanto permettono di avere un modello di harvesting più preciso e di conseguenza un modello di consumo più efficiente.

Il modello di Kansal tuttavia non richiede conoscenza sulla futura produzione di energia: le sorgenti devono essere —anche solo in parte— prevedibili, ma la stima di produzione energetica futura si basa esclusivamente sulle precedenti osservazioni; Nonostante ciò il modello benificia da una maggior accuratezza in caso esistano dei modelli più avanzati di previsione.

22.24.3 Explain the concept of energy neutrality

Mentre il goal principale anni fa era di massimizzare la durata della batteria, oggi si cerca di ottenere la **neutralità energetica**: una produzione di energia che permette di ottenere lo stesso livello di performance per tutta la durata di un intervallo di tempo, i.e. la produzione e il consumo sono bilanciati e sostenibili nel tempo.

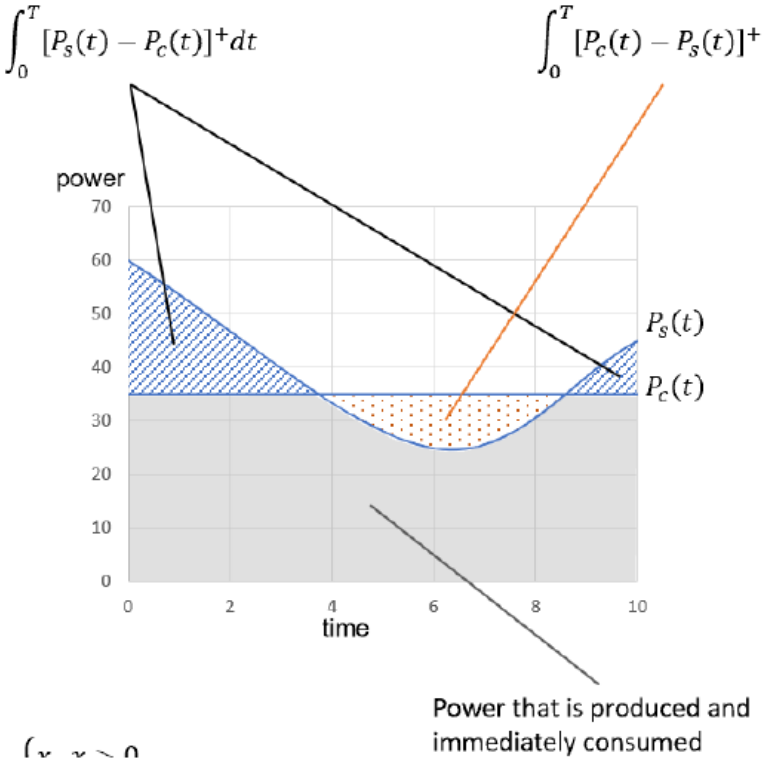
Kansal la formalizza stabilendo che $B(K+1) \geq B(1)$, ovvero la batteria al termine del periodo $K+1$ deve avere

almeno la stessa carica di quella iniziale.

Ci sono due aspetti chiave

- ◇ **Energy-Neutral Operation**:
  Scegliere le operazioni da eseguire in modo che, in qualunque time frame, l'energia usata sia sempre inferiore di quella prodotta.
- ◇ **Maximum Performance**:
  Assicurando l'operatività energy-neutral (scritta sopra), determinare qual è il massimo livello di performance che si può ottenere in un dato ambiente di harvesting.

22.25 Consumo di energia con buffer



Rappresentazione grafica del modello **Harvest-Store-Use** in cui si hanno le due funzioni che rappresentano l'energia prodotta e quella consumata, e gli integrali rappresentano l'energia che viene immediatamente consumata appena prodotta (caso in cui *consumo* > *produzione*) oppure l'energia che viene immagazzinata (caso in cui *produzione* > *consumo*). In questo modello si considera un buffer ideale senza perdite e con capacità infinita.

Per ottenere l'energy neutrality dobbiamo garantire che l'area azzurra sia maggiore dell'area arancione, in modo da consentire di mantenere il dispositivo attivo anche quando l'energia prodotta è inferiore rispetto al consumo.
Inoltre dobbiamo anche massimizzare l'energia disponibile, per garantire le migliori performance possibili.

Per far ciò, Kansal formalizza il problema come un problema di ottimizzazione risolubile tramite la programmazione lineare, in cui si cerca di massimizzare l'utility del dispositivo, ovvero la somma delle utilità di tutte le operazioni eseguite[4], avendo come vincolo che il consumo di tali operazioni non ecceda l'energia disponibile.

22.26 Esercizio harvesting

platform	v_{min} (volts)	v_{max} (volts)	v_{ref} (volts)	quantization levels (bit)	x_{min}	x_{max}	size of battery (mAh)	x (voltage sampling)	Battery charge (mAh)
RPI	3,5	5	0,004883	10	716	1023	3000	1000	2798
Arduino	7	9	0,008789	10	795	1023	3000	800	359
Tmote	1,8	3	0,000732	12	2457	4095	3000	3277	1652

[4]duty cycle per Kansal, task nel task-based model

v_{max} indica la tensione fornita dalla batteria a piena carica (B_{max}), mentre v_{min} corrisponde alla tensione fornita con la carica minima (B_{min}) che consente al dispositivo di funzionare. Notare che tale carica minima, *non* è una proprietà della batteria, bensì del sistema.

I dati rappresentati su d bit x_{min} e x_{max} (e x per un generico voltaggio v) che vengono letti dal ADC che ha in input il voltaggio, possono essere stimati matematicamente:

$$x_{max} = 2^d - 1 \tag{22.1}$$

$$x_{min} = \left\lfloor \frac{v_{min}}{v_{max}}(2^d - 1) \right\rfloor \tag{22.2}$$

$$x = \left\lfloor \frac{v}{v_{max}}(2^d - 1) \right\rfloor \tag{22.3}$$

Il reference voltage v_{ref} si calcola dividendo v_{max} per il massimo intero rappresentabile con i bit dell'ADC ($2^d - 1 = x_{max}$), e indica quanto "pesa" un singolo bit.

Le ultime due colonne propongono un esempio di lettura di x dall'ADC e calcolo del corrispondente livello di batteria, assumendo —per comodità di calcolo— che $B_{min} = 300mAh$, con la seguente formula:

$$B = B_{min} + \frac{B_{max} - B_{min}}{x_{max} - x_{min}}(x - x_{min}) \tag{22.4}$$

Tale formula assume una relazione lineare fra il voltaggio e la capacità della batteria, che è una approssimazione accettabile anche se in realtà dipende dalla batteria.

Inoltre è opportuno osservare che il voltaggio della batteria decresce "linearmente" (o progressivamente potrebbe rimanere circa costante) fino a una certa percentuale di carica, intorno al 10%, dopo il quale il voltaggio cala rapidamente verso 0, come in Fig. 22.3. Tale punto dipende dalla batteria, nell'esempio si presume sia `300mAh`, circa un decimo della capacità totale.

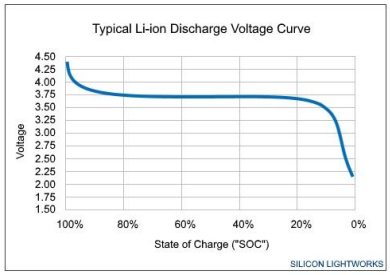
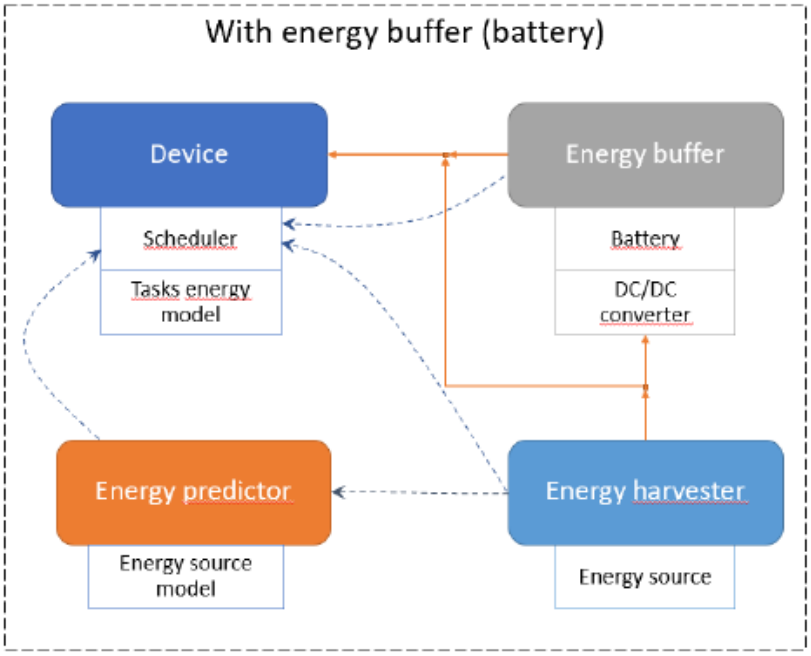


Figure 22.3: Example of discharge graph of a Li-ion battery

22.27 Approccio di Kansal



Il modello di Kansal per la neutralità energetica richiede delle assunzioni sulla fonte energetica, ovvero che sia non controllabile ma prevedibile, o quantomeno che l'energia da essa prodotta sia approssimabile ad una funzione lineare.
Lo schema in figura è una rivisitazione di quello standard Harvest-Store-Use per implementare Kansal.
L'*Energy predictor* realizza una stima della futura produzione di energia suddividendo il tempo in slot di dimensione fissa e usando l'EWMA come modello predittivo; infine comunica i risultati allo *scheduler*.
Per ogni possibile duty cycle dc si può calcolare un'utilità u —che va massimizzata— e il consumo previsto c. Lo scheduler, in base alle stime fornite dal predictor sull'energia e le informazioni u e c di ciascun dc, può assegnare ad ogni slot dc, massimizzando u e garantendo la neutralità energetica.
Nella pratica questo può tradursi in duty cycle più o meno intensi a seconda dell'energia disponibile.

22.27.1 Teorema di Kansal

Il teorema stabilisce quali sono le condizioni sufficienti (non necessarie) per poter ottenere l'energy neutrality su un sistema.

$$\begin{cases} \eta\rho_s \geq \rho_c + \rho_{leak} & \text{Trend produzione maggiore di trend consumo e leak} \\ B_0 \geq \eta\sigma + \delta & B_0 \text{ sufficiente nel worst case} \\ B_{max} \geq B_0 & B_0 \text{ è un valore legale} \end{cases} \tag{22.5}$$

Nella seconda equazione —forse— $\eta\sigma$ è un termine che starebbe a sinistra con segno negativo, ad indicare che invece di avere una produzione media (compresa fra $+\sigma$ e $-\sigma$) non si produce nulla. η = coefficiente di efficienza di carica

Drawback

Uno dei due principali drawback di questo modello è proprio che l'unico parametro che può essere variato per modificare il load è il **duty cycle**, rendendolo poco adattabile alla gestione di comportamenti più complessi.
Inoltre in caso di sensori, alterare il duty cycle significa avere una misurazione con una frequenza di **campionamento** variabile, che può rendere scomoda l'analisi dei dati.

Il modello di Kansal assume che l'energia prodotta si possa approssimare ad una fonte lineare, ovvero che cresce limitata da due rette parallele che hanno un angolo pari a ρ_s. Da qui abbiamo che l'energia E_T prodotta in un dato intervallo $[0, T]$ sia uguale a $E_T = \int_0^T P_S(T)dt$ e sia limitata come segue $\rho_s \cdot T - \alpha \leq E_T \leq \rho_s \cdot T + \alpha$.

Invece per quanto riguarda il carico (load), il modello di Kansal assume che sia limitato solo superiormente da una retta inclinata di un angolo ρ_c e quindi abbiamo che il carico L_T in un intervallo $[0, T]$ è uguale a $L_T = \int_0^T Pc(T)dt$ e quindi il carico sarà limitato nel seguente modo $0 \leq L_T \leq \rho_c \cdot T + \delta$

Nel modello di Kansal il concetto di utility è legato al duty cycle nel seguente modo:
- $u(dc) = 0$ se $dc < dc_{min}$
- $u(dc) = \alpha * dc + \beta$ se $dc_{min} \leq dc \leq dc_{max}$
- $u(dc) = u_M$ se $dc > dc_{max}$

Con u_M utility massima, u_m utility minima, dc_{max} duty cycle massimo, dc_{min} duty cycle minimo e con

$$\alpha = \frac{u_{max} - u_{min}}{dc_{max} - dc_{min}} \tag{22.6}$$

$$\beta = u_{min} - \alpha * dc_{min} \tag{22.7}$$

Dove in un grafico con assi xy duty cycle dc e utility u, α è il coefficiente lineare (*inclinazione*) della retta, mentre β è il punto in cui la retta incrocia l'asse y, ovvero l'*offset* della retta.

22.27.2 Modello di Kansal

- k il numero di slot in un giorno
- $B(i)$ la carica della batteria all'inizio dello slot i
- $B(k+1)$ la carica della batteria alla fine del giorno
- $\tilde{p}_s(i)$ la stima della produzione di energia nello slot i

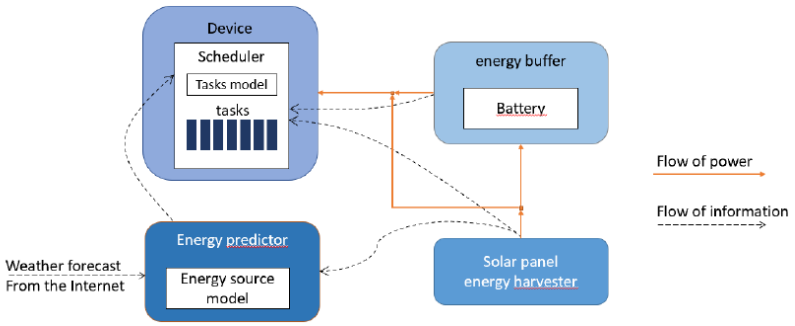
  Kansal assegna un $dc(i)$ (dunque una utility $u(i)$) a ciascuno slot i in modo da massimizzare la somma delle utility di tutti gli slot, garantendo che $B(k+1) \geq B(1)$, ovvero che la carica della batteria alla fine del giorno sia maggiore uguale a quella all'inizio del giorno.

22.27.3 Algoritmo Kansal

Si inizializzano gli slot assegnando dc_{max} ai "sun slot" e dc_{min} ai "dark slot". A questo punto —rispetto alla stima di produzione di energia $\tilde{p}_s(i)$— si può riscontrare:

1. **Overproduction**
   Si massimizza il *dc* dei dark slot finché non si esaurisce l'energia sovrapprodotta
2. **Underproduction**
   Si minimizza il *dc* dei sun slot finché non si va in pari con l'energia prodotta

22.28 Modello basato su task



Il task-based model si basa sull'idea che il ciclo di funzionamento di un applicazione di un dispositivo IoT sia tipicamente:

1. **Sensing**
2. **Storing**
3. **Processing**
4. **Transmitting**

Ciascuna di queste fasi può avere più implementazioni possibili con consumi energetici potenzialmente diversi. Ad esempio, potrebbero essere disponibili sensori diversi, sampling rate variabili... Si potrebbe fare processing e storing solo di alcuni dati, o di trasmettere dati compressi o meno, già processati o da processare, cifrati o in chiaro, ecc... Dunque è possibile definire l'implementazione di un'applicazione IoT come un insieme di **task**, ciascuna con un consumo energetico per unità di tempo e un'utilità.

Il numero di task è limitato dai vincoli del dispositivo e dallo scheduling ma le funzioni generali non cambiano, cambia l'implementazione che ha effetto sul comportamento del dispositivo.

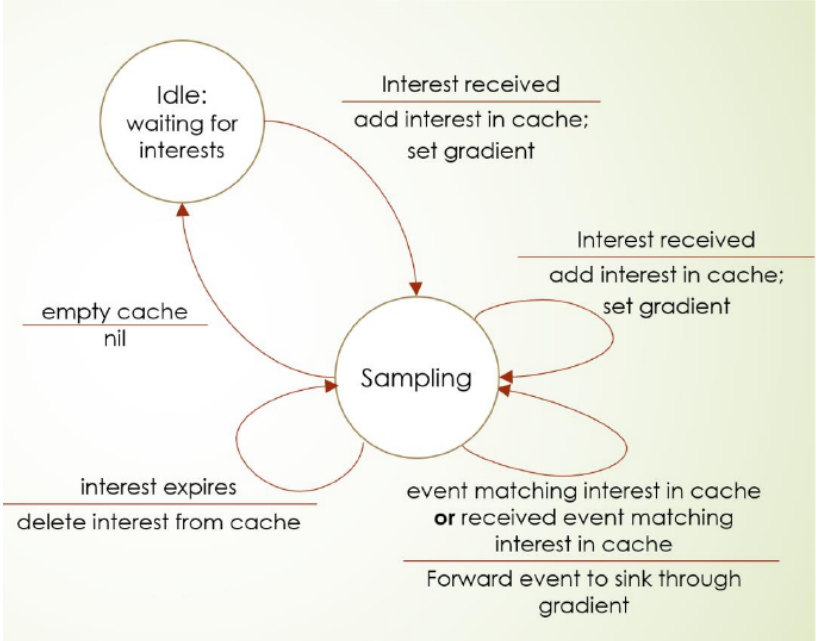
Similmente a come accade per Kansal, sulla base della stima di produzione di energia e del consumo di ciascun task, uno scheduler assegna un task per ciascuno slot di tempo, risolvendo un problema di ottimizzazione NP-Hard attraverso una soluzione (leggermente approssimativa) pseudo-polinomiale basata su programmazione dinamica.

Il vantaggio rispetto a Kansal è che al posto di duty cycle diversi, si assegnano task, permettendo un più agevole e modulare adattamento del carico di lavoro.

22.29 Diffusione diretta

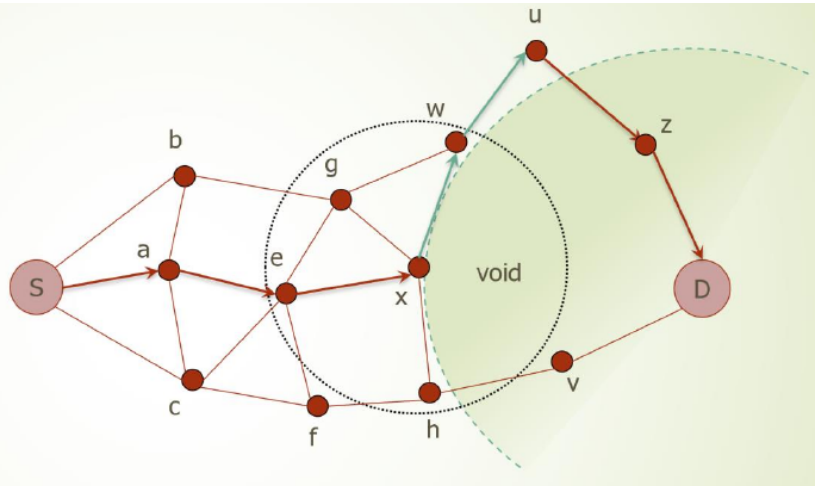
Protocollo di rete per WSN che specifica come eseguire sensing distribuito e rispondere a query basate sul contenuto dei dati, i quali sono associati a coppie **attributo-valore**. Può essere diviso in 3 passi:

1. *Comunicazione basata sull'interesse*: la comunicazione si basa sull'interesse espresso dai sink node (nodi che fungono da output gateway verso reti esterne) verso tipi di dati o attributi, il quale propaga la query nella rete;
2. *Propagazione dei dati basata su gradiente*: ricevute le query dai sink i sensori iniziano a raccogliere i dati al più ampio sampling rate fra quelli dei gradienti locali.
   Un **gradiente** esprime una *direzione* (verso il sink richiedente) e un data rate.
   I nodi inoltrano i dati verso i sink node solo se hanno in cache l'interesse (con relativo gradiente) che matcha il dato e se il dato non è già stato inoltrato prima.
3. *Aggregazione e Reinforcement*: mentre i dati confluiscono verso il sink node, i nodi sensore intermedi mettono in cache gli interessi ricevuti (fino all'expire time) e aggregano i dati relativi a uno stesso interesse con sampling rate differenti.
   Il sink può ricevere dati da più sorgenti, e per evitare ciò, può scegliere di fare *Reinforcement* su uno dei path (quello che il maggiore data rate) e inviarvi un interest con data rate maggiore, che verrà propagato lungo quel path. In questo modo vengono empiricamente (?) preferiti i dati di migliore qualità.



Nell'immagine si vede la macchina a stati di un sensore, partendo dall'`idle` state, in cui il dispositivo è in attesa di un interest. Quando viene ricevuto un interest questo viene aggiunto alla cache, viene impostato il gradiente avente come direzione il nodo da cui proviene il messaggio di interest e come data rate quello specificato nell'interest (campo `interval`); il nodo passa allo stato di `sampling`, se non vi è già.
Quando il sensore rileva un evento associato con un interesse in cache (e.g. "sta passando un elefante"), inizia il campionamento dell'evento e vengono inviati al sink node in accordo con il gradiente registrato per quell'interest.
A ciascun interest è associata una `duration`, al termine della quale l'interest viene rimosso dalla cache, che quando vuota porta allo stato `idle`.
La macchina a stati è implementata su interruzioni —similmente a TinyOS— permettendo a un dispositivo di agire contemporaneamente come router e come sensore.

22.30 GPSR Modalità Greedy forwarding



La DD richiede alcune assunzioni:

- ◇ Un singolo nodo Sink con $id = 0$ permanentemente connesso alla rete (se va down la rete non funziona più)
  In realtà forse potrebbero esserci più sink
- ◇ Ogni nodo con *id* univoco
- ◇ Il sink inizializza e mantiene il routing tree
- ◇ I messaggi dei nodi sono unicast diretti verso il sink
- ◇ Solo il sink può mandare messaggi broadcast

Notare che la DD non sfrutta le capacità di processing dei nodi e non permette neanche il rilevamento di eventi "compositi" distribuiti su più sensori. La Directed Diffusion non è scalabile per reti grandi e porta a colli di bottiglia

in prossimità dei sink con topologie ad albero. Topologie diverse richiedono strategie di routing diverse, e per questo motivo è stato proposto il protocollo GPSR.

Greedy Perimeter Stateless Routing (o GPRS) è un protocollo alternativo al protocollo Direct Diffusion che permette il routing dei pacchetti nodo-a-nodo senza mantenere uno stato globale e con un basso overhead. Anche questo protocollo necessita alcune assunzioni, ma meno stringenti:

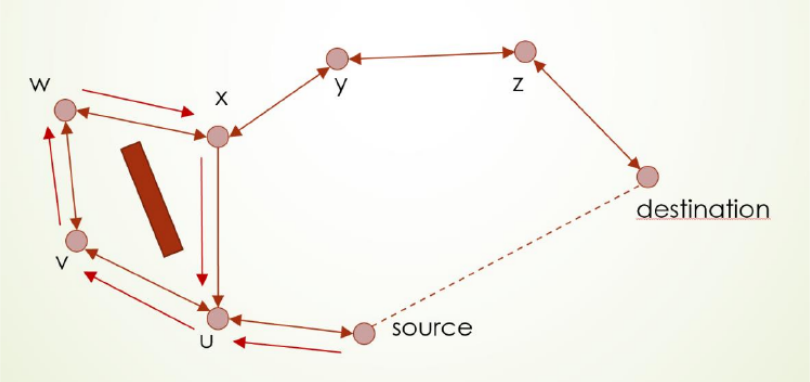
- ◇ i nodi sono disposti in uno spazio bidimensionale;
- ◇ i nodi conoscono la posizione geografica dei loro vicini;
- ◇ il nodo sorgente S conosce la posizione geografica del nodo destinazione D.

GPRS può operare sotto due modalità diverse, ***greedy forwarding*** e ***perimeter mode***.

Inizialmente si applica il *greedy forwarding*: la sorgente S per inviare un pacchetto alla destinazione D inoltra il pacchetto al nodo neighbour geograficamente più vicino alla destinazione.
Questa modalità "fallisce" quando si entra in una regione di vuoto, i.e. non ci sono nodi che più vicini a D; Quando ciò si verifica avviene lo switch in *perimeter mode* che identifica il perimetro della regione vuota e lo percorre (seguendo LHL o RHR rule) fino a trovare un nodo che possiede un vicino geograficamente più vicino alla destinazione rispetto al nodo che ha fatto lo switch in perimeter mode[5], momento in cui torna alla *greedy mode* per raggiungere la destinazione.

Nell'immagine su x si passa in perimeter mode, fino ad arrivare a u, che ha come vicino z che è geograficamente più vicino a D rispetto ad x.

22.31 GPSR Modalità Perimeter forwarding



In questo caso c'è un ostacolo che può portare a un loop.

1. Nel nodo sorgente avviene subito lo switch alla perimeter mode.
2. l'arco (x, u) è unidirezionale, dunque u inoltra il pacchetto a v.
3. v non vede x, e dunque inoltra il pacchetto a w.
4. w inoltra il pacchetto a x.
5. x se utilizza `RHR`, il primo arco che trova in senso antiorario partendo da (w, x) è (x, u), e dunque inoltra il pacchetto a u, portando a un **loop**.
   Se invece si utilizza `LHL`, gli step precedenti non cambiano, ma x inoltrerà il pacchetto a y, dove si passerà alla greedy mode e si giungerà alla destinazione.

Mutual Witness, estensione di planarization algorithm può risolvere questo problema usando sempre link bidirezionali; tuttavia, ci sono casi in cui l'aggiunta di tali link bidirezionali porta a grafi non planari e a loop, rendendolo non idoneo per scenari reali.

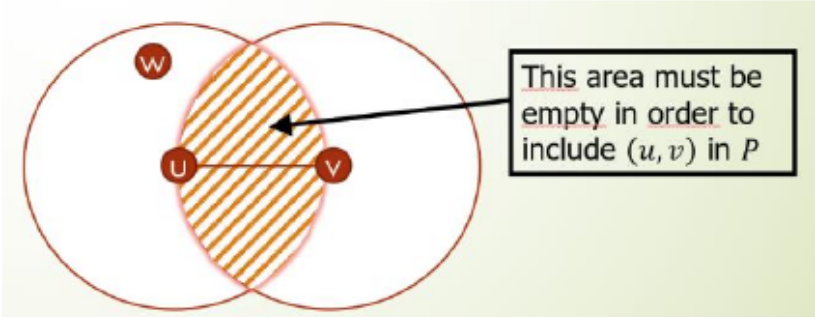
22.32 GPSR Planarizzazione del grafo

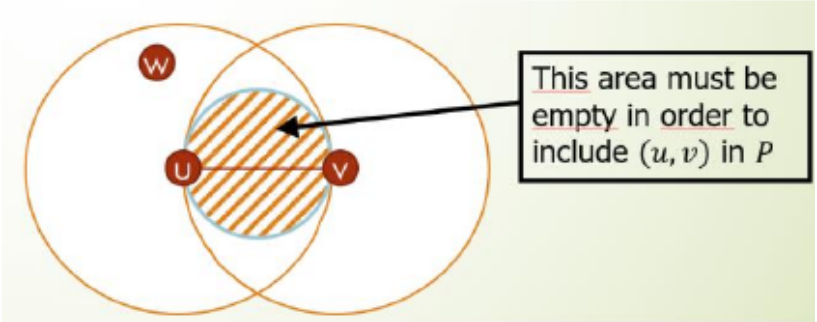
Il grafo della topologia di rete tipicamente non è *planare* (i.e. gli archi potrebbero incrociarsi), per questo motivo è necessario costruire una rappresentazione del grafo che lo sia. Abbiamo discusso due metodi:

- ◇ **Relative neighborhood Graph of G (RNG)** does not exist a third point w that is closer to both u and v than they are to each other

$$(u, v) \in P \Leftrightarrow (u, v) \in G \wedge d(u, v) \leq \max_{\forall w \in N(u) \cup N(v)} (d(u, w), d(v, w))$$

[5]Non può avvenire appena si esce dalla void region, in quanto ciò potrebbe portare a loop





Considerati tutti i vicini di u e v, non esiste un nodo w la cui distanza sia da v che da u sia minore di d(u,v), cioè la distanza fra u e v stessi

- ◇ **Gabriel Graph (GG)**
  Questa opzione porta a un grafo planarizzato più denso.
  La legge matematica per costruirlo è ~~simile a Pitagora ☺~~ :

$$(u, v) \in P \Leftrightarrow (u, v) \in G \wedge d(u, v)^2 \leq d(u, w)^2 + d(v, w)^2 \quad \forall w \in N(u) \cup N(v)$$

Considerati tutti i vicini di u e v, non esiste un nodo w la cui somma dei quadrati delle distanze fra u e v sia minore del quadrato della distanza d(u,v). In altre parole, preso il triangolo u,v,w, il quadrato del lato fra u,v è minore della somma fra gli altri due

Entrambi gli algoritmi creano un sottografo P a partire da un grafo G dato, senza aggiungere archi, ma solo rimuovendoli.

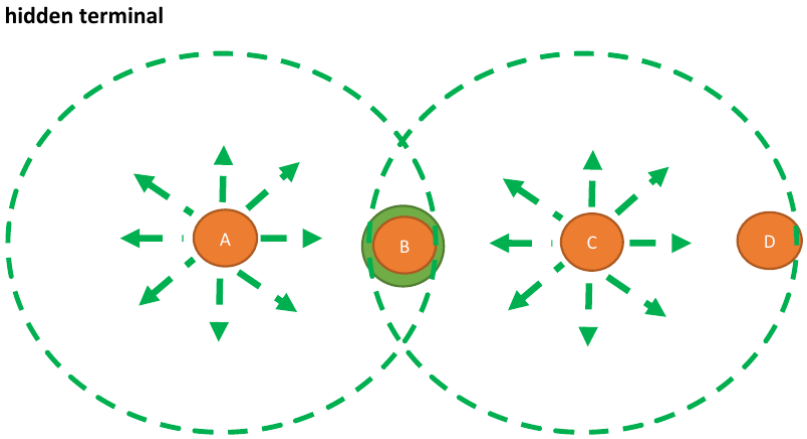
Alcune proprietà vengono mantenute: ad esempio, se un grafo è connesso allora anche il suo sottografo sarà connesso.

Inoltre, entrambi gli algoritmi minimizzano l'overhead di memoria e di calcolo eseguendo una knowledge discovery distribuita e localizzata per identificare gli archi del grafo planarizzato. Questo è reso possibile perché ogni nodo ha eseguito la planarizzazione del grafo in anticipo.

Chapter 23

Risposte orale Paganelli

23.1 Hidden terminal



Una delle sfide dei terminali nelle reti wireless è la limitata conoscenza dei terminali che non sono nel loro raggio d'azione.
Il problema degli hidden terminal è un problema che si verifica quando due o più station che sono rispettivamente fuori dal range dell'altra vogliono trasmettere in modo simultaneo allo stesso device, generando in questo modo delle collisioni.

Nell'immagine è possibile osservare come il nodo A risulti essere un hidden terminal per il nodo C e viceversa durante le rispettive comunicazioni con B. Questo accade perché A, non essendo nel radio range di C, non sarebbe in grado di rilevare un'eventuale comunicazione da C verso B. Quindi, se A iniziasse a comunicare con B mentre C sta già comunicando con B, questo porterebbe ad una collisione in B.

23.2 Exposed terminal

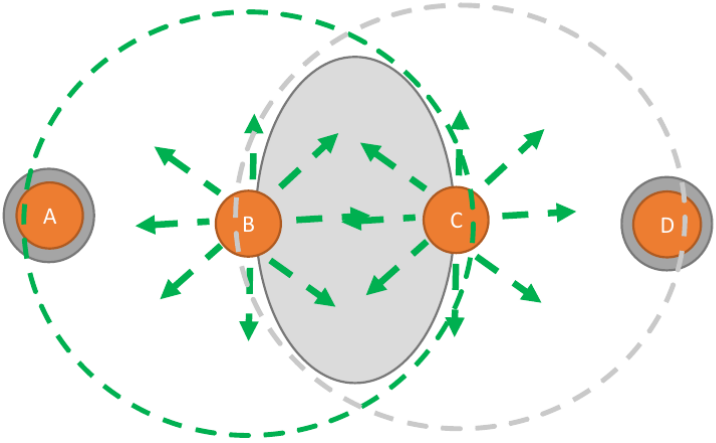
Il problema degli exposed terminal è un problema che si verifica quando una stazione è esposta alle comunicazioni di un'altra stazione e risulta essere impossibilitata ad inviare messaggi a sua volta, anche quando le comunicazioni non sono rivolte verso lo stesso terminale.

Nell'immagine è possibile osservare come il nodo C sia esposto alla comunicazione da B verso A, impedendo una possibile comunicazione da C verso D, che potrebbe comunque avvenire in parallelo.

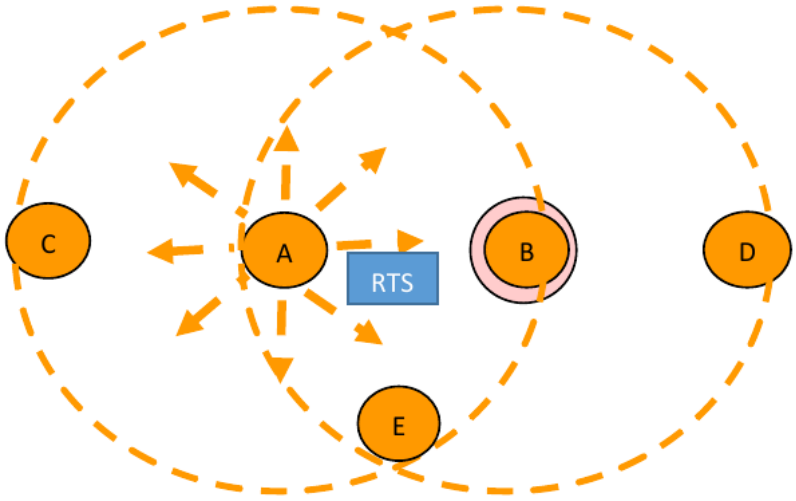
23.3 RTS/CTS

Il meccanismo dei RTS (Request To Send) e dei CTS (Clear To Send) viene utilizzato all'interno del protocollo MACA (Multiple Access with Collision Avoidance). In questo protocollo, per cercare di evitare (non risolvere) le collisioni e il problema degli hidden/exposed terminal viene inviato uno short frame precedentemente all'invio del data frame, così da far astenere gli altri dispositivi dall'invio di messaggi.

exposed terminal



RTS/CTS



Nell'immagine è mostrata la comunicazione dal nodo A al nodo B:

- Il nodo A invia un RTS (Request-to-send) al nodo B, che viene ricevuto anche dai nodi C e da E. L'RTS contiene anche la lunghezza del data frame.
- Successivamente B, pronto ad accettare il messaggio, risponde ad A con un CTS (Clear-to-send), che viene ricevuto dai nodi A, D ed E.
- A questo punto, A può inviare il data frame a B.

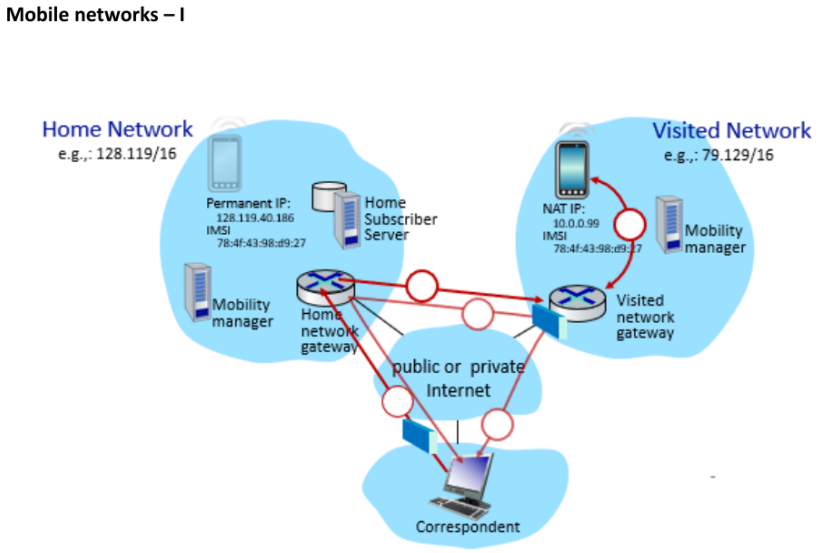
Questo protocollo però non risolve il problema degli hidden/exposed terminal. Infatti, C è un exposed terminal, in quanto riceve l'RTS di A ma non il CTS di B: quindi C è libero di trasmettere, ma tutto quello che riceverà farà collisione con i dati mandati da A; D è un hidden terminal perché riceve il CTS da B ma non l'RTS da A, quindi D non può più trasmettere fino a quando la trasmissione del data frame è completata Trasmissione che non sentirà. MACAW aggiunge un ACK finale —a ricezione dati ultimata— inviato da B per ovviare al problema, (giusto?).

Cosa accade se B e C mandano contemporaneamente l'RTS al nodo A? C'è collisione tra gli RTS, quindi non viene generato un CTS. La soluzione al problema consiste nell'utilizzo da parte di B e C del *Binary Exponential Backoff* per provare a rinviare l'RTS.

Cosa accade se un nodo al di fuori del range di A e B prova a comunicare con E? E non risponderà all'RTS del nodo esterno, dato che E ha sentito il CTS di B. Il nodo esterno proverà (sempre utilizzando *Binary Exponential Backoff*) a rinviare l'RTS fino a quando E non risponderà.

MACAW (MACA for Wireless networks) è un miglioramento del protocollo MACA: aggiunge ACK frame inviato dal nodo ricevente per fare l'acknowledge dei dati ricevuti, e altri meccanismi per scambio di informazioni: il Data Sending frame, inviato dal mittente quando inizia a inviare informazioni; il RRTS (Request to RTS), inviato da un nodo che ha ricevuto un RTS, ma non era in grado di rispondere a causa di un'altra comunicazione attiva.

23.4 MN indirect routing



L'immagine mostra il funzionamento dell'***indirect routing*** per la mobilità tra una home network e una visited network. La home network rappresenta la rete "nativa" di un dispositivo con un abbonamento ad un mobile provider (e.g. *Verizon*), dove le informazioni relative ai device abbonati vengono salvate nel HHS (Home Subscriber Server). Il visited network è un qualsiasi altro network gestito da un provider diverso da quello con cui il device è abbonato.

Nell'indirect routing sono necessari tre protocolli fondamentali:

- un protocollo di associazione mobile-device-to-visited-network, che permette l'associazione alla visited network quando si entra in contatto con questa e viceversa la disassociazione quando il dispositivo lascia la visited network;
- un protocollo di registrazione visited-network-to-home-network-HSS, che permette alla visited network di registrare la posizione del device mobile dentro l'HSS della sua home network.
- un protocollo di datagram tunneling tra home network gateway e il visited network gateway routers, necessario per effettuare incapsulamento e forwarding, da parte dell'home network gateway verso la visited network gateway, e decapsulamento, traduzione NAT e forwarding da parte del visited network gateway verso il mobile device.

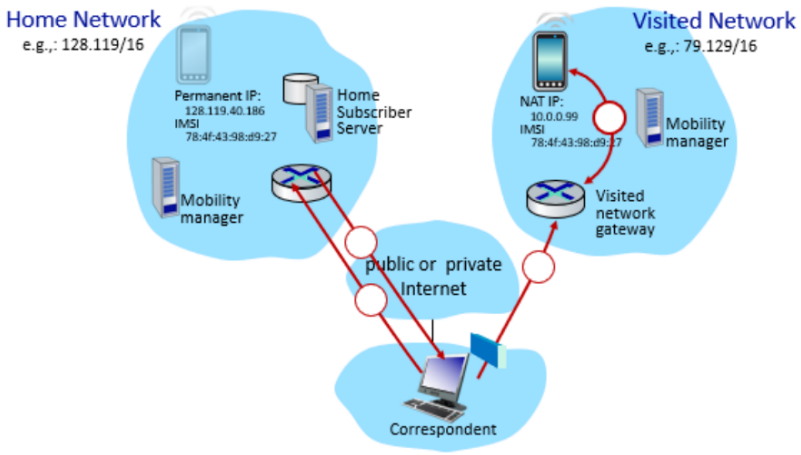
Vediamo i passi descritti nella figura:

1. Il mittente (Correspondent) utilizza l'home address del device con cui vuole comunicare come indirizzo di destinazione del datagram.
2. Il home network gateway riceve il datagram, e lo inoltra (utilizzando il precedentemente descritto protocollo di *tunneling*) al visited network gateway, dopo aver consultato il HSS per scoprire la posizione del dispositivo mobile.
3. Il visited network gateway inoltra il datagramma al dispositivo mobile (tipicamente dopo aver utilizzato NAT)
4. Il visited gateway router inoltra la risposta al mittente (Correspondant) direttamente (**b**) o utilizzando la home network (**a**).

L'indirect routing risulta inefficiente nel caso in cui un dispositivo B è un dispositivo mobile che si è spostato dalla sua home network alla stessa rete del dispositivo A (mittente, che in questo caso ha home network diversa). Nonostante entrambi i dispositivi si trovino ora nella stessa rete, l'instradamento dei messaggi dal Dispositivo A al Dispositivo B non avviene direttamente. Invece, segue questo il percorso a 'triangolo' dell'indirect routing, portando ad un overhead non necessario.

Il IMSI (o International Mobile Subscriber Identifier) viene utilizzato come una sorta di MAC address del device per poterlo identificare, insieme al suo Permanent IP assegnato dal suo home network. Per far riferimento a un mobile device durante l'invio dei messaggi, viene utilizzato il suo IP permanente come riferimento; Per far riferimento a un mobile device per billing, autenticazione, ecc.., viene utilizzato il suo IMSI come riferimento.

23.5 MN direct routing



L'immagine mostra il funzionamento del ***direct routing*** per la mobilità tra una home network e una visited network. La home network rappresenta la rete "nativa" di un dispositivo con un abbonamento ad un mobile provider (e.g. *Verizon*), dove le informazioni relative ai device abbonati vengono salvate nel HHS (Home Subscriber Server). Il visited network è un qualsiasi altro network gestito da un provider diverso da quello con cui il device è abbonato.

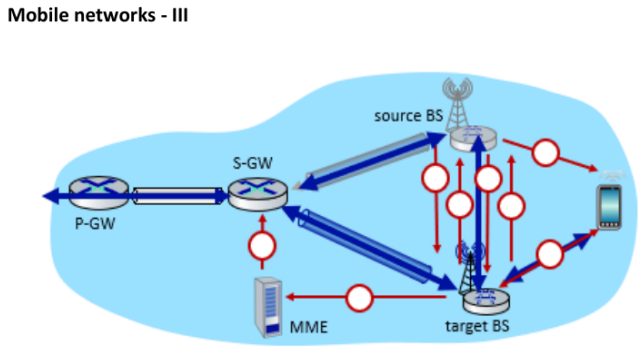
A differenza dell'indirect routing, in cui le comunicazioni tra Correspondent e dispositivo mobile passano dalla home network, utilizzando il direct routing il mittente (Correspondent) invia direttamente il datagram packet al dispositivo mobile nella visited network:

1. Il mittente (Correspondent) contatta la home network del dispositivo mobile a cui vuole mandare un messaggio attraverso il suo indirizzo IP permanente.
2. La home network risponde al mittente (dopo aver chiesto al HSS) con l'IP assegnato al dispositivo mobile nella visited network.
3. Il mittente (Correspondent) invia il messaggio al mobile device utilizzando l'IP appena ottenuto.
4. Il visited network gateway inoltra il datagramma al dispositivo mobile (tipicamente dopo aver utilizzato NAT)

Il direct routing risolve il problema del 'triangle routing', a costo però di non essere trasparente nei confronti del Correspondent, visto che deve procurarsi da solo l'indirizzo assegnato al mobile device dalla visited network. Nell'indirect routing il Correspondent non deve gestire la mobilità del dispositivo mobile (che può anche avvenire più volte in un breve periodo), rendendo l'indirect routing il protocollo più utilizzato.

23.6 MN HO tra BS nella stessa rete

L'immagine mostra l'handover di una connessione telefonica tra due base station appartenenti alla stessa rete telefonica in una rete 4G. L'handover è il processo di trasferimento di una chiamata o di una sessione dati in corso tra una base

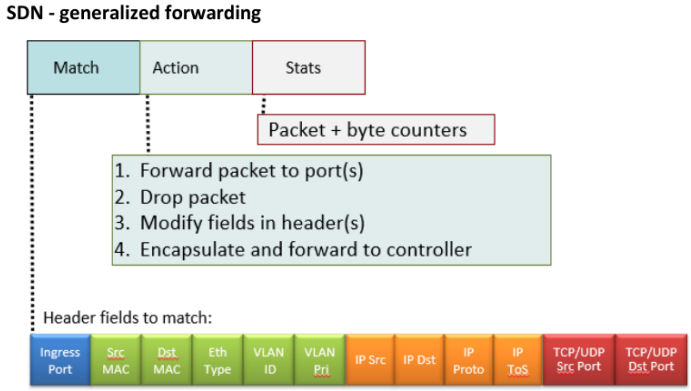


station e un'altra mantenendo la continuità di comunicazione. Questo processo è necessario quando un device si muove da una cella ad un'altra e il segnale della cella a cui era attaccato diventa troppo debole per mantenere la connessione.

In figura possiamo osservare la presenza di 7 step necessari per portare a compimento l'handover:

1. La BS corrente sceglie di selezionare un handover (per un deterioramento del segnale tra cellulare e BS o per overloading): seleziona una BS target e gli invia un **Handover Request message**;
2. La BS target pre-alloca dei radio time slot, risponde con un **Handover Request ACK** con informazioni per il dispositivo mobile;
3. La BS sorgente informa il cellulare della nuova BS che ora può essere utilizzata per l'invio dei dati e così l'handover risulta completo dal punto di vista del al dispositivo mobile;
4. La BS sorgente allora termina l'invio di datagram al cellulare e invece inoltra alla nuova BS (che a sua volta inoltra al dispositivo mobile).
5. La target BS informa il MME (Mobile Management Entity) che è lei la nuova BS per il dispositivo, così il MME istruisce il S-GW per cambiare il tunnel endpoint per essere sulla nuova BS target.
6. La target BS invia un ACK di conferma del completamento dell'handover alla BS sorgente;
7. I dati inviati dal dispositivo mobile adesso passano attraverso la nuova BS e attraversano il tunnel creatosi con il S-GW.

23.7 SDN generalized forwarding



Il generalized forwarding (approccio seguito da OpenFlow) è uno dei servizi implementati dal data plane SDN (Software-Defined Networking[1]) e consiste nel fornire ad ogni router una forwarding table (o flow table) che usa l'astrazione "match plus action[2]". Il generalized forwarding utilizza diversi campi dell'header per determinare quale azione effettuare: le azioni possibili su un dato pacchetto/flusso sono `drop`, `forward`, `modify` e `send` (al controller). La figura sopra rappresenta tutte le informazioni che possono essere usate su i diversi campi dell'header (link: verdi; network: arancioni; transport: rosse) per impostare le matching rules all'interno di una flow table.

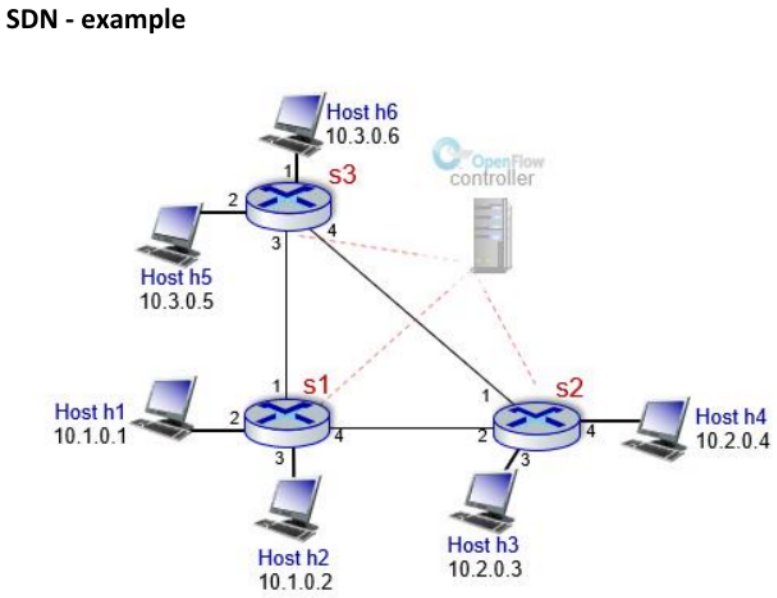
L'astrazione "match plus action" permette di unificare più tipi di device in base al loro comportamento:

- ◇ Router: il `match` è il prefisso IP più lungo, l'`action` è inoltrare verso un link
- ◇ Switch: il `match` è i MAC address di destinazione, l'`action` è l'inoltro o il flood (inoltro a tutti) del pacchetto;
- ◇ Firewall: il `match` è l'indirizzo IP e il numero di porta TCP/UDP, l'`action` permettere o negare il pacchetto;

[1]SDN: approccio dove un controller remoto calcola e manda le forwarding tables ad ogni router
[2]fare matching con i bits in arrivo, e di conseguenza effettuare quale azione inffettuare

- NAT: il `match` è l'indirizzo IP e la porta, l'`action` di sostituire un indirizzo IP privato con un indirizzo IP pubblico

23.8 SDN example



Un semplice esempio di rete SDN composta da 3 switch e 6 host in totale, l'SDN controller (nello specifico OpenFlow controller) programmerà gli switch in modo tale da inserire in ognuno di essi una flow table con tutte le regole di forwarding per la gestione del traffico di rete.

La cruciale differenza rispetto alle reti con switch tradizionali è il disaccopiamento fra control e data plane[3], che invece di coesistere nei singoli device, sono separati permettendo una gestione centralizzata, opposta alla programmazione dei singoli switch.

Nel caso si volessero scrivere delle regole per far passare per `s1` i datagrammi provenienti dagli host `h5` e `h6` e destinati agli host `h3` e `h4`, le regole necessarie sarebbero:

- `s3`:
  - `match`: IP Src = 10.3.*.*, IP Dest[4] = 10.2.*.*;`action`: forward(3);
- `s1`:
  - `match`: ingress port = 1, IP Src = 10.3.*.*, IP Dest = 10.2.*.*; `action`: forward(4);
- `s2`:
  - `match`: ingress port = 2, IP Dest = 10.2.0.3; `action`: forward(3);
  - `match`: ingress port = 2, IP Dest = 10.2.0.4; `action`: forward(4);

Alcuni dei comandi chiave di OpenFlow —discussi anche nella lezione di lab sul network slicing— sono:

- `features` richiede features allo switch
- `flowmod` aggiunge/elimina/modifica entries nelle table openflow
- `packet-in` invia un pacchetto da switch a controller per farglielo gestire
- `packet-out` specifica la porta di output di uno switch per un pacchetto, eventualmente ricevuto con `packet-in`

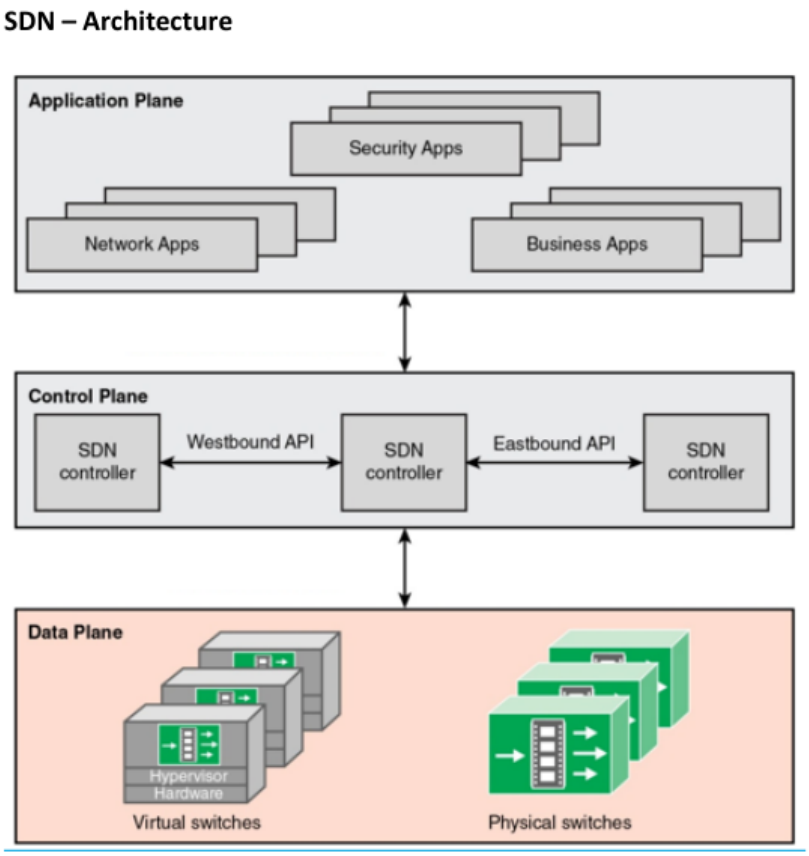
Gli ultimi due comandi sono utilizzati —ad esempio— dal protocollo `LLDP`:

1. Il controller genera un pacchetto LLDP per ogni porta attiva di ogni switch e lo invia con `packet-out`.
2. Uno switch che riceve tale pacchetto dal controller lo inoltra agli switch adiacenti
3. Uno switch che riceve tale pacchetto da un altro switch, lo incapsula in pacchetto LLDP e lo invia al controller con `packet-in`, aggiungendo `switchID`, `portID` e altri metadata.
4. Il controller può ricostruire i link esistenti fra gli switch

23.9 SDN architecture

L'immagine mostra l'architettura del SDN, partendo dal data plane, dove avviene il forwarding dei frame, passando per il control plane, dove avvengono le decisioni su dove indirizzare il traffico, fino ad arrivare all'application plane, che astrae dal control plane permettendo una gestione più intuitiva —tipicamente con un approccio dichiarativo— della rete.

[3] Rispettivamente decidere dove il traffico deve essere indirizzato e indirizzarlo verso la giusta destinazione
[4] Superfluo in questa specifica topologia, ma comunque opportuno metterlo



L'architettura del data plane (quella evidenziata in figura) è composta principalmente da switch fisici e switch virtuali: ogni switch deve implementare un modello, o astrazione, di forwarding dei pacchetti basato sulle istruzioni fornite dai SDN controllers.
Il controller dialoga con gli switch del data plane attraverso un protocollo standardizzato denominato *OpenFlow*. Questo protocollo consente al controller, attraverso comandi inviati su livello 2, di programmare gli switch affinché eseguano determinate azioni: in particolare abbiamo discusso del pattern `match + action`, ma non ci si limita a quello.
Alcune key features del data plane sono:

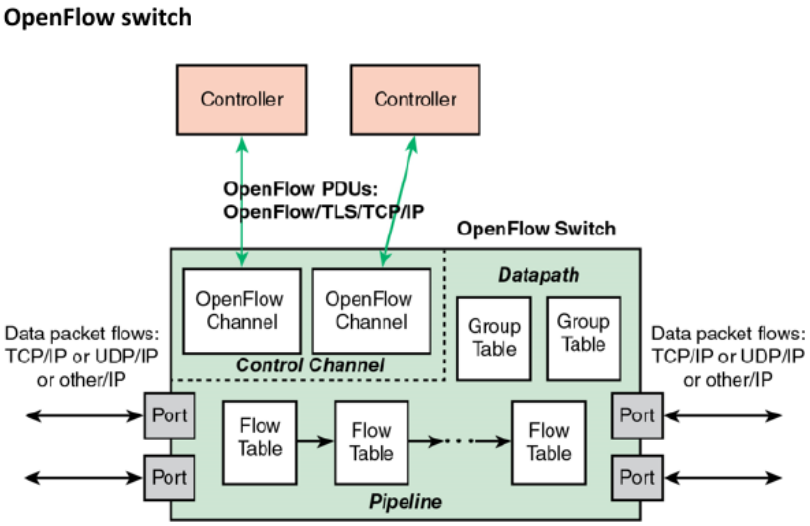
- ◇ **Packet forwarding**: sono responsabili dell'inoltro pacchetti in base alle regole fornite dal controller.
- ◇ **Traffic shaping**: possono implementare politiche di traffic shaping per gestire il flow del network (e.g. dare priorità ad un certo tipo di traffico, come quelli voce o video, in modo che traffico prioritario sia inviato senza delay)
- ◇ **Security**: può implementare politiche di security shaping per prevenire accessi non autorizzati al network (e.g. filtrare pacchetti in arrivo da indirizzi notoriamente pericolosi)
- ◇ **QoS** (Quality of Service): può implementare politiche di QoS per assicurarsi che un certo tipo di traffico arrivi con un determinato grado di qualità.
- ◇ **Load balancing**: può implementare politiche di distribuzione del traffico per evitare congestioni e assicurarsi che il traffico sia efficientemente distribuito.

Il SDN è definito su una serie di API che permettono l'interfacciamento con i vari livelli: dal control plane al data plane si parla di southbound API (e.g. OpenFlow), e permettono al control plane di ottenere informazioni sul traffico, e di conseguenza specificare le politiche di forwarding agli switches presenti nel data plane. I controller espongono northbound API, permettendo a sviluppatori e a network managers di "applicare" applicazioni network personalizzate.

23.10 OpenFlow switch

Gli OpenFlow switch sono dispositivi di rete (fisici o virtuali) programmabili da OpenFlow controller, utilizzando comandi OpenFlow. Essi utilizzano una serie di flow table per gestire il traffico di rete, e sono formati dai seguenti componenti:

- ◇ Una serie di flow table organizzate in **pipeline**.
- ◇ Una serie di **group table** all'interno del datapath, che permettono di specificare un comportamento più complesso che si riferisce a gruppi di flusso (stabilendo regole basate sugli attributi dei flussi).
- ◇ Gli OpenFlow Channels supportano la comunicazione con i controller nel control plane. Tipicamente gli switch



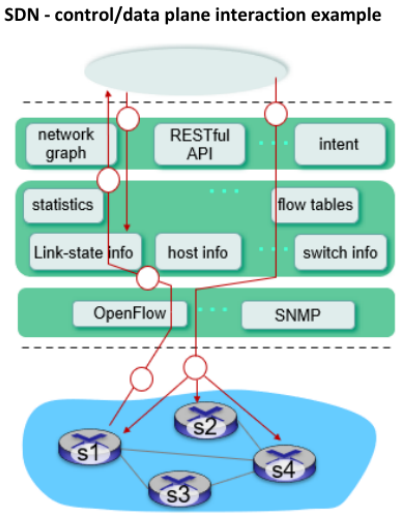
sono collegati a più controllers, dato che è possibile avere una serie di controlli distribuiti. La configurazione più semplice è quella dello master-slave, con un solo controller che comunica con lo switch, altre più complesse fanno uso di più controller con ridondanza "active-passive".

- ◇ Ports: questi switch hanno diversi tipi di porte, tra cui porte fisiche, porte logiche (versioni virtualizzate di quelle fisiche) e porte riservate.

Le flow table sono disposte a pipeline all'interno dello switch. Quando un pacchetto entra nella pipeline, inizierà a scorrere la prima tabelle fino a quando non troverà un match in una delle tabelle. Se ci sono più match in una table, verrà selezionato quello con priorità più alta. Se l'unico match è quello di *table-miss entry*[5], allora una delle tre azioni è possibile:

- ◇ passare il pacchetto alla successiva flow table (se presente).
- ◇ delega al controller: il pacchetto viene delegato al controller, che in base alle politiche a lui assegnate deciderà se creare un nuovo flow per quel pacchetto, oppure dropparlo.
- ◇ droppare il pacchetto.

23.11 Interazione control plane/data plane



Il controller è suddiviso in:

- ◇ API di astrazione per interfacciarsi con le applicazioni di network control.
- ◇ Database distribuito su stato di link, switch, servizi, ecc...
- ◇ Componenti comunicazione con gli switch

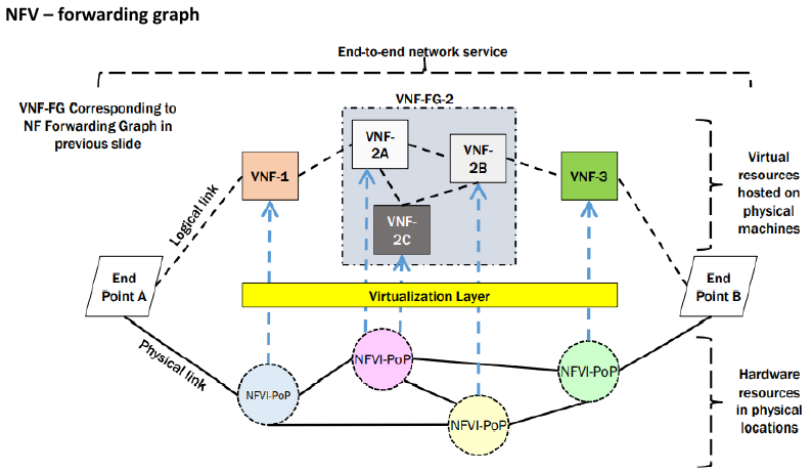
L'immagine mostra un esempio di interazione tra il control plane e il data plane nel caso in cui uno switch rilevi un

[5] match di default che specifica il comportamento dello switch in caso un pacchetto non effettui un match con nessun'altra regola

errore in uno dei suoi collegamenti (il link da `s1` a `s2`). Tale interazione è composta da 6 step totali:

1. Lo switch S1 ha un errore di collegamento (poiché ha perso il link a S2), quindi invia un OpenFlow `port-status` message per notificare il controller;
2. SDN Controller riceve il messaggio e aggiorna le informazioni di stato di quel link;
3. Ipotizzando che l'applicazione che implementa l'algoritmo di Dijkstra per il routing sia eseguita fuori dal controller, sarà compito del controller avvisare tale applicazione del nuovo stato della topologia di rete.
4. L'applicazione accede alle informazioni del grafo di rete, alle informazioni dello stato dei link e calcola nuove rotte;
5. L'applicazione di link state routing interagisce con il componente che produce le flow-table nel controllore SDN, che calcolerà le nuove flow table;
6. Il controller infine usa OpenFlow per installare le nuove flow-table negli switch che devono essere aggiornati inoltrando un `flow-mod` message agli switch

23.12 NFV forwarding graph



Il crescente utilizzo di internet e di servizi che richiedono connessioni brevi ad alta velocità hanno portato alla necessità di acquisto di hardware specializzato per ognuna delle funzioni richieste, portanto ad una distribuzione ad alta densità di infrastrutture network. Gli operatori di rete stanno progressivamente passando da soluzioni basate solo su hardware a soluzioni basate anche su software. La NFV (Network Function Virtualization) affronta questi problemi sfruttando le tecnologie di virtualizzazione: permette di **disaccoppiare** l'hardware dal software, consentendo di migrare ed eseguire funzioni di rete dove necessario.

Un network service può essere scomposto in un insieme di *Virtual Network Functions* (VNF) (e.g. Firewalls, IDS, load balancers, ...)pu, quindi può essere anche visto come un forwarding graph di network function e endpoint. I link logici che interconnettono ogni network function possono essere gestiti tramite controllori SDN e devono essere supportati da path fisici attraverso l'infrastruttura di rete sottostante.

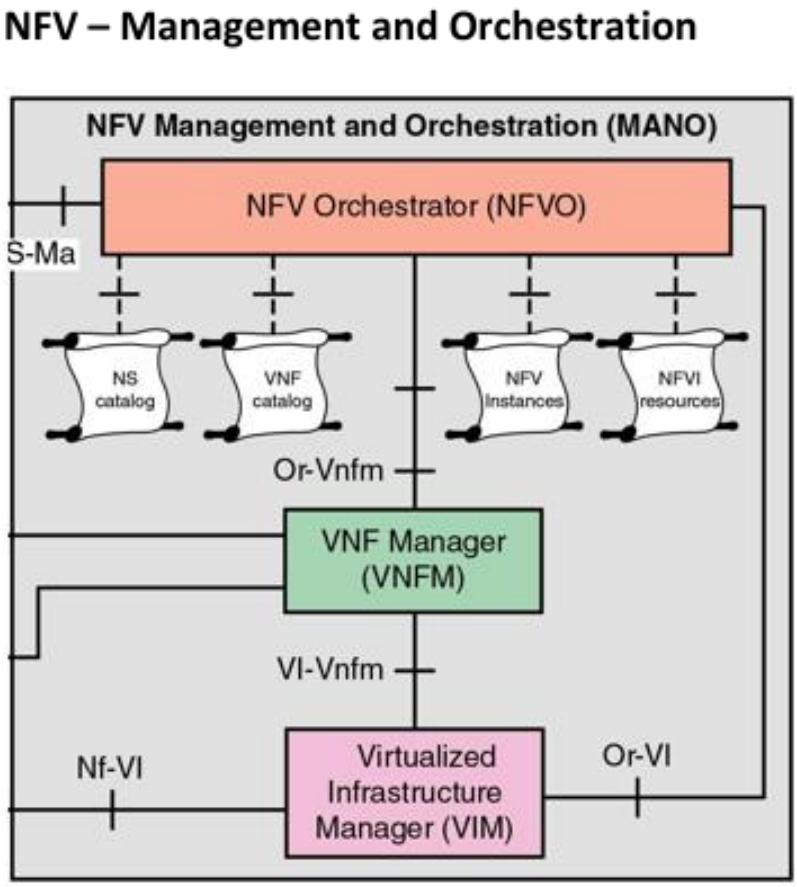
Ogni network service può essere implementato da un singolo operatore o da una rete di più operatori.

Un network service può essere composto da una o più VNF che sono deployate attraverso l'uso di VM (Virtual Machine) o container (in questo caso si parlerebbe di CNF (Containerized Network Function)), i quali comunque devono essere ospitati da una macchina fisica, detta *Point of Presence* (**PoP**). Il problema del **collocamento** delle VNF non è banale e deve tenere conto di QoS (latency e min bandwidth), performance, bandwidth, risorse PoP, massimizzando una *utility function*. L'ottimalità può essere definita in base a vari fattori, come latenza complessiva, costo deployment, bandwidth utilizzata... Non c'è una soluzione standard al problema, i requisiti possono essere variabili e gli approcci per risolverlo altrettanto diversificati, e.g. algoritmi genetici, ML, programmazione lineare.

Un esempio applicabile all'immagine potrebbe essere: `VFN-1` servizio di cache posizionato vicino all'end point (quindi sull'edge della rete). È possibile che una VNF sia composta da diverse VNF, come accade in figura per `VFN-FG-2`. È possibile che le performance di un servizio possano benificiare da una corrispondenza fra link logici e fisici: un servizio di cache o firewall istanziato sul nodo fisico più lontano dall'endpoint ingress può significare multipli hop per dati che dovrebbero invece avere un rapido accesso.

Grazie alla virtualizzazione è possibile migrare e istanziare dinamicamente VNF e allocare risorse a seconda delle necessità, ed eventualmente automatizzare il tutto grazie ad un orchestrator.

23.13 NFV MANO



La figura mostra la struttura interna di un NFV MANO formato da diverse componenti, le principali delle quali sono: NFVO (o NFV Orchestrator), VNFM (o VNF Manager) e VIM (o Virtual Infrastructure Manager). Questi componenti forniscono le funzionalità richieste per istanzare i VNFs, fornendo le configurazioni dei VNFs e le configurazioni all'infrastuttura in cui i VNFs verranno eseguiti.

Utilizzando un approccio top-bottom troviamo:

- il NFV (NFV Orchestrator), responsabile della creazione, dell'installazione, della configurazione, e del lifecycle dei network services, composti a loro volta da pacchetti di VNF. Questo componente fa affidamento a delle repositories per ottenere informazioni sui servizi:
  - il *NS Catalog*, è una repository che contiene una lista di network services utilizzabili (con relativi deployment templates per NS in termini di VNFs)
  - il *VNF Catalog*, una repository contenente tutti i VNFD (descrittori che descrivono le funzioni da un punto di vista delle risorse necessarie per poter istanziare tale funzione)
  - *NFVI* (Network Function Virtualization Infrastructure) *resources*, repository con lista di risorse necessarie per costruire e mantenere l'infrastruttura NFV e i servizi attivi.
  - *NFV instances*, repository con lista contenente dettagli relativi al deployment dei network services e corrispondenti VNF.

  L'orchestrator si occupa anche di comunicare con componenti hardware di rete tradizionali

  Il collegamento `Se-Ma` permette di far arrivare le richieste di network service da un manifesto contenente i dettagli richiesti di rete?.
- il VNFM (VNF Manager), responsabile della gestione del ciclo di vita di ogni singola VNF. Si occupa di istanziazione, aggiornamento, query, di aumentare o diminuire il numero di istanze (scale up/down) e infine di terminare una VNFI (VNF Instance). È anche il responsabile della raccolta di metriche delle VNFs, e di informazioni legate ad eventuali fault/errori.
- il VIM (Virtualized Infrastructure Manager), responsabile del controllo e della gestione dell'interazione di una VNF con le risorse di computing, storage e rete dell'infrastruttura fisica sotto il suo controllo (e.g. per un network service serve un set di VM con determinate caratteristiche). Una singola istanza di VIM è in grado di gestire un intero dominio d'infrastruttura[6] sotto il controllo di un operatore. Per amministrare l'intera rete, potrebbero essere necessarie più istanze di VIM in un singolo MANO.

[6] set di risorse fisiche e virtuali gestite da un operatore di rete in una specifica area

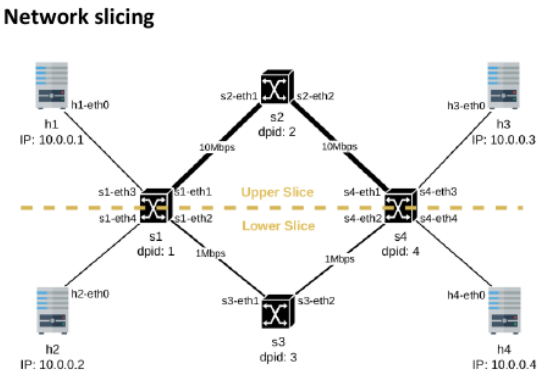
23.14 Esempio di network slicing

Il network slicing è una tecnica utilizzata nelle reti SDN (Software Defined Network) per isolare le risorse di rete e soddisfare diverse esigenze su un'infrastruttura fisica condivisa.

Questa immagine illustra l'implementazione del network slicing in un SDN per consentire l'isolamento delle risorse di rete. L'obiettivo di questo esempio è mostrare che diverse necessità possono essere soddisfatte utilizzando il network slicing su un'infrastruttura fisica condivisa. L'architettura impiega una topologia multi-hop: la rete comprende quattro host (h1, h2, h3, h4) e quattro switch (s1, s2, s3, s4). La rete deve essere isolata in due slice per due servizi:

- Slice del traffico video (upper slice): Il traffico video (porta UDP 9999) può utilizzare un massimo di 10 Mbps di larghezza di banda. Il diritto del traffico video di utilizzare questa larghezza di banda non è influenzato dagli altri tipi di traffico.
- Slice per altri tipi di traffico (lower slice): Tutti i traffici, che non siano di tipo UDP porta 9999, possono utilizzare qualsiasi percorso della rete, ma non devono influenzare il traffico video.

Tutti gli switch sono collegati (anche se non mostrato in figura) ad un controller.




```python
def __init__(self, *args, **kwargs):
    super(TrafficSlicing, self).__init__(*args, **kwargs)

    # out_port = slice_to_port[dpid][in_port]
    self.slice_to_port = {
        1: {1: 3, 3: 1, 2: 4, 4: 2},
        4: {1: 3, 3: 1, 2: 4, 4: 2},
        2: {1: 2, 2: 1},
        3: {1: 2, 2: 1},
    }

@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    msg = ev.msg
    datapath = msg.datapath
    in_port = msg.match["in_port"]
    dpid = datapath.id
    out_port = self.slice_to_port[dpid][in_port]

    actions = [datapath.ofproto_parser.OFPActionOutput(out_port)]
    match = datapath.ofproto_parser.OFPMatch(in_port=in_port)
    # add_flow(self, datapath, priority, match, actions)
    self.add_flow(datapath, 1, match, actions)
    self._send_package(msg, datapath, in_port, actions)
```

21

```python
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
    datapath = ev.msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    # install the table-miss flow entry.
    match = parser.OFPMatch()
    actions = [
        parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
        ofproto.OFPCML_NO_BUFFER)]
    self.add_flow(datapath, 0, match, actions)

def add_flow(self, datapath, priority, match, actions):
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    # construct flow_mod message and send it.
    inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
        actions)]
    mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
          match=match, instructions=inst)
    datapath.send_msg(mod)
```

Il codice mostrato definisce il comportamento del *controller* in alcune situazioni.

Nel primo snippet, c'è una entry di un dizionario per ciascuno switch, e per ciascuna di queste il mapping delle porte che definisce deve essere inoltrato il traffico. In sostanza qui si definiscono le informazioni necessarie per generare le regole di flusso, che dovranno poi essere applicate agli switch.

La topologia viene implementata nel file network.py, e definisce hosts, switch, e collegamenti fisici.

Un'applicazione Ryu può registrare l'ascolto di specifici eventi, con handler da lanciare una volta rilevato l'evento. Il primo evento descritto nell'immagine è quello di packet in (`EventOfPacketIn`), quindi in `_packet_in_handler` viene definito il comportamento per un pacchetto in arrivo. Il controller riceve dagli switch un *packet in* message, permettendogli di sapere ID switch e porta ingress del pacchetto, e dunque di comunicare allo switch il flow e le azioni da eseguire utilizzando la funzione `add_flow`, in base allo slicing del primo snippet. `_send_package` invia allo switch un *packet out* message per gestire lo specifico pacchetto che lo switch non aveva saputo gestire, e per cui quindi aveva

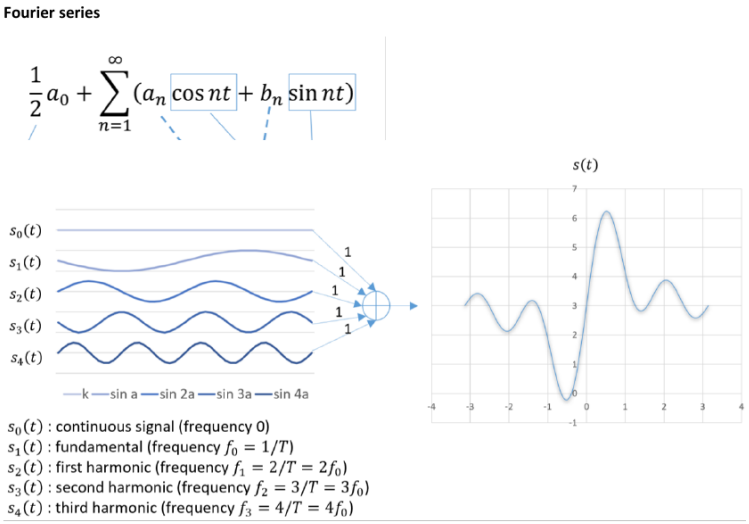
invia un *packet in* message al controller.

In questo modo, la prima volta che arriva un pacchetto, lo switch lo invia al controller il quale determina su quale porta dello switch debba essere inoltrato (in base alla topologia definita) e glielo comunica. Poi lo switch manderà autonomamente pacchetti con tale ingress port sulla porta output indicata.

`switch_features_handler` serve per gestire l'evento di ricezione di un messaggio di *feature reply* (`EventOfPSwitchFeatures`), mandato in risposta dagli switch al periodico *feature request* dei controller. Lo snippet, alla ricezione di tale messaggio, installa nello switch una entry con un `flow_mod` message per indicare allow switch di propagare al controller la gestione dei pacchetti qualora non trovasse un match nelle sue table.
In sostanza `match` = *nessuna entry disponibile* + `action` = *inoltra a controller*.

`add_flow` costruisce un messaggio `flow_mod` in base ai parametri e poi lo invia a `datapath`, che è un oggetto che incapsula l'ID e l'address dello switch destinatario.

23.15 Serie di Fourier



L'immagine in figura mostra la definizione della serie di Fourier, utilizzata per la decomposizione di un segnale continuo e periodico (in $[-\pi, \pi]$) nei suoi armonici (che sono generalmente funzioni trigonometriche che oscillano a diverse frequenze). Possiamo vedere come $s(t)$ a destra sia formato dalla composizione delle 4 armoniche fondamentali e da una componente costante $s_0(t)$. Sopra abbiamo invece l'equazione che definisce le serie di Fourier per $s(t)$ definita da $\mathbb{R} \to \mathbb{R}$ nell'intervallo $[-\pi, \pi]$: è utilizzata per decomporre il segnale in una somma di funzioni continue infinite (periodiche e che oscillano a diverse frequenze), dove:

$$a_0 = \frac{1}{\pi}\int_{-\pi}^{\pi} s(t)dt \tag{23.1}$$

$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} s(t)cos(nt)dt \tag{23.2}$$

$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} s(t)sin(nt)dt \tag{23.3}$$

$$\tag{23.4}$$

a_0 è la componente costante, $sin(nt)$ e $cos(nt)$ sono gli armonici, mentre a_n e b_n sono le rispettive ampiezze degli armonici. Non sono conosciute le condizioni *necessarie* per cui $s(t)$ possa essere sviluppata in una serie di Fourier, ma esistono due condizioni *sufficienti* (*Dirichlet*):

- ◇ $s(t)$ sia periodica
- ◇ $s(t)$ sia continua a tratti

⇒ ed è possibile affermare che in questo modo la serie di Fourier di $s(t)$ esiste e converge in $\mathbb{R}$

La serie di Fourier può anche essere scritta con base esponenziale, sfruttando l'esponenziale di eulero (periodo $T = 1/F$):

$$s(t) = \sum_{n=-\infty}^{+\infty} S_n e^{j2\pi nFt} = \sum_{-\infty}^{+\infty} S_n(cos(\pi nFt) + jsin(\pi nFt)), \text{ con } S_n = \frac{1}{T}\int_0^T s(t)e^{-j2\pi nft}dt$$

S_n rappresenta il contributo (*ampiezza*) di ciascuna funzione $e^{-j2\pi nft}$ al nostro segnale di partenza $s(t)$

Qualora un segnale fosse non periodico non se ne può ottenere la serie di Fourier, in quanto per un tale segnale avremmo che il periodo tendente ad infinito $T \to \infty$ e di conseguenza una frequenza fondamentale $F = 1/T \to 0$, portando gli armonici (multipli di F) ad essere infinitamente vicini gli uni agli altri. In tal caso si può applicare la ***Trasformata di Fourier Continua***, che restituisce lo *Spettro* $S(f)$ del segnale, indicante come il segnale è distribuito nel campo delle frequenze.

$$S(f) = \int_{-\infty}^{+\infty} s(t)e^{-j2\pi nft}dt, \text{ con } s(t) \text{ segnale non periodico: } s(t) = \int_{-\infty}^{+\infty} S(f)e^{j2\pi nft}dt$$

23.16 DFT (Discrete Fourier Trasform)

DFT

$$S_f = \sum_{n=0}^{N-1} s_n e^{-j\frac{2\pi f}{N}n} \; with \; f = 0, 1, \ldots, N-1$$

Abbiamo visto come la serie di Fourier permetta di scrivere un segnale continuo e periodico (in un intervallo) come la somma di segnali, o meglio, *armonici*. La trasformata di Fourier ci permette di ottenere la serie ordinata di coefficienti di ampiezza degli armonici S_n, detta *Spettro* del segnale. Il valore assoluto di $|S_f|$ rappresenta l'ampiezza dell'armonico di frequenza $f = 1/N$, con f frequenza fondamentale.

Ciascun armonico è la somma di *sin* e *cos*, ciascuno moltiplicato per un coefficiente indicante l'*ampiezza* (in modo informale il "peso" dell'armonico). La serie di Fourier può anche essere scritta con base esponenziale, sfruttando l'esponenziale di eulero (periodo $T = 1/F$:

$$s(t) = \sum_{n=-\infty}^{+\infty} S_n e^{j2\pi nFt} = \sum_{-\infty}^{+\infty} S_n(cos(\pi nFt) + jsin(\pi nFt))$$

$$s(t) = \sum_{n=-\infty}^{+\infty} S_n e^{j2\pi nFt} \quad S_n = \frac{1}{T}\int_0^T s(t)e^{-j2\pi nFt}dt$$

In effetti $e^{j\Theta} = cos(\Theta) + jsin(\Theta)$

La trasformata di Fourier di un segnale continuo ma *non periodico* sfrutta l'esponenziale di Eulero, e prevede di integrare da $-\infty$ a $+\infty$; ciò avviene perché per rappresentare un segnale non periodico diciamo che abbia periodo $T \to \infty$, e che dunque gli armonici siano potenzialmente infiniti. Dunque il risultato fornisce non una serie di coefficienti, bensì lo *spettro* $S(f)$ del segnale continuo $s(f)$:

$$S(f) = \int_{-\infty}^{+\infty} s(t)e^{-j2\pi ft}dt \tag{23.5}$$

Per segnali non periodici, la situazione si complica un po'.

La funzione in figura rappresenta la definizione di una trasformata di Fourier discreta, da applicare a un discreto non continuo. Tale segnale si ottiene tramite il *sampling* di un segnale continuo, che fornisce una serie di osservazioni separate da un intervallo di tempo T.

Dato che tali osservazioni sono "puntiformi" nel grafico, si assume che ogni sample abbia un *"impulso"* (i.e. un rettangolo nel piano), dunque invece di calcolare un integrale come in 23.5, si scompone tale integrale in una sommatoria che arriva fino ad N, ovvero il numero di osservazioni.

$$S(f) = \int_{-\infty}^{+\infty} s(t)e^{-j2\pi ft}dt = s(0)e^{-j2\pi f0} + s(1)e^{-j2\pi f1} + \ldots + s(N-1)e^{-j2\pi f(N-1)}$$

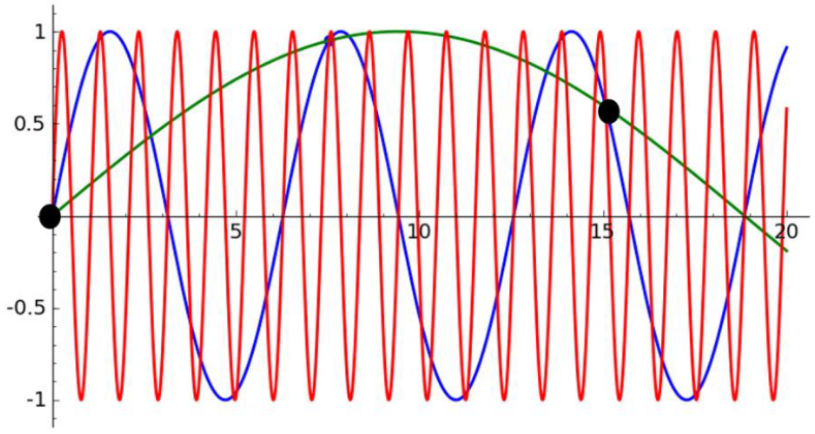
$$\text{i.e. } S(f) = \sum_{0}^{N-1} s(t)e^{-j2\pi fn}, n \in \{0, 1, \ldots, N-1\}$$

Dato che i sample sono divisi da un tempo $T = 1/N$ sono periodici, prendiamo f non in $\mathbb{R}$, bensì nell'insieme della frequenza fondamentale $1/N$ e dei suoi multipli (armonici) $f = 0, 1/N, 2/N, \ldots, N-1/N$.
Per comodità di notazione nella formula riportata, $f = 0, 1, \ldots, N-1$ e nell'esponente si divide per N.

Quindi considerando tutte queste assunzioni, la formula nell'immagine mostra come è possibilie calcolare la trasformata di Fourier discreta, a partire da un numero finito di sample di un segnale. La formula inversa per ottenere i valori dei sample è la seguente:

$$s_n = \frac{1}{N} \sum_{0}^{N-1} S_f e^{j2\pi f \frac{n}{N}}$$

23.17 Sampling and Aliasing



Ci sono due tipi di errori che rendono la trasformata di Fourier discreta diversa rispetto allo spettro del vero segnale sul quale sono stati effettuati i samples:

- ◇ Aliasing: non utilizzare un rate di sampling appropriato per un segnale che varia molto frequentemente in un periodo potrebbe portare a difficoltà nel ricostruirlo e nell'analisi del comportamento della sua frequenza. Per esempio, dati dei sample, applicando la FT è possibile risalire a un segnale che abbia come spettro anche una sola frequenza che corrisponda al segnale campionato, quando invece questi potrebbe essere la somma di più armonici, oppure semplicemente ad un armonico della frequenza fondamentale ($s(t)$ e $3s(t)$ potrebbero apparire uguali da campionati).
  Nell'immagine le tre sinusoidi differiscono di $1/6$, i.e. $f' = f/6$, e matchano tutte i campioni neri, ma sono tre segnali diversi!
- ◇ Leakage: durante il sampling di un segnale periodico, può accadere che il primo e l'ultimo sample del periodo abbiano valori diversi (presumibilmente a causa di un'interruzione di sampling non avvenuta a fine periodo). Questo porta a discontinuità tra gli end-point dei sampling periodici, creando possibili false ricostruzioni.
  Per arginare il problema, una soluzione potrebbe essere quella di far tendere a 0 il primo e l'ultimo sample.

Un segnale discreto $g(nT)$ si può ottenere con un sampling uniforme di un segnale continuo $s(t)$:

$$g(nT) = s(nT)$$

Dove T è il periodo di sampling mentre $F = 1/T$ la frequenza di sampling.

Sapendo che f_M è il limite oltre il quale lo *spettro* di $s(t)$ è nullo, il Sampling Theorem ci dice che $s(t)$ è rappresentata dai suoi campionamenti se:

1. essi avvengono in intervalli regolari $t_n = nT_c$ (con $n \in \mathbb{Z}$)
2. con un periodo di sampling di $T_c \leq \frac{1}{2f_M}$, o detto in altri termini, la frequenza minima di campionamento deve essere:

$$f_{c_{min}} = \frac{1}{T_{c_{max}}} = 2f_M \rightarrow \textit{frequenza di Nyquist} \tag{23.6}$$

In altre parole, $f_{c_{min}}$ è il doppio della frequenza più alta in $s(t)$. Soddisfatte queste condizioni, il segnale originale può essere ricostruito tramite interpolazione.

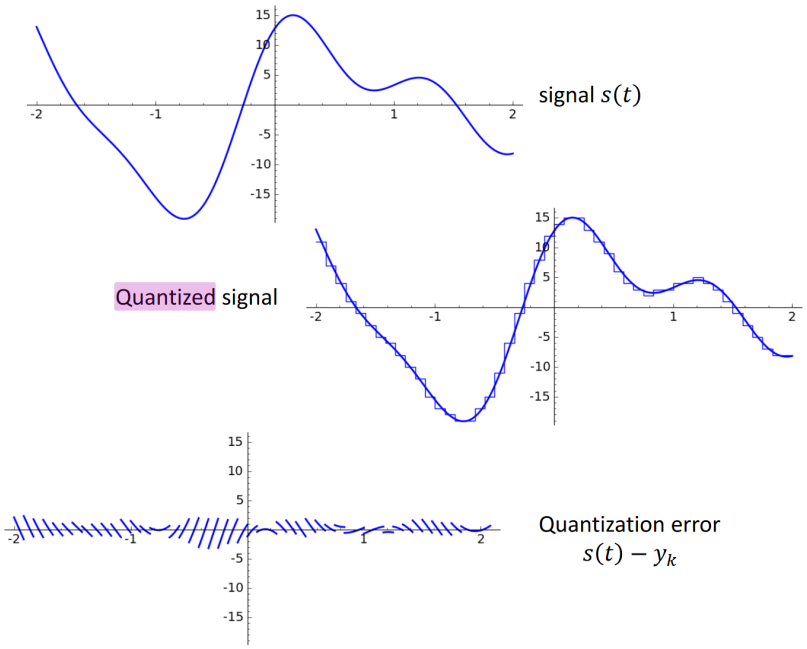
Il motivo per cui Nyquist richiede che i segnali abbiano un tetto massimo di frequenza è che ciò permette di evitare di considerare frequenze più elevate che matchino i campioni osservati. Il vero limite di Nyquist, più della frequenza di sampling richiesta, è proprio richiedere l'esistenza di tale tetto massimo sulla frequenza del segnale, difficile da riscontrare nel mondo fisico (nel mondo reale la situazione raramente è "ideale"): nessun segnale è perfettamente limitato in banda. Anche se un segnale è limitato in frequenza, non è possibile campionarlo e ricostruirlo perfettamente a meno di non prendere un numero infinito di campioni.

Il teorema stabilisce una frequenza minima di campionamento, ma non il numero di campioni necessari, il che implica che un numero finito di campioni non può garantire una ricostruzione perfetta.

Un problema che potrebbe avvenire se il tetto massimo di frequenza non è selezionato correttamente è quello di ricostruzione non accurata. Questo perché la distanza tra il centro di una replica e il centro di un'altra è proprio la frequenza di campionamento, quindi più questa frequenza è alta, più le repliche del segnale periodico sono distanti una dall'altra.

Detto ciò, il teorema fornisce comunque una buona base (periodo di sampling) per una ricostruzione che è sufficientemente accurata per le necessità applicative, considerando che comunque esistono segnali che sono quasi limitati in banda (e.g. voce). In caso è anche possibile applicare dei filtri (non esistono filtri ideali, ci sarà sempre un margine di errore) per eliminare delle frequenze sopra una certa soglia, in modo da soddisfare meglio i requisiti del teorema.

23.18 Quantization



Quantizzare con un "quantizzatore scalare a R-bit". significa approssimare il valore di un sample x_k in un intero y_k rappresentabile nel range $[0, 2^R - 1]$ (e.g. Arduino usa 10 bits). Questo è necessario perché la rappresentazione dei sample in floating point (4 o 8 byte) potrebbe essere non possibili a causa di restrizioni hardware (caso comune in IoT). Più bit sono disponibili, più la risoluzione della quantizzazione è alta. Applicando una quantizzazione uniforme, la risoluzione di un segnale con valori tra $[0, M]$ è di $\frac{M}{R^2}$.

Chiaramente tale approssimazione porta un errore nella nella rappresentazione del sample —e nell'eventuale ricostruzione del segnale—, che in casi estremi può sfociare nell'**overload** ($s(t) > 2^R - 1$) o nella mancata osservazione di **rumore granulare**, i.e. il segnale varia talmente poco in un intervallo I_k che viene rappresentato con un unico y_k.