University of Pisa

Department of Computer Science

Master Degree in Computer Science

Notes of Mobile and Cyber-Physical Systems

Based on the lectures of prof. S. Chessa, F. Paganelli

A.Y. 2022-2023

Contents

This notes have been written in Obsidian and automatically converted to LaTeX: there are several type of errors, missing labels and caption, wrong/absence of references and both grammar and conceptual errors.
If you're one of those shits that sells notes, Richard Stallman is looking at you.

Chapter 1

IoT & Smart Environment

There is no universal accepted definition of *Smart Environments*. According to Journal of Ambient Intelligence and Smart Environment:

> *"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****"*

They are considered smart for the final user because can **recognize context, situations and activities**, **able to figure out use need at right time** and **provide services**. They also have in common several characteristic like **persasive**, **unobtrusive cyber-physical devices**.

With the term IoT or **Internet of Things** we mean physical objects embedded with electronics, software and sensor, with **network connectivity**. Generally, they are objects with **sensors** and **actuators** that respectively represent the two flow of data in-and-out of the IoT system because **sensors** can receive input data and elaborate them by resulting in a performed physical action on the physical environment by **actuators**.

IoT Devices Each IoT device can be seen as composed by 4 main components:

- **Sensors/actuators**
- **Microcontroller**
- **Wireless interface**
- **Software for business logic**

We can identify in IoT a *layered architecture* pictured in 1.1 And, by istantiating this model we can

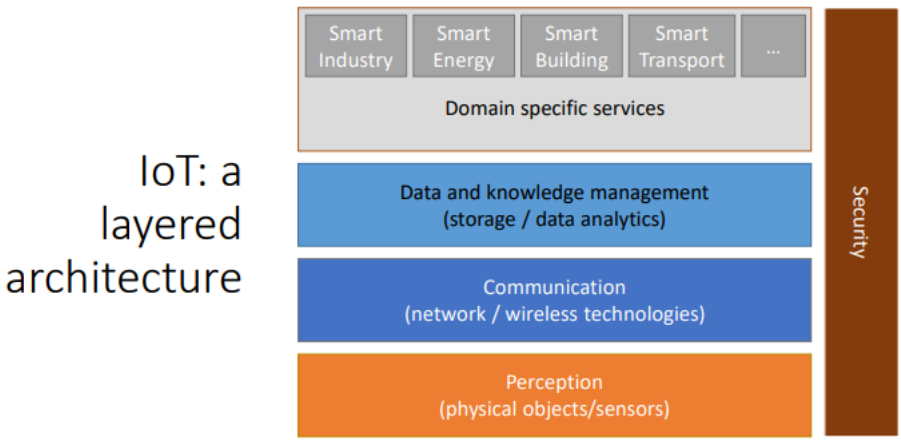


Figure 1.1: IoT layered architecture

obtain the following scenario 1.2.

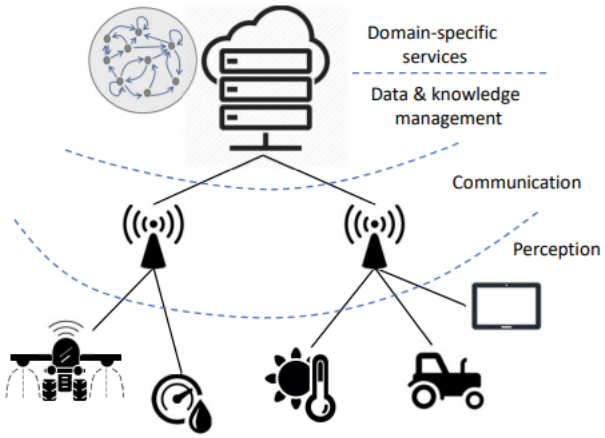


Figure 1.2: Simple IoT schema device

Sensors at **perception layer** can be of different type based on the nature of activity that need to be carried out:

- Specific: they're a fully-functional component and not need to be istantiate along other components to accomplish the desiderated task
- Aspecific: they're part of a chain of interaction with other component and this interaction (*e.g. Camera that effect facial recognition task*) allows to perform the intended task

Platforms for IoT Sensors and actuators are the edge of the cloud: behind internet, data is stored, processed and presented in the cloud. So **IoT Platforms** provide software layer between IoT devices and applications. Their functionalities may be distributed between devices themself, gateways and server in the cloud or at the edge. Those platform not allow only data collection but can perform several complex functionalities with a various range of activities (*as mentioned, facial recognition can be a service offered/performed via AWS SageMaker connected to AWS IoT*). Those platforms allows a wide range of functionalities:

- **Identification**: provides a unique way to identify things in the platform. There are several established standard, based on a given context:
  - *IP Address*: for Internet
  - *MSISDN – Mobile Station International Subscriber Directory Number*: for telephony identification
  - *URI - Universal Resource Identifier*: on resources exposed to web
  - *UUID - Universally Unique Identifier*: in computer system and bluetooth-based communication system
- **Discovery**:A mean to find devices, resource and/or services within an IoT deployment inside the same network of devices. This allows also to obtain properties, features and services and locate via identification.
- **Device Management**: as *initial setting* allows to pair, secure setting and key distribution, manage configuration and binding of devices with services, calibrate the sensors, localize devices etc. Managing involves also **Automatic Software Updates** (*both for software and firmware*), **device real-time monitoring** (*battery level, internal tempeture, energy consumption*), **diagnostic/default detection**, **remote control of devices** (*by activate pheriperals and remote rebooting, without human interaction*) and **system logging and audit management**. Clearly there are different standard based on a given context. The most popular are:
  - **OMA DM (Open Mobile Alliance Device Management)**: mainly for management activity of mobile terminals
  - **OMA LWM2M (Open Mobile Alliance Lightweight Machine2Machine**: used for management of IoT devices

  - **BBF (Broadband Forum) TR-069**: management for end-users appliances (*e.g. DSL terminals, etc.*)

- **Abstraction/Virtualization**: an IoT device is seen as a service so the physical device is associated with its *digital twin*. This provide a way to represent IoT devices and their context, enabling a broad of function on the virtual-device like reasoning and AI-processing over data.

- **Service Composition**: build a composite service by integrating services of different IoT devices and SW components as sketched in 1.3

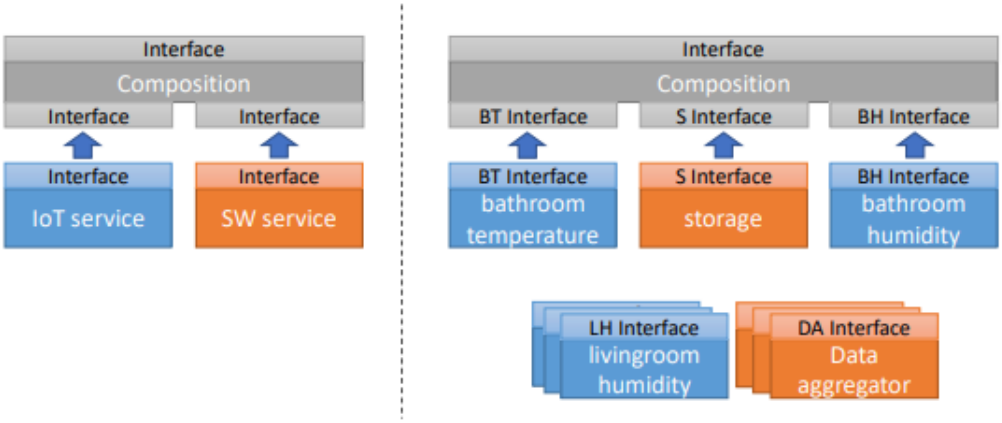


Figure 1.3: Example of different service composition

Based on the previous layered architecture model, we add the details just mentioned and provide a general picture on how those are distributed among the layers, obtaining the model pictured in 1.4

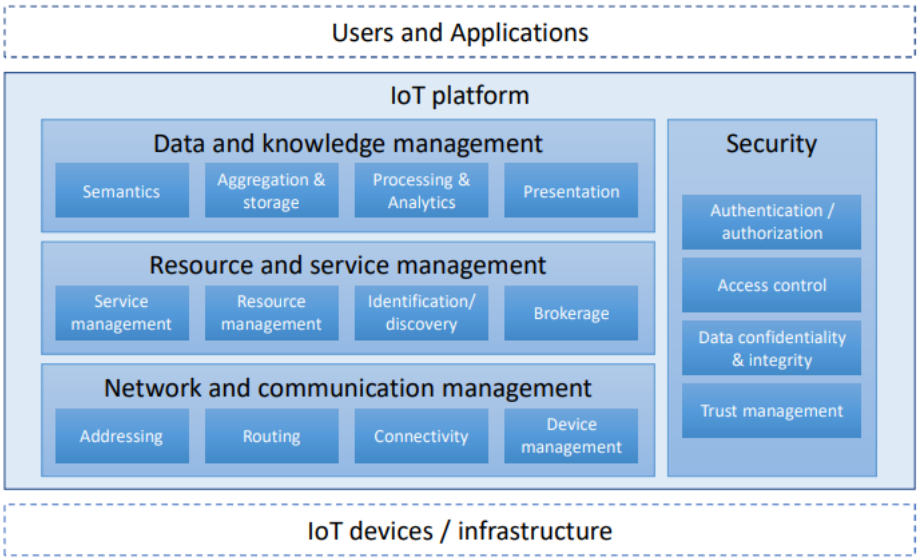


Figure 1.4: General layered model for IoT devices

NoSQL databases

No-SQL databases are used in big data and real-time application: they're suitable to scale on horizontal cluster of machines.

As main example, we use **MongoDB**: what in SQL were *Records* in MongoDB are **documents**. Documents have a *JSON*-like data syntax: in the following snippet, they're all *field:value* entry.

```
{
name: "sue",
```

```
age: 26,
status: "X";
group: ["news", "sport"]
}
```

A set of *name/value* pairs are separated by commas: a `value` can also be an array. So MongoDB **documents** correspond to **native data types** in many programming languages: can embed other documents and arrays to reduce the need for expensive joins.

What were **tables** in SQL, in *MongoDB* are **collections**: documents can be grouped in collection but, differently from SQL, different documents in the same collection can have different structure.

Queries Usually a **query** targets a specific **collection**: it specify criteria and conditions that identify a set of documents in the collection. As SQL, a query can also include a *projection* that specifies the fields to return or include modifier of the output (*like sorting of result, manipulating data format, etc*). Here an example:

```
db.sensedData.find({time:{$gt: 1900}}).sort({time:1})
```

which is represented by the image pictured in 1.5.

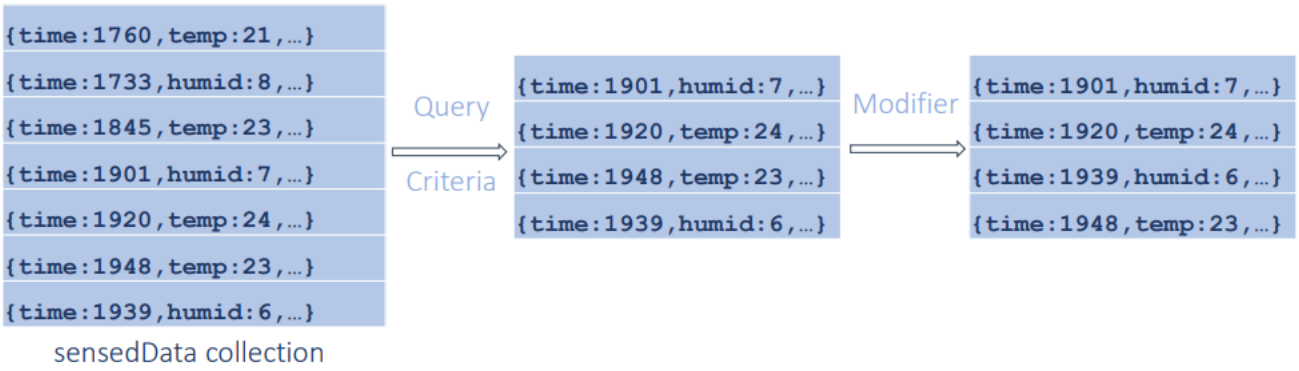


Figure 1.5: `sensedData` query perform

We can **modify the data** by update/create/delete: the `update` and `delete` operations can specify the criteria to select the documents to update or remove. Here an example of `insert` query (*referring the pictured in 1.6*):

```
db.sensedData.insert(
{
time:2011,
humid: 5,
... }

)
```

1.0.1 IoT issues

There are many issues in IoT and they're not easy to address:

- **Performance**: because microcontroller have a very low computation capacity there are physical and computational constraints to evaluate based on the type of activity to perform. This induces to optimize various parameters and metrics.
- **Energy efficiency**: each features it's based on a component that require a minimum level of power to work properly. The *features level* of a device can be tuned, as we will see when discussing about **duty cycle** 8.5.1.
- **Security**
- **Data analysis/processing**: how perform analysis and processing in a constrained device or orchestrating the load between several devices in the same network, balancing the overhead between differences neighbors devices.

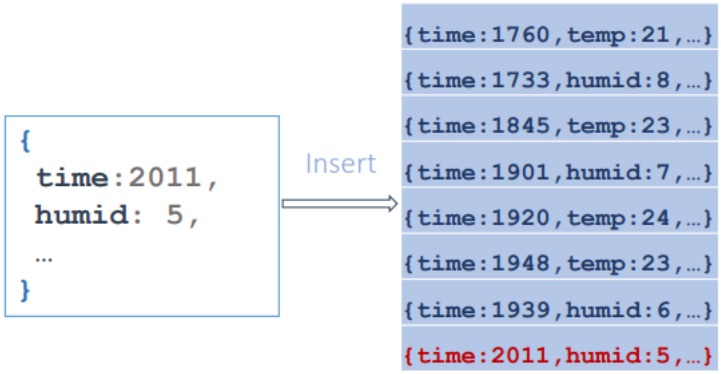


Figure 1.6: `insert` query execution

- **Communication/brokerage/binding**: How to bring together data producers (*sensors*) with consumers (*users/actuators/applications*).
- **Data representation**: data formats and standardization to simplify the multi-vendor device communication.
- **Interoperability**: many standard already exists at different level, like:
  - **MAC Level**: *Bluetooth, IEEE 802.15.4*
  - **Network level**: *ZigBee, Bluetooth, 6LowPan,*
  - **Application level**: : *MQTT, CoAP, oneM2M*

Let's discuss the issues related to **latency** and **reliability**, proposing some solutions: this issues arises because the physical devices can be (and they are) far from each other, this without any doubt adds latency to the IoT network. With reliability we mean the process of the network to lose nodes or packets due to some type of **failure/incident**. We saw 2 approaches:

- **Everything is on the Cloud:** this means that all the processing is done in the cloud, the devices at the perception level are meant only to acquire data and send it to the cloud servers, where all the processing is done. Here latency impacts a lot, since the amount of data sent to the cloud can be significant in term of needed bandwidth.

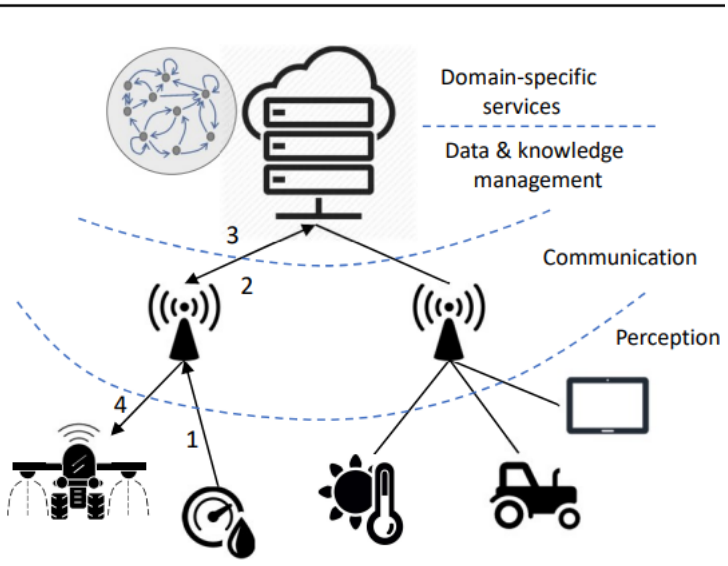


Figure 1.7: First approach: everythign on the cloud

- **Some Processing on the Edge**: We can adopt some *perception devices (still sensors)* with a little bit of computational power (*small, cheaper cpu*) to process some of the input data, in this

way we can reduce the amount of packets sent to the Cloud servers. It's not mandatory to use the cloud, if your devices can make all the processing alone, then you don't need any cloud.

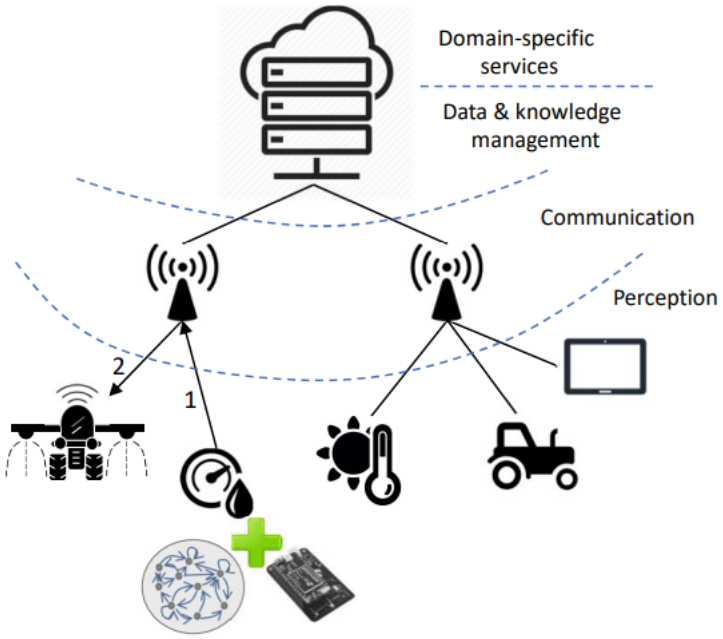


Figure 1.8: Second approach: processing on the edge

Based on the type of devices, scenario, services offered and computational powered provided closer or farest from the data generation point, we can identify three different *computational models* to apply in an IoT context, as pictured in 1.9.

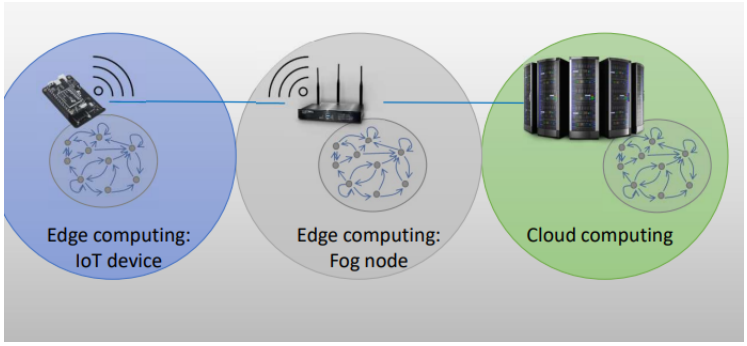


Figure 1.9: Three approaches to IoT computational load

1. **Edge**: the edge of a typical enterprise network is a network of IoT-enabled devices consisting of sensors and perhaps actuators, these devices may comunicate with one another. A cluseter of sensors can comunicate with a central device that aggregate the data to be collected by a higher level entity. A **gateway** interconnects the Iot devices with the higher level comunication networks, it performs the traslation between the protocols used in the comunication networks (on top of **perception** layer) and those used by devices. Perception layer uses some protocol that needs to be translated to enter the network. In *a few words*, in the edge we find the **physical** devices of the perception layer.

2. **Fog**: It's an infrastructure of local servers and access points, connected together to provide fast response time to the **edge devices** and to reduce the amount of data that goes to the cloud, typically the fog devices are deployed **physically near the edge**. So rather than storing data **permanently** or for **long period** in a central storage we do as much processing as possibile on the **Fog** level. Processing elements at these levels may deal with high volumes of data and perform data transformation operations, resulting in the storage of much lower volumes of data.

You can think of Fog as the opposite of Cloud, because Cloud **centralizes storage and processing for a relatively small number of users** while the Fog approach **distributes processing and storage resources close to the massive number of IoT devices**.

3. **Core**: the core **network connects far fog networks and provides access to other networks.** The core network uses high-performance routers, high-capacity transmission lines and multiple interconnected routers for redundancy and capacity. It may connect to other type of devices, such as high performance servers and databases or private cloud capabilities. **This means that you can connect fog networks to the cloud.**

1.0.2 AI approach to IoT

AI aims at getting computers to behave in a smarter manner, **either through curated knowledge or through machine learning**.

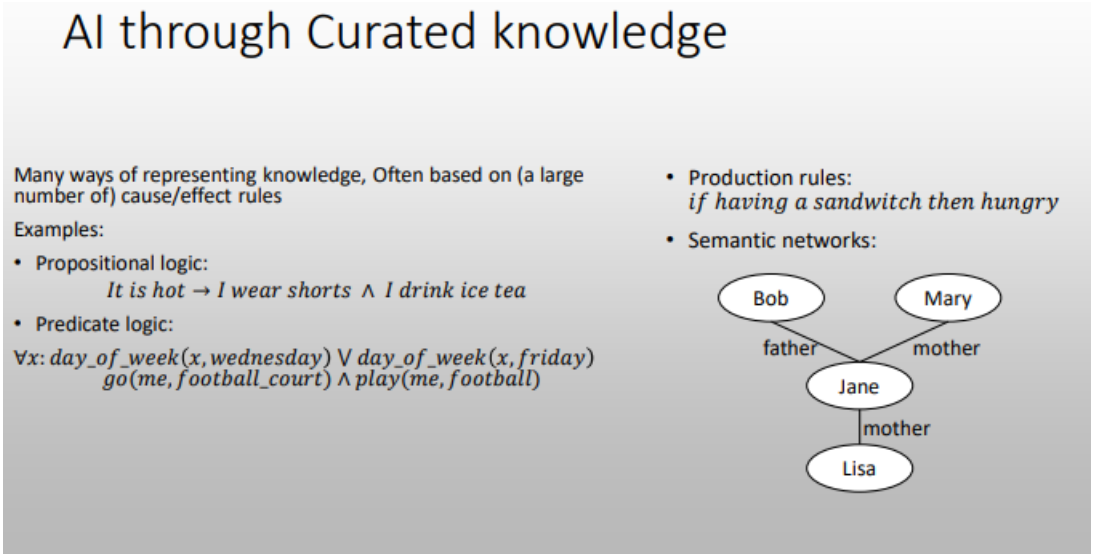


Figure 1.10: Curated Knowledge paradigm

With **Machine learning** we indicates a subfield of AI that deals with automatic systems that can learn from data. The system is fed by examples to learn to associate input with output, then when an input (**never seen**) is given, the model/system produces anyway an output. If **well trained** the output will mostly be correct, due to generalization capability of ML.
Mostly three ML paradigms are used today: the first is the *Unsupervised Learrning* in which the algorithm is presented with a dataset and must find structure or relationships within that dataset on its own. The goal is to identify underlying patterns or groupings within the data. *without any type of labeled data.*
The following 2 approaches are the most used in IoT:

- **Supervised learning**: learn from past examples, **every example is a pair input + desired output**, aims at predicting the future or interpret the present.
- **Reinforcement learning**: learn from examples, **but the main difference is that the example is a pair input + reward**, in this way the system knows that the answer is correct if there's a reward. (*E.g. to learn a game the reward can be +1 for winning, -1 for losing, 0 otherwise*).

IoT and emerging Paradigms

Blockchain is a **shared and trusted public ledger** for making transactions, everyone can inspect and nobody can alter it. the blockchain thus provides a single point of truth: it is shared and **tamper-evident**, in case of tampering it's quickly noticed.
Blockchains shifts the IoT paradigm from **centralized storage** to a **decentralized one** in a distribuited ledger. Supports the expanding of the Iot Ecosystem. Since the **ledger is public** this implies a **reduce in maintenance costs** and provides trust in data produced. An implementation of this approach can be seen on having different companies that work in a supply chain, everyone of them needs to **check**

the quality of the product along the chain. Each company in supply chain can **query the ledger** and check the latest transaction, thanks to **smart contracts** that are used to certify each intermediate transaction.

1.0.3 Interoperability & Reference Standards

A straight implementation of an IoT solution is not a problem by itself, you can design the solution from the bottom (*physical layer*) up to the application layer. This approach is called **Vertical Silos** (*as there is no external comunication, so even the infrastructure itself is inside the silos*) since this system only has your devices and every change/update requires **your intervention**. This creates a **Vendor Lock-in** for your clients because the silos prevents them to use devices from other vendors. Vendor lock-in forces high costs to **migrate** to another vendor, customer need to fully redesign and deploy a new solution.
Standards are useful to fix this problem. The first standards we saw are for **Wireless technologies**:

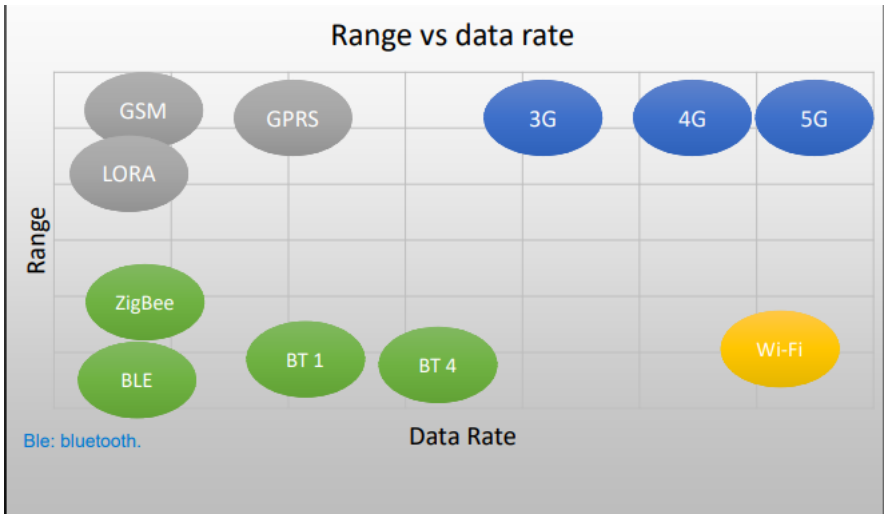


Figure 1.11: Standard comparison for range and data rate performances

1. **IEEE 802.11** aka **WIFI**: is a **family of standards** useful for networks where devices are close to each other (*Range = 100 meters*) but need high data rate. There's a new standard IEEE 802.11A, B, AC. They added new features such as additional frequency bands (5GHz). Since Wi-Fi networks operate on specific frequency bands, with this band the network is faster and less congested. Allows to **Increase trasmission range and bit rate.** Another useful feature is Roaming between access points, this means that a device changes to the nearest access point in the network.

2. **IEEE 802.15.4 and Zigbee**: is a wireless communication standard for low-rate wireless networks, where devices are very close to each other (low range). It defines **both physical and MAC layers**. Usually used when developing low-power sensor networks. The *Low power* property involves also having a **low throughput** (*115 Kbps, at most*) and low duty cycle (*percentage of time sensor is sensig, as we will see*). Thanks to the fact that Zigbee network can create a **mesh topology**, we can create larger range networks. In partial mesh topology, nodes basically forward the data to the desired node, in this way the network can grow. The general idea is pictured in 1.12.

3. **Bluetooth**: Higher data rate than ZigBee, but the range is basically the same. Used to make multimedia comunication. Small networks with **master-slave comiunication**, master sends the data and the slave acquire it.

Why standards? Usually motivated by a reduction of the costs for development of a technology, the price for a new technology is high but as soon as other companies try to create the same one this will **instantly drop the price**. For instance think about a technology that has been developed on the lower layers, in this case comunication layer (Wifi) the companies can move to higher layers in order to create their own technology and sell it at their price.
Coopetition stand for *competition and cooperation* so can be a good approach for companies, they work with each other to develop a standard but still deploy their own technologies, built on top of those standards. As said before, this happened for wireless comunications, we saw lot of different standards.

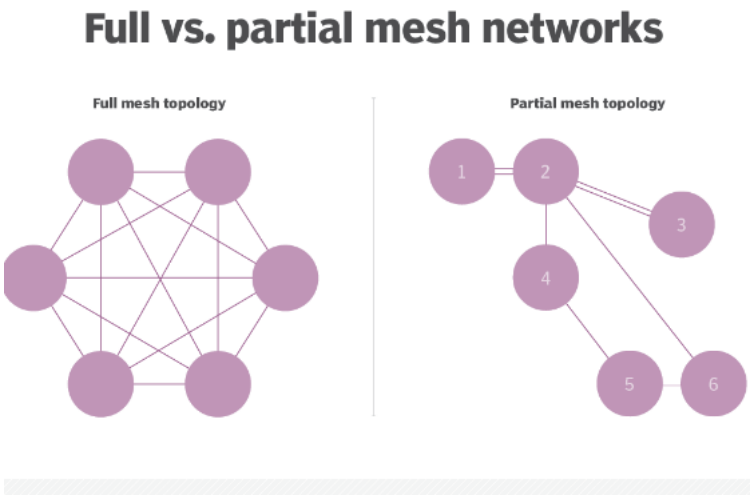


Figure 1.12: Full mesh network vs partial mesh network

Interoperability problems moved to the higher layers.
The problem of **interoperability** arises between consortium of standards. Nowadays, the problem of interoperability and thus of standarization is moving up at middleware/application layer. For ZigBee, it covers many layers: it defines a network, transport and application layer specific for ZigBee and incompatible with others.
When there are too may standard available and they are not compatible, the solution is to use the concepts of *endpoints* or **integration gateways**: allows to translate from a given standard to another. The gateway is limited to a single layer but they're able to translate a layer protocol to another one.
We have different scenarios/configurations as sketched in 1.13

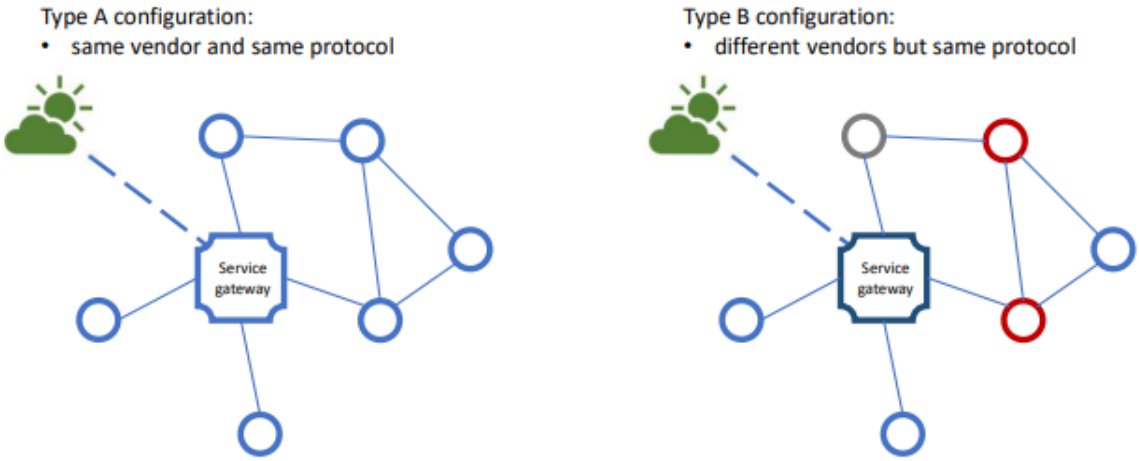


Figure 1.13: Service layer within the same standard vs Integration layer for interoperability

In the left case, even if the same vendor with same protocol, the nodes need a communication to the service gateway that provides access to the internet. In the right case, we have different vendors that commonly use the same protocol, respecting a common standard: even in this case we need the service gateway to access the rest of the world. The service gateway allows to map *the rest of the world* to interact with it.
For devices that are not using internet, there is no need to provide external connectivty: differently for ZigBee or MQTT in which the service gateway provide the functionality of transmitting external data.
Different scenario is pictued in 1.14: in the left scenario, we have 3 networks which talks using a common standard provided by the **integration gateway**: in each network, the node talk each other using specific protocols, different from network to network.
A **integration gateway** is able to speak different languages/protocols and able to map one protocol in another. The mapping can be complicated becase can define different behaviours of the devices: in case

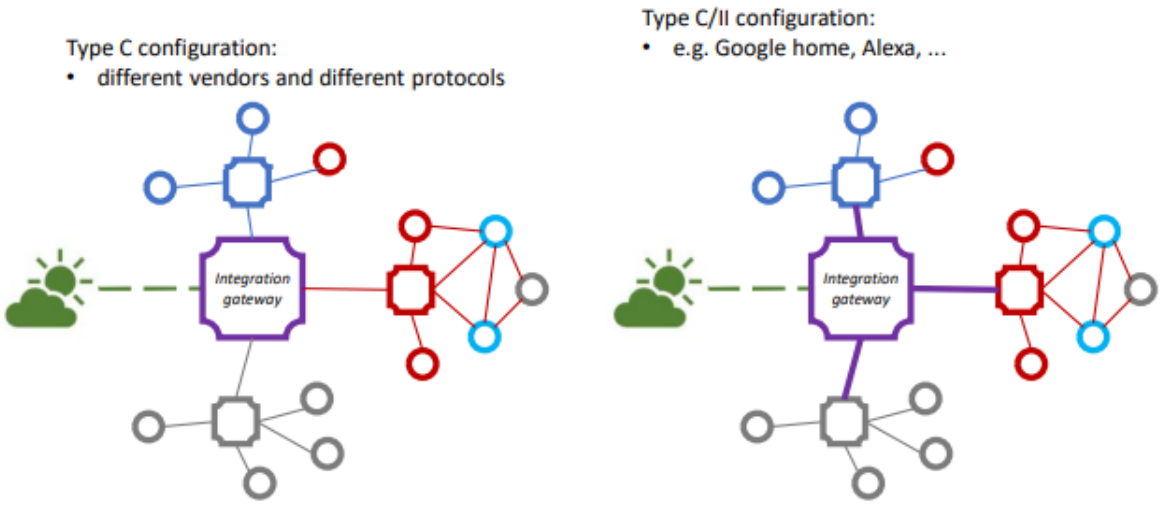


Figure 1.14: Integration gateway driven by different vendors and standard consortium

of a protocol that *push data* and other that differently *pop data* are not translable in one another. So the integration gateway need to translate the *behaviour*: it also need to complete the packet structure, frame tramission, etc. In type C/II scenario (*on the rigth*), the key idea is that the protocol standard or under-the-hood technology is driven by the players (*Google, Amazon*) that in practice determine the specific technology of *integration gateway*, pushing other players to develop in accordance to their technology to guarantee interoperability.

Provide a general purpose integration service is complex: a better solution is forming a *network* of integration gateway that allows to map one protocol to another one or subset of specific-translated protocol. So we obtain a network of **distributed integration gateways** as sketched in 1.15

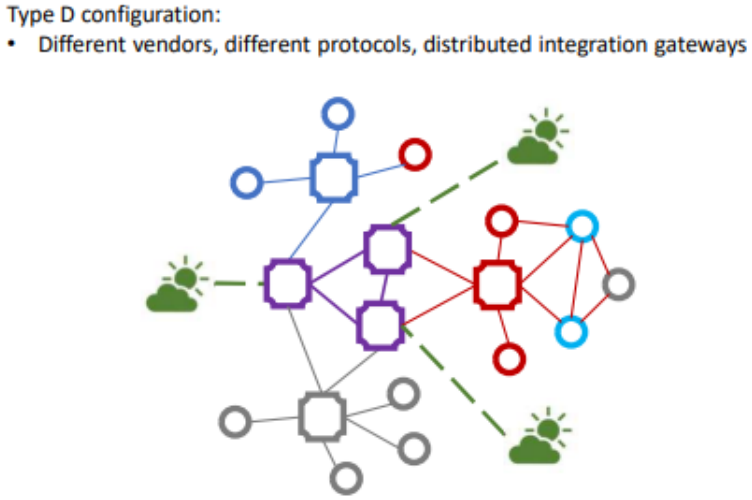


Figure 1.15: Distributed integration gateway network

1.0.4 Brief Security in IoT

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. The device manufacturers focus is the functionality of the devices itself so end-users may hve no means of patching the system, having little information about how and when patching.
The result is hundreds of millions of internet-connected devices vulnerable to attacks: this lead a problem with sensors that, if attacked, can inject false data into the nework. The general context which we refer is pictured in 1.16 Some devices are *constrained, unconstrained* or provide security features: some of them, despite not having security features, they're connected directly with internet.
Apply those requirements try to address the problem of security in IoT: the pressure of time-to-market

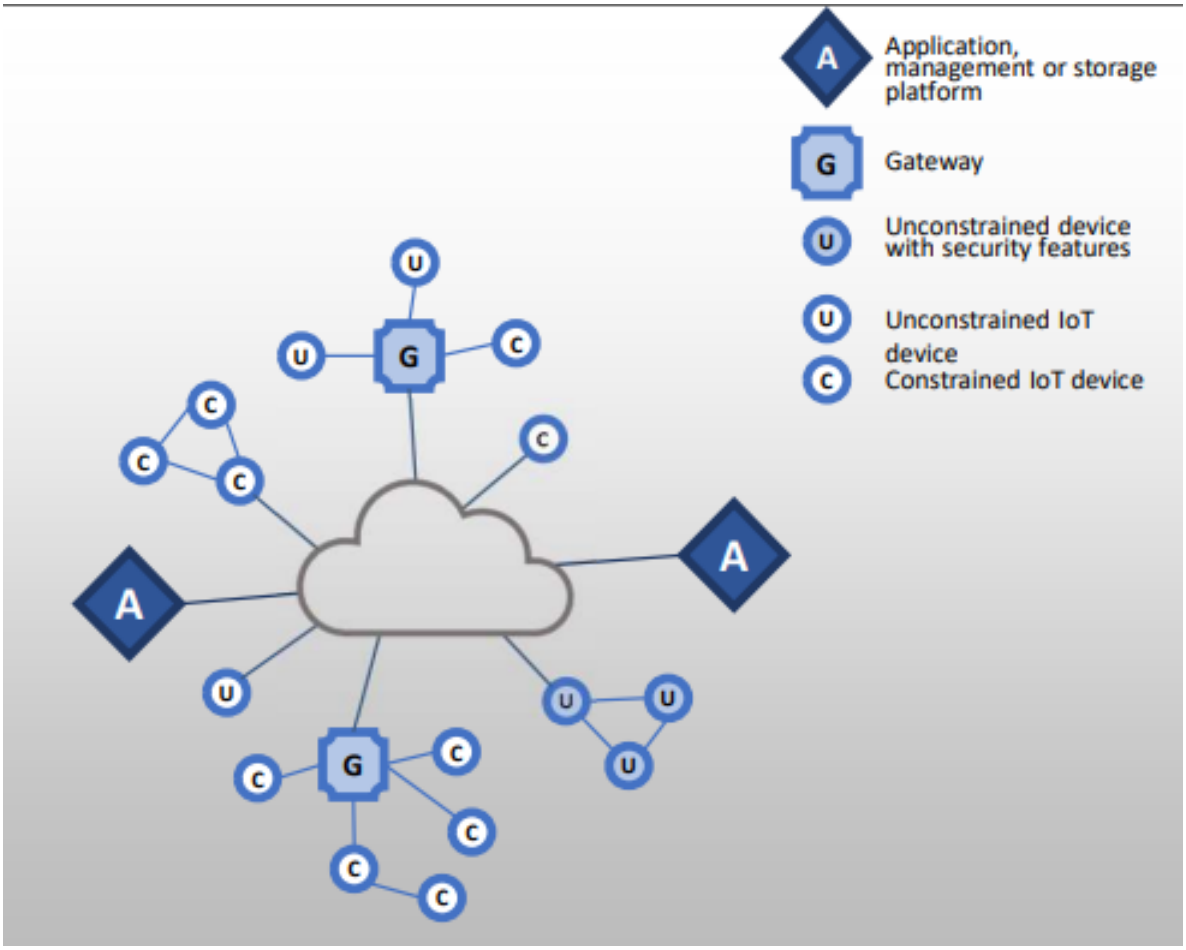


Figure 1.16: IoT gateway security: scenario 1

and fasten development phases set the problem of security on devices that are constrained in terms of capabilities or powers so security sometimes became optional because it's not fittable in that types of devices.
Usually those devices are not capable to run a full-fledge OS but only a simple OS without security features that are necessary to implement policies, even during updating the software-specific features.
The problem arises in the topics of *confidentiality* and *authentication* (*e.g. wearable, physiological sensors*).

Security Standards The *IUT-T Standard Recommendation Y.2066* includes a list of **security requirements** for IoT: those are functional requirements during *capturing, storing, transferring,a ggregating, processing* the data. The requirements concern:

- **Communication security** (*secure, trusted, and privacy protected communication capabilities*): enforces confidentiality and integrity of data during data transmission or transfer
- **Mutual authentication and authorization**: mutual authentication and authorization between devices (or device/user) according to predefined security policies. Before a device (or an IoT user) can access the IoT. The authentication pattern can follow 2 main methods:
  - Remote: directly to the cloud but poses complexity due to device's constraints
  - Local: between devices or authentication service at most one-hop far way or with low overhead for devices The mutual authentication is meant as *in both direction between devices and gateway.*
- **Data management security**: enforces confidentiality and integrity og data when storing or processing data
- **Service provision security**: deny unauthorized access to service and frauolent service provision. Ensure privacy protection for IoT users.

- **Integration of security policies and techniques**: ability to integrate different security policies and techniques. Ensure a consistent security control over the variety of devices and user networks.
- **Security audit**: any data access or attempt to access IoT applications are required to be fully transparent, traceable and reproducible, according to apporpaite regulations and laws. Support security audit for data transmission, storage, processing and applciation access.

The **gateway** is the main building block of the security mechanism and the devices implement only a subset of logic in accordance to the security operations carried out by the gateway.
Authentication can happen at different levels: authenticate devices only at physical layer is important but itself it's not a guarantee that the behaviour at the application level is correct (*e.g. tampered device at physical layer*). There are also other features like protect privacy for devices and gateway, self-diagnosis and self-repair. A critical phase is the deployment of the devices because initially they're not fully configured and may have not been still identified the correct gateway to connect.
An **IoT gateway** allows the identification of each access to the connected device: the authentication is performed based on application requirements and device capabilities so can either be *mutual* or *one-way*. The first type is more robust than the second one. The *mutual authentication* can also be perfomed between applications.
The *security of the data* is based on different security level, defined by the data storage location:

- data stored in devices and the gateway
- data transferred between the gateway and devices
- data transferred between the gateway and applications

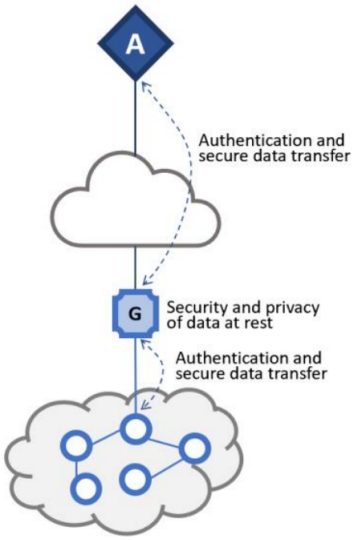


Figure 1.17: IoT gateway security - scenario 2

These requirements may be difficult to achieve if they involve **constrained devices**: if the gateway should support security of data stored in devices, without encryption this may be impractical to achieve. These requirements make several refences to *privacy*:

- with massive IoT, goverments and enterprises will collect massive amount of data about individuals like *medical information, location, application usage*
- privacy is an area of growing concern with the widespread IoT

Chapter 2

MQTT - Message Queueing Telemetry Transport

MQTT it's a *lightweight publish/subscribe reliable messaging transport protocol.* It's lightweight thanks to a small code footprint and low network bandwidth, with minimal packet overhead (slightly better than TCP).
It's builds upon *TCP/IP* with port `1883` or `8883` for using MQTT with SSL (*with overhead*). MQTT covers application, session and presentation layer (*both directly or undirectly by using middleware*).
MQTT uses an instance of **publish-subscribe** model between users and consumer: internally, as we will see, is implemented as a client-server architecture. To connect to internet, the IoT devices must adopt internet protocol suite. However, internet stack is too resource-compsuntive respect to constrained IoT devices that usually are *lossy, low power, etc.*
Internally MQTT uses a **client/server** architecture: this bought the major complexities on the server side. It also provide basic **Quality of Service (QoS)** and it's *data agnostic* so it's suitable both to *M2M (Machine-to-Machine)* and IoT.

2.1 Publish/subscribe model

The main components of the model are **publishers, subscribers** and **broker or event service**. The fist two are both seen as clients and do not know each other, the latter it's the server known both by publishers and consumer. The **publisher** produce event or any data: interact only with the **broker** and does not know if the data is consumed/read by other entities. The subscribes express interest for an event (*or a pattern of events*): receive the notifications from the broker whenever the event is generated. Publishers and subscribers never interact each other: they interact indirectly only by the *broker(s).* Those pattern allows to be publisher and subscriber to be decoupled in **time, space and syncrhonization**.

The broker know publisher/subscribers and receive all incoming messages from the first, filter them all and distributed them to the interested subscriber. It also need to manage request of subscription/unsubscription.

A **publish/subscribe** interaction can be implented in different ways: the boker is usually an *independent* agent and manage, coordinating with publishers/subscribers, the operations of `publish, subscribe, notify, unsubscribe` as briefly introduced in 2.1
The **broker** is delegated to storage and management of subscriptions: in figure 2.2 an example in which different subscribers register themself to one or more *topics* (*smartphone subscribe to `Temperature & Humidity` topic* while *computer subscribe only to temperature topic*). This model allow to **decouple the space** because pub/sub do not need to known each other and do not share anything (*they don't known the IP/port of each other and how many peers are subscribed*). It allows also the **time decoupling** because they don't need to run at the same time, guaranteeing the **asyncrhonicity** of the model. The asyncrhonous model is pictured in 2.3.

The decoupling increments the **scalability** of the actors involved: the operations on the broker can be easily parallelized and are event-driven, allowing scalability to a very large number of devices by parallelizing the broker.

Filtering

The filtering features managed by the broker can be based on three different criteria:

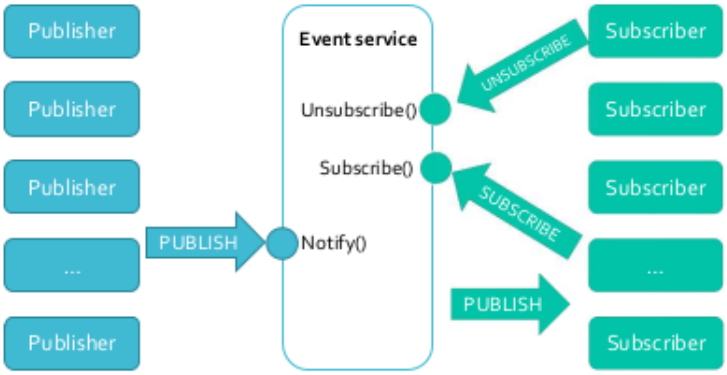


Figure 2.1: Operations summary

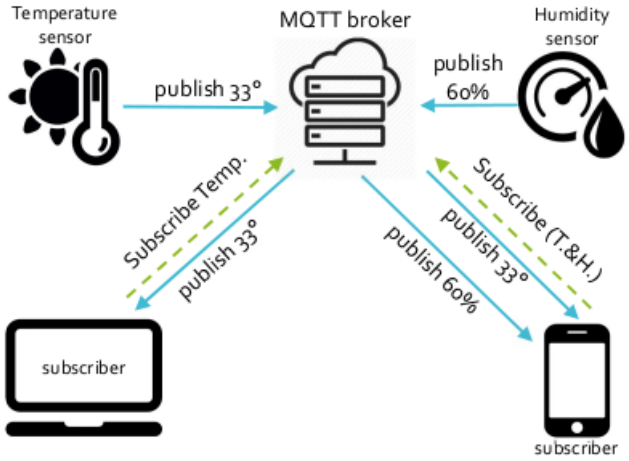


Figure 2.2: Example of `subscribe` for temperature topic

- *Based on* ***subject topic***: the subject (*or topic*) is a part of the message and clients subscribe only to a specific topic. They are usually represented by a string, possible organized in a hierarchical taxonomy.
- *Based on* ***content***: the client subscribe for a specific query (*like* `temperature > 30°`) and that query is used by the broker to filter the messages. This methods involves to understand the semantics of the data transmitted or at least be able to *read* the data so add complexity con context in which security mechanism (*like encryption*) are used to transmit/store data.
- *Based on* ***type***: filtering of event is based on both *content* and *structure* so the type refers to the type/class of data that can be customized by the subscriber by defining in a strongly typed language the desired object structure. This methods introduces complexities in case publishers and subscribers are written in different language and there is no a unique *translation of type* because their type system is not uniquely determinate. In those scenarios, tight integration of the middleware and language paradigms (*like Object Programming*) can ease the presented problem.

Highlights In the proposed model (*widely sketched in 2.3*), publishers and subscriber **need to agree on topics beforehand** and publisher cannot assume that somebody is listening to the messages because is there are no subscriber messages are not readed by anyone.
Despite publisher and subscribers do not need to known each other, they need to known the *hostame/port* of the **broker** beforehand, to connect and establish a TCP connection to it. In most application, the delivery of messages is near-real-time but in cases when the subscribers are offline, the broker is able to store the messages *only if* subscribers have been connected with a **persistent session** and they're already subscribed to the *topic*. In case of delay or unreliable message the retransmissions is necssary and the latency issues have to been taken into account. Also the deployment phase take time due to the time needed both by clients to connect to brokers.

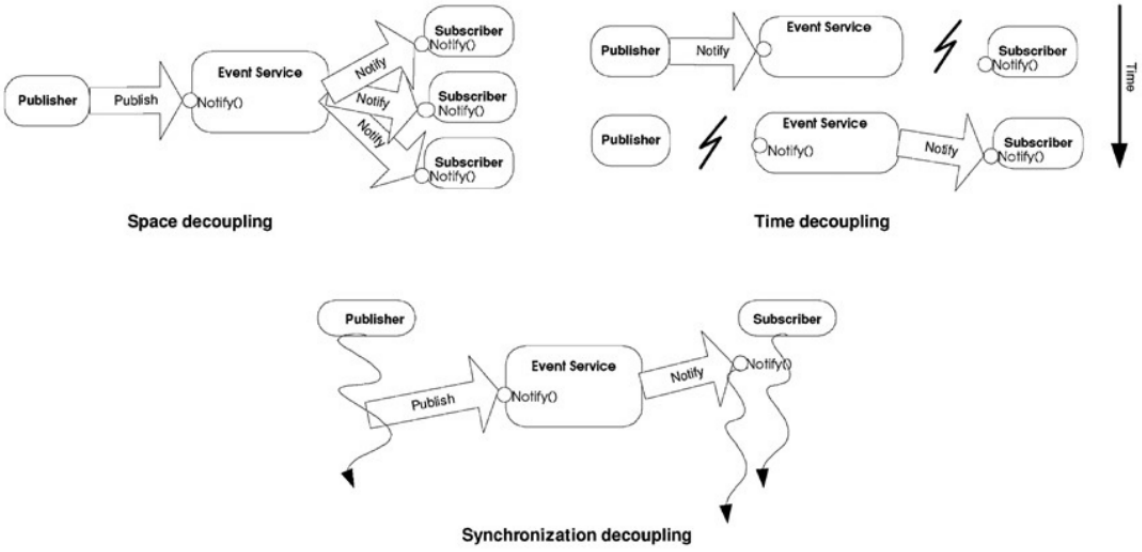


Figure 2.3: Asyncrhonous model flow of operations

MQTT decouples the syncrhonization allowing to obtain an **asyncrhonous model**: this model implies the use of callback functions on the subscriber side. Libraries that implements MQTT allows this async model by enabling callbacks. The complexities of the MQTT protocol are shifted to the **broker work** which in general represent a much more powerful machine respect to subscribers/clients.

As mentioned, the *subject-based* filtering of a message involves the use of *topic*: topic is based on a hirerarchy of topics that are meaning to the purposes of the appication. MQTT offers additional *QoS* in term of message reliability: it's based on top of TCP by adding an application level ackwnoledge. It's expressed by 3 levels of QoS (0, 1 ,2) where the last two levels ensure the ACK at application level, while the level 0 corresponds to TCP reliability level.

2.1.1 MQTT operations

MQTT includes operations at layer 5-6 of ISO/OSI layer, as indicated in 2.4. In the following chapter

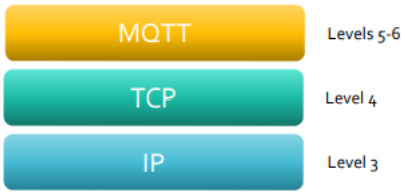


Figure 2.4: MQTT in ISO/OSI layer specification

we'll present the main types of messages used in MQTT, how they interact and contribute to the overall protocol flow.

CONNECT A client connects to a broker by sending a `CONNECT` message. This enstaurate a *TCP connection* with the broker. The `CONNECT` message contains:

- *Client ID*: it's an optional identifier, if absent is setted to 1. It's necessary in case of persistent connection: allows to restore connection even in case the connection with the broker is broken, allowing to restore the session from the last *time* the exchange of the message happened. Using no client ID implicit declare the *no session state* so this setting the *Clean Session* flah to true.
- *Clean Session (Optional)*: it's FALSE if the client request a persistent session.
- *Username/Password (optional)*: No encryption unless used with SSL

- *Will flags (optional)*: allows in case of disconnection ungracefully to sent a **last will message** to the subscribers by the broker (*in case of broken connection with the broker and/or damage*).
- *KeepAlive (optional)*: the broker expect to receive from the client periodic message of alive. If not received, the broker can assume that the client is dead and close the TCP connection (*possibly sending the last will message to each subscriber*). The `KeepAlive = 0` turn of this mechanism, indicating that the subscriber will never send the alive messages to the broker.

The broker acknoledges to the CONNECT message with the CONNECTACK message that states if the connection is accepted or rejected and also informs the client (in case of persistent session) information about the previous session.

PUBLISH

Each message contains a **topic** and a **payload** of whatever type because its managed as a bunch of *bytes*. Only the subscriber is able to understand the semantics of the payload.

The publish message is sent by the publisher to the broker and forgetted. Then it's forwarded by the broker to the subscriber. The `PUBLISH` message contains the following data (*summarized in 2.5*):

- `packetId`: an integer, is 0 if the QoS level is 0
- `topicName`: string possibly structured in a hierarchy wih slash `/` delimeters
- `qos`: 0,1 or 2 (*see later*)
- `payload`: the actual message in any form (*usually interpreted as bytes*)
- `retainFlag`: tell if the message is to be stored by the broker as the last knoen value for the topic. If a subscriber connects later, it will get this message
- `dupFlag`: indicates that the message is duplicate of a previous un-ACKed message. This field is meaningful only if the `qos` is greather than 0.

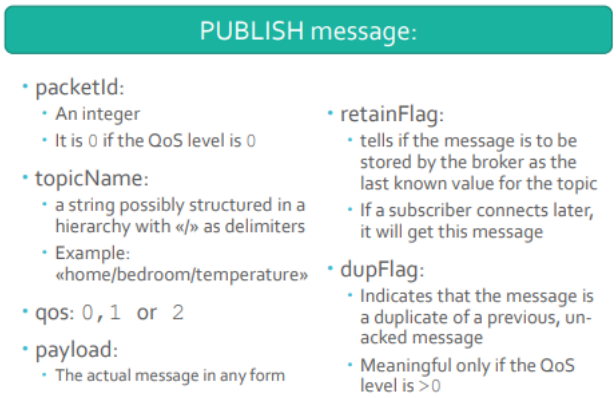


Figure 2.5: Publish message structure

The `retainFlag` allows to send a specific message for later subscriber, storing it to the broker.
When the **broker** receives a `PUBLISH` message, it:

- acknowledges the message (*if requested, QoS ¿ 0*)
- processes the message (*identify its subscribers*)
- delivers the message to its subscribers

The **publisher**:

- leaves the message to the broker
- does not know whether there are any subscribers and whether or when they will receive the message

SUBSCRIBE There can be a difference of QoS between **publisher-broker** and **subscriber-broker**: even in case of 3 different subscriber with different QoS, the broker will manage them independently. The structure of the `SUBSCRIBE` message is the following:

- `packetId`: an integer
- `topic1`: a string (*as in publish message*)
- `qos1`: 0,1 or 2 Usually a `SUBSCRIBE` message contains a list of `(topic, qos)` because a subscribe message can refer to multiple topics with the relative QoS.

The **broker** acknowledges the subscriber with a `SUBACK` message. It contains the following fields:

- `packetId`: the same integer of the relative `SUBSCRIBE` message
- `returnCode`: one for each topic subscribed. The value `128` indicates failures (*e.g. the subscriber is not allowed or the topic is malformed*) while `0,1,2` indicates success with the corresponding QoS granted (*that can be lesser than the QoS requested*).

UNSUBSCRIBE A client can unsubscribe a topic to stop receiving the related messages. The `UNSUBSCRIBE` message is composed by a `packetId` and a list of topics `topic1, topic2, ..., topic_n`. The `UNSUBACK` message have also a `packetId` and contains the same data of the `UNSUBSCRIBE` message.

2.1.2 Topics

Topics are strings that are organized into a **hierarchy** in which each level is separated by a `/` *slash*. The subscriber can use **wildcards** to specify a group of topics:

- `home/firstfloor/+/presence`: select all presence sensors in all rooms of the first floor
- `home/firstfloor/#`: select all sensors in the first floor If using only a dash as a topic, you will subscribe to all topics, including the reserver one. Topics that begins with **"$"** are reserved for internal statistics of MQTT so they cannot be published by clients.

An HiveMQ example of 5 different topic specification:

```
$SYS/broker/clients/connected
$SYS/broker/clients/disconnected
$SYS/broker/clients/total
$SYS/broker/messages/sent
$SYS/broker/uptime
```

Topics must do not contain space, are short for memory storage optimization and packet size, use ASCII char UTF-8. It's also common practice to embed the clientID in topic or an *unique identifier* like `sensor1/temperature` so that onlt the client with the same `clientID` can publish such a topic.

2.1.3 Quality of Service

The QoS is an agreement between the send and the receiver of a message: the underlying TCP allows the QoS to gurantee delivery and ordering of the messages. In MQTT the QoS is an agreement between the *clients (publishers/subscribers)* and the *broker*. MQTT define three levels of QoS:

- **At most once** (`level 0`): best-effort delivery without guarantees
- **At least once** (`level 1`): guarantees that a message is delivered at least one time to the receiver
- **Exactly once** (`level 2`): guarantees that each message is received only once by the intended recipients

So QoS is used both between publisher and broker and between broker and subscriber.
The **QoS level 0** is called *best effort* delivery because messages are not acknoledge by the receiver. When used between publisher and broker messages are not stored by the broker and immediately forwarded. It provides the same guarantees as the TCP Protocol by guaranteeing the delivery as long as the connection remains. If one of the two peers disconnect there's not guarantee anymore.
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 1. A message may be delivered more than once so it's suitable when the subscribers can handle the management of duplicated messages. Subscribers send acknowledgements by sending `PUBACK` packets.

The **QoS level 2** is the slowest mechanism to guarantee QoS because check that each message is received exactly once by the recipient.
It uses a **double two-way handshake** by sending `PUBLISH` and the relative ACK by the `PUBREC` Packet. Respectively, the client send `PUBREL` to acknoledge the reception of the message: the broker now can drop the message from its storage and declare the transmission complete by sending a `PUBCOMP` packet.
So for each message of QoS level 2 there is need four times the exchange of packets required by the level 0. To summarize QoS level 2, the figure 2.6, describe the entire exchange.

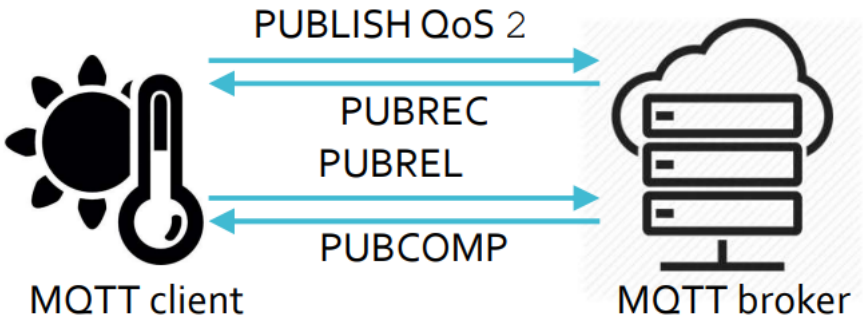


Figure 2.6: MQTT QoS 2 message exchange diagram

In *QoS level 2*, the **broker**:

1. receives the PUBLISH with a message
2. Process the PUBLISH
3. Sends back a PUBREC
4. Keeps a reference to the message until it receives PUBREL

The **client**:

- Sends PUBLISH with a message
- Waits for PUBREC and then store PUBREC and discards the message
- Sends back a PUBREL to inform the broker
- Waits for PUBCOMP and then discards the PUBREC

The *double two-way handshake* is necessary because the first handshake allows to send the message, the second is to agree to discard the state. The `PUBREC` message alone is not sufficient because if it's lost, the client will send again the message so the broker needs to keep a reference to the message to identify duplicates. With `PUBCOMP` the client know that it can discard the state associated to the message. This level of QoS is suitable when the clients *cannot manage duplicates* so the complexity to guarantee avoiding duplicates is managed by the broker that keep tracks of ACK and stores messages, with high overhead.

2.1.4 Persistent sessions

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

- All subscriptions to topics
- All QoS `1,2` messages that are not confirmed yet

- All QoS `1,2` messages that arrived when the client was offline

A persistent session is requested at `CONNECT` time (*`cleanSession` flag setted to FALSE*): the `CONNACK` message confirms whether the session is persistent.
Also clients have to **store state** in persistent session:

- Store all messages in QoS `1,2` flow, not ACKED By the broker
- All received QoS `2` messages not confirmed by the broker

Messages of persistent sessions will be stored as long as the system allows it. The persistent mechanism can be avoided when refers to publishing-only clients that uses only QoS level 0 or if old messages are not important or when miss some data is not a problematic.

2.1.5 Retained messages

A publisher has to gurantee that its messages are actually delivered to the subscriber, even for later subscriber. When a client connects to the broker and subscrbes a topic, it does not known when it will get any message. For each topic, we can have **only one retained messages**, generally the *last one message* sent by the publisher. This way a new subscriber is immediately updated with the *state of the current communication.*
A retained message is a normal message with the flag `retainFlag` setted to `TRUE`: the message is stored by the broker so if a new retained message is published, the broker will keep the last one received. When a *client* subscribes the topic of the retained message, the broker immediately sends the retained message for that topic: this can also works combined with **wildcards**.
The mechanism of **retained messages** and **persistent sessions** are completely decoupled: the second are messages kept by the server even if they had already been delivered. To delete a retained message is sufficient to publish a retained **empty** message of the same topic. This mechanism is usefull for *unfrequent updates* of a topic: for *example*, consider a device that update its status (`ON/OFF`) on topic `home/devices/device1/status` so if it publish ON, this status will remain for long time and if the message if retained, all subscribers will easily know the device is on.

2.1.6 Last will and testament

Last will and testament feature is used to **notify other clients about the ungraceful disconnection of a client**. At `CONNECT` time, a client can request the broker a specific behavior about its last will: it's a normal message with `topic, retainFlag, QoS` and `payload` stored by the broker as a last will. When the broker detects the client is **abruptly disconnected**, it send the last will message to all subscribers of the topic specific in the last will message. If the client send `DISCONNECT`, the stored last will message is discarded because the underlying assumption is that before correctly disconnect the clients has been already notified other clients. So, the broker sends a last will message if:

- Occurs an IO Network error
- The client does not send the `KeepAlive` message in time
- The client closes the network connection without sending `DISCONNECT`
- The broker closes the connection with the client because of a protocol error

The last will is specified in the `CONNECT` Message and contains 4 optional fields:

- `lastWillTopic`: a topic
- `lastWillQoS`: either 0,1,2
- `lastWillMessage`: a string
- `lastWillRetain`: a boolean flag

The last will message can be combined with **retained messages**: imagine a scenario in which a devices updates the status (`ON/OFF`) on topic `home/devices/device1/status` so it's powered and publish `ON` with a *retained message*. If the device crashes and then it abruptly disconnect, it does not publishes `OFF`: *last wll message* can be useful here because could be a retained message with payload `OFF` so the **subscribers and the futures ones are properly informed**, even in case of unwanted disconnection.

2.1.7 KeepAlive

The KeepAlive mechanism assure that a client and its connection with the broker is still **alive**: the client sends periodics messages to the broker that prove its liveness. The *frequency* of these messages is declared in the `CONNECT` message. The keep alive message must be sent by the client before the connection expiration of the interval set with the CONNECT message. The entire process is sketched in 2.7

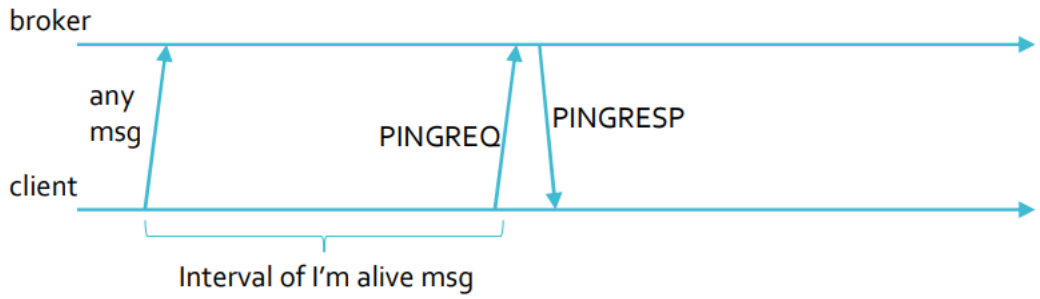


Figure 2.7: *KeepAlive* message flow

KeepAlive timer (*managed by the broker to control the liveness*) is resetted by both `PINGREQ` messages and by pulbish a message on a topic: this allows to be consired alive by the broker. So if the client does not send `PINGREQ` in time or any other message, the broker turns off the TCP connection and sends the last will message.

2.1.8 Packet format

The structure of an MQTT **control packet** is the following:

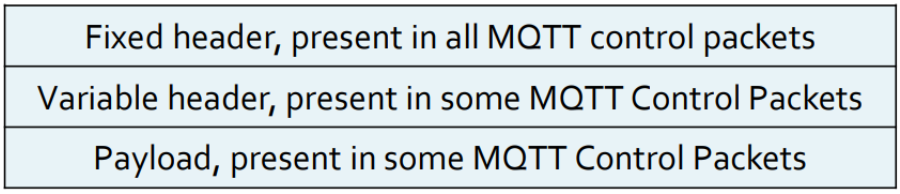


Figure 2.8: Control packet structure

The **fixed header** is composed by **2 bytes**, the first contains (2.9):

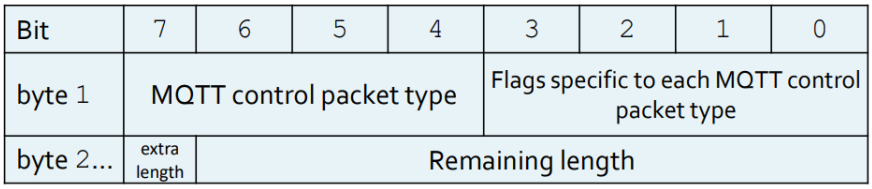


Figure 2.9: Fixed header structure

- *MQTT Control packet type*: 4 bit (*range 7 to 4*)
- *Flags specific to each MQTT control packet type*: 4 bit (*range 3 to 0*)

The second byte contains the **remaining length** which is the **length** of the variable header and payload: 7 bits are used to encode the remaining length while one bit (*the MSB, index 7*) is a flag that specifies that there is another field. The **control packet type** are pictured in 2.10

The **variable header** contains the `packetId` (*encoded with 2 bytes)*, only the packets regarding the `CONNECT` and `CONNACK` does not contains the `packetId` while the `PUBLISH` packet contains this information only if `qos > 0`. It can 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).*
The **payload** can be empty or, in case of `CONNECT` message it contains data about the client or for

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

Figure 2.10: Control packet type

`PUBLISH` can be optional (*empty payload are allowed, e.g. to reset retained message*). For example, the `CONNECT` packet includes:

- client identifier (mandatory)
- will topic (optional)
- will message (optional)
- Username (optional)
- Password (optional)

In the table 2.11 is listed when the payload is required.

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 2.11: MQTT required payload commands

2.2 MQTT on Arduino

Include only a subset of MQTT specific, excluding:

- SSL/TLS

- QS level 2
- Payload limited to `128` bytes

2.2.1 MQTT Competitors

HTTP is a valid alternative to MQTT, despite it's not speficially designed to operate for IoT devices at appliaiton level. A comparison is sketched in 2.12.

	MQTT	HTTP
pattern	publish/subscribe	client/server
focus of communication	single data (byte array)	single document
size of messages	small and small header	large, details encoded in text form
service levels	3 QoS levels	same service level for all messages
kind of interaction	1 to 0; 1 to 1; 1 to N	1 to 1

Figure 2.12: MQTT vs HTTP

HTTP used in IoT device can be used in two different ways:

1. Clients are server accept connection from services
2. Service is the server and accept incoming connection from clients (devices). This paradigm poses problems on scalability, also considering the size of messages and the focus of communication (*because HTTP is not suitable for constraned devices*). In real world application both solution are deployed (*e.g. TeamSpeak, AWS IoT*) because nowadays devices are not entirely constrained on power/computation capacity.

There are several **limitations** on MQTT: the need for a centralized broker can be limiting in scenario where we have several point-to-point communication because the overhead of a broker may easily become not compatible with end devices capabilities as the network scales up. Also, the broker is a **single point of failure** and the underlying TCP protocol does not come for free: it's not cheap for low end devices. Exists also an MQTT based on UDP, not standardized nowadays.

2.2.2 Brief CoAP (Constrained Application Protocol)

Standardized in `RFC-7252`, it's specialized for **web transfer** and it's suitable for machine-to-machine for use with constrained nodes and constrained networks. It used under the hood the UDP Protocol. The general key idea is pictured in 2.13

CoAP also support a direct connections between devices, without relying on an intermediate bridges to translate/support the communication.

CoAP is designed to work with nodes with *8 microcontrollers* with small amounts of ROM and RAM. Constrained networks such as IPv6 over 6LoWPANs: it concerns the size of the header, compressing addresses and header used in IPv6.
The main strenghts are that is *native UDP, support multicast, security mechanism are embedded in the RFC itself, suppport asyncrhonous communication by design.* There are also weakes as *standard maturity* and *message reliability not quite sophisticated, similar to MQTT QoS.*

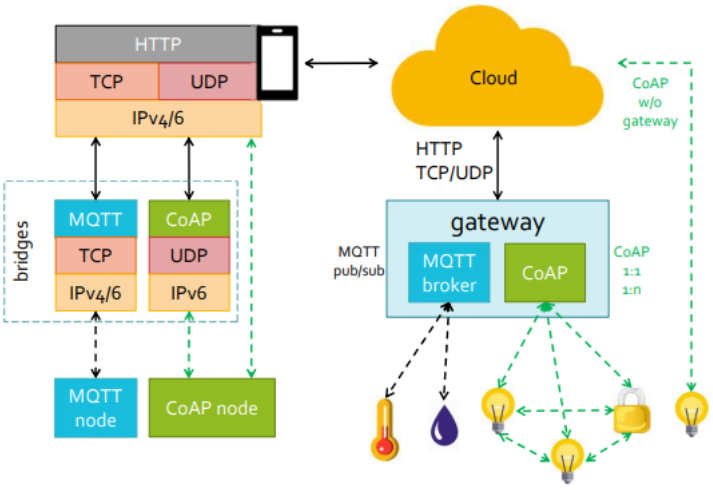


Figure 2.13: MQTT vs CoAP

Chapter 3

ZigBee

ZigBee covers almost all layer from *phyisical* to *application* layer: it's a standard protocol for wireless sensors and actuators networks thus is considered an IoT standard even if use TCP/IP as its internal mechanism. The protocol is developed and promoted by industrial private alliance called ZigBee alliance. It covers different application cases like *home automation, health care, consumer and industrial automation.* It's a competing standard with Bluetooth, especially in eHealth environment where ZigBee failed but had more successfull application in other sectors.

The main requirements that drove the designing of the protocol were:

- *Network completely autonomous*: self-organiziting with minimal or nothing manual intervention
- *Very long battery life*: ideally a ZigBee device should last 10 years
- *Low data rate*: allows to guarantee a very long battery life by optimizing performances. This allows trading performances for battery life.
- *Interoperability* of ZigBee devices from different vendors

This criteria resulted in some **main features**, like being *standard based, low cost* and *globally-enabled* by allowing a range of private frequencies based on user parameters (*there are constraints country-specific for frequencies*), *reliable and self-healing*, *large number of node support* by allowing up to 30.000 devices in a short ranges (200 *meters*) and also for larger range by supporting *multi-hop network.* Those combination of charateristics allows an easy deploy and a very long battery, also guarateeing advacend *security mechanism.*
Despite is not explicitly mentioned in the specific itself, the *transport layer* features are implemented by some **sub-layers** between network and MAC layer.

IEEE 801.15.4 Standard Define the specification of the **physical** and **MAC layers** for *low rate PAN.* It operates on top of the frequencies of WiFi and Bluetooth: using the same frequencies but not perceived or talk each other, just consider each other as *noise.* The frequency that are defined by the specification are:

- 868–868.6MHz (*e.g., Europe*) with a data rate of 20 kbps
- 902–928MHz (*e.g., North America*) with a data rate of 40 kbps
- 2400–2483.5MHz (*worldwide*) with a data rate of 250 kbps

3.1 ZigBee Standard

The components described in the ZigBee specification are pictured in 3.1.

The Application layer of ZigBee is defined by 3 sub-layer:

1. **Application Framework**: contains up to 240 ***Application Objects (APO)*** in which each APO is a *user-defined ZigBee application.*
2. **ZigBee Device Object (ZDO)**: provides services to let the APOs organize into a distributed application. Guarantee interoperability between different devices of different vendors

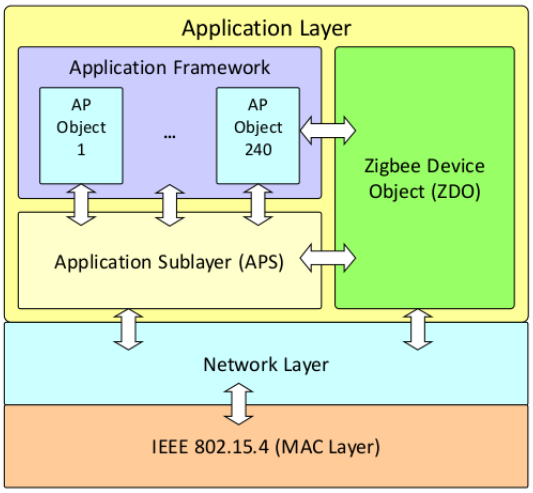


Figure 3.1: ZigBee Specification overview

3. **Application Support sublayer (APS)**: provides data and management services to the APOs and ZDO.

The workflow of those three sublayers will be addressed in detail in 3.1.5.

3.1.1 Service primitives

Each layer provides its data and magement service to the upper layer so **each service** is specified by a set of primitives of four generic types. Between all the services and all the layer, only **4 primitives** are used/consumed between each layer as sketched in 3.2:

- **Request**: It is invoked by the upper layer to request for a specific service;
- **Indication**: It is generated by the lower layer and is directed to the upper layer to notify the occurrence of an event related to a specific service;
- **Response**: It is invoked by the upper layer to complete a procedure previously initiated by an indication primitive;
- **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. Services may not use all the four primitive (*see later*).

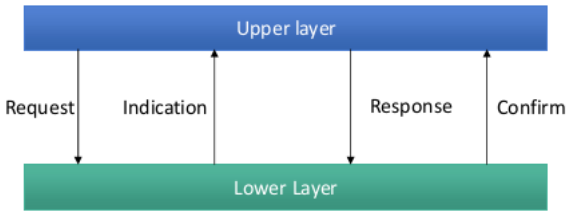


Figure 3.2: Layer interaction

In figure 3.3 an example of *service primitives* that allows to visualize the flow: same levels of different nodes interact by means of those 4 primitives.

Data transfer The data transfer between node assume an uniform name called **Data Unit**, encapsulated at different level with different packet layers, here specified in 3.4.

For each layer the *data unit* are specific:

- *Application Protocol Data Unit*
- *Network Protocol Data Unit*

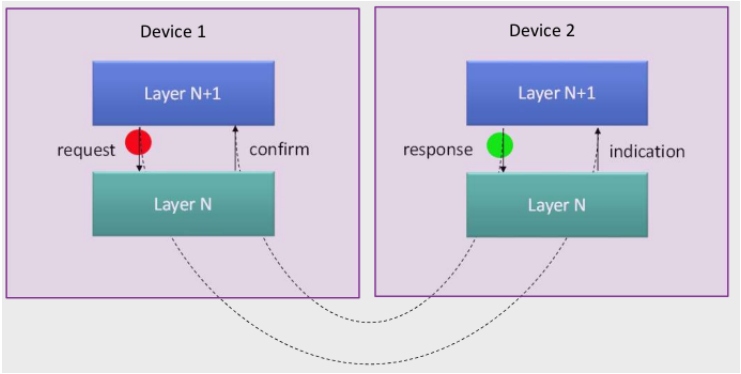


Figure 3.3: Communication between two devices, using basic primitives

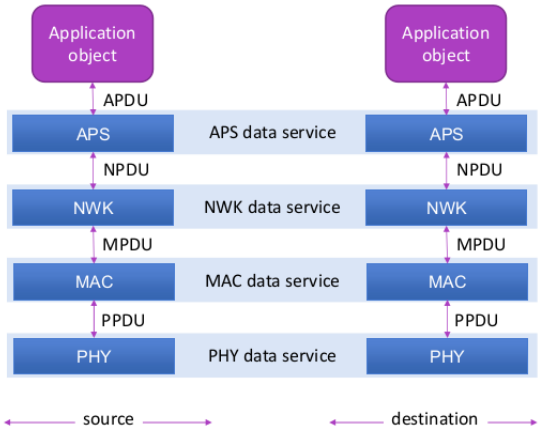


Figure 3.4: Data Unit per layer

- MAC Protocol Data Unit
- *Physical Protocol Data Unit*

3.1.2 Network Layer

At network level we identify 3 types of devices:

1. **Network coordinator**: Despite been a infrastructurless protocol, a coordinator have the function of create the network by taking decisions/interfacing with the rest of the network. Allows to bootstrap the network.
2. **Router**: provide routing features in a large network and can also have services capability. From the point of view of routing capabilities, they have the same capabilities of a coordinator.
3. **End device**: they are not able to communicate alone with the rest of the network and use coordinators/routers to communicate. They must be attach to the coordinator to be able to communcate over the network. They do not need to implement the entire standard, differently from router/coordinators that must have the capabilities to implement the standard fully. Generally they correspond to a RFD *Reduced Functional Interface* while routers/coordinator are FFD *Full Functional Device*.

We can have different **network topologies** as shown in 3.5. In a **star shaped topology**, the router have the same roles as end device: they uses the *superframe*. In the **tree topology**, the *multi-hop network* (*not fully addressed in those notes*) functionalities are supported and the nodes can communicate only via the intermediary routers: they can or cannot use *superframes*. In **mesh networks**, the communication is possible even between nodes witout relying on the *superframe infrastructure* (*superframe is the topic of the lecture on CSMA/CA, see later chapters*).

The **network layer** provides services for:

1. Data transmission (*both unicast & multicast*)

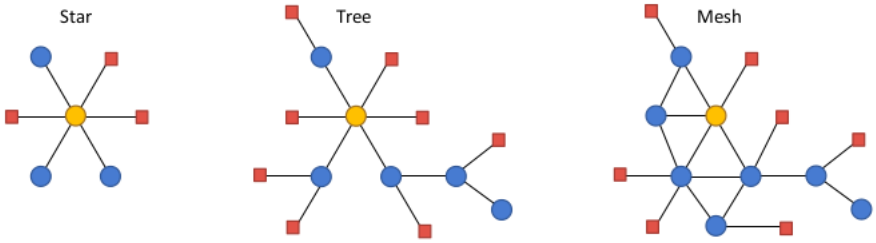


Figure 3.5: ZigBee supported network topologies

2. Network initialization
3. Devices addressing
4. Routes management & routing
5. *Management of joins/leaves of devices*: because the networks is formed incrementally, formed a network with an arbitrary topology.

The 3.6 table summarizes which services uses the 4 primitives already mentioned:

Name	Request	Indication	Confirm	Description
DATA	X	X	X	Data transmission service
NETWORK-DISCOVERY	X		X	Look for existing PANs
NETWORK-FORMATION	X		X	Create a new PAN (invoked by a router or by a coordinator)
PERMIT-JOINING	X		X	Allows associations of new devices to the PAN (invoked by a router or by a coordinator)
START-ROUTER	X		X	(Re-)initializes the superframe of the PAN coordinator or of a router
JOIN	X	X	X	Request to join an existing PAN (invoked by any device
DIRECT-JOIN	X		X	Used by the coordinator or by the routers to force an end device to join their PAN
LEAVE	X	X	X	Leave a PAN
RESET	X		X	Resets the network layer
SYNC	X		X	Allows the application layer to synchronize with the coordinator or a router and/or to extract pending data from it
GET	X		X	Reads the parameters of the network layer
SET	X		X	Set parameters of the network layer

Figure 3.6: Operations that exploit the 4 basic primitives

As highlighted by the table, not all four primitives are used in the listed operations: particularly, the `GET` and `SET` are functionalities used **locally to set/get parameters of the network layer**, internally, without using communication means.

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

1. Form a network so it's a **ZigBee coordinator**
2. Join an existing network so it's a **ZigBee router/end-device** The role of the device is chosen at **compile-time** by specifing the code ad application level when writing the **Application Object** code.

3.1.3 Network Formation

The *network formation* is initiated by the `NETWORK-FORMATION.request` primitive: it's invoked by a device that acts as a **coordinator** and that is not already in another PAN. The operation uses the MAC layer services to **look for a channel** that does not conflict with other existing networks: it **selects a PAN identifier** which is not alredy in use by other PANs. The process is pictured in 3.7

1. At application layer is invoked the `NETWORK-FORMATION.request`
2. The network layer execute an *energy scan* to verify if there is noisy on the channel

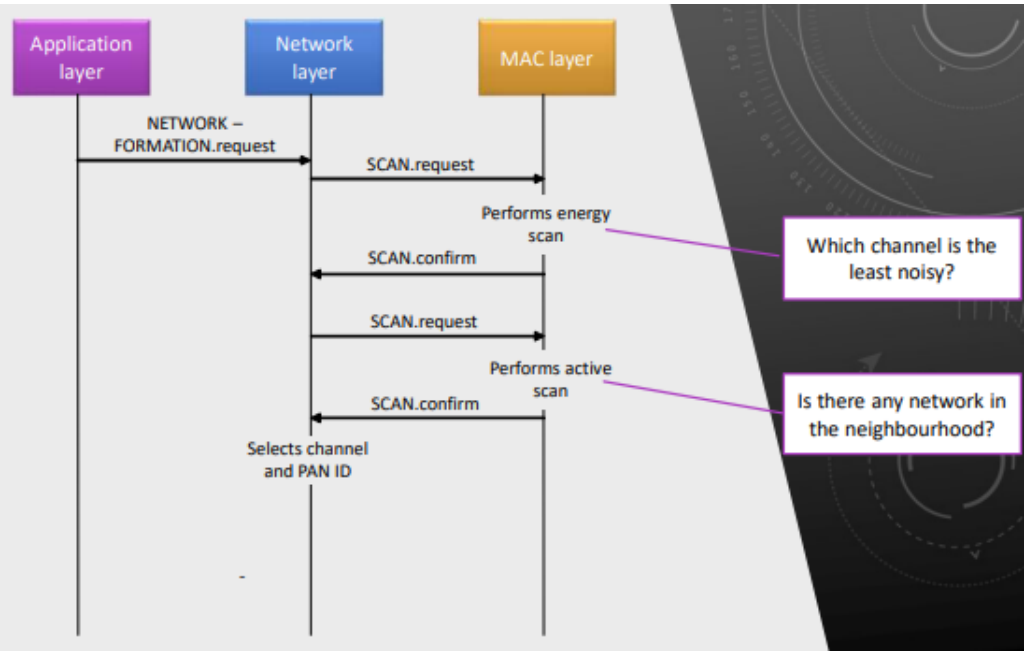


Figure 3.7: Network Formation: `SCAN` request exchange and PAN ID selection

3. The MAC layer return the result of the energy scan: at this point, the network layer request a second scan, differently from the first because it's an **actve scan** to lookup nearest PAN networks. This is due to avoid to have two different PAN with the same identifier so a preventive scan allows to select a PAN ID unique among the already existing.

4. The MAC Layer return a list of existing PAN ID and the Network Layer now can set the PAN ID.

The process of `NETWORK-FORMATION` continues as showed in 3.8

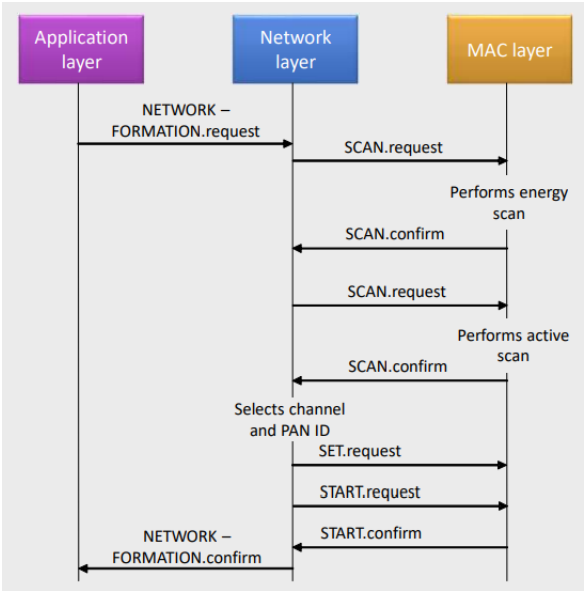


Figure 3.8: Network creation confirmation after `START.request` is completed

Now the Network layer can configure the MAC layer starting from the time the `START.request` is sent thus the MAC layer can start to send beacon frames. After that, the MAC layer can answer with a `START.confirm` that will be translated also by the network layer with a `NETWORK-CONFIRMATION.confirm` to the application layer to confirm that the PAN has been created.

3.1.4 Device join

To **join** inside a PAN network there are two ways:

1. The end-device or the router can communicate with the coordinator to request the access: this method is called **Join Through Association**
2. The coordinator request a device to join the network: this method is called **Direct Join** (*no further addressed in this notes*)

Join through association (`CHILD-SIDE`) The protocol flow is pictured in 3.9

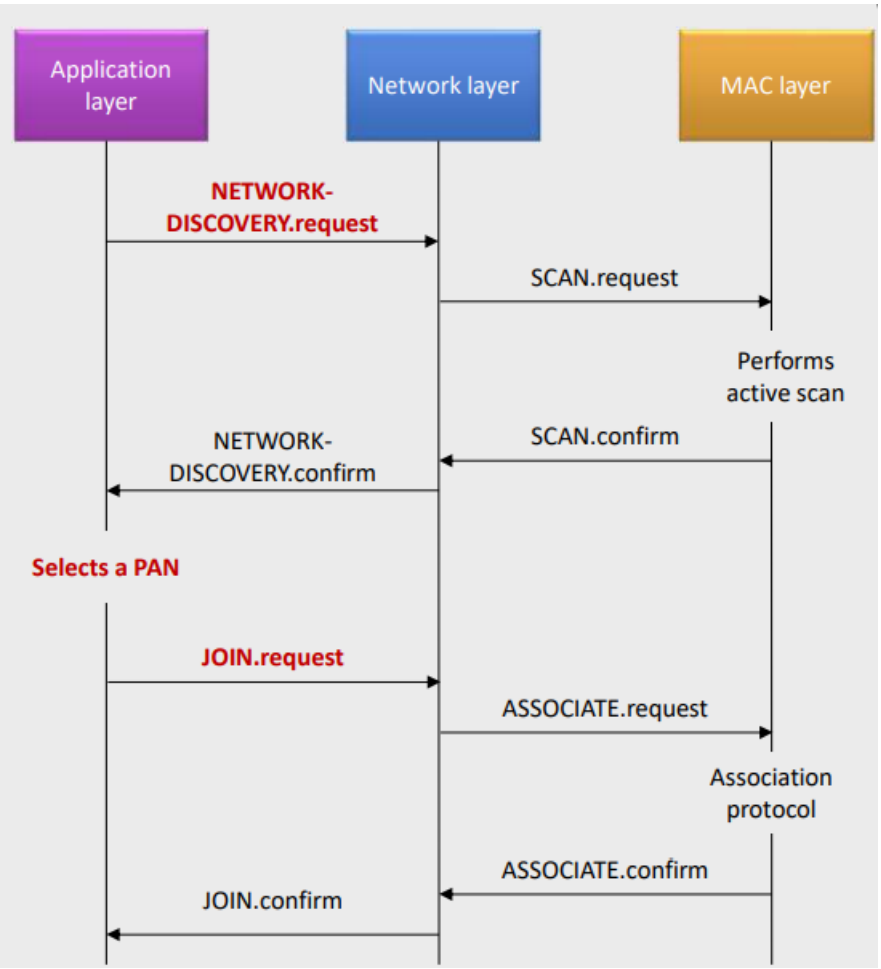


Figure 3.9: Join Through Association

The device that wish to join a PAN:

1. Performs a `NETWORK-DISCOVERY` to look for existing PANs
2. Select a PAN
3. Invokes `JOIN.request` with parameters like the *PAN Identifier* of the slected network, a *flag* indicating whether it joins as a router or as an end device. This request in the *network layer* selects a "*parent node P*" (*in the PAN*) from its neighbourhood. In a **star topology** `P` is the coordinator and the device join as end-device, while in a **tree topology** `P` is the coordinator or a router and the device can join as a router or as an end-device.
4. The association protocol obtains from `P` a $16-bit$ **short address**: the `ASSOCIATE.confirm` returns to the network layer the short address that will be used in any further communication over the network. The `JOIN.request` is triggered at application level: this require preliminarly to scan the network to find a free channel to communicate. The scan is perfomed troguht `NETWORK-DISCOVERY` which return a list of PAN network to connect: this allows to select the PAN and send the `JOIN` request.

Depending on the three network topologies supported by ZigBee, the **parent-child** relationship established as a result of *nodes joins* shapes the network in a **tree** form, where:

- the ZigBee *coordinator* is the **root**
- the ZigBee *routers* are **internal nodes** *(or even leaves)*
- the ZigBee *end-devices* are the **leaves**

The *tree-based topology* is exploited by the **ZigBee Network Layer** to assign the **short network addresses** based on a specific policy defined by the *ZigBee coordinator*, defined on three main parameters:

- the **maximum number of routers** `Rm` that each router may have as a children
- the **maimum number of end-devices** `Dm` that each router may have as a children
- the **maximum depth of the tree** `Lm`

Each router is assigned with a range of addresses to be recursively assigned to its children: devices join as high to the three as possible to *minimize the number of hops*. Despite the addresss are assigned according to a tree structure, the actual network topology can be a *mesh*.

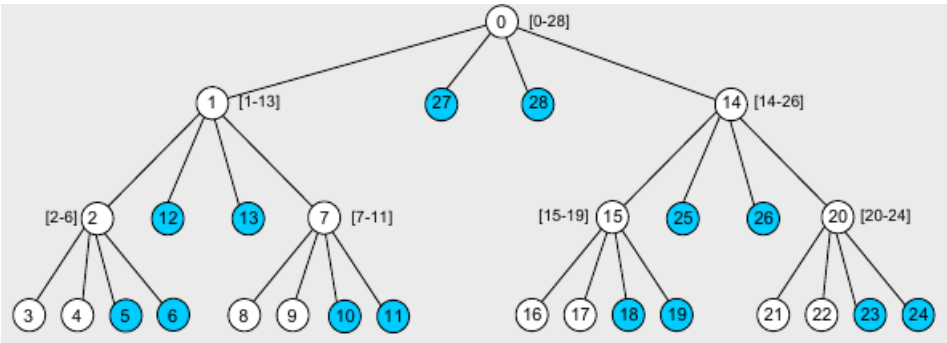


Figure 3.10: Address assignment with $Rm = 2, Dm = 2, Lm = 3$ (*router are white, end-devices are blue*)

In 3.10, the coordinator have index 0 and considering the left children 1, the range $[1-13]$ is assigned based on the parameter Rm, Dm, Lm, considering the worst-case scenario in which all addresses in this range are used by either a device or a router. The maximum number of nodes (*both end-devices and routers*), in the given example is 28 because is the threshold given by the three parameter already specified: to have a larger number of nodes it's necessary to set those three parameter to values that allows more nodes. This value limit the number of new devices that can join the network, so new devices cannot enter the network if this limit is reached.
This mechanism allows to **pre-define the set of the addresses to be assigned to joining nodes**: in advance we already know the size of a given subtree (*knowing the max depth and number of children per node*) thus we can allocate the addresses range for each sub-tree.
The range assignation is performed following a DFS order, considering sub-nodes children routers (*in 3.10, nodes* $2, 7$) range as previously.
At network layer we can also define the specific type of **routing protocol** that ZigBee can use:

- *Broadcasting*
- *Tree routing*
- *Mesh routing*

The last two types are for *point-to-point* communication.

Tree routing Consider the scenario pictured in 3.11.

The node 3 that want to communcate with node 25 must forward the packet to node 2 because is his father. The node 25 is outisde of node 2's range so the packet will be forwarded to node 1 and again to node 0 which is the root node. The packet will be forwarded to 14 and then to node 25. All those communciation happen between routers, except the final one that is between the router 14 and the end-device 25: for the last hop, the node 14 can advertise the node 25 of the presence of a new message in its *beacon message*, so the node 25 will request the message itself to node 14.

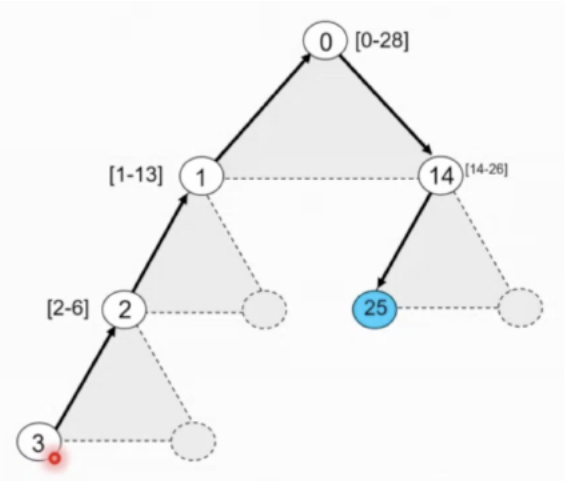


Figure 3.11: Tree routing scenario

Mesh routing It's based on *AODV protocol* (*which is not in the scope of the following notes*): the key idea is that each router have its own routing table, mantaining destination, next hop and a set of information about the path to a specific node.

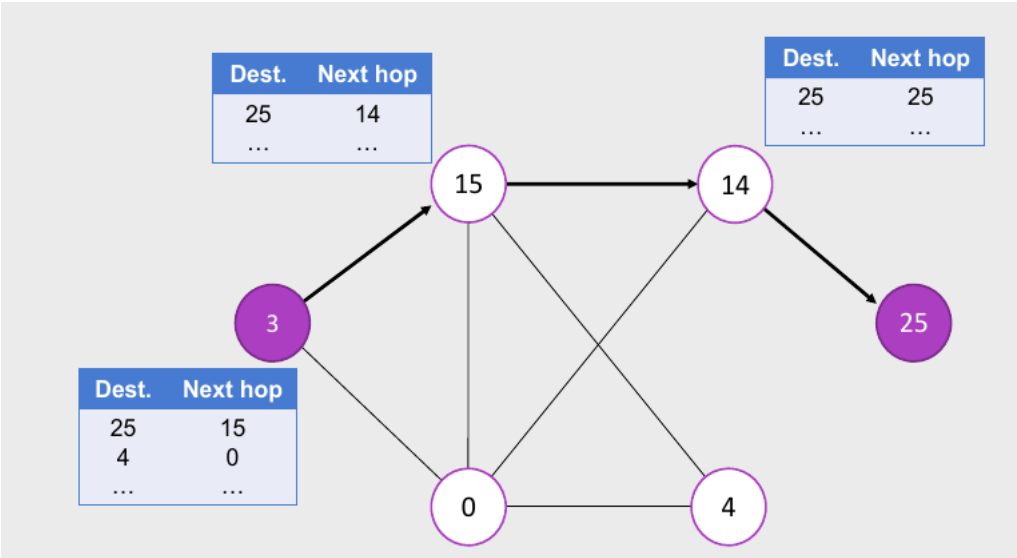


Figure 3.12: Mesh routing scenario

If the communication happen between the node 3 and node 25, the source node will verify that the destination exists in an entry of its routing table: if the entry does not exists, a **path discovery** operation is triggered. The path discovery operation exploit the *broadcast* method, containing the *request ID* and the *path cost*, initialized to 0: the request is propagated inside the network so each intermediary router can answer if an entry with the target destination is found in one of the node's routing table.

3.1.5 Application Layer

The application layer comprends various components (*as shown in 3.13*):

1. **Application Framework**: contains up to 240 **Application Objects (APO)** in which each APO is a user defined ZigBee application.
2. **ZigBee Device Object (ZDO)**: provides services to let the APOs organize into a distributed application. Guarantee interoperability between different devices of different vendors
3. **Application Support sublayer (APS)**: provides data and managemnt services to the APOs and ZDO.

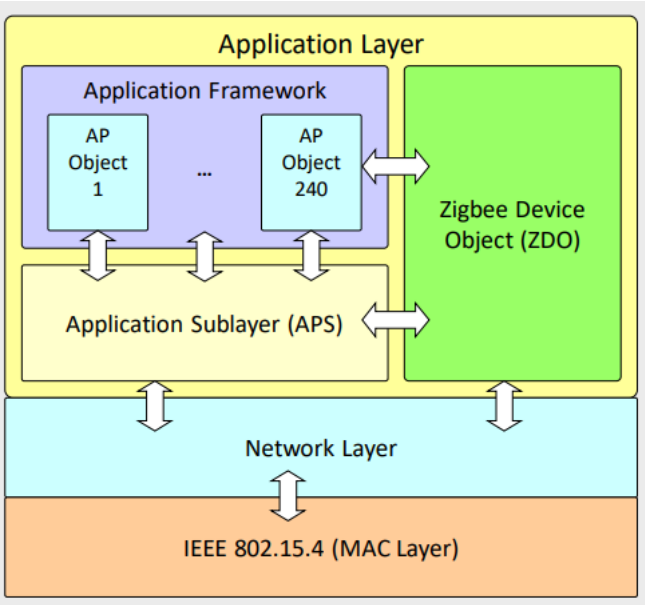


Figure 3.13: ZigBee Application layer segmentation in three sub-layers and interaction with network layer

3.1.6 1. Application Framework

The application framework can contains up to 240 different *Appication Object (APO)*, each one uniquely identified inside the netowrk. Simplest APO can be queried with *Key Value Pair data service (KVP)* by using primitives like `Set/Get/Event` transactions on the APOs attributes: native KVP disappeared in the second version of ZigBee but it's now present in the cluster library. Complex APO can have conpelx states and communicate with the *Message data Service*.

In figure 3.14 an example of complex APO: we have two devices, the first with two endpoint, the second with three endpoint. The logic of this device can be easily implemented using two `ON-OFF` attribute and two endpoint in the device A. **APOs 5B, 6B, and 8B** have a single attribute containing the status of the bulbs (on/off) that is configured as servers at the application layer while **APO 10** and **APO 25** are switches, and they are configured as clients at the application layer. The attributes of **APOs 5B, 6B**, and **8B** can be set remotely from the **APOs 10A** and **APO 25A**.

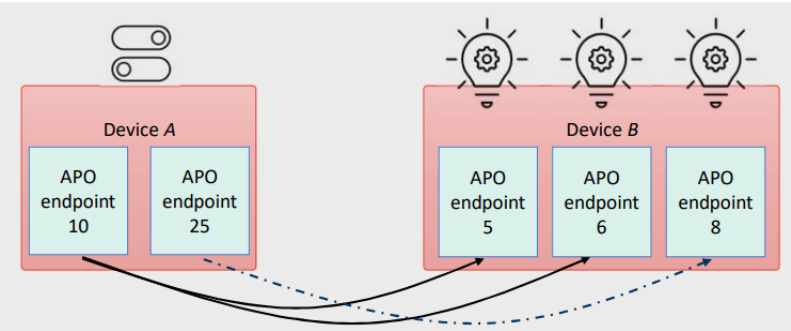


Figure 3.14: APO bulbs example

3.1.7 2. Application Support Sublayer (APS)

The APS define the **endpoint, cluster, profile IDs and device IDs**. It's responsible for:

- Data service (*light transport layer*) that provides support for the communication's reliability and packet filtering
- Management services which includes mantaining a local binding table, a local group table and a local addresss map.

The **endpoints** are identified by a number between 1 and 240: an endpoint can be seen as the equivalent of a Unix socket, connecting virtual wires to applications. The **cluster** is a protocol that can be used to implement the functionality of a given application: cluster defines the attribute that contains the information about the APO. A cluster is defined by a **clusterID** of ***16 bit*** (*as shown in 3.15*), local inside the **profile**. In ZigBee a **profile** defines a set of application layer protocols that enable interoperability between devices that perform specific functions or serve specific purposes. A profile specifies the standard ways in which devices should communicate and exchange information, making it easier for devices from different manufacturers to work together.

A profile consists of one or more **clusters**, which are groups of related functionality that can be supported by a device.

A cluster defines a specific set of **messages and commands** that devices can use to communicate with each other. *For example*, the ZigBee Home Automation profile includes clusters for controlling lights, thermostats, and door locks. Each cluster specifies the **data format and message types** that devices should use to exchange information related to that functionality.

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

Figure 3.15: Prefefined clusters ID

In the previous table, the *Basic Cluster* with ID `0x0000` allows to retrieve basic information about the device, the *Power Configuration Cluster* allows to set the *minimum and the maximum temperature* of the device. Also, the *OnOff Cluster* allows to set the *switch* ON or OFF, as expected.

The *APS* ***Device IDs*** range from `0x0000` to `0xFFFF`, this value has two main purposes:

1. To allows human readable displays (*e.g. an icon is realted to a device*)
2. Allows ZigBee tools to be effective also for humans: a device may implement the on/off cluster but you don't know whether it's a bulb or a oven so you only kwno you can turn it on or off. The DeviceID tells you what it is, but it does not tell you how you can communciate with it: this is given by the IDs of the cluster it implements.
   ZigBee discovers services in a network based on profile IDs and cluster IDs, but not on device IDs alone because each device is specified by the combination of Application profile ID and cluster ID.

In figure 3.16 an example of cluster IDs.

APS Services

The **APS** ***Services*** provides data service to both APOs (*Application Objects*) and ZDO (*ZigBee Device Object*), allowing to ***bind a service to the ZDO*** and ***group management services*** (*see later*). APS data service enables the exchange of messages between two or more devices within the network: here

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

example from the Home Automation Profile

Figure 3.16: Cluster ID example

the data exchange is defined in terms of primitiviles like `request(send), confirm(return status of trasnmission), indication(receive).`

The reliability of a data service is ensured by using the **ACKNOWLEDGE messages** for data trasmission as the schema 3.17 indicates.

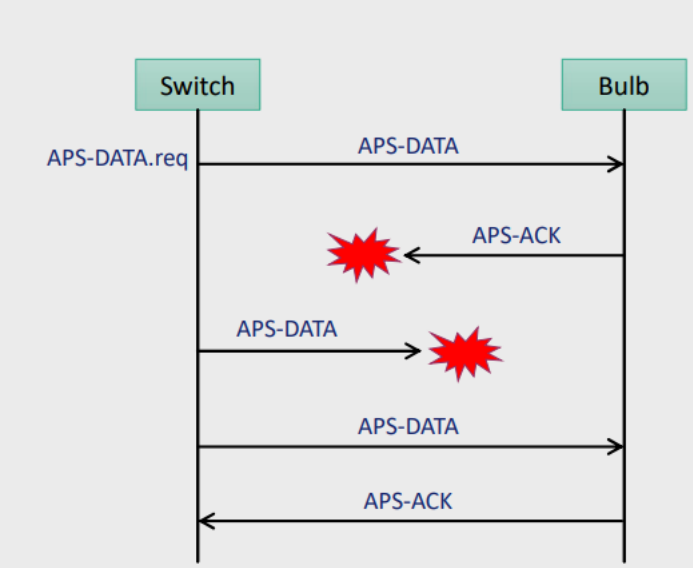


Figure 3.17: APS exchange diagram

The ***APS Group Management*** provides service to build and mantains groups of APOs: a group is identified by a $16-bit$ address and each APO in the group is identified by the pair network *address/endpoint*. The `ADD-GROUP` primitive allows to add an APO to a group and `REMOVE-GROUP` primitive to remove an APO from a group: takes group number and endpoint number; if the group does not exists it's created. The information about groups are stored in a *group table* in APSs.

APS Binding The binding operation allows an endpoint on a node to be connected (*bound*) to one or more endpoints on another nodes: the binding is **unidirectional** and can be configued only by te ZDO of the coordinator or of a router.
So the binding provide a way to specify the destination address of the messages. This method is called **Indirect Addressing**: when a Message is routed to the destination APO based on its pairs `<destination endpoint, destination network address>` the method is usually called **Direct Addressing**. Direct addressing might be unsuitable for extremely simple devices while *Indirect addressing* exploit the ***Binding table***.

Two main primitives are used for binding in **Indirect addressing**, respectively `BIND` and `UNBIND`:

1. `BIND.request` creates a new entry in the local binding table. It takes in input the tuple

`<source address, source endpoint, cluster identifier, destination address, destinatione endpoint>`

2. `UNBIND.request` deletes an entry from the local binding table

In the **Indirect addressing**, the binding table matches the *source address* `<network addr, endpoint addr>` and the *cluster identifier* into the pair `<destination endpoint, destination network address>`. The binding table is stored in the APS of the ZigBee cordinator and/or of the other router: it's updated on explicit request of the ZDO in the routers on in the coordinator. It's usually initialised at the network deployment.

In the scenario of light bulb and a switch, can be complex from the switch side to know to which light bulb connect but, this information is known by the installer. Despite operating directly on the device, the installer can set and entry in the **coordinator's binding table** so that the messages arriving to the switch will be forwarded to the light bulb.

Here an example of **binding table** with a request `APS-DATA.req` of cluster 6 from endpoint 5 on node `0x3232` that in **indirect routing** will generate 4 data requests:

- Node `0x1234`, endpoint `12`
- Multicast to group `0x9999`
- Node `0x5678`, endpoint `44`

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 3.18: Binding table example

In the binding table showed in 3.18, the `Src Addr` and the `Dest Addr` are formatted as a MAC addresses: this choice has been made because the devices can disconnect/re-connect to the network for many reasons, requiring a reconfiguration of the addresses in the binding table. The column `Addr/Grp` allows to tag as destination a group of devices (`G`). Also, the same devices can be the source of different messages (`Src EP = Endpoint`).

The *APS* ***Address Map*** 3.19 allows to associated the $16-bit$ network address with the $64-bit$ IEEE MAC address: ZigBee end devices (*ZED*) may change their 16 bit network address when they leave and join again. In that case, an announcement is sent on the network and every node updates its internal table to preserve the binding.

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

Example of APS address map

Figure 3.19: Address map example

3.1.8 3. ZigBee Device Object (ZDO)

ZDO is a *special application* attached to ***endpoint 0***: it implement the basic functionalities expected by the devices. It also implement the end-device or router or coordinator based on the specific configuration given by the **ZigBee Device Profile** that describe the cluster that must be supported by any ZigBee device, defines how the ZDO implements the services of discovery and binding and how it maanges the

network and security.
So the main ZDO services are *device and service discovery, binding management, network management, node management.*

Device and service discovery

The **ZigBee Device Profile** specifies the device and the *service discovery mechanisms* to obtain any information about devices and services in the network. The **Device Discovery** allows a device to obtain the (*network or MAC*) address of other devices in the network: starting from an unknown address it allows to identify the relative device inside the network. It follow an **hierarchical implementation** in which a router return to its parent its address and the address of all the end-devices is associated to itself. Then the coordinator returns the address of its associated devices.
A newly joined device does not know anything about the network: to provide a service it must also known, require and use the services offered by other devices. This actions are performed thanks to **ZigBee Device Profile** that allows the service discovery of other devices.
The service discovery send queries that are based on *cluster ID, profile ID* and optionally on *service descriptors*: the coordinator responds to service discovery query by returning lists of endpoint addresses matching with the query. The hierarchical implementation is effective because *each router collect information from its associated devices and forwards it to its parent.*

The other mechanism to understand what service a device is providing is called **Service Discovery**: the network is created according to a parent-child relation (with a coordinator), building as a tree (*as the addresses tree previously mentioned*). To known which service a device offer, a joined device ask to its coordinator that may not have fully all the information and may ask to it's parent node, recursively. Each node is responsible for the subtree under its domain: in case of missing information, intermediate routers prepare information report to answer the queries.
The **Service Discovery** provides a report that tells for a single device what are the cluster to which it responds: for example, a device can responds telling that implement a ON/OFF cluster. The service discovery can be direct to the coordinator and is it's responsability to collect information and answer for the whole network. The requests can be directed also to end-device but due to low-power devices, they may not be ready to answer the queries regarding the services they offer. The service discovery does not tell where the device is physically located, only the service offered and their relative identifier (Cluster ID, Device ID, etc): to obtain the location is necessary that a device implements the **Identity Cluster** that force a device to identify itself by blinking a led.

Binding management The ZDO processes the binding requests received from local or remote EP: both to add entries and delete entries from the APS binding table (*by invoking primitives*). This process require having an IEEE address.

Managing network and nodes The *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. Differently, the *node management* involves the ZDO to serves incoming requests aimed at perfoming network discover, retrieving the routing and binding tables of the device and managing `join/leaves` of nodes to the network.

3.1.9 ZigBee Cluster Library (ZCL)

The ZCL is a repostory for cluster functionalities: it's a *working library* with regular updates and new functionalities. The develoeprs are expected to use the ZCL to find relevant cluster funcionalities to use for their application. The main goal is to avoid re-inventing the wheel, supports **interoperability** and facilitates maintanability: two devices must implement the same application layer to use each other. The ZCL allows to implement the same set of commands and functionalities, standardized by ZCL.
For example, in a *base home automation system* we have many application execute at the same moment: the ZCL allows to specify the behaviours of the involved devices based on a ***client-server model***.

A **cluster is a collection of commands and attributes**, which define an interface to a specific functionality. The **device** that stores the attributes, by keeping an internal state, is the **server of the cluster** while the device that manipulates/writes the attributes is the **client of the cluster**. It's not necessary that a server have more computational power than end-devices.

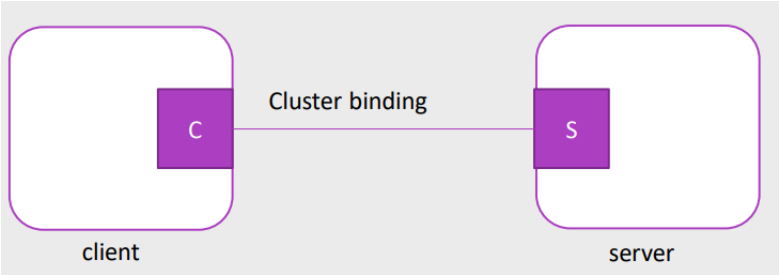


Figure 3.20: Client-server cluster binding operation

If we think an application that switch the light, it will have an attribute `ON/OFF` and the client will be the switch that turn on/off the light while the server will be the light bulb itself. Cluster are grouped in **functional domains**:

- *General*: allows the access and control of any device, irrespective of their functional domain. Includes general basic functions, implemented by all devices.
- *Closures*: for shade controllers, door locks.
- *HVAC*: for pumps (*like fan, heating, etc*)
- *Lighting*: control lights
- *Measurement and sensing*: illuminance, presence, flow, humidity
- *Security and Safety*: security zone devices (*e.g. passive infrared*)
- *Protocol interfaces*: interconenct with other protocols

Each functional domain, for each server, defines **commands** and **message structure**. The **commands** are messages (*with their own header and paylaod*) with a format specified by ZCL: the commands can be read/write attribute or also can be used for dynamic attribute reporting (*useful to periodic reading of the sensors*). The *types of command* are generally three:

1. Read/write an attribute
2. Configure a report and read a reporting response: allows to request periodic attribute/configuration reading. Requests for report whenever an attribute/configuration changes. The client ask the server to report periodicly about the state of the server: the server will send periodically a report to the client, withoyt any further request. Also defines parameters of the reporting like *duration, period, minimum change, etc.*
3. Discover attributes: to discover the IDs and types of the attributes of the cluster that are supported by the server

The *schemes of use* 3.21 show a typical use of **device configuration and installation cluster:** the configuration phase allows to setting up some parameters by using several clusters. The `Basic` domain allows to retrieve software version, manufacturer ID, etc. The `PowerConfig` return information about battery and power configuration. The `Temperature Configuration` allows to set treshold, etc. Last, the `Identify` cluster allows to identify physically a device.

Following the schema, usually the installer configure the devices by using specific configuration tools that allows to connect to the devices, apply the specific settings and read their status. As an example, the `Identify` allows to recognize the device by enabling a blinking led.
Applied to our `ON/OFF` previous example, the relative scheme obtained is pictured in 3.22
In this case we want to configure a set of APO, we have one ON/OFF switch which is connected with a "*Simple Lamp*", we have to create the association between the lamp and the switch, in the same way we must also create the association between the "dimmer switch" and the "dimmable lamp".

This association is created by the **Configuration Tool**, working as a client sets the configuration of the On/Off switch device in such a way that when it wants to turn the lamp on or off it will find the corresponding device and make the request for turning on or off. The same goes for configuring the Dimmer Switch. The configuration tool must also configure the simple lamp. The configuration allows

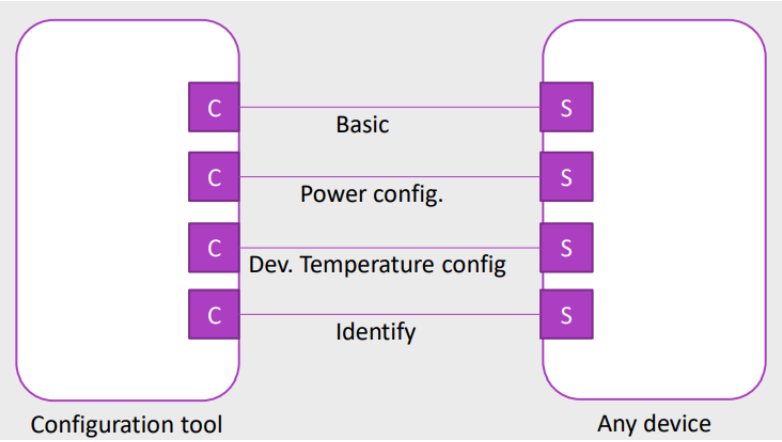


Figure 3.21: Common functional domains use between client and server

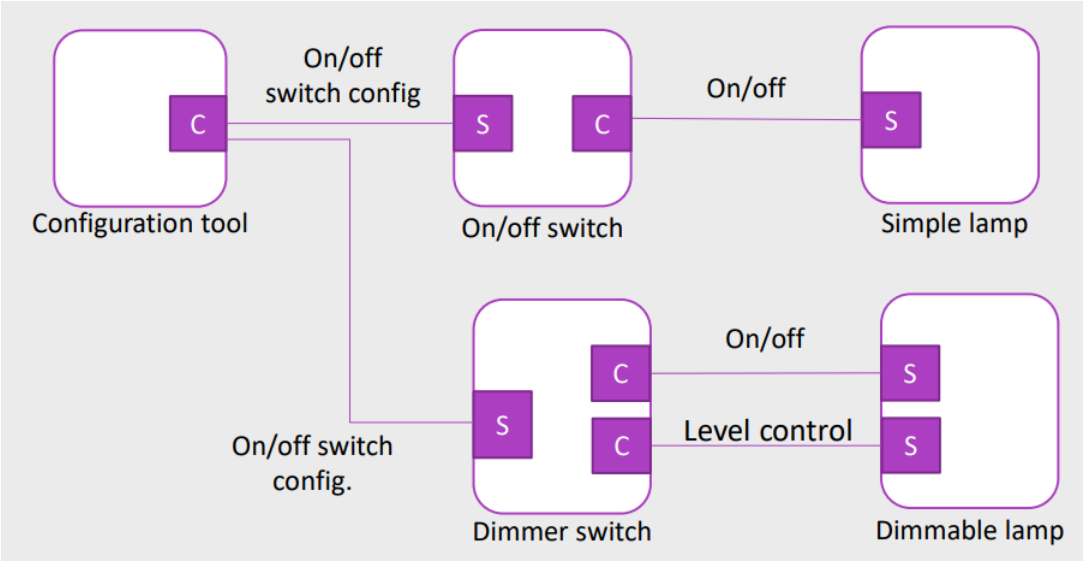


Figure 3.22: Lamp switches configuration diagram and interconnection

to setup the **binding table** to the **coordinator** by knowing the addresses of the devices and relative APO. This can be done by invoking device and service discovery, alongside physical discovery.
Typically what happens is that the installer has a tablet and it connects to the Zigbee Network, using an application you can access information about the network. When I run this configuration each device knows what it is and what it can do, a lamp for example knows it is a lamp but it is necessary to carry out a **commissioning procedure** so that the lamp is connected to an on/off switch which allows the ignition. It's necessary to execute the procedure of *commissioning* to simplify the deployment of complex smart system, like in a scenario with hundreds of devices to be deployed and configured.
Each devices has many of information assigned to it, contained in the `BASIC` cluster, like the ones listed in 3.23.
The `ZCLVersion` is mandatory to be sure that the ZCL is the right one implemented at application level, specially due to continue updating. Also the `PowerSource` is mandatory because provide information about power sources. Usually the attributes in `BASIC` cluster are in *read-only* access.
Among those information, highly relevant are the `PowerSource`, an 8-bit number that can have the values listed in 3.24.
The value in the last table indicates the **type of battery**, if there is an emergency power supply, etc. Those devices have also the attribute called **tempetature measurement cluster** that allows to kown the tempetature *inside* the device itself. The attribute to know this temperature are specified by `Temperature` cluster as listed in 3.25.

The cluster specified uses only generic commands to *read attributes, discover attributes, configure and report.* There are no specific commands to program a simple devices as a temperature measurement.For complex devices that also implement part of the `Temperature` cluster there is usually an accessory ap-

Identifier	Name	Type	Range	Access	Default	Mandatory / Optional
0x0000	*ZCLVersion*	Unsigned 8-bit integer	0x00 – 0xff	Read only	0x01	M
0x0001	*ApplicationVersion*	Unsigned 8-bit integer	0x00 – 0xff	Read only	0x00	O
0x0002	*StackVersion*	Unsigned 8-bit integer	0x00 – 0xff	Read only	0x00	O
0x0003	*HWVersion*	Unsigned 8-bit integer	0x00 – 0xff	Read only	0x00	O
0x0004	*ManufacturerName*	Character string	0 – 32 bytes	Read only	Empty string	O
0x0005	*ModelIdentifier*	Character string	0 – 32 bytes	Read only	Empty string	O
0x0006	*DateCode*	Character string	0 – 16 bytes	Read only	Empty string	O
0x0007	*PowerSource*	8-bit enumeration	0x00 – 0xff	Read only	0x00	M

Figure 3.23: `BASIC` cluster information

Attribute Value	Description
0x00	Unknown
0x01	Mains (single phase)
0x02	Mains (3 phase)
0x03	Battery
0x04	DC source
0x05	Emergency mains constantly powered
0x06	Emergency mains and transfer switch
0x07 – 0x7f	Reserved

Figure 3.24: `PowerSource` attribute

pendix that specify other commands.
ZCL is designed to enforce a hierarchical approach to device functionality by supporting backward compatibility and interoperability, as shown in 3.26: instead of design every possible ZCL cluster for specific *lamp*, it's incremental and allows compatibility by also let the possiblity to create custom features not standardized by ZigBee.

3.1.10 ZigBee Security Specification

Security services provided for ZigBee include methods for:

1. Key Establishment: create the keys without prior information
2. Key Transport: exchange keys previously setted/stored at configuration time
3. Frame Protection: by using encryption
4. Device Management

Security services are optional and have an overhead, minimum for low power devices. These services form the building blocks for implementing security policies within a ZigBee device.
The underlying assumptions are that **level of security** depends on:

- the safekeeping of the symmetric keys: by avoiding stolen information about keys, also physically
- the protection mechanisms employed,

Identifier	Name	Type	Range	Access	Default	Mandatory / Optional
0x0000	*MeasuredValue*	Signed 16-bit integer	*MinMeasuredValue* to *MaxMeasuredValue*	Read only	0	M
0x0001	*MinMeasuredValue*	Signed 16-bit integer	0x954d – 0x7ffe	Read only	-	M
0x0002	*MaxMeasuredValue*	Signed 16-bit integer	0x954e – 0x7fff	Read only	-	M
0x0003	*Tolerance*	Unsigned 16-bit integer	0x0000 – 0x0800	Read only	-	O

Figure 3.25: Temperature cluster data definition

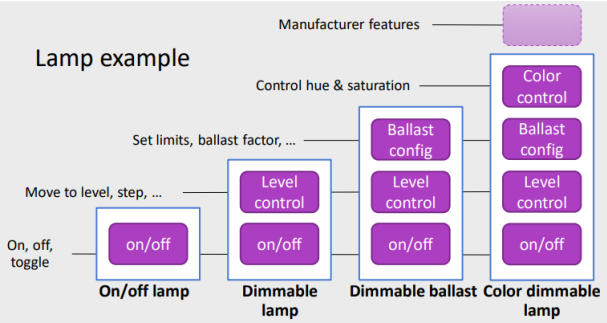


Figure 3.26: Hierarchical approach for lamp example

- the proper implementation of the cryptographic mechanisms and associated security policies involved. **Trust** in the security architecture comes from:
- trust in the ***secure initialization*** and installation of keying material and
- trust in the ***secure processing and storage*** of keying material.

Further assumptions are:

- **correct implementation of all security protocols**: the implementation must follow the standard, according to ZigBee standard and deployment operations
- correct random number generators
- secret keys do not become available outside the device in an unsecured way. The only exception to this is when a device joins the network.

Due to the *low-cost nature of ZigBee devices* we **cannot generally assume the availability of tamper resistant hardware**: it's possiible to some extent but it's not provided by the ZigBee standard (*can be possible by anti tamper hardware up to a given point*). The level of protecntion is both at network level of physical level. 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. Also, **different APO 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.

Security Design Choices The security is granted by the **Trust Center** that it's a module in the *Application Layer*, tipycally allocated in the *ZDO* that distribute the security keys to the devices inside the same network. Two main keys are provided, at different level:

- **Single key per network** (*network level security*): each Zigbee network is assigned a unique network key, which is a symmetric encryption key. This network key is shared among all devices within the network and is used to encrypt and decrypt data transmitted between them. Since it is a symmetric key, the same key is used for both encryption and decryption.

- **Single key per link** (*device-to-device level security*) The process flows: the devices inside the network uses a **Master key** to connect to the **Trust center**: this master key can be pre-assigned to the device (*write in ROM/Firmware*) or can be injected later in the device. The master key allows to establish a connection to request the **Network keys** or **Link keys**.
  Those three key allows ZigBee to provide a level of encryption corresponding to $AES-128$ that it's enough for low power devices. The keys are provided by the APS. The low power and computing constraints of the devices implies that the encryption is **symmetric** and the decryption is made by the keys provided by the *application layer*. It aims to the protection of individual devices but not individual applciations in the same device: this allows the re-use of the same keying material among the different layers of the same device.

The layer that originates a frame is responsible for initially securing it: if a network command need protection this must be ensured at network level security. If protection from *theft of service* is required, network layer security must be used for all frames (*except the one when new device joins*). The reuse of key materials among different layers in a device and link-keys allows to have additional security from source to destination: an application can extends the security by adding other mechanism but it's entirely upon itself.

An *application profiles* should include policies to:

- Handle error conditions arising from securing and unsecuring packets: error conditions may indicate loss of synchronization of security material or may indicate ongoing attacks.
- Detect and handle loss of counter synchronization and counter overflow.
- Detect and handle loss of key synchronization.
- Expire and periodically update keys, if desired.

The keys The keys can be obtained by means of:

- **Key transport**: a mechanism for communcating a key from one device to another dervice or other device. It's a sensible phase because at that time the communication is unencypted: can be encrypted but with more overhead, depending the scenario context.
- **Key establishment**: a mechanism that involves the execution of a protocol by two devices to derive a mutually shared secret key
- **Pre-installation**: at manufacturing time, install the key into the device or during the deploying phase

The **Network key** are *128-bit* shared by all devices: they are acquired either by key-transport or pre-installation and generally are of two types (***standard vs high security***).
The **Link keys** are 128 bit, shared by all devices: they are acquired by key-transport, key-establishment or pre-installation. The key establishment is based on pre-existing master key. Two types are considered (***Global vs Unique***).
The **Master key** is a 128 bit key acquired either by key transport or pre-installation. Other keys are used to implement the secure transport of key material: they are derived from the link key with one-way functions.

The **Key Establishment** it's stated by the specification: it describe a variation of Diffie-Hellmann and it's called **SKKE (Symmetric-Key Key Establishment)**. Usually the key establishment involves two entities, an initiator device and a responder device. It evolves as following:

1. *trust provisioning based on trust information (typically the master key)*: the master key can be either pre-installed during manufacturing or it may be installed by a Trust Center (*for example, from the initiator, the responder, or a third-party device acting as a Trust Center*), or it may be based on user-entered data (*for example: PIN, password, or key*)
2. exchange of ephemeral data (a random number)
3. the use of this ephemeral data to derive the link key.
4. confirmation that the link key was computed correctly

The **Trust Center** is the device trusted by devices within a network: it's a funcionality implemented in the coordinator or a specified device. In a simple network can be the coordinator and it's usually used to obtain new **network/link key** by connecting to it and requesting new keys. All members of the network shall recognize exactly one Trust Center and there shall be only one in each secure network to provide a trust agreement between all the device in the same network.
The device that joins the network:

- obtain the keys from the trust center according to different protocols, depending on which keys they already have (or no key)
- in low-security applications the Trust Center give it the master key by using unsecured transport
- If the device already has the master key, it can establish a secure communication link with the Trust Center to transport the keys.

Chapter 4

IoT Design Aspects

IoT are special custom devices that need to manage the energy efficiency differently from standard market device. The devices we refer are usually charaterized by *low power, low cost, small size* and *entirely autonomous*, being able to perform operative operations without the human intervention.

Each device is a full micro-system that embeds:

- Processor
- Memory
- Radio transceiver
- Battery powered or energy harvesting methods: depending on the specific scenario, both allows to avoid wired powered devices.

The **energy efficiency** based on battery-powered implies that the working life is limited in time: this implies the need to intervene on the field to change the battery. This operation can be costly, respect to the device cost itself. Some other issues are *adaptability to changing conditions* that arises for dynamic network management and programming, adapting its behavior as the environment change. This implies the necessity of *dynamic programming* to also preserve the interoperability.
Another problem is that the devices comes in term of very *low complexity and overhead*, with reduced complexity of operation execution and constrains on computational resources. Another issues concerns *multi-hop communication*: standard routing protocols like Distance Vector or similar relies on address table but due to constrained capacity both in term of storage and computational resoures, those model does not scale in IoT context multi-hop network.

This also introduce the problem of the *mobility* that need for dyamic routing protocols to scale efficiently, provided the underlying infrastructure.

Moore's Law *"The number of transistor that can be embedded in a chip grows exponentially (it doubles every two year)"*. The application of Moore's law to IoT is peculiar and subject to different interpretation, like:

1. The *performance doubles every two years* at the same cost
2. The *chip's size halves* every two year at the same cost
3. The size and the processing power remain the same but the cost halves every two year.

In IoT all those interpretation are true, based on the specific scenario: there are application that require small sized sensors and or that have low power comsuption, can require higher processing capabilities. There is the need to balance the computational power to the costs, based on the specific application.
The Moore's law does not necessarily solve the problem in IoT design, at least in the near future.

4.1 Energy efficiency

It represent one of the critical aspects because devices are usually battery-powered, avoid wired powered devices and avoid maintenance cost. The energy efficiency is one of the main aspects in IoT design: the following diagram shows how the evolution of *processor, HD capacity and memory* grows faster than

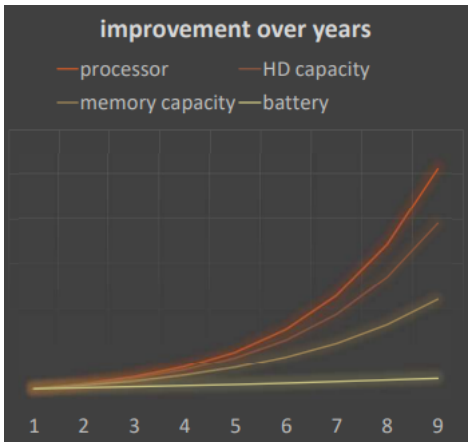


Figure 4.1: Processor and memory storage evolution over the years

battery efficiency as pictured in 4.1 The improvements of the battery efficiency is not bound to the Moore's Law: this is also due to the physic limit. The consequences are that if we want to improve the battery efficiency, we need to improve the design aspects of an IoT system, improving the design of the components that heavily rely on battery.

In a laptop, most of the energy in spent on a screen, as shown in 4.2 image.

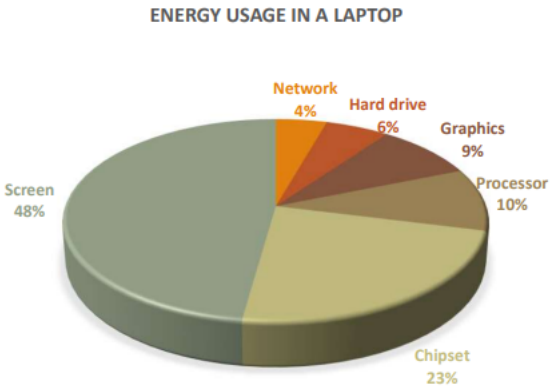


Figure 4.2: Energy expenditure for a commercial laptop

In a wireless sensor, the energy usage follows a different pattern than in laptop as pictured in 4.3: 20% of battery is used for sensing activities, the 40% is used for wireless and network interfaces activities and the remaining part, about 40%, is used by the processor and the chipset.

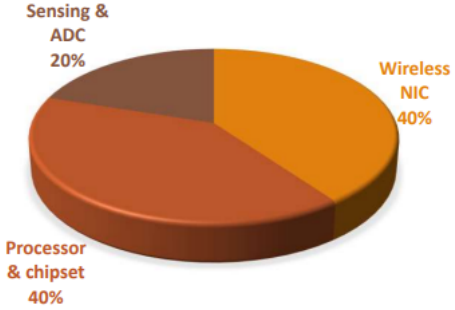


Figure 4.3: Energy expenditure for a commercial IoT device

The percentage areas can vary based on the specific sensors. There is a trade-off between sensing and transmission: *"a general rule state that transmit 1 byte is equal to compute 1k byte"*. Usually the sensing activation frequency it's mandatory based on a given context: reducing it have a direct impact on the

quality of sensing data. The opmitizable component is surely the **processor area** but, differently, the sensing area cannot be opmitized muchly because it both transmit and receive: the optimizable part is the one that concern the communication timeframe and behavior. A radio for **WiFi Network Interface** can be in 4 modes, each with different energy consumption:

1. **Sleep Mode**: consume very less and can get the radio operative in short time. The energy consumption is around $10mA$.
2. **Listen Mode**: $180mA$
3. **Receive Mode**: $200mA$
4. **Transmit Mode**: $280mA$

For a sensor (*specifically the moto-clone*) the energy consumption is:

1. Sleep Mode: $0.16mW$
2. Listen Mode: $12.35mW$
3. Receive Mode: $12.50mW$
4. Transmit Mode: varies from $12.36mW$ to $17.76mW$

The power level can change based on the distance that are needed to communicate, amplyfing the signal. In case of receive mode, it's also necessary to activate a **decoder** (*from a given analogical signal*) that require more energy. Those numbers show how it's not useful to put the **sleep mode** only, but remaining in a **listen mode**, letting the device comunicate. This had an impact on the MAC protocols for IoT which differs from the standard definition.
So for **sensors**, as a rule of thumb, the approach is:

- transmit power is *less* than receive power
- listen power is *more or less* equal to receive power
- radio should be turned off as much as possible

Switching on and off a component require energy: the decision to switch the status must not be reversed in a short timeframe to avoid energy wasting.
The IoT device we're considering are not general purposes devices so it's not needed to be active all the time providing the service: the key idea is to save energy by **reducing the period of activity of a sensor**. This is called **duty cycle** and define the *repetitivity of the activities of a sensors*:

1. *Sense*: it's more efficient if it's repeated periodically and it's **constant** in a given timeframe
2. *Process and store*: process sampled data to transmit or for local purposes, storing locally if requested by the device features itself.
3. *Transmit and receive*: A sensor alternates a period of activity and a period of inactivity (*defining a duty cycle*). During the inactivity the energy consumption is very low but the process, radio and I/O need to be freezed.

In general, the ***duty cycle*** of a system is defined as **the fraction of one period in which the system is active**. The radio activities can represent an exception to this, based on the specific activity that need to be carried out even during an inactivity period. The duty cycle is commonly expressed as a ratio or as a percentage. An example taken form Arduino is the pictured in 4.4: the `delay` instruction does not power off any component. The period of activity of this duty cycle is 400 ms, but the duty cycle itself does not guarantee to save energy consumption.

Instead, consider the following code in which we turn off the components:

```
1  void loop() {
2    // reads the input from analog pin 0:
3    turnOn(analogSensor);
4    int sensorValue = analogRead(A0);
5    turnOff(analogSensor);
6
7    // converts value into a voltage (0-5V):
8    float voltage = sensorValue * (5.0 / 1023.0);
9
```

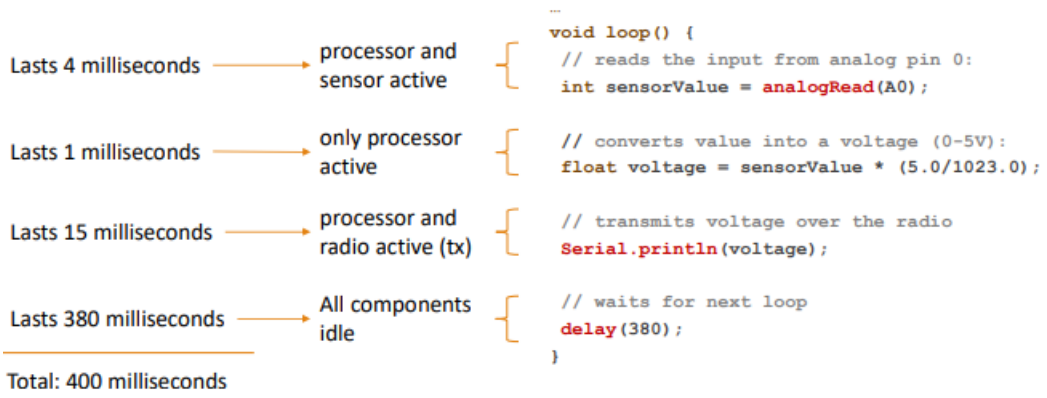


Figure 4.4: Duty cycle on Arduino device

```
10      // transmits voltage over the radio
11      turnOn(radioInterface);
12      Serial.println(voltage);
13      turnOff(radioInterface);
14
15      // waits for next loop
16      idle(380);
17  }
```

The `turnOn/turnOff` turn on/off a given components and the `idle` isntruction puts the processor in idle state with low power consumption for `y` ms. The `idle` instruction allows to do not power-off the processor but to obtain a duty cycle that by firing a call when the `y` ms are passed, the processor can return operative. The above code defines 3 different duty cycles, one for each component as is sketched in 4.5.

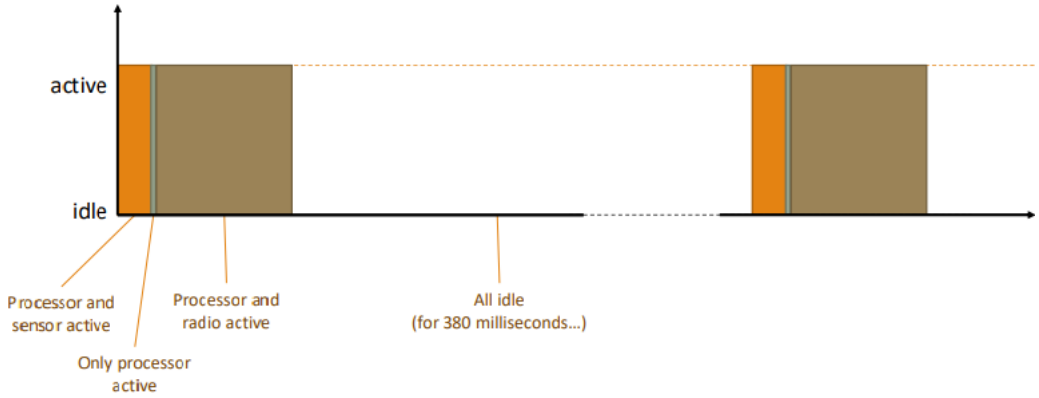


Figure 4.5: Three different duty cycle over active and idle period

The duty cycles allows to define or compute the current absorbed by each component based on the general state of each component: the table in 4.6 define the specific value for each state of the considered component. As an example, the processor have two state (*full operation, sleep*) that consume different amount of current (*respectively 8 mA and 15 η A*).

The battery gives a given amount of power but part of the energy is lost even without plugging it: it estimates to 3% per year on the total provided energy. This also have an impact on the charge efficiency of the battery that decrease overtime.

Example 2 Consider two different models: the first model have a 100% DC so it's always fully active while the second model have a 5% DC as listed in 4.8, or, in an alternative visualization as pictured in 4.7.

The sum of two different activity period form a duty cycle: in the figure 4.7 there is only indicated a single period.

	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	%

Figure 4.6: Ampere for each component in each different mode

	model 1: 100%DC	model 2: 5%DC	units
Micro Processor			
current (full operation)	100	5	%
current sleep	0	95	%
Radio			
current in receive	50	4	%
current xmit	50	1	%
current sleep	0	95	%
Logger			
write	1	1	%
read	2	2	%
sleep	97	97	%
Sensor Board			
current (full operation)	100	1	%
current sleep	0	99	%

Figure 4.7: Two different duty cycle with percentage of usage for each component

4.1.1 Measuring energy

Energy and power are mesured in Joule J and Watt W, respectively $1J = 1W * sec$. In electromagnetism, $1W$ is the work performed when a current of $1A$ (Ampere) flows through an electrical potential difference of $1V$ volt, so:

$$1W = 1V * 1A$$

Since we use direct current and the electrical potential difference is almost constant, the power and the energy only depend on the current (*Ampere*). Hence we can express both the energy stored in a battery and the energy consumed in *mAh* (*milli-Ampere per hour*).

Example on computing consumption per duty cycle

Consider the two different models, introduced in 4.8, in which, for each component, the Duty Cycle *(DC)* follows the percentage presented in 4.9: Note that the first model have a Duty Cycle of 100% so it's an always up system while the second model present a Duty Cycle of 5%.
To compute the energy total cost we must first compute the energy cost of each component, considering its duty cycle period.
The **energy cost of a microprocessor** E_η ***per cycle*** can be expressed as:

$$E_\eta = C_\eta^{full} * dc_\eta + C_\eta^{idle} * (1 - dc_\eta) \tag{4.1}$$

where:

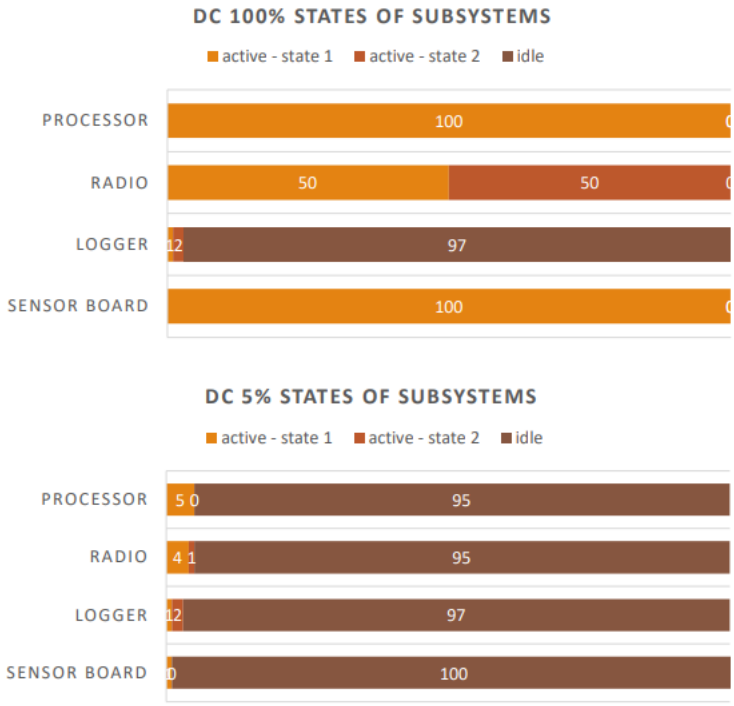


Figure 4.8: Two different duty cycle usage percentage

- C_{η}^{full} full energy cost microprocessor
- C_{η}^{idle} idle energy cost microprocessor
- dc_{η} % duty cycle microprocessor

The C_{η}^{idle} cannot be dropped because it's multiplied with a factor that is not neglectible $(1 - dc_{\eta})$. The **energy cost radio** E_{ρ} ***per cycle*** can be expressed as:

$$E_{\rho} = C_{\rho}^{T} * dc_{\rho}^{T} + C_{\rho}^{R} * dc_{\rho}^{R} + C_{\rho}^{idle} * (1 - dc_{\rho}^{T} - dc_{\rho}^{R}) \tag{4.2}$$

where:

- C_{ρ}^{T} radio transmission energy cost
- C_{ρ}^{R}radio receival energy cost
- C_{ρ}^{idle} idle energy cost
- dc_{ρ}^{T} percentage transmit duty cycle radio
- dc_{ρ}^{R} percentage receive duty cycle radio

The **energy cost logger** E_{λ} **and sensor board** E_{σ} can be calculated as before, obtaining the **total energy cost per duty cycle** of:

$$E = E_{\eta} + E_{\rho} + E_{\lambda} + E_{\sigma} \tag{4.3}$$

.The **lifetime**, expressed in number of duty cycles, is:

$$Lifetime = \frac{B_0 - L}{E} \tag{4.4}$$

where B_0 is the initial battery charge and L is the battery charge lost during the lifetime due to **battery leaks** and E is the total cost of energy. Because L depends on lifetime and it's based on charge/loss cycle ϵ (*estimated to 3% yearly*), we can obtain a recurrence equation:

$$B_n = B_{n-1} * (1 - \epsilon) - E \tag{4.5}$$

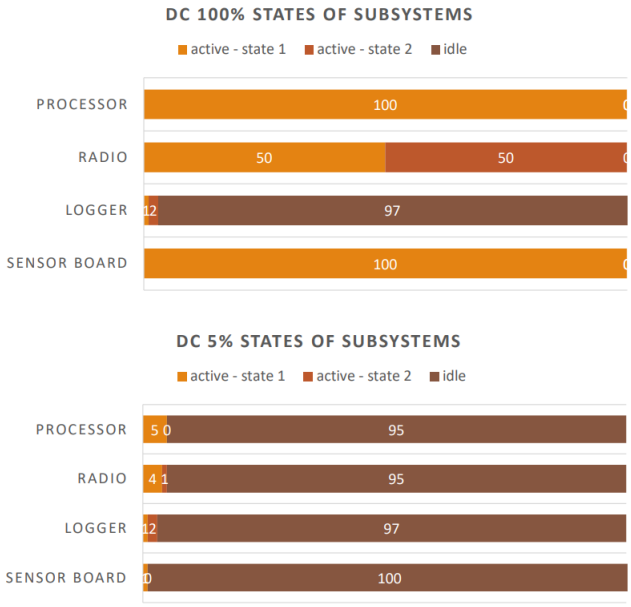


Figure 4.9: Example 3

where B_n is the battery charge at cycle n. By solving the recurrence equation we obtain that:

$$B_n = B_0 * (1-\epsilon)^{n-1} + \frac{E((1-\epsilon)^n) - 1}{\epsilon} \tag{4.6}$$

The device lifetime is given by n, when $B_n = 0$.
Regarding the two models already cited, we can plot both the battery life (*expressed in months on x axis*) and the battery capacity (*expressed in milli-Ampere per hour on y axis*) as pictured in 4.10.

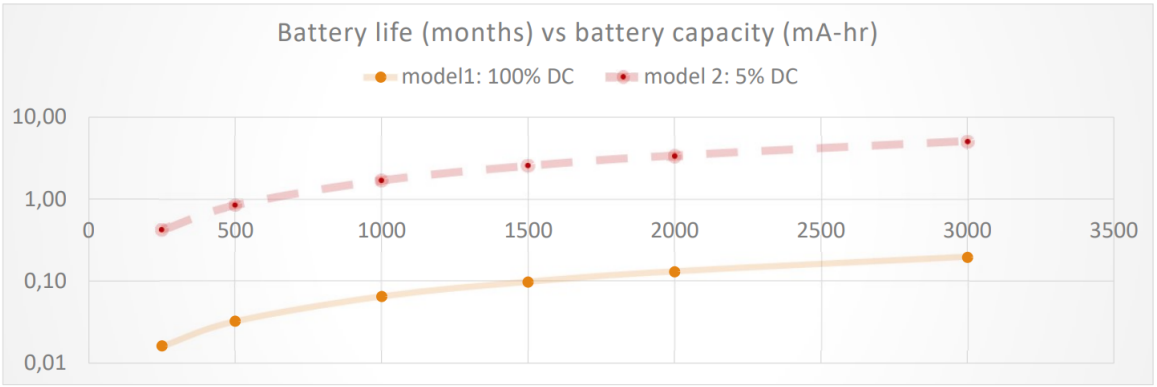


Figure 4.10: Comparison between the two models

The previous graph (4.10) show how the second model that have 5% of DC have a longer battery life than the first-one, thanks to longer duration of idle state of its components. The graph, which in expressed in logarithmic scale, show how enlarging the battery capacity the correspondent battery life for the first model is limited to a couple of days while for the second model the battery life can last up to some months.

The second graph (4.11) show that given two devices, increase the DC corresponds to a decrease of the lifetime battery. To obtain an efficient device, it's necessary to work on the DC to minimize it and obtain a longer battery life.

To gain *energy efficiency* the solution is to reduce the *Duty Cycles (DC)*, however ***turning off the processor is a local decision*** so the node scheduler knows that are the activires and when the processor should run. Also, turning off the radio is a **global decision** that implies no communication, no reception of incoming messages/commands and the impossibility to act as a router in multihop network.

At **MAC Layer**, MAC Protocols arbitrate the access to the shared communcation channel: in IoT also implements strategies for energy efficiency like **synchronize the devices** and **turn off the radio when it's not needed** (*so excluding the device from the network*).

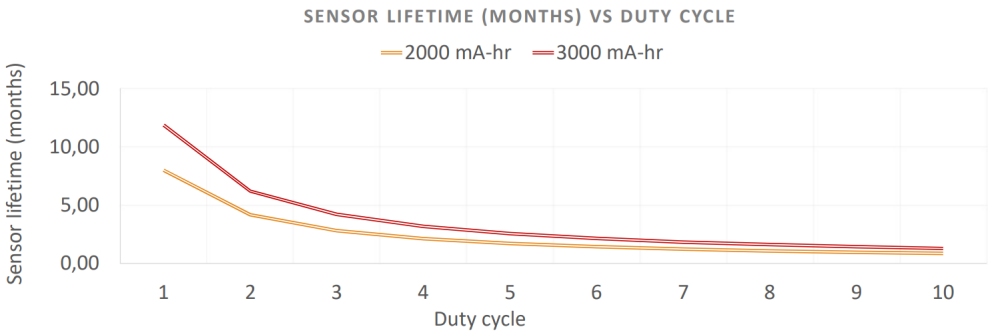


Figure 4.11: Lifetime correlation with DC for the two models

4.1.2 Exercise

Consider the following program and the table 4.12 of energy consumption in the different states. Compute:

- the energy consumption of the device per single hour
- the expected lifetime of the device

```
void loop(){

//4 milliseconds
  turnOn(analogSensor);
  int sensorValue = analogRead(A0);
  turnOff(analogSensor);

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

//15 milliseconds
  turnOn(radioInterface);
  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	1	mA
current sleep	20	µA
Sensor Board		
current (full operation)	5	mA
current sleep	5	µA
Battery Specifcations		
Capacity	2000	mAh

Figure 4.12: Exercise 1 parameters

Answer

Consider the energy consumption per hour:

	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 Specifications		
Capacity	2000	mAh

Figure 4.13: Exercise 2 - Parameters specification

1. The processor has a duty cycle of:
2. The radio has a duty cycle of:
3. The sensor has a duty cycle of:
4. The energy consumption in one hour is:
5. The lifetime of the device is:

Exercise 2

Consider the sensor specification in the table pictured in 4.13
The device measures the hearth-rate (HR) of a person:

- Samples a photo-diode on the wrist at 20 Hz
  - sampling the sensor takes 0.5 ms
  - it requires both the processor and the sensor active
- HR is computed every 2 s (based on 40 samples)
- Transmit (from time to time... see below) a data packet to the server:
  - The average time required to transit is 2 ms
  - Requires both processor and radio active Compute the energy consumption and the lifetime of the device if it sends all the samples to a server:
- Stores 5 consecutive samples from the photodiode
- Transmits the stored 5 samples to the server
- The server computes HR (hence the device does not compute HR) Disregard battery leaks.

Solution 2

Duty cycle of sampling: DC of processor + sensors: 0.5 milliseconds (sampling time) / 0.05 seconds = (sampling period)= 0.01

Duty cycle of transmitting: DC of radio + processor: 2 milliseconds (transmit time) / 0,25 seconds (transmission period) = 0.008

- Sensor power consumption: $E_\eta = 5mAh * 0,01 + 5uAh * 0,99 = 0,05 + 0,005mAh = 0,055mAh$
- Processor power consumption: $E_\rho = 0,018 * 8mAh + 0,982 * 15uAh = 0,144 + 0,0147mAh = 0,1587mAh$

	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 Specifications		
Capacity	2000	mAh

Figure 4.14: Exercise 3 - Parameters specification

- Radio power compsution: $E_\lambda = 0,008 * 20mA + 0,99992 * 20\eta A = 0.16mA + 0.0198mA = 0.1799mAh$

$$E = E_\eta + E_\rho + E_\lambda$$
Battery specification $(in\ table)$ $B_0 = 2000mAh$
Power consumption: $2000mAh/0,3935mAh$ $(\frac{B_0}{E})$
obtaining **Lifetime**: $5082h$

Exercise 3

Consider the sensor specification in the table pictured in 4.14.
The device measures the hearth-rate (HR) of a person:

- Samples a photodiode on the wrist at 20 Hz
  - sampling the sensor takes 0.5 ms
  - it requires both the processor and the sensor active
- HR is computed every 2 s (based on 40 samples)
  - Computing HR in the device takes 5 ms
- Transmit a data packet to the server:
  - The average time required to transmit is 2 ms
  - Requires both processor and radio active Compute the energy consumption and the lifetime of the device if it computes HR itself:
- Transmits every 5 values of HR computed (1 packet every 10 seconds)

Disregard battery leaks.

Solution 3

- Duty cycle of sampling: $0,5ms$ (sampling time) / $0,05s$ (sampling period)= $0,01$
- Duty cycle of processing: 5 milliseconds / 2 seconds = 0,0025
- Duty cycle of transmitting: 2 milliseconds (transmit time) / 10 seconds (transmission period) = 0,0002
- Power consumption of sensor (in 1h):
  - $E_\eta = 5mAh * 0,01 + 5uAh * 0,99 = 0,05 + 0,005mAh = 0,055mAh$
- Power consumption of processor (1h):

	value	units
Micro Processor (Atmega128L)		
current (full operation)	8	mA
current sleep	0,015	mA
Radio		
current xmit	1	mA
current sleep	0,02	mA
Sensor Board		
current (full operation)	5	mA
current sleep	0,005	mA
Battery Specifications		
Capacity	2000	mAh

Figure 4.15: Exercise 4 - Parameters specification

- $E_\rho = 8mAh * 0,0127 + 15uAh * 0,9873 = 0,1016 + 0,0148mAh = 0,1164mAh$

- Power consumtpion of radio (1h):
  - $E_\theta = 0,0002 * 20mAh + 0,9998 * 20uAh = 0,004 + 0,02mAh = 0,024mAh$

$$E = E_\eta + E_\rho + E_\theta = 0.1954mAh$$
Battery specification (*in table*) $B_0 = 2000mAh$
Power consumption: $2000mAh/0,1954mAh$ $(\frac{B_0}{E})$
Lifetime: $2000mAh/0,1954mAh = 10.235h$

4.1.3 Exercise 4

Consider a Mote-class sensor with the parameters pictured in table 4.15.
Assume that the device performs a sensing task with the following parameters:

- The sensor board is activated with a rate of 0,1 Hz to perform the sampling; this operation takes 0.5 milliseconds. At the end the sensor board is put in sleep mode. During each sensing operation the processor is always active.
- After each sampling the processor performs a computation that takes 2 milliseconds.
- Then the processor activates the radio and transmits the data. The transmission takes 1 millisecond and, during it, the processor is active. At the end the radio and the processor are both set in sleep mode. Compute the duty cycle of each component (*sensor board, radio and processor*), and the lifetime of the device (*assuming that the sensor stops working when its battery charge becomes 0*).

Solution 4

- Sampling takes $0.5ms$ with a rate of $0,1Hz$
- Processing: $2ms$
- Transmitting: 1 ms
- **Duty cycle of sampling (processor and sensors)**: $0.5ms/10s = 0,00005$
- **Duty cycle of processing (only processor)**: $2ms/10s$ $(10000ms) = 0.0002$
- **Duty cycle of transmissions (radio&processor)**: $1ms/10s = 0.0001$
- Power consumption of sensor (in 1h):
  - $E_\eta = 5mA * 0,00005 + 0,005mA * (1 - 0,00005) = 2.5 * 10^{-4}mA + 0.05mA = 0.00525mA$
- Power consumption of processor (1h):
  - $E_\rho = 8mA * 0.00035 + 0,015mA * (1 - 0.00035) = 0.0028mA + 0,015mA = 0,0178mAh$

- Power consumtpion of radio (1h):
  - $E_\theta = 0,0001 * 1mA + (1 - 0,0001) * 0,02mA = 100nA + 20\eta A = 0.0201mA$

- $E = E_\eta + E_\rho + E_\theta = 0.04315mAh$

Battery specification (*in table*) $B_0 = 2000mAh$
Power consumption: $2000mAh/0.04315mAh$ ($\frac{B_0}{E}$)
Lifetime: $2000mAh/0,04315mAh = 46.349h$

Chapter 5

MAC Protocols

In order to efficient the power consumption, is necessary to *reduce and optimize* the duty cycle: this does not apply to **radio interface** directly so careful observation must be taken to address the activity of the radio interface at MAC layer. By operating at MAC layer we can reduce the radio duty cycle while mantaining network connectivity. The optimal solution involves a trade-off between *energy, latency* and *bandwidth*. In this scenario, MAC protocols change their objectives, taking the role of implementing DC by developing:

1. **Syncrhonization of nodes** (*as in S-MAC or IEEE 802.15*): implement a distributed protocol so the nodes agree on the activity period of the network. It's based on explicit syncrhonization.
2. **Preamble sampling** (*as B-MAC*): another way to obtain synchronization despite it's not a fully standardized mechanism.
3. **Polling** (*IEEE 802.15.4*): there is an asymmetry between the roles of the nodes in synchronizing process, with an unbalanced workload between the sender node and the receiver node.

Those approaches are implemented as **distributed coordination protocol**: the syncrhonization of the nodes is performed by agreeing **on when can turn on/off the radio**.
To build large multi-hop network those protocols try to minimize the impact of *latency* on large communcations: the latency adopted by the synchronization protocols can be problematic for hundreds of devices (*e.g. multi-hop ZigBee network with high number of devices*) because each hop introduce additional delay due to the processing and transmission time of each node. This behavior can determine packet loss because the communication is is performed during an *inactive period* of the neighbors's node. The **syncrhonization of nodes** allows the node to turn on the radio simultaneosly by coordinating between all nodes and when the radios are active the network is connected while when the radio is inactive there is no network activity available. The goal is to have a low duty cycle for the radio, minimizing the overall energy consumption by remaining inactive for most of the time.

5.1 Syncrhonization: S-MAC

S-MAC is a *medium access control protocol* for wireless multi-hop network. The key idea is that it uses only **local syncronization**: the nodes implement synchronization locally by exchaging synchronization packets only with around neighbors. The farest node can be totally un-synchronized respect to a given node.
The syncronization mechanism of exchanging `SYNC` messages allows to synchronize the clock to a common timeframe of radio activation: to communicate they just to wait the activity period of the communication timeframe. A `SYNC` frame contains the ***schedule*** of the node thus if a node detects adjacent nodes with a pre-defined listen period will use the same period, otherwise it chooses its own period. The choosen period is advertised to the neighbors by `SYNC` frames. So nodes alternate *listen* and *sleep* periods in which, during sleep time, the sensor cannot detect incoming messages. Two different nodes can decide different schedule of activity for the radio but the *"most popular schedule"* among the neighbors it's the most suitable to communicate with the highest range of devices. A node may revert to someone else schedule if its own schedule it's not shared by any other device.

A node **receives** frames from the neighbors during its own **listen period**: node `A` can send a frame to node `B` only during the listen period of `B` (*as pictured in 5.1*):

- node `A` may need to turn on its radio also outside its listen period
- node `A` should know all the listen periods of its neighbors
- At startup a node preliminary listens the `SYNC` frame of its neighbors

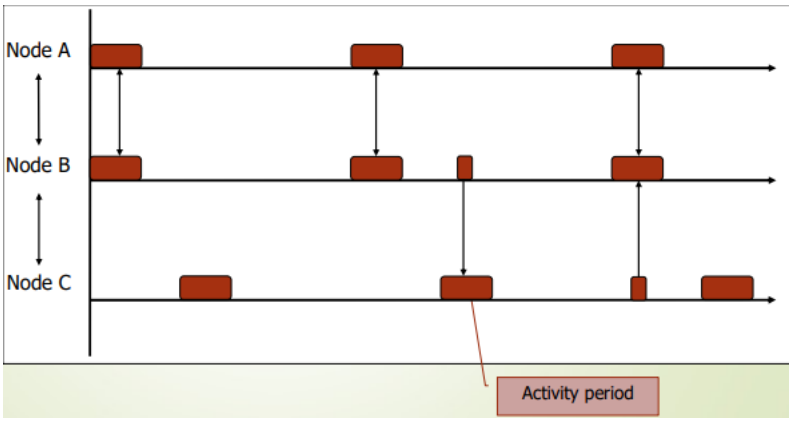


Figure 5.1: Activity perioid utilisation

In the previous exchange 5.1, `A` and `B` have the same listen period so they're able to communicate without problem: the communication between `B` and `C` require that the node `B` activate its radio in the listen period of `C`, despite it's outside of its own active period.

So the frame are sent during the listen period of the receiver: the problem of **collision** arises if two nodes have the same listen period, they will send packets on the same time. To avoid collisions, the adaptive duty cycle use a **Carrier Sense** check it's executed before the transmissison and if the channel is busy and a node fails to get the medium, the frame is delayed to the next period. So if an *RTS* ***(Request To Send)*** or *CTS* ***(Clear To Send)*** is received (*more details in section 10.0.7*), the radio is keeped up until the end of the transmission just incase it's the next hop of commnication. This simple optimization allows avoiding collisions on the communciation channel.

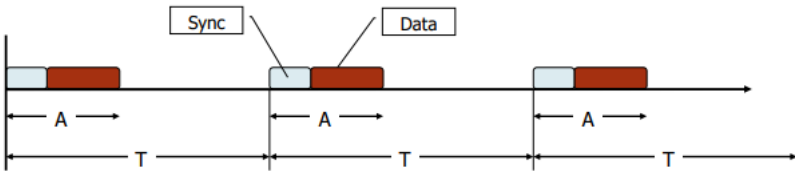


Figure 5.2: Differentiation between sync time and data transmission time in each period

The ***S-MAC Energy Consumption*** follows the diagram pictured in 5.3. The diagram shows the *aggregate energy consumption over a 10-hops path*: as the data shows, the adaptive duty cycle is a little optimization used by S-MAC so if a node overhears a RTS or a CTS it keeps its radio on until the end of the transmission just in case it's the next hop of communication. As the diagram shows, the **larger is inter-arrival period, the smaller is the load of the network**.

Latency

While traveling across a multi-hop path, a frame may have to wait (*in the worst case*) for the listen period of each intermediate node. This is mitigated by the fact that (*hopefully*) a number of nodes will converge towards the same schedule (*not guaranteed anyway*).

Latency must be well addressed, as shown by the exchange pictured in 5.4: from `A` to `C` the communication can happen within one activty period but between `C` and `D` there is a *waiting time*, the same happen with `E` and `F`: this *waiting periods* increase the overall latency. A trade-off must be achieved because a larger duty cycle ca surely lower the latency but increasing the energy consumption.

Another problems that affects the Iot devices is the **quality of clock component** that usually it's not suitable for exact sycnrhonzation: they are cheap and are affected by environmental conditions. There are conditions that make clocks of different devices diverge (*e.g. a device with a shield vs no shield respect to temperature impact*), impacting the syncrhonziation between the devices.

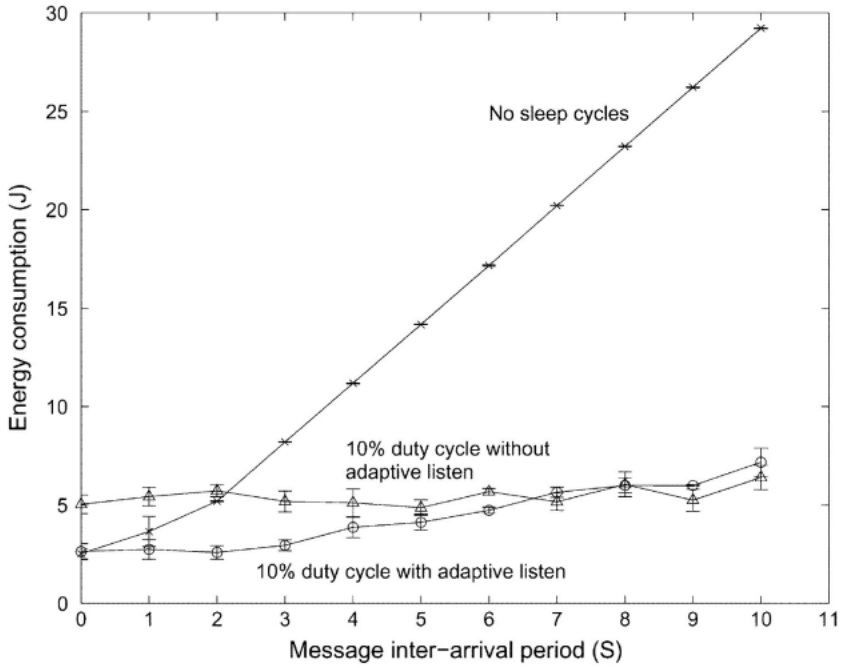


Figure 5.3: S-MAC Energy consumption

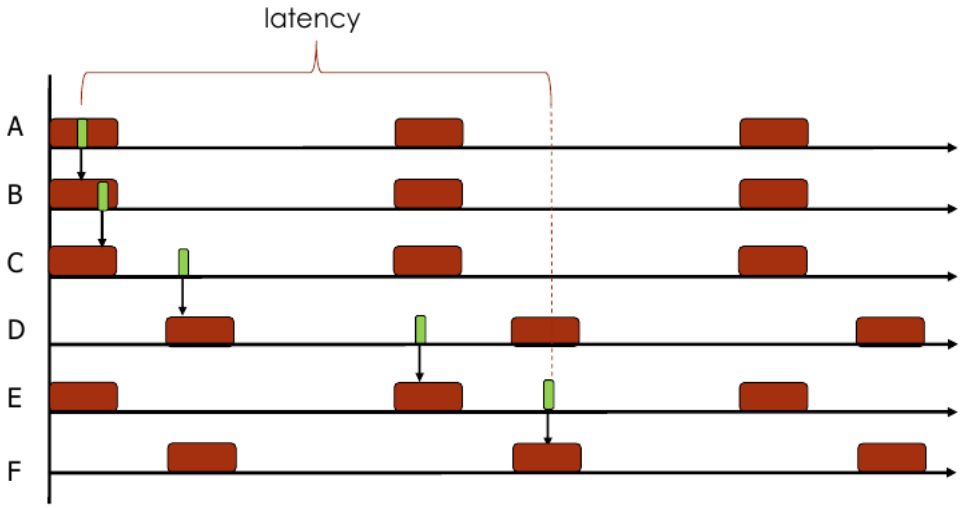


Figure 5.4: Cumulative latency in multihop transmission over 10-hops path

5.2 Preamble Sampling: B-MAC

The key idea behind *B-MAC* is to reduce the complexity of the protocol, obtaining syncrhonization by differenciating the behavior of the receiver and the transmitter. A **sender** sends whener it want that contains a very long preamble in it's header while the **receiver** activates its radio periodically to check if there is a preambe on the air. This last activity is called **preamble sampling** and it's based on ***Low-Power Listening (LPL) mode***.
The general schema is shown in 5.5: if the receiver during the *preamble sampling* detects a preamble **keeps the radio on to receive the frame**, otherwise turns off the radio. The idea is to spend more in transmission but save energy in reception because the preamble sampling should be very short and cheap. The costs of radio activation/deactivation at the receiver side are balanced by a *low rate sampling* Surely to be effective, this implies that the **preamble should be longer than the sleep period**.

The green peaks in 5.5 indicates the **preamble sampling** for small fraction of time while the interval between two *peaks (sleep period)* it's a fixed parameter given by the B-MAC protocol. The *sampling duration*, differently, cannot be enforced by the protocol but depends on the specific device/hardware

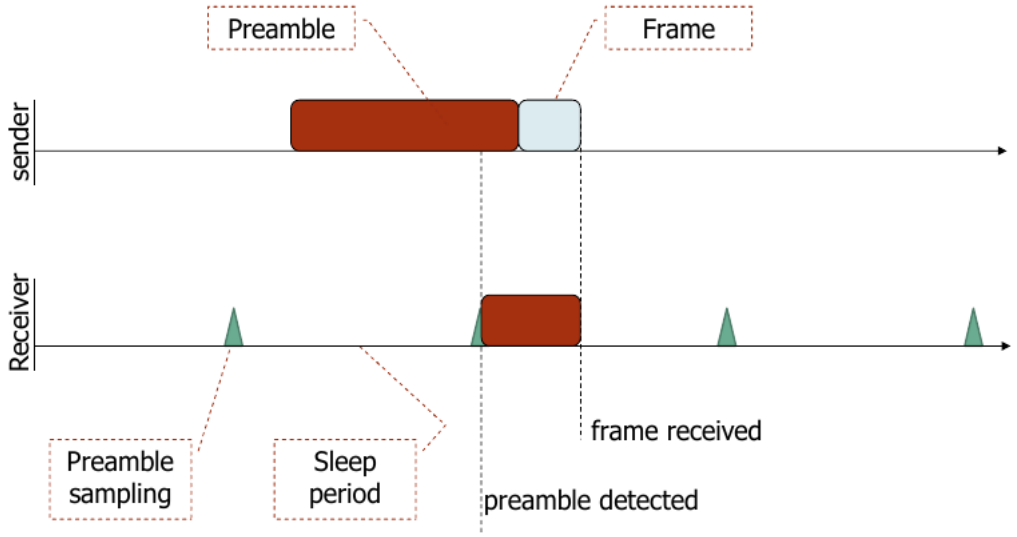


Figure 5.5: B-MAC syncrhonization mechanism

performances. To avoid partially communication by receiver side, the **long preamble** it's used: it does not contains anything useful but the only purpose it's to signal that meaningful packets will be sent at the end of the preamble packets. The protocol work because the lenght of the preamble is longer than the interval between two periods.

The idea is to give *more work to the transmitter, lesser to the receiver*: in multi-hop devices network this allows to save the overall energy in a scenario with multiple receivers.

As mentioned, the preamble sampling it's based on **Low-Power Listening (LPL)** that also have its own overhead: the diagram in 5.6 shows the Current consumption over time for the *Mica-Mote* radio during its initialization/startup/trasnmission/sleep phases.

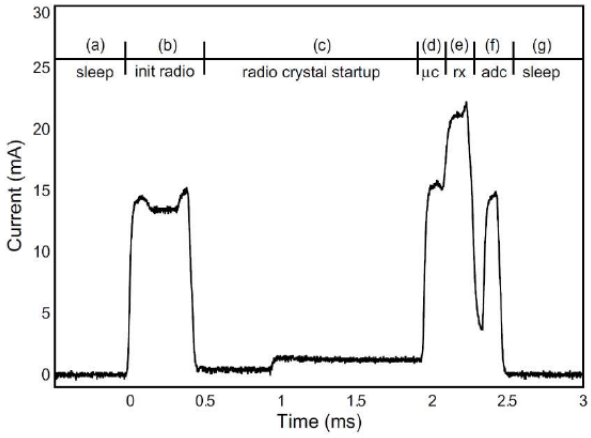


Figure 5.6: Radio current usage over different modes

The radio keeps an internal clock to be synchronized to the bit flowing on the radio channel: this require an initial setup by the clock (`init radio`). The diagram also indicates that the *init* phase must be avoided over time to reduce the current consumption. Another result is that the transmission represent the activity with the most current expenditure, alongside the ADC (Analog-Digital Converter) activity period.

This simple process allows to define a **model to compute the** $lifetime$, both for transmitter and receiver sketched in 5.7. Based on this general schema, to **model the lifetime** we need make some assumption like that there is only *one receiver and one transmitter*. Their parameter are:

1. f_{data} : frequency of **transmitted data** (*in hertz* $\frac{1}{sec}$)

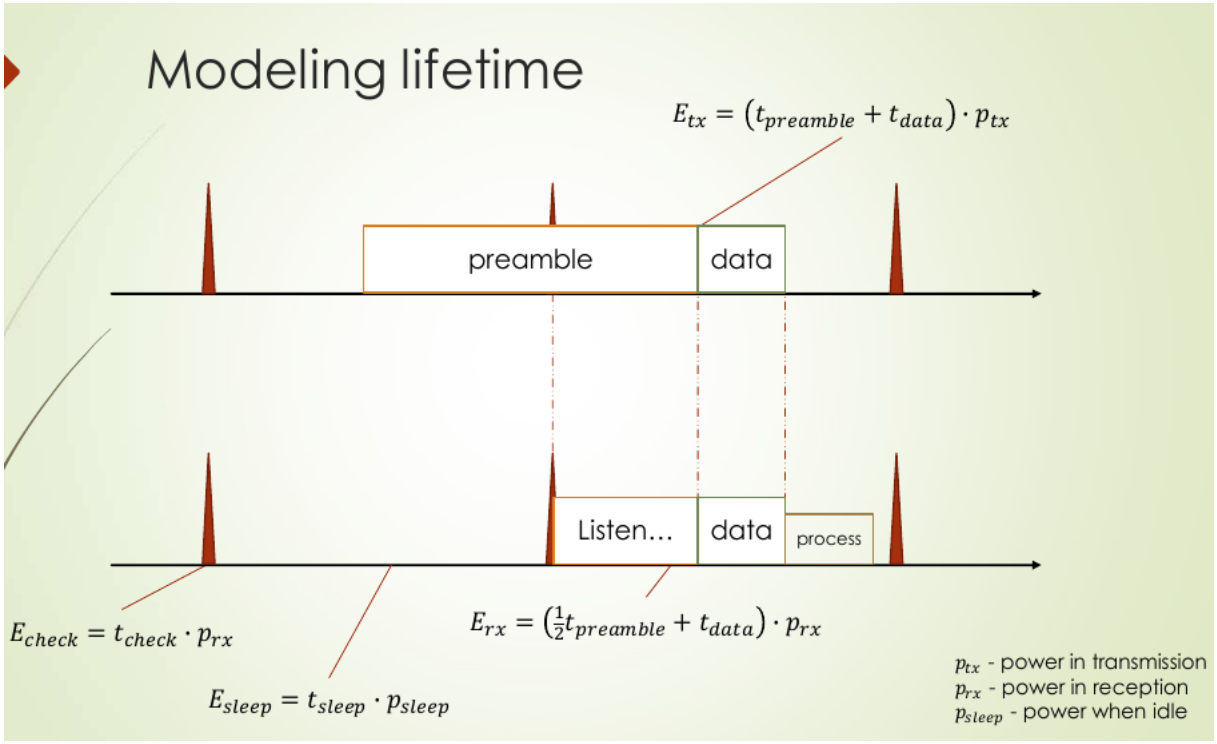


Figure 5.7: Transmitter and receiver lifetime model

2. f_{check}: frequency of **preamble samplig**

Hence, at the **transmitter** (*considering only the radio activity*), we have:

- **Duty cycle data**: $DC_{tx} = f_{data} * (t_{preamble} + t_{data})$
- **Duty cycle check**: $DC_{check} = f_{check} * t_{check}$
- **Energy (*in joule*) spent in t seconds at the transmitter side**:

$$ET(t) = t * (p_{tx} * DC_{tx} + p_{rx} * DC_{check} + p_{sleep} * (1 - DC_{tx} - DC_{check}) \tag{5.1}$$

While, at the **receiver side** we have:

- **Duty cycle data**: $DC_{rec} = f_{data} * (\frac{1}{2} t_{preamble} + t_{data})$
- **Duty cycle check**: $DC_{check} = f_{check} * t_{check}$
- **Energy (*in joule*) spent in t seconds at the receiver side**:

$$ER(t) = t * (p_{rx} * DC_{rec} + p_{rx} * DC_{check} + p_{sleep} * (1 - DC_{rec} - DC_{check}) \tag{5.2}$$

Thus, the overall **lifetime** are:

- **Transmitter**: $lifetime = \frac{battery_{charge}}{ET(1)}$
- **Receiver**: $lifetime = \frac{battery_{charge}}{ER(1)}$

In the table 5.8 are sketched some value of the specified parameter from the original BMAC paper for the *CC1000 radio*.

Lifetimes

The following results assume a battery of $3000mAh$ and disregard the processing, sensing and battery leaks in the computation. The data in 5.9 shows the lifetime at *transmitter side*, showing that by decreasing the sample frequency interval (t_{check}) (*from 1 sample every min to 1 sample every 20 min*) we can increase te overall lifetime. Analogues results are obtained for the receiver 5.10.

Parameter	value
p_{tx}	60 mW
p_{rx}	45 mW
p_{sleep}	0,09 mW
preamble	271 bytes
data frame	36 bytes
byte length	4,16 E-04 s
check length	3,5 E-04s
preamble length	1,13 E-01 s
frame length	1,5 E-02 s

Figure 5.8: BMAC Parameters

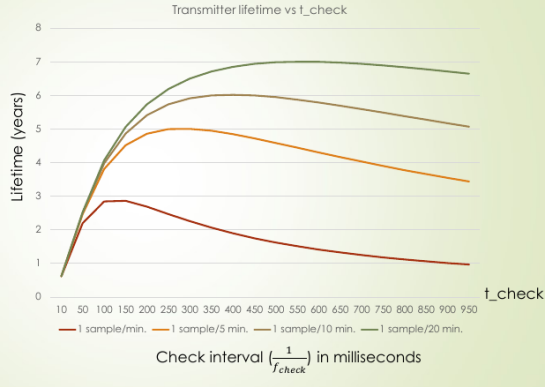


Figure 5.9: Transmitter lifetime vs *t_check*

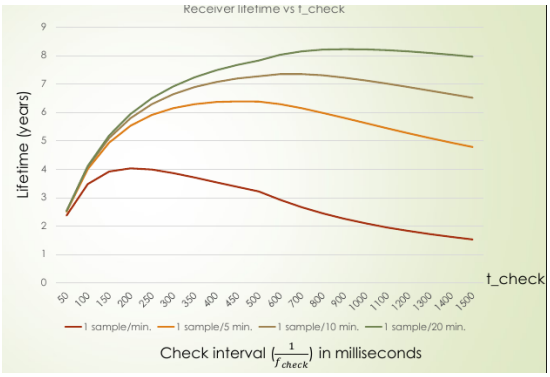


Figure 5.10: Receiver lifetime vs *t_check*

The previous model consider only two nodes (*one receiver and one transmitter*): even considering several transmitters, the lifetime trend does not change and still result in the same shown in 5.9. The node *lifetime* depends on the **check interval** (*of preamble sampling*) and the total amount of traffic in the network cell. Each line in the picture 5.11 diagrams shows the lifetime of a node at that sampe rate and LPL (*Low-Power-Listening*) check interval: the circles occur at the maximum lifetime (*indicating optimal check intervals*) for each sample rate.

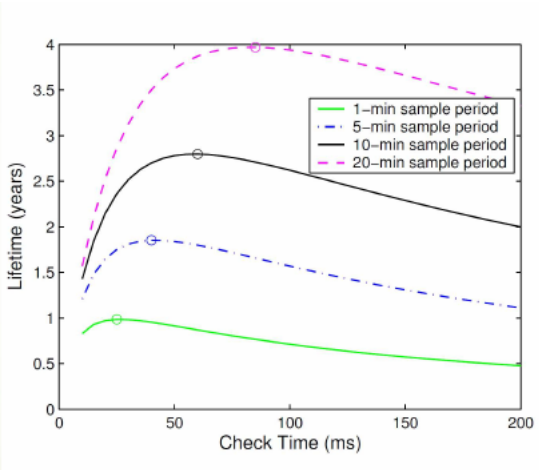


Figure 5.11: Lifetime with several transmitter nodes

Stenghts and drawbacks

As mentioned previous, the clock must be not necessarily syncrhonzied but they need to measure the time in the same way, guaranteeing that 1 second must corresponds to 1s in every device, despite, as previously mentioned, the decreasing quality of measuring of the clocks used in IoT device. The **strengths** of the B-MAC Protocol are:

1. It's not a network organization protocol
2. It's simple to use and configure (*having only one parameter*)
3. It's transparent at the higher layers

While the **drawbacks** are:

1. Long check intervals have an impact on the througput of the network but in IoT this is acceptable. B-MAC is suitable for low transmission rate.
2. Long check interval imply long, expensive preambles
3. In some cases it may result more expensive than using other form of synchronization

5.2.1 X-MAC

X-MAC is an evolution of B-MAC aimed at **reducing the impact of long preambles**. It allows a *receiver* to stop the preamble transmission: the preamble this time contain also information of the **ID of target node** so a receiver can check if it's the recipient of the frame and interrupts the preamble transmission to request the data frame. In figure 5.12 comparison schema between *LPL (B-MAC)* and *X-MAC*.

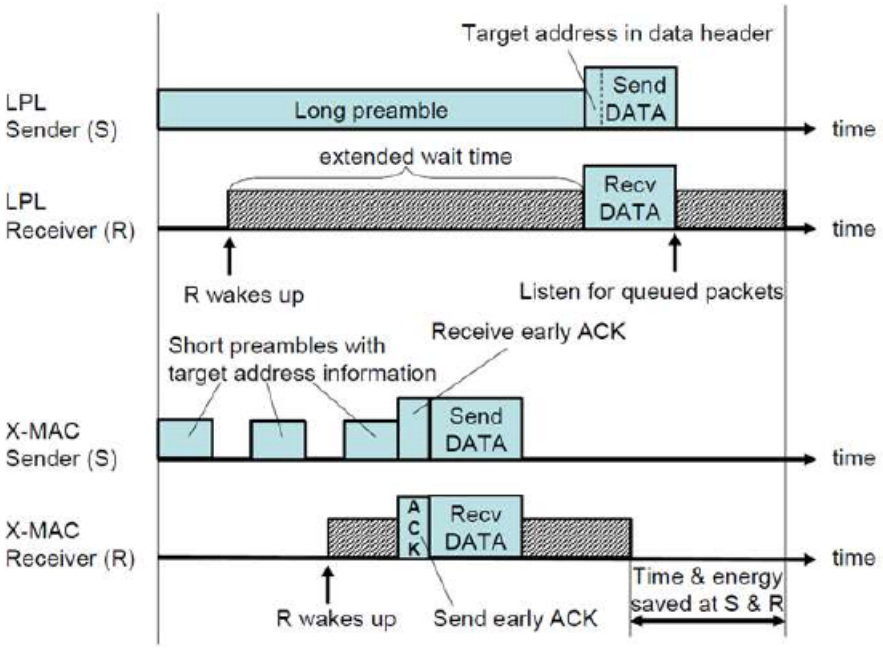


Figure 5.12: B-MAC (*LPL*) vs X-MAC periods

During the *transmission phase*, the transmitter (S) send ***short preambles*** containing the target address information: when the receiver (R) wakes up, catch the short preamble and reply with an **early ACK** that allows S to stop the sending of short preamble and start sending the **data frame**.
After receive a frame, the receiver R does not turn off the radio immediately to allows other nodes that wanted to transmitduring the preamble transmission but were not allowed (*as they sensed the channel was occupied*), so the receiver remain active to catch eventually other communciations: this is indicatd by the gray area after R's reception phase (*low part diagram X-MAC*).

X-MAC vs B-MAC The comparison pictured in 5.13 is made by simulation on **total duty cycle of transmitters**: the scenario was a *star connected network* with 1 receiver and several transmitter that varies over time. Each station transmits every 9 seconds with randomized jitter, using also different LPL intervals (*different check intervals of B-MAC*).

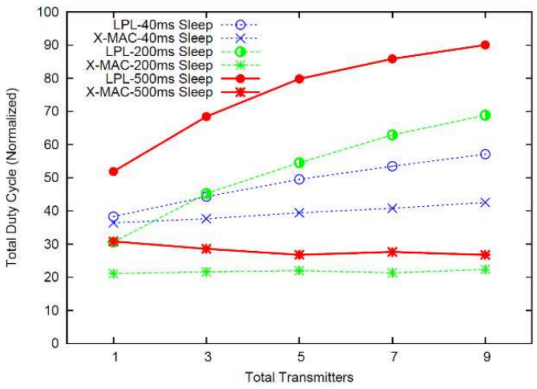


Figure 5.13: X-MAC vs B-MAC duty cycles

BoX-MAC Another following development of X-MAC is known as ***BoX-MAC***: it still try to reduce the impact of long preamble by attaching inside the preamble message also the data itself, so the receiver send an ACK after having received the packets containing both the preamble and the data frame. This is suitable for **small data packets**. The general schema of BoX-MAC is pictured in 5.14

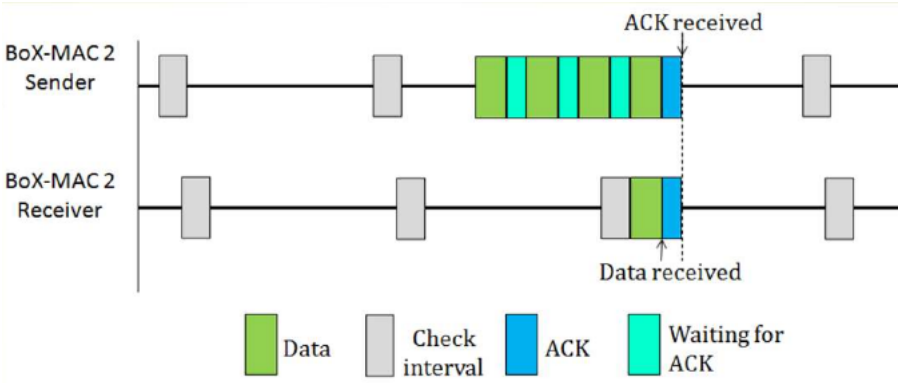


Figure 5.14: BoX-MAC single hop transmission

For a **multi-hop routing** in BoX-MAC, we refer the schema in 5.15 : the transmission involves three different nodes. The first node $i1$ try the transmission multiple time before having success: the node $i2$ first need to sample the channel and then receive the communication so it's allowed to send a $ACK\text{-}TX$ to $i1$ and the same process is repeated between $i2$ and $i3$.

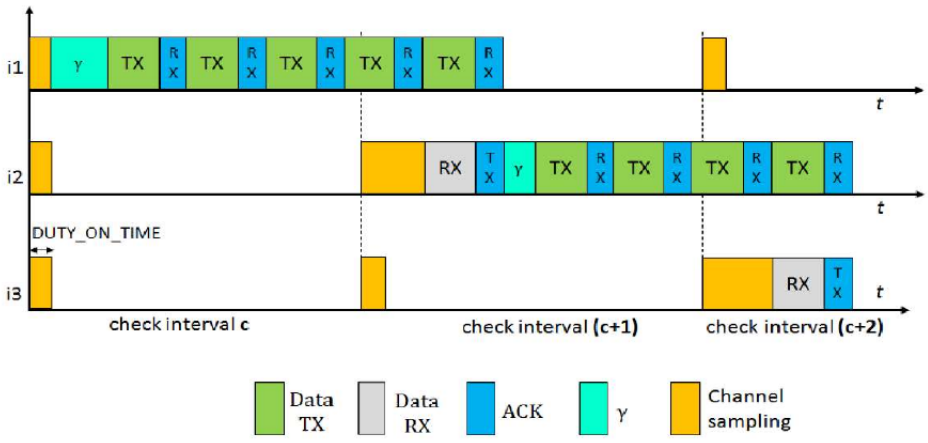


Figure 5.15: BoX-MAC multihop transmission

Polling This methods is widely used in IEEE 802.15.4: can be combined with other syncrhonization methods and usually allows an **asymmetric organization** of the nodes by having *one master* that issues periodic beacons and *several slave nodes* that can keep the radio off whenever they want. (*See section 6 on IEEE 802.15.4 for more details*).

Chapter 6

IEEE 802.15.4

The **IEEE 802.15.4 standard** specifies the **physical** and **MAC layers** for low-rate wireless *Personal Area Networks (PAN)*. Its protocol stack is simple and flexible, it does not require any infrastructure and it is suitable for short-range communications (*typically within a range of 10 meters*). For these reasons it features ease of installation, low cost, and a reasonable battery life of the devices.
The *physical layer* of the IEEE 802.15.4 has been designed to coexist with other IEEE standards for wireless networks such as *IEEE 802.11 and IEEE 802.15.1 (Bluetooth)*. It features activation and deactivation of the radio transceiver and transmission of packets on the physical medium. It operates in one of the following three license-free bands:

- 868–868.6MHz (*e.g., Europe*) with a data rate of 20 kbps;
- 902–928MHz (*e.g., North America*) with a data rate of 40 kbps; or
- 2400–2483.5MHz (*worldwide*) with a data rate of 250 kbps.

The MAC layer provides ***data and management services*** to the upper layers. The **data service** enables transmission and reception of MAC packets (*Protocol Data Unit (PPDU)*) across the physical layer. The **management services** include synchronization of the communications, management of guaranteed time slots, and association and disassociation of devices to the network. This is performed by a set of features like **Energy Detection (ED), Link Quality Indicator (LQI)** (*that allows to chose the best channel for a given communication*), **Channel Selection**, **Clear Channel Assessment (CCA)** (*more information later*). In addition the MAC layer implements basic security mechanisms.

6.0.1 Physical layer

There are several channels operating at different frequencies (Mhz) as pictured in 6.1: the lower channels are used for short range devices (*baby monitor, automatic gate remote controller, etc*) while the upper channels (*11-26*) are for large range of communications.

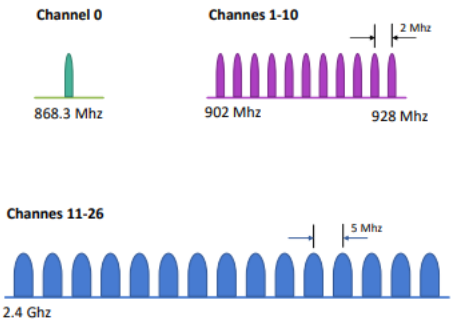


Figure 6.1: Physical channels with relative Mhz

It's designed to operate well even in highly noisy environment: the physical channel is designed to be more robust respect to Bluetooth as shown in 6.2 because the higher is the SNR, the easier is to extract signal from noise. The *BER - Bit Error Rate* indicates the probability that a transmitted bit is receveid

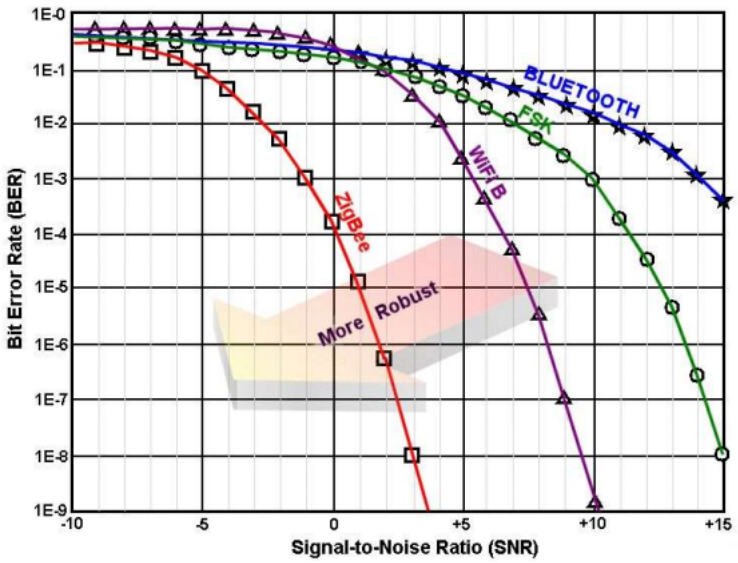


Figure 6.2: ZigBee transmission efficiency respect to SNR-BER correlation

wrongly by the receiver. Overall, increase the transmission power correspond to an increment of SNR thus to a decrease of BER.

As mentioned in the previous chapters (10.0.2), the **Signal-to-noise ratio (SNR)** is a measure of the strength of a signal relative to the background noise present in a given environment or system. It is expressed as a ratio of the *signal power* to the *noise power*, typically measured in decibels (dB).

In communication systems, the signal refers to the information or data that is being transmitted, while noise refers to any unwanted or random fluctuations in the signal. **The higher the SNR, the better the quality of the signal and the more accurate the transmission of information**. Conversely, a low SNR indicates a weaker signal that is more susceptible to errors and interference from noise.

The **Energy Detection (ED)** is used to find a free channel and for carrier sense: the estimation time for ED equals the average over 8 sumbols interval. The **detection threshold** is at $10dB$ above the sensitivity level. The **Link Quality Indicator (LQI)** indicates the quality of data packets that are received by a node: it's based on ED or on the *signal/noise ratio*. It's assesed each time a packet is received and must contemplate 8 different levels of quality. The estimated value for LQI is forwarded to the network and application layers.

For the **Channel Assessment**, the main objective is to detect if the channel is *busy* and operates in 3 modes:

- **Mode 1**: uses ED, if the energy level exceeds the detection threshold, then the channel is marked as *busy*.
- **Mode 2**: perform *Carrier Sense*, the channel is *busy* if the detected signal has the same characteristics as the sender.
- **Mode 3**: Combination of modes 1 and 2 (in *AND/OR*).

Data services At the *physical layer*, the *PPDU (Physical Protocol Data Unit)* reports the result of the transmission to the upper layer (*if successfull or failed*). The main reasons for a transmission to fail are:

1. radio transceiver *out of order*
2. radio transceiver is in the *reception mode*
3. radio transceiver is *busy*

The **frame** have the structure pictured in 6.3:
A frame at the physical layer contains:

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

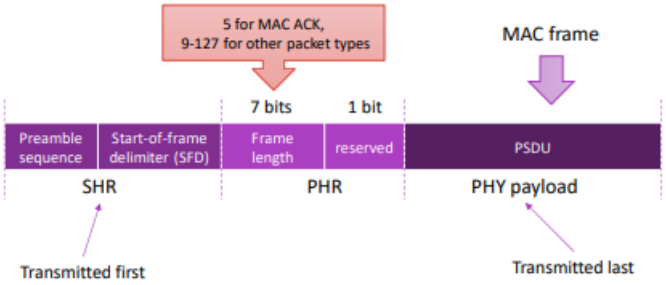


Figure 6.3: 802.15.4 Physical Layer frame structure

- **PHY Payload**: the MAC frame
- **SFD (Start-of-frame delimiter)** : tell the receiver that the real data is beginning
- **PSDU (Physical Service Data Unit)**: it's the actual payload, up to 127 bytes. It's not a good idea to use all of them all despite are only few bytes: with long frames there are problem of carrier sense and multiple access.

6.0.2 MAC Layer

Also at the MAC Layer, we can differentiate three different types of services:

1. ***Data services***: transmission and reception of **MAC frames (MPDU)** across the physical layer.
2. ***Managament services***: perform synchronization of the communications, channel access, management of guaranteed time slots,association and disassociation of devices.
3. ***Security services***: allows *data encryption, access control, frame integrity and sequential freshness.*

The MAC layer defines two types of nodes: ***Reduced Function Devices (RFDs)*** and ***Full Function Devices (FFDs)***.
RFD are meant to implement end-devices with reduced processing, memory, and communication capabilities which implement a subset of the MAC layer functions. In particular the RFD can only associate to an existing network and they depend on **FFDs** for communication. One *RFD* can be associated to only one FFD at a time. Example of RFD devices are simple sensors or actuators like light switches, lamps and similar devices.
The ***FFDs*** implement the full MAC layer and they can act aither as the *Personal Area Network (PAN)* coordinator or as a generic coordinator of a set of RFDs. The PAN coordinator sets up and manages the network, in particular it selects the PAN identifier and manages associations or disassociation of devices. In the association phase the PAN coordinator assigns to the new device a 16 bit address. This address can be used in alternative to the standard 64 bit extended IEEE address which is statically assigned to each device.

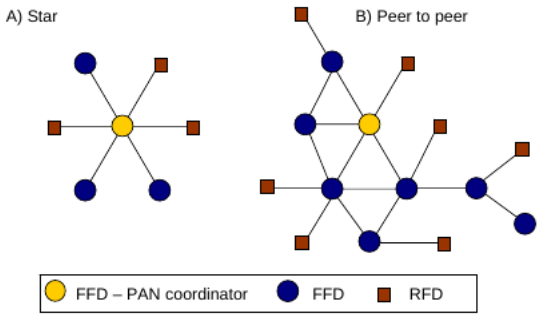


Figure 6.4: Star topology vs Peer to peer topology

The FFDs cooperate to implement the **network topology** as pictured in 6.4. The actual network formation is performed at the network layer, but the MAC layer provides support to two types of network

topologies: ***star*** and ***peer-to-peer*** (*as shown in 6.4*). In the **star topology** one FFD is the PAN coordinator and it is located in the star centre. All the other FFD and RFD behave as generic devices and they can only communicate with the coordinator which synchronizes all the communications in the network. Different stars operating in the same area have different PAN identifier and operate independently of each other.

In the **peer to peer topology** each FFD is capable of communicating with any other device within its radio range. One *FFD (normally the FFD which initiated the network)* act as PAN coordinator, and the other *FFDs* act as routers or end devices to form a multihop network as shown in the previous figure. The *RFDs* act as end devices and each RFD is connected only with one *FFD*.

6.0.3 Channel access

The MAC protocol have two channel access:

1. **Superframe structure**: used in **star topologies** (*it can also be used in peer to peer topologyes organized in trees*) and provides synchronization between nodes to enable energy savings of the devices.

2. **No superframe structure**: is more general and can be used to support communications in arbitrary **peer to peer topologies**.

Access with superframe A superframe is composed by an **active** and an **inactive** portion as shown in 6.5. All the communications happen during the active portion, hence the *PAN coordinator (and the connected devices)* may enter a *low power (sleep)* mode during the inactive portion. The active portion comprises up to *16* **equally sized time slots**. The first time slots is the ***beacon frame*** and is sent by the PAN coordinator to begin the superframe. The beacon frames are used to synchronize the attached devices, to identify the PAN, and to describe the structure of the superframes.

The actual communications between the end devices and the coordinator take place in the *remaining time slots*. The time slots in the active portion are divided into a ***Contention Access Period (CAP)*** and a (*optional*) ***Contention Free Period (CFP)*** as shown in 6.6.

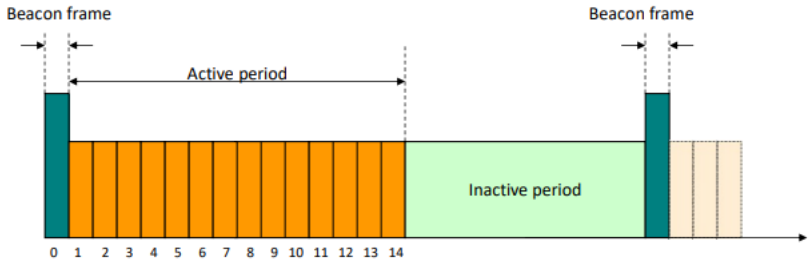


Figure 6.5: Superframe division between beacon, active period and inactive period

In the **CAP period** the devices compete for channel access using a standard slotted *CSMA-CA protocol (Carries Sense Multiple Access with Collision Avoidance)*. This means that a device wishing to transmit data frames first waits for the beacon frame and then it *randomly selects a time slot for its transmission*: if the selected time slot is busy because another communication is already ongoing (*this is detected using carrier sense*) then the device selects randomly another time slot. If the channel is *idle*, the device can begin transmitting on the next slot.

The **CFP period** is optional and it is used for low-latency applications or applications requiring specific data bandwidth. To this purpose the PAN coordinator may assign portions of the active superframe (*called* ***Guaranteed Time Slots*** *or GTS*) to specific applications. The GTSs form the **contention-free period (CFP)**, which always begins at the end of the active superframe starting at a slot boundary immediately following the CAP. Each GTS may comprise more than one time slots and it is assigned to an individual application which access it without contention.

In any case the PAN coordinator *always leave a sufficient number of frames for the CAP period for the other devices and to manage the ssociation/disassociation protocols.* Note also that all contention-based transactions shall be complete before the beginning of the CFP, and each device transmitting in a GTS shall complete its transmission within its GTS.

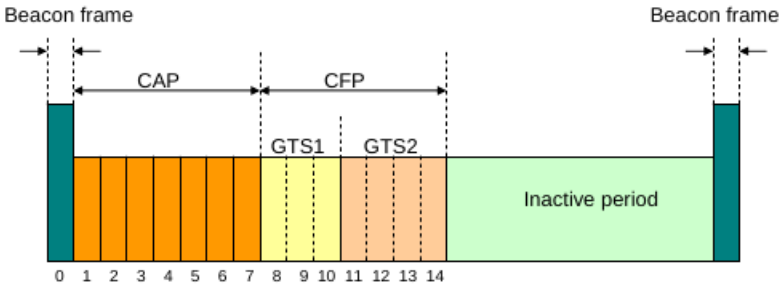


Figure 6.6: Division between CAP, CFP and GTS over active period

The PAN coordinator may optionally avoid using of the **superframe structure** (*thus the PAN is called* ***non beacon-enabled***). In this case the PAN coordinator never sends beacons and communication happens on the basis of the ***unslotted CSMA-CA protocol***. The coordinator is always on and ready to receive data from an end-device while data transfer in the opposite direction is **poll-based**: the end device periodically wakes up and polls the coordinator for pending messages. The coordinator responds to this request by sending the pending messages or by signalling that no messages are available

6.0.4 Data transfer models

The standard supports three models of data transfer:

1. end device to the coordinator (*or router*),
2. coordinator (*or router*) to an end device
3. Peer to peer

The **star topology** use only the first two models because the data transfers can happen only between the PAN coordinator and the other devices. In the peer to **peer topology** all the three models are possible since data can be exchanged between any pair of devices. The actual implementation of the three data transfer models depends on whether the network supports the transmission of beacons, thus obtaining two network models: *beacon-enabled vs non-beacon enabled*.

Data transfers in beacon-enabled networks

Data transfer from an end-device to a coordinator: The end device first waits for the network beacon to synchronize with the superframe. When the beacon is received, if it owns a GTS it directly use it, otherwise it transmits the data frame to the coordinator using the slotted CSMA-CA protocol in one of the frames in the CAP period. The coordinator may optionally acknowledge the successful reception of the data by transmitting an acknowledgment frame in a successive time slot. (*Figure A of 6.7*).

Data transfer from and to a coordinator to an end device: The coordinator stores the message (*a data frame*) and it indicates in the network beacon that the data message is pending. The end-device usually sleeps most of the time and it *periodically listens to the network beacon to check for pending messages*. When it notice that a message is pending it explicitly requests the message to the coordinator using the **slotted CSMA-CA in the CAP period**. In turn, the coordinator sends the pending message in the **CAP period** using the slotted CSMA-CA. The device thus acknowledges the reception of the data by transmitting an acknowledgment frame in a **successive time slot** so that the coordinator can remove the pending message from its list (*Figure B of 6.7*).

Peer-to-peer data transfers: If the sender or the receiver is a end device then **one of the above schemes is used**. Otherwise both the source and the destination are coordinators and they issue their own beacons because two end-devices *cannot communicate directly*. In this case the sender must **first synchronize with the beacon of the destination** 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 and are thus left to the upper layers. The syncrhonization procedure is sketched in 6.8.

Data transfers in non beacon-enabled networks

Data transfer from an end device to a coordinator: In this case the end device transmits directly its data frame to the coordinator using the **unslotted CSMA-CA protocol**. The coordinator acknowledges

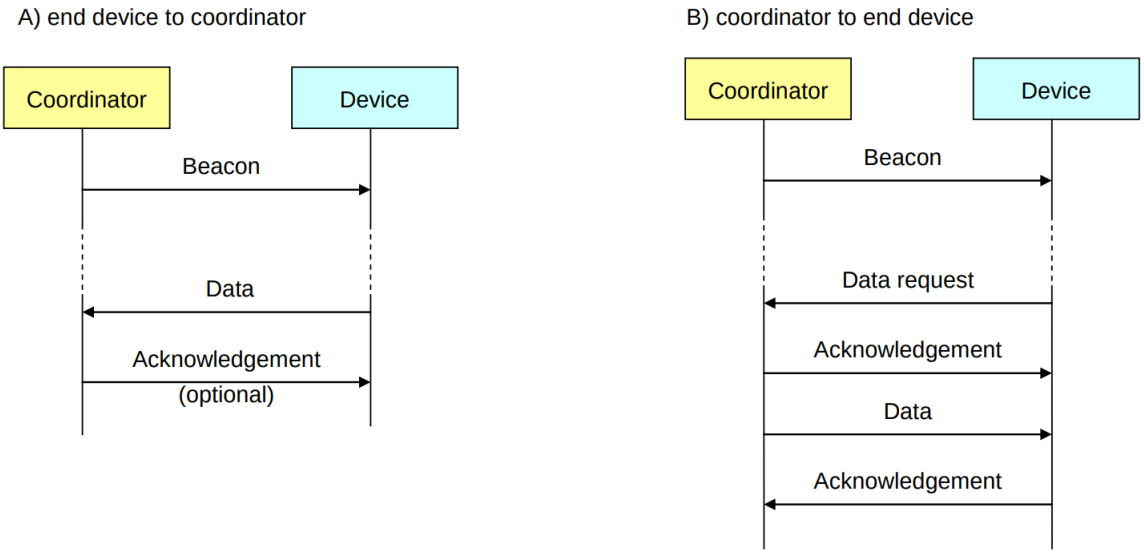


Figure 6.7: Data transfer models in beacon-based networks

Figure 6.8: P2P Data Transfer in Beacon-based network

the successful reception of the data by transmitting an optional acknowledgment frame. (*Figure A of 6.9*).

Data transfer from a coordinator to an end-device: The coordinator stores the message (*a data frame*) and **waits for the appropriate device to request for the data**. A device can inquiry the coordinator for pending messages by transmitting a request using the *unslotted CSMA-CA protocol* (*this request happens at an application-defined rate*). The coordinator acknowledges the successful reception of the request by transmitting an **acknowledgment frame**. If there are pending messages, the coordinator transmits the messages to the device using the *unslotted CSMA-CA protocol*. Otherwise, if no messages are pending, the coordinator transmits a message with a **zero-length payload** (*which indicates that no messages are pending*). The device acknowledges the successful reception of the messages by transmitting an acknowledgment frame so that the coordinator can discard the pending messages (*Figure B of 6.9*).

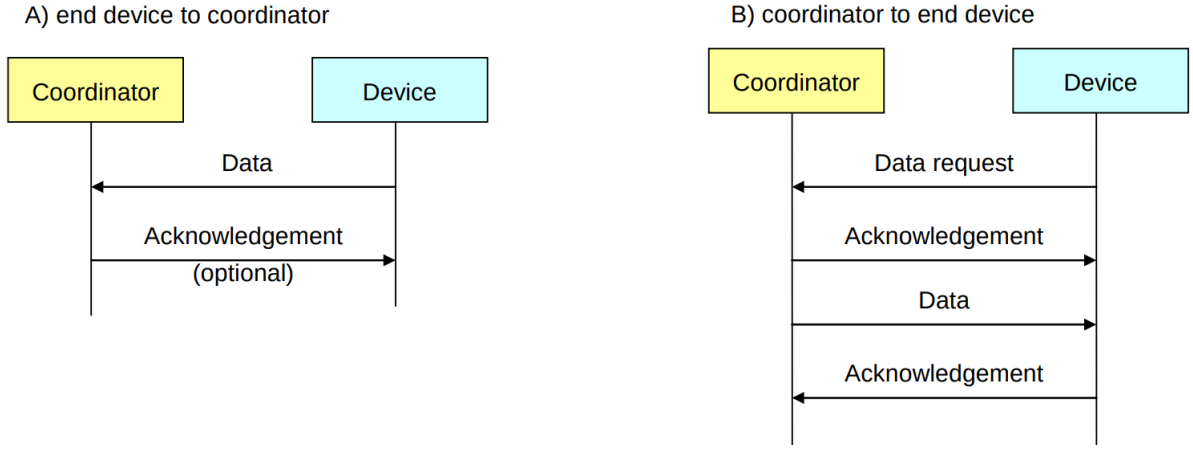


Figure 6.9: Non beacon-based data transfer model

Peer to peer data transfer: : In peer-to-peer PANs each device can communicate with each other device in its *radio range*. In order to do this effectively, the devices wishing to communicate need either to:

1. keep the **radio constantly active** in order to be ready to receive incoming messages or
2. to synchronize with each other.

In the former case the device can directly transmit the data using *unslotted CSMA-CA* while in the second case the device has to wait until the destination device is ready to receive. Note however that the devices synchronization is beyond the scope of the IEEE 802.15.4 standard and it is left to the upper layers.

6.0.5 MAC Layer Service primitives

The MAC layer provide data and management services to the upper layer (*normally the ZigBee network layer*). Each service is specified by a **set of primitives** which can be of four generic types (*as pictured in 6.10*):

- **Request**: It is invoked by the upper layer to request for a specific service;
- **Indication**: It is generated by the MAC layer and it is directed to the upper layer to notify the occurrence of an event related to a specific service;
- **Response**: It is invoked by the upper layer to complete a procedure previously initiated by an indication primitive;
- **Confirm**: It is generated by the MAC layer and is directed to the upper layer to convey the results of one or more service requests previously issued.

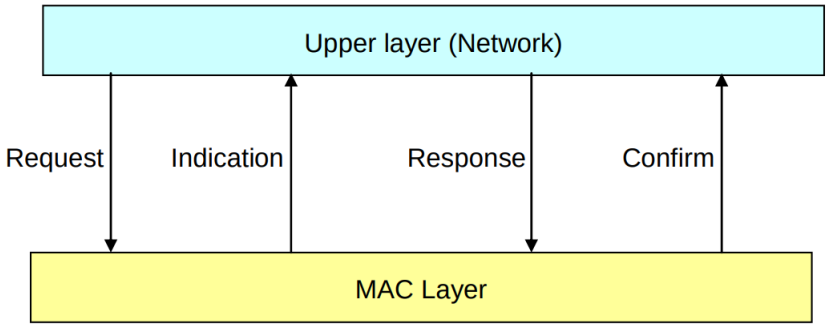


Figure 6.10: MAC Layer services primitive

Data service The data service comprises one main service which exploits only the `request`, `confirm` and `indication` primitives. The `DATA.request` primitive is invoked by the upper layer to send a message to another device. The result of a transmission requested with a previous `DATA.request` primitive is reported by the MAC layer to the upper layer by the `DATA.confirm` primitive, which returns the status of transmission (*either success or an error code*). The `DATA.indication` primitive 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. The overall exchange is sketched in 6.11

Management service The management services of the MAC layer include functionalities for *PAN initialization, devices association/disassociation, detection of existing PANs* and other services exploiting some of the features of the MAC layer. The main management services are summarized in **Table 6.12** : in the table symbol X in a cell corresponding to **service** S and **primitive** P denotes that S uses the primitive P, while symbol O means that primitive P is optional for the **RFDs**.

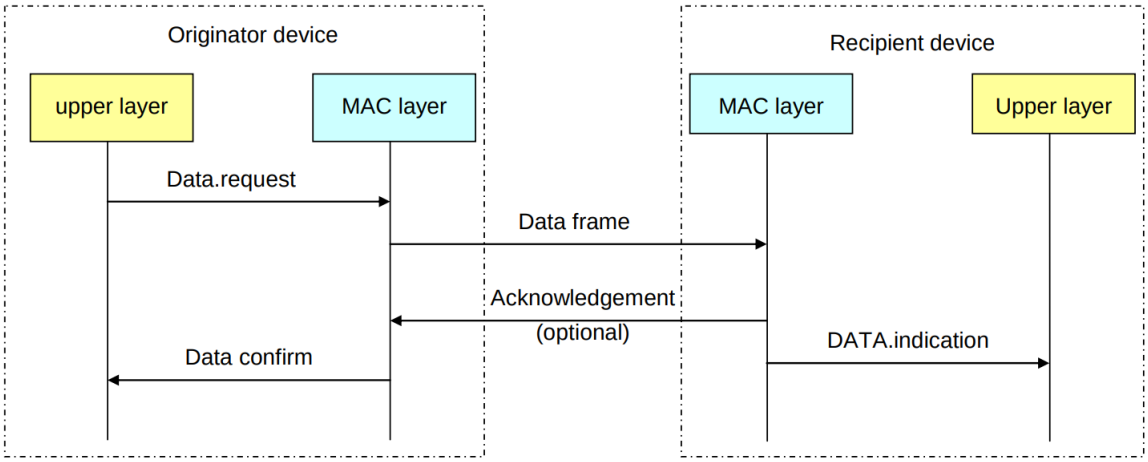


Figure 6.11: Data service exchange using MAC layer primitives

Name	Request	Indication	Response	Confirm	Functionality
ASSOCIATE	X	O	O	X	Request of association of a new device to an existing PAN.
DISASSOCIATE	X	X		X	Leave a PAN.
BEACON-NOTIFY		X			Provides to the upper layer the received beacon.
GET	X			X	Reads the parameters of the MAC.
GTS	O	O		O	Request of GTS to the coordinator.
SCAN	X			X	Look for for active PANs.
COMM-STATUS		X			Notify the upper layer about the status of a transaction begun with a response primitive.
SET	X			X	Set parameters of the MAC layer.
START	O			O	Starts a PAN and begins sending beacons. Can also be used for device discovery.
POLL	X			X	Request for pending messages to the coordinator.

Figure 6.12: Management service functionalities

`ASSOCIATE` service flow example

Let' describe the **ASSOCIATE** service (*as sketched in 6.13*): this service is invoked by a device wishing to associate with a PAN which it have already identified by preliminary invoking the **SCAN** service. The `ASSOCIATE.request` primitive takes as parameters (*among others*) the *PAN Identifier, the coordinator address, the 64-bit extended IEEE address* of the device. The primitive sends an association trquest message to a coordinator (*either PAN coordinator or a router*): since the association procedure is meant for **beacon-enabled networks**, the association request message is sent during the **CAP period** using *slotted CSMA-CA Protocol*.

The coordinator **acknowledges immediately** the reception of the association messages, however this *acknowledgement does not mean that the request has been accepted*.

On the **coordinator side** the association request message is passed to the upper layers of the coordinator protocol stack (*using the `ASSOCIATE.indication` primitive*) where the decision about the association request is actually taken. If the request is accepted the coordinator selects a **short 16 bit address** that the device may use later in place of the **64-bit extended IEEE address**. The **upper layers** of the coordinator thus invoke the `ASSOCIATE.response` primitive of the coordinator MAC layer: this primitive takes as parameters the 64 bit address of the device, the new 16 bit short address and the status of the request (*which can be association successful or an error code*). The primitive thus generates an association response command message which is sent to the device requesting association using **indirect transmission**, *i.e., the command message is added to the list of pending messages stored on the coordinator*.

The **MAC layer** of the device automatically issues a *data request message* to the coordinator after a pre-defined period following the acknowledgement of the association request command. The coordinator

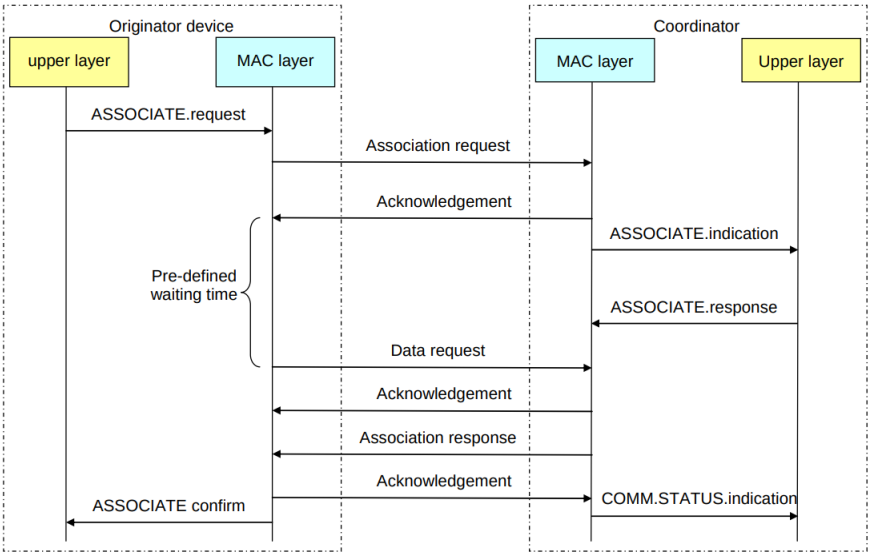


Figure 6.13: `ASSOCIATE` primitives invokation

then sends the association response command message to the device. Upon receiving the command message, the devices' MAC layer issues a `ASSOCIATE.confirm` primitive, while the MAC layer of the coordinator issues the `COMM-STATUS.Indication` primitive to inform the upper layer that the association protocol is concluded either with success or with an error code.

MAC layer security

The IEEE 802.15.4 MAC layers provides a basic support for security, and it leaves advanced security features (*such as keys management and device authentication*) to the upper layers. All the security services are based on symmetric-keys and use keys provided by the higher layers. The MAC layer security services also assume that the keys are *generated, transmitted, and stored by the upper layers* in a secure manner. Note also that the security features of the MAC layer are optional and the applications can decide when and which functionality they use. The security services provided by the MAC layer are the following:

- **Access control**: allows a device to keep a list of devices (called the Access Control List, or ACL) with which it is enabled to communicate. If this service is activated, each device in the PAN maintains its own ACL and it discards all the incoming packets received from devices not included in the ACL.
- **Data encryption**: uses symmetric cryptography to protect data from being read by parties without the cryptographic key. The key can be shared by a group of devices (typically stored as the default key) or it can be shared between two peers (stored in an individual ACL entry). Data encryption may be provided on data, command, and beacon payloads.
- **Frame integrity**: uses an integrity code to protect data from being modified by parties without the cryptographic key and to assure that the data comes from a device with the cryptographic key. As in the data encryption service the key can be shared by a group or by pairs of devices. Integrity may be provided on data, beacon and command frames.
- **Sequential freshness**: orders the sequence of input frames to ensure that an input frame is more recent than the last received frame.

Chapter 7

Embedded Programming

We can define *embedded systems* as a combination of hardware, software and mechanical component. Mainly they're based on a microcontroller: it's a chip that embeds a microprocessor, memory, IO interfaces and optmized mechanism over IO components. The microcontroller interacts with an electro-mechanical devices to which it provides control. Given this controllimg aim, it often has *real-time* constraints.

On the market exists different types of microcontrollers, having different processors, memory and guaranteed performance: they can be specific, also known as *ASICs - Application-Speficic Integrated Circuits* or be general purpose, thus adaptive based on the specific application scenario. In terms of capacity, those devices are highly limited with a very small memory footprint of about $16kB$ of program memory based on flash memory technology and $8kB$ of SRAM for data. Usually there isn't any type of *file system* thus this features must be provided (*and implemented by libraries*) by the developer itself, detailing the controlling steps of the features.

On this devices usually there aren't any **conventional operating system** but they contains only a set of libraries to be able to execute specific functions. During the compilation, those libraries are statically linked to our injected code. Different systems, like *RaspberryPi*, have the features of having a full OS. In this chapter we'll consider only devices without an OS.
This implies also that is not possible to write and compile the code directly on the microcontroller but it's necessary to write the code on a full-fledge computer/OS and compile the given code using a *cross-compiler*, specifying at compilation time the type of system on which the code will run and then upload it on device firmware. The executable file include the user's code, a set of libraries that implement the interconnection with the hardware and mimic the OS functions.

Usually the code is written in C both for performance reasons (*mainly being able to manipulate low-level registers*): as explained later, the `main` is provided by the *OS* itself. This methodology is known as *co-design*, in which both hardware and software are designed and bundled together, without a proper user interface to interact with them.

Based on the specific purpose, when uploading the binary files into the devices can be necessary to add some more information to be able to carry out peculiar function like the *device identifier* to implement *routing functions*. This scenario shift the *programming paradigm* usually used on a classical computer: in classic programming we provide a `main` function but in embedding systems the `main` function is provided by the device itself that execute the setup operation, inizialing the device and then activates the task that implement the functionalities specific for the device. Two things are embedded in user's code: the *initial device configuration* and the *code to run at runtime*. The first is executed by the `main` function that initialize the specific data structure, than the function calls the user's specific code. This approach allows to simplify the initialization, without repeating the procedure everytime.

Usually the software execution on an embedded system is structure as a **loop cycle** (*as a form of duty cycle*) that includes:

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

Sampling sensors needs regular intervals to work properly (*as we'll see later*). All these steps are executed in a **control loop** that is a function invoked by the `main` after device initialization: the user only provide

the function to be repeated, the *loop* itself it's embedded in the device control behavior.
Due to it's simplicity, the interaction with hardware at low level happen by the code interacting with the components by means of **commands** and **interrupts**: a *command* activates the hardware component (*e.g. read from a transducer*) while an *interrupt* signals that the command has been executed.
In parallel with a classical OS, the command are available in terms of *system calls* invoked by the user's code: if there is a waiting time, the invoking thread is suspensed. The interrupts are managed by the OS kernel and re-activates the suspended threads, allowing *async event management* due to the thread/process abstraction and the interfaces exposed by the kernel. So in conventional OS, the programmer's code just see a library stub function that execute the commands and returns a value: the suspension mechanism of a thread is transparent to the user but require also a *stack* (*e.g. TCB*) for each thread to store its context. This approach guarantee asyncronicity, abstracting the passing time between the requested execution and the final result.
For the type of **constrained devices** considered, usually the RAM memory is very small (*e.g. Arduino 1 had 4kB RAM*) that serves for communication buffers, variables and all the basic setup functions of the devices. In case of insufficient memory, this devices usually adopts mainly two different strategies:

1. **Event loop**: single thread that can be suspended but without including thread scheduling. Mainly used in Arduino 1.

2. **Event based programming and task**: manage directly interrupts, implementing **event handlers** for the actions to be carreid out in response to those events. This model is present in TinyOS but also supported by Arduino.

7.1 Arduino model

The flow process of Arduino is defined in a *single loop function* that may also invoke other functions: the loop is executed by a **single thread**, without suspension thus the I/O makes the thread in waiting until the IO is complete, but without contemplate any thread scheduling. If you want to *delay* the execution explicitly it's necessary to use delays function in the code itself: this mainly happen to impose a strict time in sampling.

In figure 7.1 is sketched the event loop model adopted in Arduino.
The ordered steps are:

1. The first `init` call is executed by the device itself and it's used to initialize the device to be able to communicate with the hardware components.

2. At the end of the `init` function, the `main` loop is invoked. This phase contemplates the send of commands to the hardware. When the command has been executed, the Arduino RTS (*Run-Time Support*) signals the end of command communication and return.

3. After the command execution is terminated, the main loop is re-activated by the main thread. If the *active workload* is already done, we can explicitly invoke a `delay` function that send a command to a timer, suspending the CPU until the timer reactivates the main loop by firing a signal.

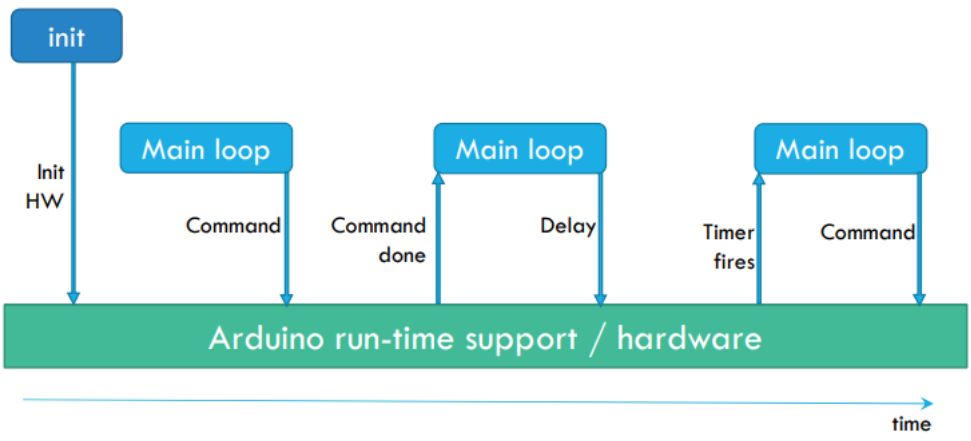


Figure 7.1: Arduino event loop model

The Arduino simple model fits well with simple sensing and controlling activities, including natively also libraries for actuators. Despite it's not born for long,stable and asyncrhonous communications, recent version includes libraries and HW to support **asyncrhonous event management** (*requiring to manage communications through serial line*) that is as the foundation of communication between devices. Mainly, sensing activities are syncrhonous, while communication and specific sensing are usually asyncrhonous.

7.2 Case Study: TinyOS

TinyOS hasn't the concept of event loop but was designed to support **asynchronous management of event** by the means of:

- **Commands**: offers functions to program and activate the hardware
- **Events**: it abstract the *interrupts* in form of *upcall*, the programmer has to define handler function for each event or use handler function provided by the RTS.
- **Tasks**: defines **non-preemptive tasks** that are execute sequentially, allowing to manage different independent activities. When the task execution is terminated and there aren't anymore to execute, the CPU goes in *idle mode* so avoiding to store the context of a given task. Still, interrupts should be handled as soon as they arrive by means of an event handler: those events can start a task to perform complex/long computation and can be *pre-empted* **only by events**. This is done with little overhead using the stack. This is the only exception to task managent and it's mainly due to the fact that:

  1. During handler execution it's not possible to receive another interrupt so the second interrupt reception would be delayed
  2. The event handler it's not suitable to implement complex task but only to update data structure, thus complex work must be posted as a **task**. The event handler should be *as short as possible*, allowing to update data structure and give commands to the HW but if the event management require more work, this work must be post as a **task**, as already said.

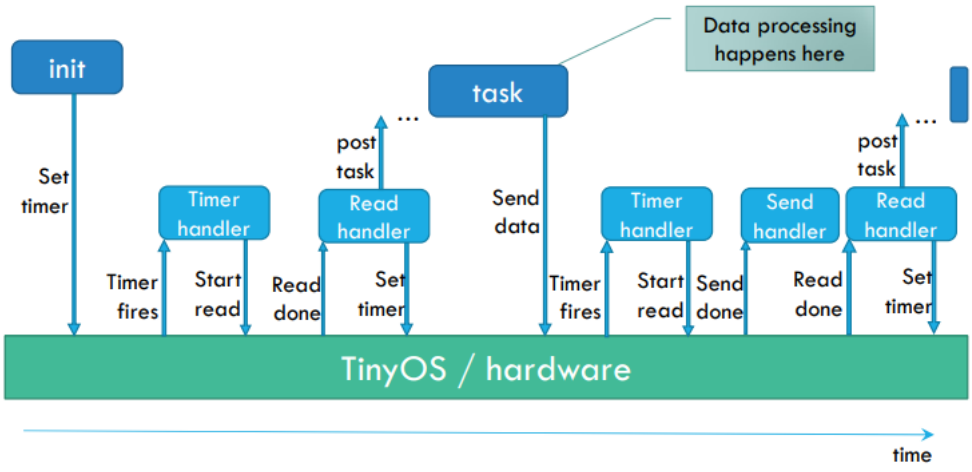


Figure 7.2: TinyOS asyncrhonous model

The control loop is not provided in the underlying RTS: in the example pictured in 7.2, the timer handler is invoked when the interrupt *timer* is fired. The handler will stop shortly after sending the *reading* command to the RTS: the result is not immediate but will arrive later after the RTS will notify that the operation is complete (*Read done*). Upon reception of the readed values, the reader handler is executed and the process is delegated to a task. In the read handler, alongside posting a task, it's necesssary to provide what to do after the task execution and this is done by setting another timer. After the task is completed, it sends data (*transmitting the data to other device*, which takes time and it's not immediately performed; when complete will be signaled by the RTS with "*Send done*") and then the RTS will signal that the timer fill fire the end signal. After a the task is performed, the operations are repeated thanks to the event-handler chain association. The general abstraction is sketched in figure 7.3.

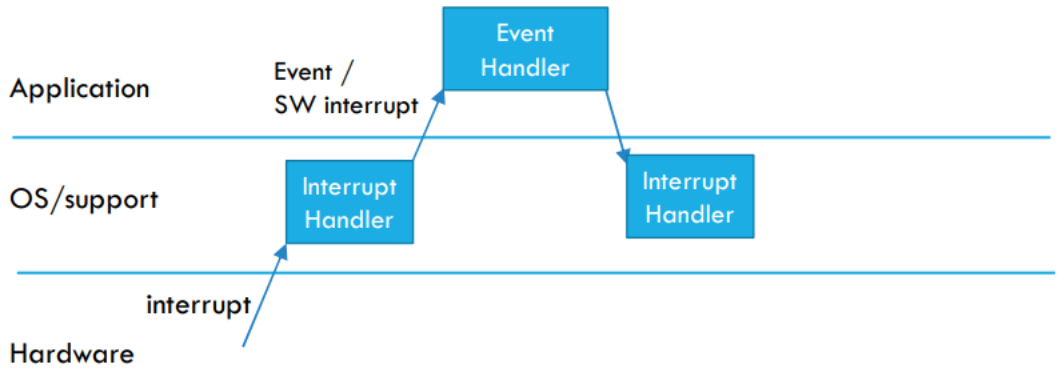


Figure 7.3: Simple abstraction: interrupt handler invoke event handler function

The *interrupt handler* it's only an invokation of the event handler and *must be the simplest one*: it's still possible to invoke a command in the event handler but **that command must relate to a different hardware component from which the interrupt was received**. Manipulate the same data structure of components from which the interrupt was received can lead to an **incosistent state**. This mostly

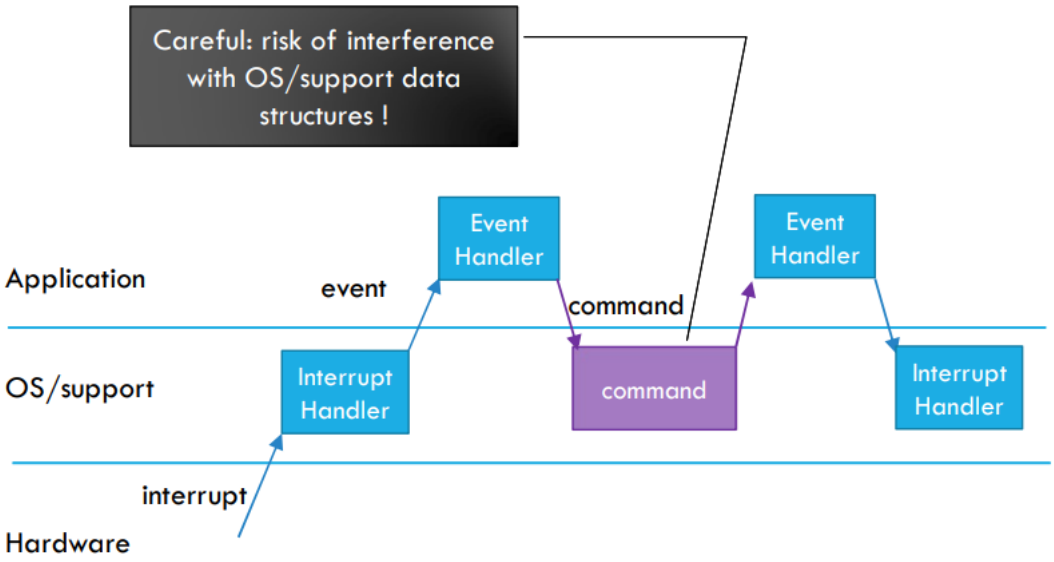


Figure 7.4: Risk of concurrent modification of RTS DS

justify the introduction of task event management: this allows to strictly decouple the interrupt handler from the event handler, adding another layer of abstraction, as sketched in 7.5.

To support asyncrhonous communcation, Arduino provides an **interrupt** mechanism to be able also to go in *low-power* mode: this allows to avoid to save state of current process. The key idea, explained in detail later, it's that *interrupts* should be handled as soon as they arrive by using event handlers. The event handlers should be as short as possible, allowing to *update data structure, send commands to hardware components (not all components)*, obtaining a model similar to TinyOS's behavior.

7.3 Case Study: Arduino

Arduino is an **open-source electronics prototyping platform** based on flexible, easy-to-use hardware and software. It comrpises three elements: a board, the IDE and the reference forum.

The following board 7.6 is *Arduino Uno* based on *AVR Arduino microcontroller Atmega328* with:

- SRAM 2KB (data)
- EEPROM 1KB

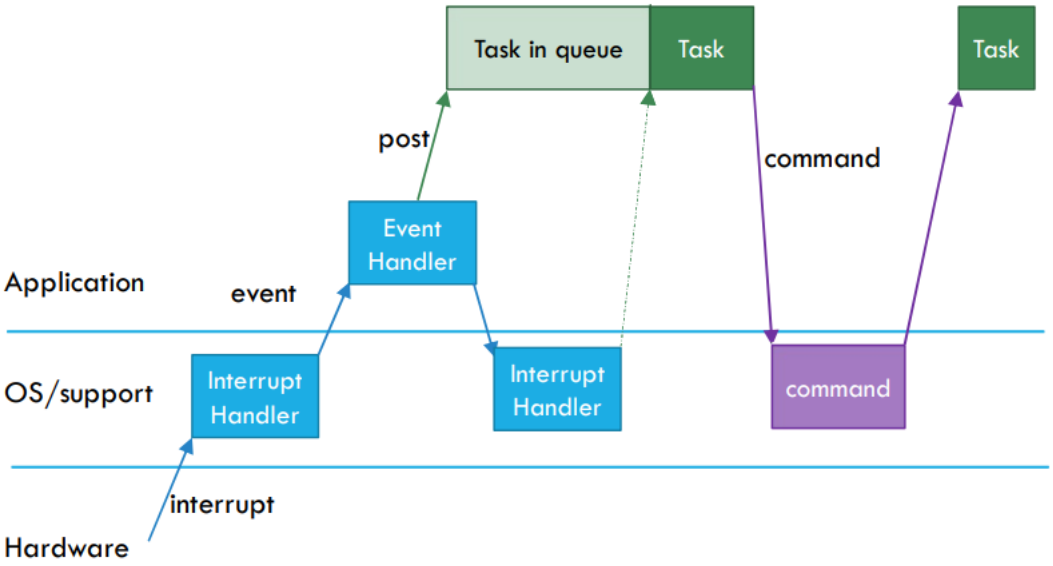


Figure 7.5: TinyOS task abstraction solution to concurrent modification of RTS data structure

- Flash memory 32 KB (program)
- 14 digital pins (both input and output)
- 6 analog pins (only input)
- other pins for power

The circuit is powered with 6V, so the value 1 correspond to 6V/1024. The ADC converts the voltage in 10 bits.

Some terminology:

- "*sketch*": a program you write to run on an Arduino board
- "*pin*": an input or output connected to something. (*e.g. output to an LED, input from a knob*).
- "*digital*": value is either **HIGH** or **LOW**. (*aka on/off, one/zero*) (*e.g. switch state*)
- "*analog*": value ranges, usually from $0 - 1023$. (*e.g. LED brightness, motor speed, etc.*)

The **variables** already defined:

- *Constants*:
  - level of energy (`HIGH`, `LOW`)
  - mode of pin (`INPUT`, `OUTPUT`, `INPUT_PULLUP`)
  - led13(`LED_BUILTIN`)
- *Types*: word, strings, etc;
- *Variable scope and qualifiers*: `volatile`;
- *Usefulness*: `sizeof()`

The overall *structure* it's based on two functions:

- `void setup()..`: called when a sketch start. Will run only once.
- `void loop()...`: user defined code.

The **builtin function** that can be used in `loop()` are:

- **Digital functions**: `pinMode(); digitalRead(); digitalWrite();`

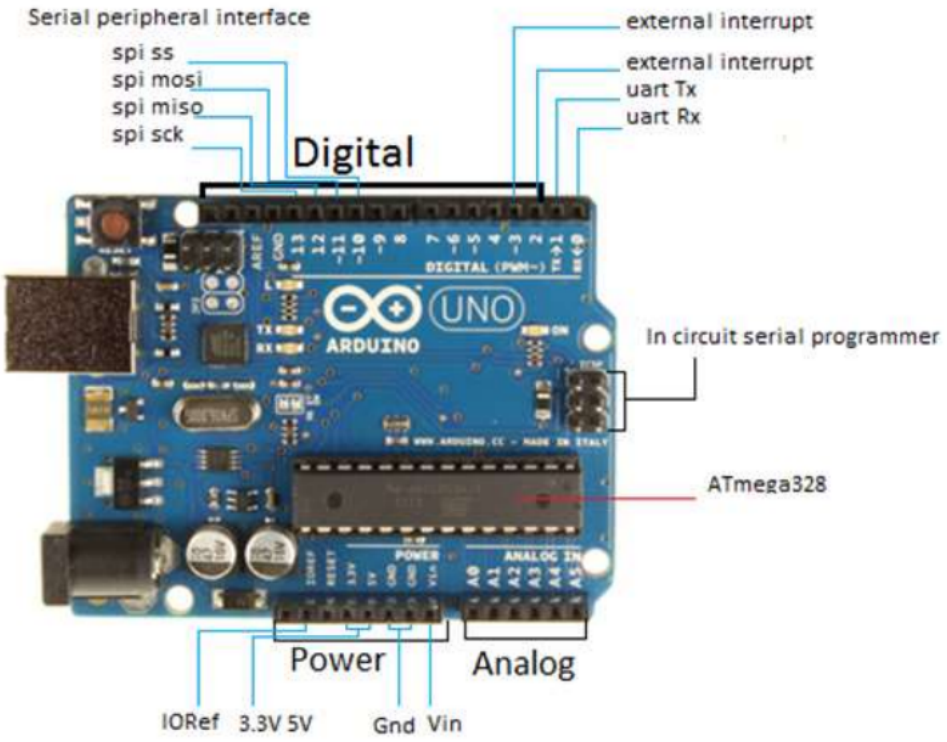


Figure 7.6: Arduino Uno analog/digital pins and power pins

- **Analog functions**: `analogReference(); analogRead(); analogWrite();`
- **Advanced I/O**: `tone(); noTone(); shiftOut(); shiftIn(); pulseIn();`
- **Time**: `millis(); micros(); delay(); delayMicroseconds();`
- **Math/trigonometry**: `min(); max(); abs(); sin(); cos();`
- **Random Numbers**: `randomSeed(); random();`
- **Bits and Bytes**: `lowByte(); highByte(); bitRead(); bitWrite(); bitSet(); bitClear(); bit();`
- **External Interrupts**: `attachInterrupt() detachInterrupt();`
- **Interrupts**: `interrupts(); noInterrupts();`
- **Communication**: `Serial; Stream;`

Here an example of *analog and digital read*:

```
1  // Reads an analog input on pin 0 and converts it to voltage;
2  // Reads a digital input on pin 2;
3  // When input on pin 2 is high prints voltage in the serial and switches the led.
4  const int buttonPin = 2; // the number of the pushbutton pin
5  const int ledPin = 13; // the number of the LED pin
6  int buttonState = 0; // variable to read the pushbutton status
7  void setup() { // runs once when you press reset
8    Serial.begin(9600); // init serial lineat 9600 bps
9    pinMode(ledPin, OUTPUT); // init. the LED pin as an output
10   pinMode(buttonPin, INPUT); // init. pushbutton pin as an input
11 }
12
13 void loop() {
14   // reads the state of the pushbutton value:
15   buttonState = digitalRead(buttonPin);
16   if (buttonState == HIGH) { // if the pushbutton is pressed
17     digitalWrite(ledPin, HIGH); // turns LED on... // ... and reads the input (in
           [0,1023] on analog pin 0:
```

```
18        int sensorValue = analogRead(A0); // Convert analog reading to a voltage (0-5V)
              :
19        float voltage = sensorValue * (5.0 / 1023.0);
20        Serial.println(voltage); // print out the voltage
21      }
22      else digitalWrite(ledPin, LOW); // turns LED off
23    }
```

Exercise Consider the following fragment of Arduino code:

```
void loop() {
int sensorValue = analogRead(A0);
Serial.println(sensorValue);
delay(100);
}
```

Compute its duty cycle assuming that:

- Reading an analog value takes 2 milliseconds
- Transmitting along the serial line takes 5 milliseconds

7.3.1 Arduino interrupts

As in TinyOS, allows management of **asyncrhonous events**: this is possible by providing an interface to manage *interrupts*, allowing async access to sensor data and actuators. There are three types of interrupts:

- **External**: a signal outside, connected to a pin
- **Timer**: internal to Arduino (*e.g. timer feature, managed by RTS*)
- **Device**: an internal signal coming from a device (*ADC, serial line, etc*). Interrupts are managed by RTS of Arduino. For **external interrupts**, the support is provided by Arduino at run-time by declaring `attachInterrupt(interrupt#, func-name, mode);` in the `setup()` function (*Shown later*).

Arduino provides only two *external interrupts pins*, respectively $INT0$ and $INT1$ which are mapped to **pins 2 and 3**: they can be set to trigger on `RISING` or `FALLING` signal edges, on `CHANGE` or on `LOW` level. The triggers are interprented by hardware, and the interrupt is very fast. The following are the possible values of `mode` parameter:

- **RISING**: the interrupt occurs when the pin passes from `LOW` to `HIGH` state. (*Shifting from* $0V$ *to* $5V$).
- **FALLING**: the interrupt occurs when the pin passes from `HIGH` to `LOW` state
- **CHANGE**: the interrupt occurs when the pin switches state
- **LOW**: the interrupt occurs whenever the pin has `LOW` state. Not necessary a change of state. If it remains `LOW` the interrupt occurs again
- **HIGH**: the interrupt occurs whenever the pin has `HIGH` state. Not necessary a change of state. If it remains HIGH the interrupt occurs again.

The connection pictured in the *breadboard* sketched in 7.7 implement:

- A **button** to digital input 2: with 10kΩ resistor
- A **led** to digital output 7: with 220Ω resistor

The code of the two function `setup` and `loop` is the following:

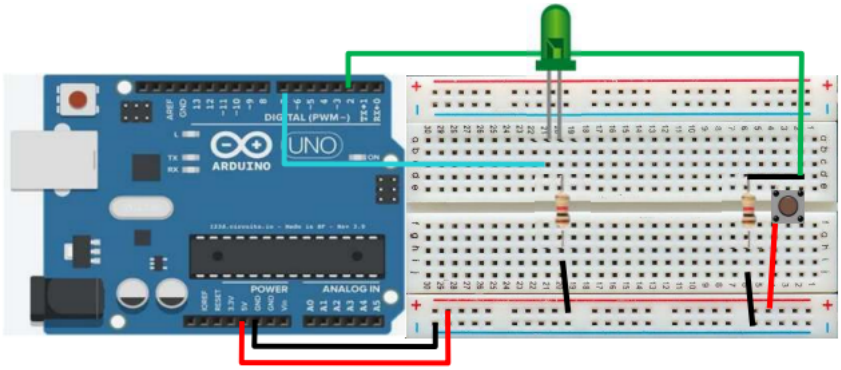


Figure 7.7: Example breadboard with button, led and two resistors

```
/** * test interrupt * */
volatile int greenLed=7;
volatile int count=0;
void setup() {
  Serial.begin(9600);
  pinMode(greenLed, OUTPUT);  //set green light as output
  digitalWrite(greenLed, LOW); //write LOW to digital pin 7
  attachInterrupt(0, interruptSwitchGreen,RISING); // 0 because it is pin 2
}
```

The `Serial.begin(speed)` specify the data rate in bits per second for serial data transmission: the rate must be the same setted in the Serial Monitor so be sure to be able to receive the data at the same rate of emission.

```
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); //turn off the green light, resetting the counter
    Serial.println("now off");
  }
}

void interruptSwitchGreen(){
  digitalWrite(greenLed, HIGH); //turn on the green light
  count=0;  //reset the counter
  Serial.print("now on");
}
```

The first thing we can notice is that the two variables we define (`greenLed` and `count`) above the `setup` function are declared as *volatile*. If you don't declare them as volatile the processor would allocate them in a register but this would mean that when one of these variables is updated, the update would have no effect in the loop where it is used. For the `count` variable there is the risk of concurrent modification both by the `loop()` function and the interrupt handler function `interruptSwitchGreen()`: if `count` is set to 0 and was not declared as *volatile*, the modification would be noticed by the two diffent function that manipulate the variable's value. In this way the compiler is forced to store the variable in memory, there is also another variable which is `greenLed` which is volatile even if it has never been written, it is in practice a *constant* and therefore in this case they could also avoid call it volatile. The code turns on a led which is initially off. In the `setup` code we declare, beside the `pinMode` and the initialization of the `greenLed`, how manage an interruption on PIN 2 by specifying the function `interruptSwitchGreen` and on which signal the execution must happen (***`RISING` signal***). When the interruption is catched on the speicifed PIN, the function is invoked and turn on the led, in the `loop` function then this led is only turned off and to turn it off we go to reset a counter.

Chapter 8

Energy Harvesting

Energy harvesting comes to a wide range of topics like *energy haversting Iot architecture, properties of the energy source, battery and load and energy neutrality.*

The IoT device that we consider are battery-powered (*either rechargable or changeable*). The battery life can be increased, for example by enlarging batteries despite it is come with major costs. Another point of view is to extends device lifetime but reducing the computational ability and limited communications. Those two are the main pathways to prolong the battery life: both can be applied. The decisions regarding the battery are done at the begginign while the duty cycle structure can be *modified*, by making it *with a dynamic behavior*: adapting the DC can prolong the lifetime, by respecting the device functional requirements.

The alternative is give by **energy harvesting** which consists in converting energy from one *form* to another. The source of energy can have different properties (*periodic, non-predictable, etc*): after identifying the source, the tradeoff between how acquiring it physically and the effective utilization in the device must be designed.

The energy harvesting implies also a complex design of the device, allowing to parametize more features of the device.

8.1 Harvesting architectures

The **energy source** is the source to be harvested while the **harvesting source** is the technology used to extract energy from the environment. In most of the cases this sources have a not controllable behavior so the device's harvesting source must take the behavior into account. The **load** is the consumption of energy in a device due to its activities: it varies over time based on the specific activities the device is executing at a time. Overall, an **Harvesting system** is a system that supports a variable load from a variable energy harvesting source. It allows to power the device even in a period when the energy source does not provide energy by storing it. To match energy supply and load, two main approaches are used:

1. **Energy buffer**: as a rechargeable battery or a supercapacitor
2. **Adapting load**: align the workload to the actual energy supply, implying also reducing functions

Neither of these approaches may be sufficient because the *load* cannot be reduced arbitrarly (*having a dynamic DC it's a complex task to perform*) and the *buffer* one it's not the ideal solution because there is no *infinite storage* and must be also considered *energy leaks*.

In a simplified scenario, we have two components, the *energy harvester* and the *device*, interacting together as pictured in 8.1.

There are some problems to be addressed: the device can be **abruptly turned off** and the state management of internal state of the device must be addressed, exploiting the energy to mantain consistency and resume operation later. Another issue is that the energy supply can be the bare minimum, **oscillating around the threshold to work properly**.

8.1.1 Direct harvesting model

In the simplest scenario, called ***Harvest-Use***, **no energy buffer is provided** to store energy thus the excess supply is *wasted*: the power produced immediately is consumed so, referring the diagram 8.2, the device can operate anytime at time t when $P_s(t) \geq P_c(t)$, where:

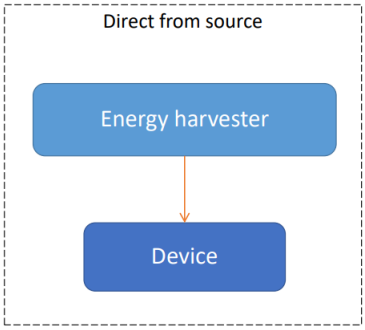


Figure 8.1: Direct harvesting scenario

- $P_s(t)$ is the **power harvested** at time t
- $P_c(t)$ is the **power consumption on the load** at time t

The *waste of energy* is given by the following conditions:

- whenever $P_s(t) < P_c(t)$: the device is turned *off* due to insufficient energy
- whenever $P_s(t) > P_c(t) : P_s(t) - P_c(t)$ is in excess

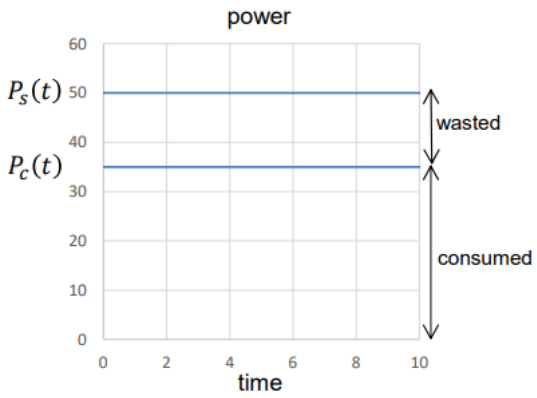


Figure 8.2: Energy base model

8.1.2 Energy buffer model

Another model is ***Harvest-Store-Use*** in which is introduced an **energy buffer** 8.3: with this model, in principle, the device can work anytime but the reality is different, as explained following.

In this model the energy is harvested whenever possible and stored for future use. The *residual energy* is tored and it is used later when either there are no harvesting opportunities or the device task require more energy.

The problem is that no exists an **ideal energy buffer** that can *store any amount of energy, does leak energy over time, has a decresing charging efficiency* $\eta = 1$ that in the real world is below 1. The device can operate any time if, for every time interval $(0, T]$:

$$\int_0^T P_c(t)dt \leq \int_0^T P_s(t)dt + B_0\, dx \tag{8.1}$$

where B_0 is the initial charge of the *ideal supply*. The integral express the **amount of energy the device has consumed** (*consume power multiplied by time*), the same is for the energy supply, as pictured in 8.4.

This model does not consider the point previously cited about the ideal energy buffer like *energy leaks and charging efficiency*. The diagram in 8.5 picture the *energy storing* using the previous model:

As shown, we can compute how much power we are using from the buffer: this is done only for values of x (*or the difference in the integral*) over 0.

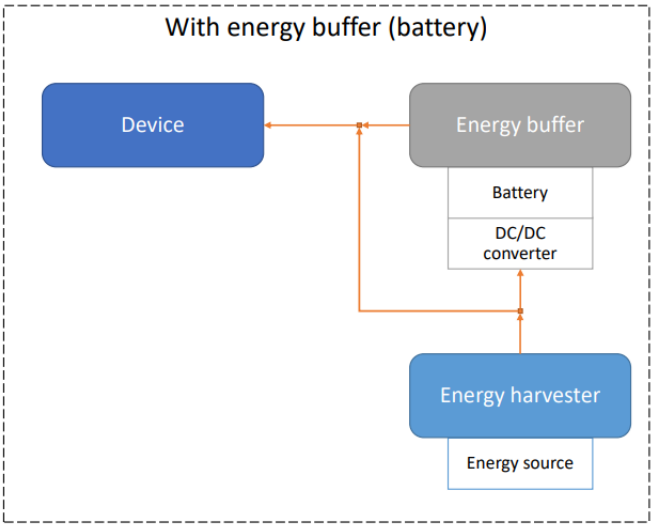


Figure 8.3: Energy buffer model

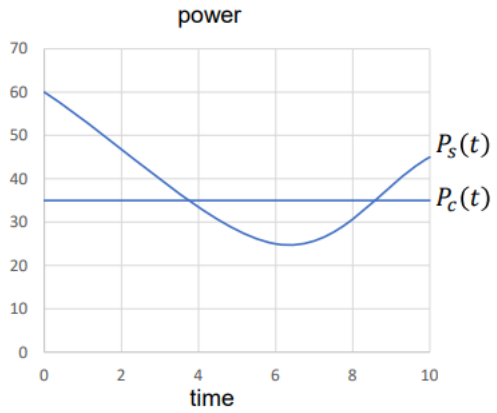


Figure 8.4: Power diagram for buffer model

8.1.3 Non-ideal energy buffer model

This model still refer to **ideal buffers** for some of the parameters that define the behavior. Differently, *real-world* buffers are defined by others adjunctive parameters:

- 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 a time t
- $\eta < 1$: 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
- **Rectifier function**:

$$x^+ = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases} \tag{8.2}$$

The energy conservation leads to the following equation $\forall T \in (0, \infty]$:

$$B_t = B_0 + \eta \int_0^T [P_s(t) - P_c(t)]^+ dt - \int_0^T [P_c(t) - P_s(t)]^+ dt - \int_0^T P_{leak}(t) dt \geq 0 \tag{8.3}$$

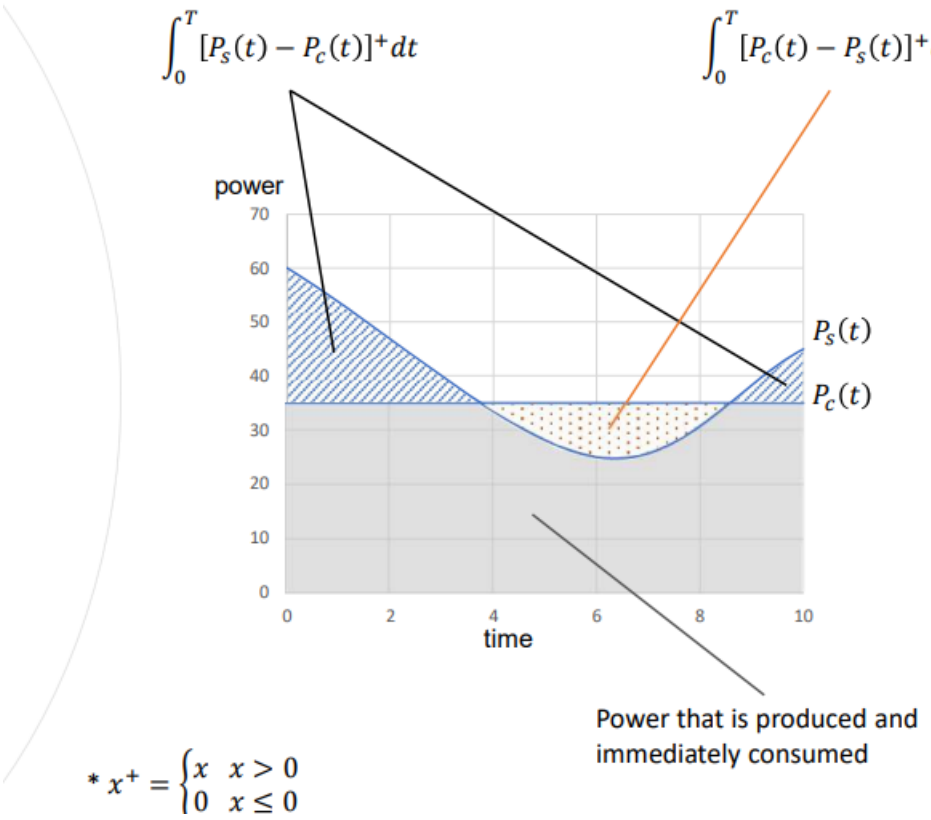


Figure 8.5: Buffer model expanded

where:

- The first integral accounts for the **energy produced in excess** (*thus not consumed*) that goes to buffer
- The second accounts for the *energy consumed from the buffer* (*when the production was insufficient*)
- The last integral accounts for **buffer energy leak**

In the previous model, what is still missing is the *buffer limit*: is necessary to guarantee that the buffer capacity is not exceeded or, better, to guarantee that there is no *wasted energy*. This can be formalized as by imposing that $\forall T \in (0, \infty]$, the value of B_{max} is:

$$B_{max} \geq B_0 + \eta \int_0^T [P_s(t) - P_c(t)]^+ dt - \int_0^T [P_c(t) - P_s(t)]^+ dt - \int_0^T P_{leak}(t) dt \tag{8.4}$$

8.1.4 Exercise

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

- In the interval $[0, 10sec]$ the energy production is constant and produces $P_s(t) = 80mA$
- In the interval $[0, 4sec]$ the load is $P_c(t) = 150mA$
- In the interval $[4sec, 10sec]$ the load is $P_c(t) = 20mA$
- The charging efficiency is $\eta = 95\%$
- The battery charge at time 0 is $400mAh$
- The energy leak of the battery is negligible

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

8.2 Energy sources

The energy sources can be classified based on the *controllability* property as:

- **Fully controllable**: energy sources that can provide harvestable energy whenever required.
- **Partially controllable**: the actual energy that you can transfer to the device may be limited by the environment. (*e.g. hydrielectric centrals does not contemplate summer months without rain*).
- **Non-controllable**: cannot be activates on-demand (*sun, wind , etc*).

For the **un-controllable** energy sources, the classification involves the *predictability* property as:

- **Uncontrolled but predicatable**: energy sources are those for which there exists reliable models that forecast the energy availability. This type of sources are the best suitable for IoT application.
- **Uncontrolled and unpredictable**: are those for which there not exists reliable models that forecast the energy availability

In the following table are reported the main features of energy sources:

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 8.6: Energy sources parameters

The interesting parameter is the *conversion efficiency* column and the *energy harvested column*. In the following we discuss some of the harvesting sources commonly used.

1. Radio Frequency (RF) 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 have own battery*).

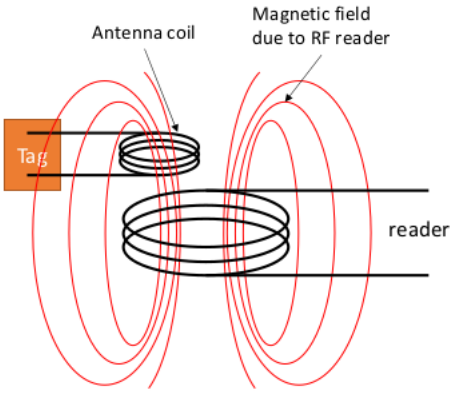


Figure 8.7: RF tag schema

RFID reader queries an *RFID tag* by sending RF signal: RFID tag is entirely powered by the energy harvested by the antenna coil. This energy is just sufficient to send back a reply.

2. Piezo-electric Use mechanical force to deform a piezo-electric material, which results in an electric potential difference. Usually are composed by:

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

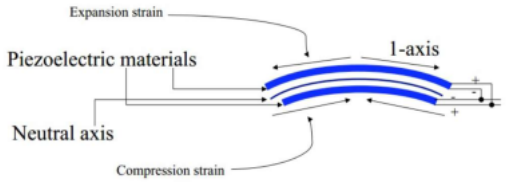


Figure 8.8: Piezo-electric schema

3. Wind turbines Current wind turines can operate at different ranges of wind speed and are suitable for IoT mostly thanks to the *small size, low height* and *operative range*even with weak wind condition. For a micro wind turbine, we report the data **??** from *"Optimized Wind EnergyHarvesting System Using Resistance Emulator and Active Rectifier for Wireless Sensor Nodes", Yen Kheng et al. IEEE Transaction Jan. 2021.*

Geometrical parameters	unit	value
Radius of wind turbine	cm	3
Volume of AC generator	cm^3	1
Mass of wind turbine generator	g	100

Mechanical & electrical parameters	unit	value
Maximum wind speed	m/s	10
Rated generator speed	Rpm	3000
Rated generator frequency	Hz	50
Rated electrical power	mW	40
Open-circuit voltage	V	4
Internal resistance	Ω	100

Those value charaterize the mechanical and electrical part of the examined turbine, allowing to derive the following diagrams: the first 8.9 shows that the wind is not fully controllable, the second 8.10 diagram shows the electrical power generated for different wind speed and load applied, highlighting that the *maximum power* is generated when the **load resistance** matches the internal resistance of the generator.

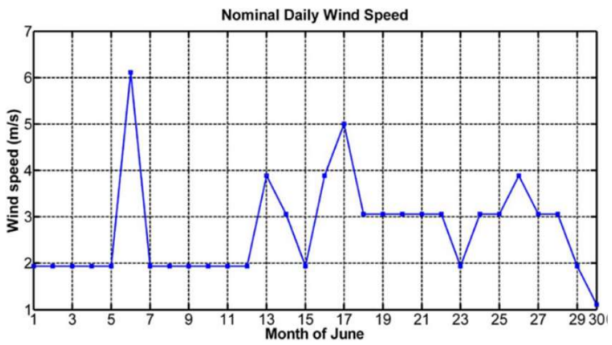


Figure 8.9: Wind relevation during the month of June

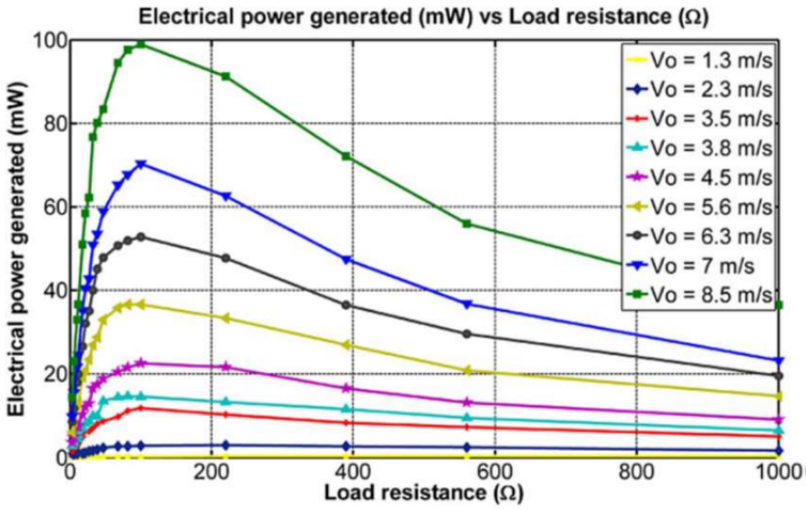


Figure 8.10: Power generated respect to load

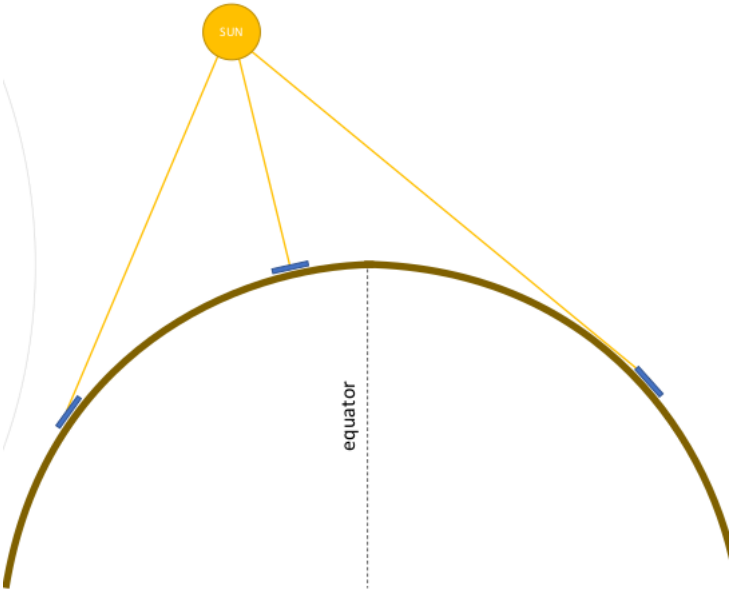


Figure 8.11: Solar irradiance

4. Solar energy The production of the panel depends on parameters like:

- **Solar irradiance**: indicates the amount of *solar rays* that are transmitted to the surface, measured in square meters (*measured as* W/m^2). The efficiency is charaterized also by the *geographical position*, the weather cnditions, the daylight of the day and all the environmental factors.
- **Size of the panel**
- **Charging efficiency**: depends on the electronics and on the materials of the panel, usually 15% for solar panel.

The **analytical model** is the theorical model to compute the daily power output and its distribution per hour in two different locations.The following diagram 8.12 picture the solar production of a solar panel in a timeframe of 24 hours: it shows that the *availability* of harvested energy varies in different period of the year thus a solution to be adopted is to **oversize** the harvesting subsystem and the battery to match with the *worst conditions*.

The following diagram shows the cloud day respect to the *irradiance value* (*refers to the transfer of energy between two bodies by means of electromagnetic waves*). The model is based on an approximation,

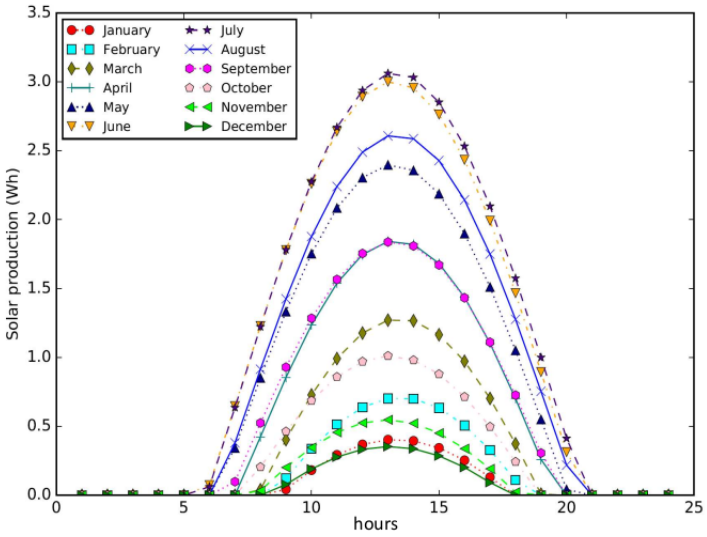


Figure 8.12: Solar panel production

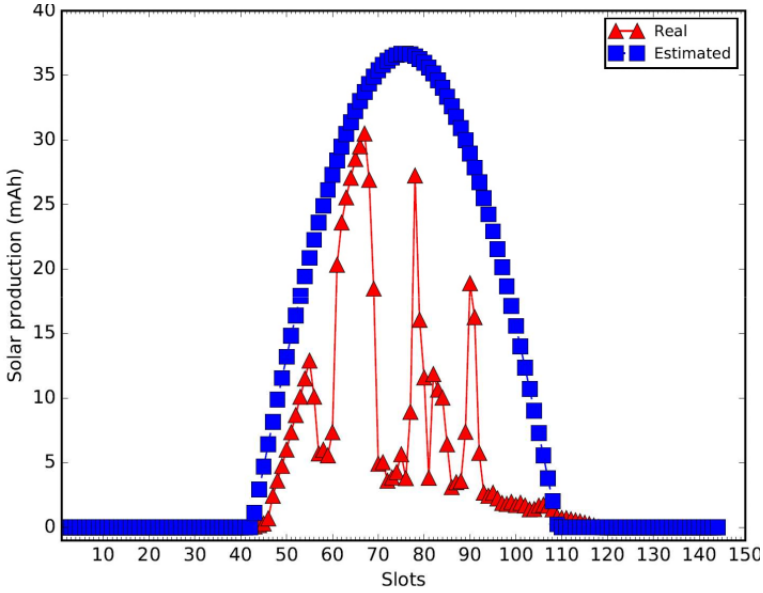


Figure 8.13: Approximated values vs real relevations

forecasted by previous timeframed observation thus there can be differenes between the *real production* and the *estimated production* that cannot be precisely forecasted.

A solution to compensate the missing point of the analytical model can be to mix and match the *solar panel* with *wind energy* production, allowing to guarantee a more realistic and stable source of energy, as shown by the pictured data [1] in 8.14.

Limitations The solution here presented does not guarantee that the device can be always operational, even for mixed solution because they depends on the capacity to forecast the energy production of the chosen sources. In case of harvesting solar and wind energy, must be considered also:

- In a night without wind there's no energy production
- The device relies only on the residual battery charge so if the residual charge is insufficient it may drop below the minimum and the device turns off. It will turn on again in the morning, when the source is return back to the minimum necessary to the device to be able to be operational.

[1] *Cloudy Computing: Leveraging Weather Forecasts in Energy Harvesting Sensor Systems", IEEE Secon 2010, Sharma et al.*

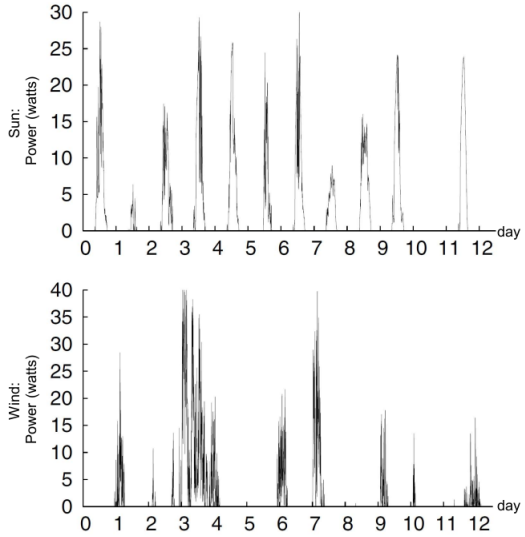


Figure 8.14: Sun vs wind avilability over 12 days

8.2.1 Storage technologies

The battery and the harvester must be sized together to guarantee the device to be operational for the time required to be functional.

The following table 8.15 indicates the *battery type* and the *capacity* for each energy cell that can be charged by **reversing the internal chemical reaction**:

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 8.15: *Sealed Lead Acid (SLA), Nickel Cadmium (NiCd), Nickel Metal Hydride (NiMH), Lithium Ion (Li-ion)*

The **Pulse charging** involves delivering a series of high-energy pulses to the battery, followed by a resting period. During the resting period, the battery is allowed to recover and stabilize before the next pulse is delivered. This charging method is typically used for lead-acid batteries and is intended to reduce the buildup of lead sulfate on the battery plates, which can reduce the battery's capacity and lifespan. Pulse charging is also sometimes used for lithium-ion batteries to extend their lifespan.

The **Tickle charging**, on the other hand, involves applying a small, constant current to the battery after it has been fully charged. This current is intended to keep the battery topped up and prevent self-discharge. Tickle charging is often used for batteries that are used infrequently or left sitting for long periods of time, such as those in standby power supplies or backup generators.

Supercapacitors

Supercapacitors are energy storage devices that can store and release electrical energy very quickly: have a unique structure that consists of **two plates separated by an electrolyte** 8.16. The plates are *coated* with a *porous material* that provides a large surface area for the electrolyte to contact. This large surface area allows the super capacitor to store more charge than a traditional capacitor, and the separation of the plates by the electrolyte allows for the quick discharge and recharge of energy.

Super capacitors are commonly used in applications where a **quick burst of energy is needed**, such as in *electric vehicles, renewable energy systems, and backup power systems.* The parameter that describe a common *supercapacitor* are reported in the following table:

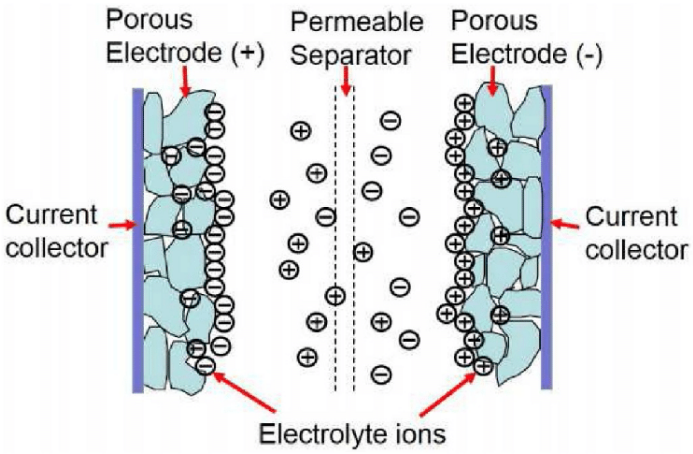


Figure 8.16: Supercapacitors structure

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 8.17: Supercapacitor working parameters

Supercapacitors also have a **higher self-discharge rate** than batteries, which means they will lose their stored energy more quickly when not in use.

In figure 8.18, the first diagram show that the *discharge curve* is linear, while the charging curve depends on many things because, fixing the same harvester, the speed of charging depends on the parameters that conditionate the harvester (*like the solar panel conditions the charge curve based on the sunny days of the year*).

8.3 Measuring energy production

All the methods for power management in energy harvesting assume that the device has fresh information about the **battery charge** and the **actual energy production** from the harvesting source but the *accurate measurement* of these parameters is complex and not required for most devices. To avoid complex and costly solution, a simple method is to exploit *alredy known* parameters to **better approximate** the battery charge and energy production.

Assume we have the information about the *measurement of the battery charge* $E_b(t)$ at time t and the *power consumption* $p_c(t)$ in a timeframe $[t1, t2]$ is known, then the energy E_e produced in the same timeframe is given by:

$$E_e = \left[\int_0^T p_c(t)dt + E_b(t2) - E_b(t1) \right]^+ \tag{8.5}$$

where:

$$[x]^+ = \begin{cases} 0 & \text{if } x \leq 0 \\ x & x > 0 \end{cases} \tag{8.6}$$

The previous formula compute the energy produced by integrating the energy consumption plus the battery charge at time $t2$ (*at the end*) and subtracting the battery charge at the *start time* ($t1$): if the total is positive the production is positive. This model refer to an ideal model with an **ideal energy buffer**, despite also other parameters can be introduce to obtain a more accurate model.

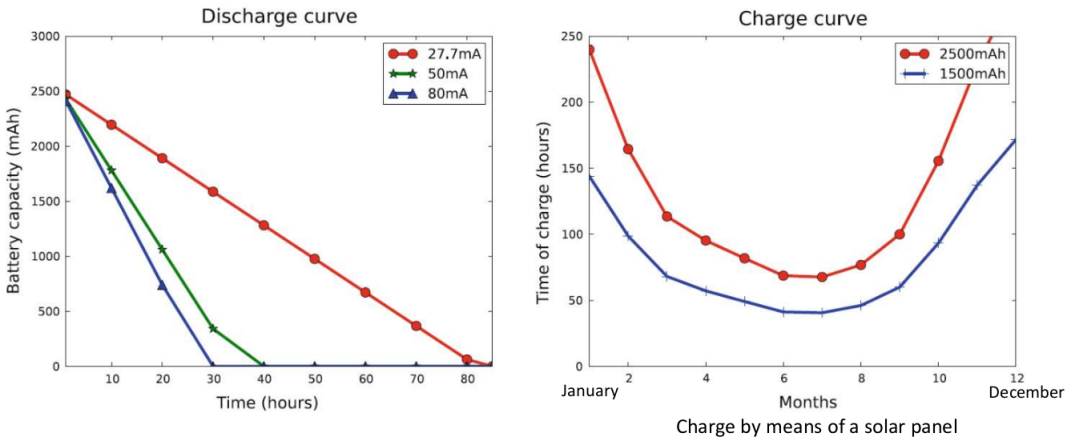


Figure 8.18: Charge and discharge relations

This *analytical* approach have some drawbacks, like:

- errors in the measurement of the battery charge imply errors in the estimation of energy production
- if $[t1, t2]$ is too large it is difficult to know when the energy was available
- the *accurate estimation of the energy consumption* $p_c(t)$ may not be easy

A better way is to **use ad-hoc electronics on board to measure the current flowing out** of the harvesting source and its voltage: the *battery charge* and the *voltage* are **linearly related**, thus the voltage drops down with the charge.

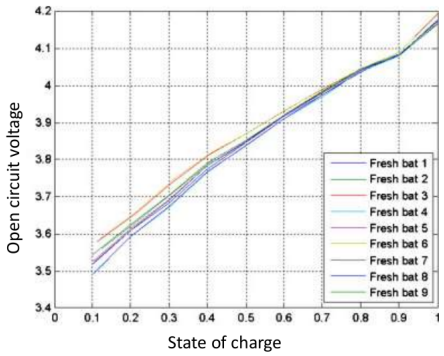


Figure 8.19: Battery charge and voltage linearity

Consider a device that measures voltage by means of d-bits **ADC**, let also define:

- B_{min}: minimum battery charge
- B_{max}: maximum battery charge
- v_{min}: minimum battery *voltage*
- v_{max}: maximum battery *voltage*

The **ADC** takes in input v_{min}, v_{max} and return in output x_{max}, x_{min} computed as:

$$x_{max} = 2^d - 1 \tag{8.7}$$

$$x_{min} = ROUND[\frac{v_{min}}{v_{max}} \times (2^d - 1)] \tag{8.8}$$

Let v be the *current battery voltage*, so the value x read in output from the ADC is:

$$x = ROUND[\frac{v}{v_{max}} \times (2^d - 1)] \tag{8.9}$$

and the corresponding current battery level B is obtained by scaling x respect to the values previously computed:

$$B = B_{min} + \frac{B_{max} - B_{min}}{x_{max} - x_{min}} \times (x - x_{min}) \tag{8.10}$$

Exercise Consider a device that samples the battery output voltage with an analog to digital converter (ADC) at 10 bits. The battery has a maximum charge of $2000mAh$, and its maximum voltage (*when fully charged*) is 10 Volts. When the battery reaches a voltage of 8 Volts the battery charge is $200mAh$ and it becomes insufficient to power the device.
Compute the **battery charge** B when the ADC outputs $920, 830, 1023$.

Solution Summarize the data we already known:

- d : 10 bits
- B_{min} : $200mAh$
- B_{max} : $2000mAh$
- v_{min} : $8V$
- v_{max} : $10V$

Now compute:

- $x_{max} = 2^d - 1 = 1023$
- $x_{min} = ROUND[\frac{v_{min}}{v_{max}} \times (2^d - 1)] = 818$

Hence, when the ADC outputs:

- $x = 920 \rightarrow B = 1096mAh$ (*half the charge*)
- $x = 830 \rightarrow B = 305mAh$
- $x = 1023 \rightarrow B = 2000mAh$ (*full charge*)

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

8.4 Energy neutrality

We need to optmize both the power consumption and the harvesting-storage phase: a solution can be applied by **modulating the load** and thus reducing the amount of work to reduce the overall energy consumption: for sampling sensors, reduce the sampling frequency can have an impact also on the quality of the data produced thus the *range of accepatance* of each component and the grade of working function must be evaluated.
Power consumption varies according to the operational mode of a device: some activities like *sampling, transmitting, receiving* may initially wake the sensor node from sleep, start the processor and radio but also other activities may be executed in background like *routing data packets and network management activities*. In figure 8.20 is shown the load of a device over time.

In figure 8.21 we compare the load versus the supply of solar energy based sensors. The battery level in sensors run differen task from 0 : 00 AM to 12 : 00 PM: sensors are recharged with solar panels and running different duty cycles and thus different load. From the collected data we highlight that the sensors 1 and 7 consume more energy than that available while the sensor 6 and 5 do not exploit all the energy produced.

Having both a lower and higher energy consumption result in *inefficiencies*: for the first case, the sensing activity is not **sustainable in the long term**, for the second, the sensing activity **could be much more work intensive**, performing major energy consumption and allowing high quality sampling. Between the two cases, there are *energy neutral devices* in which the energy consumption is the sustainable due to the fact that the energy at the beginning of a timeframe is the same energy at the end.

A device is defined **energy neutral** if it can keep the desired performance *"forever"*: this also simplify the mantainance operations, without involding battery replacing or load modulation.

The design of those devices involves two main considerations:

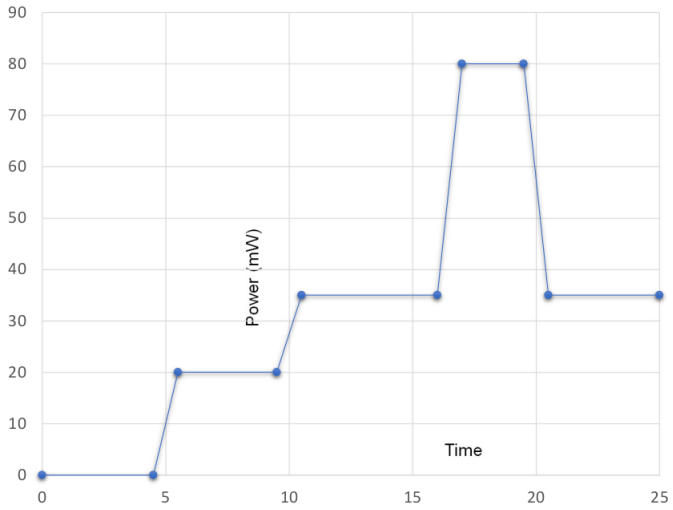


Figure 8.20: Generic load of a device overtime

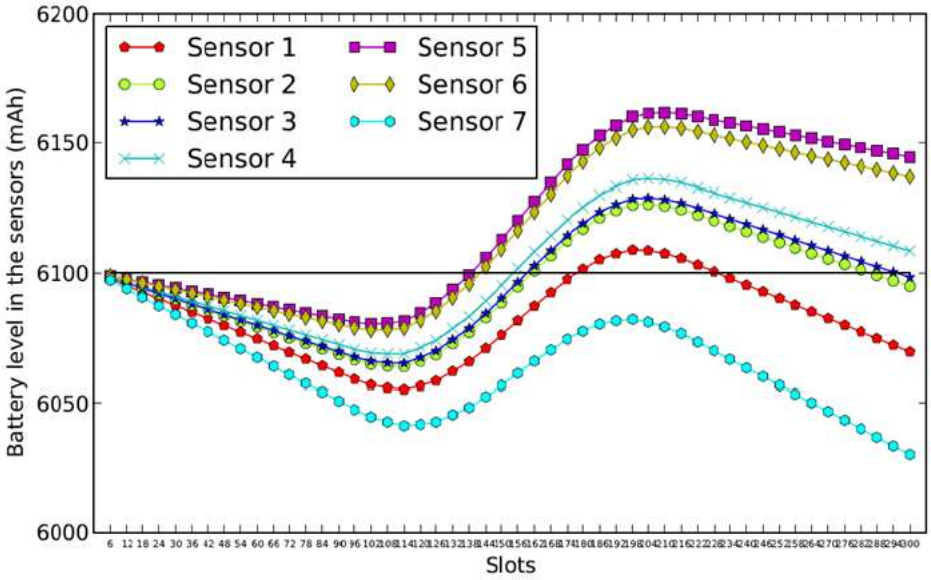


Figure 8.21: Sensors battery level over 24 hour

- **Energy neutral operations**: define how operate in a given timeframe using always less energy than the harvested one.
- **Maximum performance**: compute the maximum performance level tht can be supported in a given harvesting environment. This point can be solved by formalizing the workload behavior as a linear problem and apply operational research problem throgh a solver tool.

Those design involves tuning each of the component involved, including the network components which manages the communication with other dervices: a well-designed systems try to guarantee the energy neutrality not only to a single device but as a fleet of devices and design the server component of the entire architecture respect to the local choices of the devices (*e.g. low duty cycle must be ackoledge by a central server so it expect lower data transmission in case of energy consumption reduction*).

The schema in 8.22 shows the **load** compared to a generic **supply** over a day: the energy production increase during daytime, allowing to store it when the energy production is above the required by the device load.

The **power consumption** of the device is modulated to ensure energy neutrality: this parameter is computed by an *analytical model* as an approximation of the real power consumtpion at a given time. The other three parameters that are used in the analytical model can be measured in an accurate way, allowing to obtain:

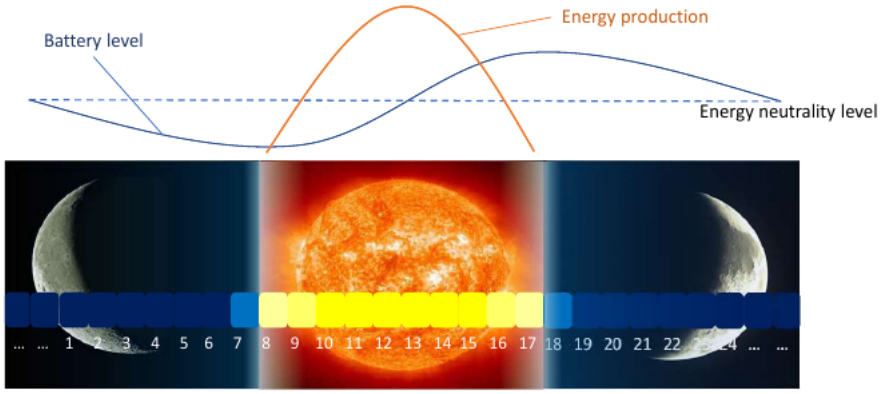


Figure 8.22: Solar energy production compared to battery level consumption over 24 hours

- The **amount of harvested energy** from each source
- The **current charge of the battery**
- The **future energy production**

The *monitoring* of those parameters allows to compare it to the analytical model and introduce optimization overtime, if necessary. The *future energy estimation* can be based on several methods like *weather forecast* despite it can be subject to errors. Usually the forecast method it's based under pessimistic assumption thus the errors are maximazed and the *expected production* is minimized.

Harvest-store-use architecture revisited

The harvest-store-use model use an harvester for future use thus the energy is used later when either there are no harvesting opportunities or the sensor taska require more energy. The general model is pictured here:

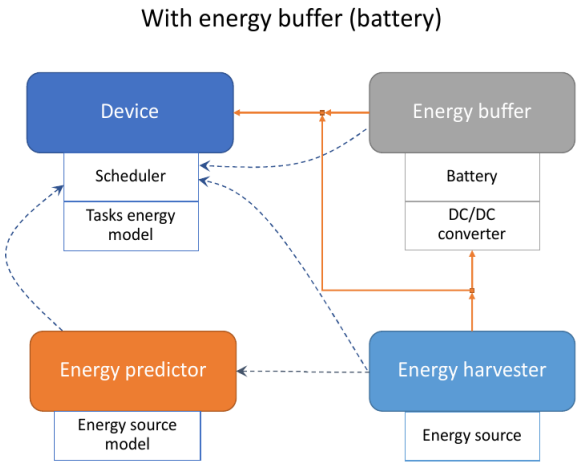


Figure 8.23: Harvest-store-use architecture with energy predictor component

The **energy prediction** components needs a model of the energy source and specific information taken from the *harvester* itself or through *external components* like weather forecast or third-party services. The **scheduler** takes in input the *energy production* from the energy harvester and the *energy consumption* from the energy buffer and take scheduling decision based on the analytical model, update by the *energy predictor* and based on the real data provided by the *energy harvester*.

Here we describe two main models for **energy neutrality** known and used in literature.

8.5 Kansal's model for energy neutrality

This approach is based on underlying assumption about the energy source: it must be **predictable** but **uncontrollable**. In presence of unpredictable source the problem of forecast the production require a more complex model of the source thus guaranteeing performance become difficult. The uncontrollability of the source constraint is given due to the fact that if it would be controllable then the energy production can be activated on demand when necessary.

The approach focuses on take the **current and expected battery charge** into account, dynamically tune the device's performance by *tuning the duty cycle* (*which is the only parameter to be tuned in the Kansal approach*) and thus modify the **load**. This allows to ensure that the device operates below minimum performance levels nor switches OFF before the next recharge cycle.

The model proposed by Kansal is valid under some hypothesis, here presented.

Conditions The energy production, according to Kansal, can be approximated as a **linear source**: this implies that the growth is bounded by *two parallel line* with an angle of ρ_s.

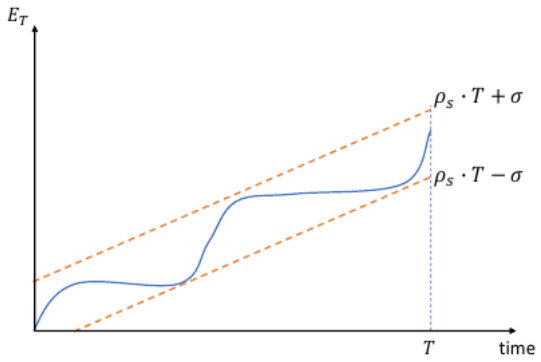


Figure 8.24: Condition on the linearity of energy production

The **amount of energy produced** E_T in an interval $[0, T]$ is:

$$E_T = \int_T P_s(T)dt \tag{8.11}$$

such that:

$$\rho_s \times T - \alpha \leq E_T \leq \rho_s \times T + \alpha \tag{8.12}$$

for some ρ_s, α *real numbers.*
The *linearity of the source* is reasonable, expecially for the sun as energy source.
The second condition is imposed on the **load**: according to Kansal, the load can be **bounded linearly** by finding the correct coefficiencies. The **amount of energy consumed** $L_T$in an interval $[0, T]$ is:

$$L_T = \int_T P_c(t)dt \tag{8.13}$$

such that:

$$0 \leq L_T \leq \rho_c \times T + \delta \tag{8.14}$$

for some ρ_c, δ real numbers.

Kansal theorem The **Kansal theorem** is valid under the previous two assumption of E_T and L_T and if the energy buffer is a real one, thus charaterized by **energy efficiency** η and **leakage** ρ_{leak}. The theorem states that a ***sufficient condition*** for energy neutrality of the system is shown in 8.26.

The first branch of the definition states that the *energy production* grows more than the *consumption and leakage*. This condition regards the **runtime load** and not the overall system during *shutdown* and at *start* operations.

The constrains given by Kansal allows to **avoid oversizing** the energy production solution by providing larger batteries and solar panels.

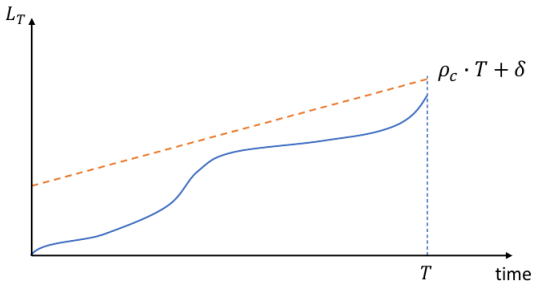


Figure 8.25: Condition on the load

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

- The trend of production (considering the charging efficiency) should exceed the trend of load and 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)

Figure 8.26: **Kansal theorem**: energy neutrality conditions formalization

Utility and duty cycle measurements The main purpose of the Kansal constraints allows also to define what is a **useful duty cycle**: he introduce the concept of the *utility* in the DC that describes the operations that are useful for the device purpose and not avoid performing operation due to insufficient energy.

The proposed model allows to ensure the two conditions already mentioned, here summarized:

1. *Condition 1*: on power production, assume it fits
2. *Condition 2*: on the load, allows to adjust dynamicaly the duty cycle

Kansal introduced a relationship between the *duty cycle* dc and *utility* $u(dc)$ of the application:

$$\begin{cases} u(dc) = 0 & if\ dc < dc_{min} \\ u(dc) = \alpha \cdot dc + \beta & if\ dc_{min} \leq dc \leq dc_{max} \\ u(dc) = u_M & if\ dc > dc_{max} \end{cases}$$

Figure 8.27: dc_{min} and dc_{max} constraints within the utility $u(dc)$ function

To this aim, we also define the **max utility** u_M and **min utility** u_m associated with the **maximum duty cycle** dc_{min} and **minimum duty cycle** dc_{min}. This parameters allow to define a range in which the operations are useful and are not a waste of energy (*e.g. sampling frequency for infrared alarm, not too fast and too slow*), as shown in 8.28.

Example Let define:

- $dc_{max} = 90\%$
- $dc_{min} = 50\%$
- $u_{max} = 100$
- $u_{min} = 10$

Hence the parameters α and β are:

$$\alpha = \frac{u_{max} - u_{min}}{dc_{max} - dc_{min}} = \frac{90}{40} = 2.25 \tag{8.15}$$

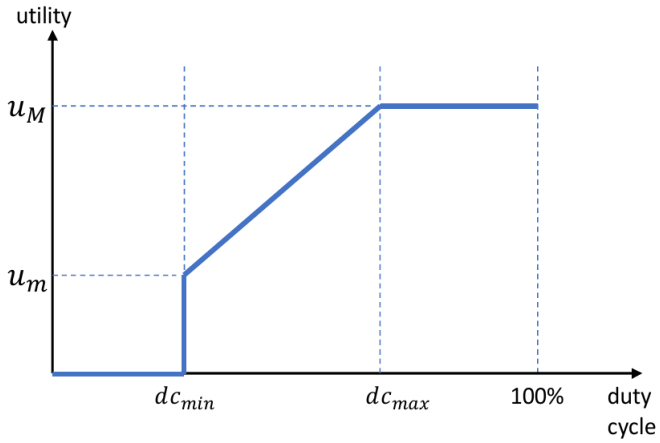


Figure 8.28: Comparison between generic dc and utility function

$$\beta = u_{min} - \alpha \times dc_{min} = -102.5 \tag{8.16}$$

Thus the *utility* of a generic dc can be computed as:

$$u(dc) = 2.25 \times dc - 102.5 \tag{8.17}$$

The relationship between the duty cycle and utility function, for this example, can be seen in 8.29.

$$\begin{cases} u(dc) = 0 & if\ dc < dc_{min} \\ u(dc) = \alpha \cdot dc + \beta & if\ dc_{min} \leq dc \leq dc_{max} \\ u(dc) = u_M & if\ dc > dc_{max} \end{cases}$$

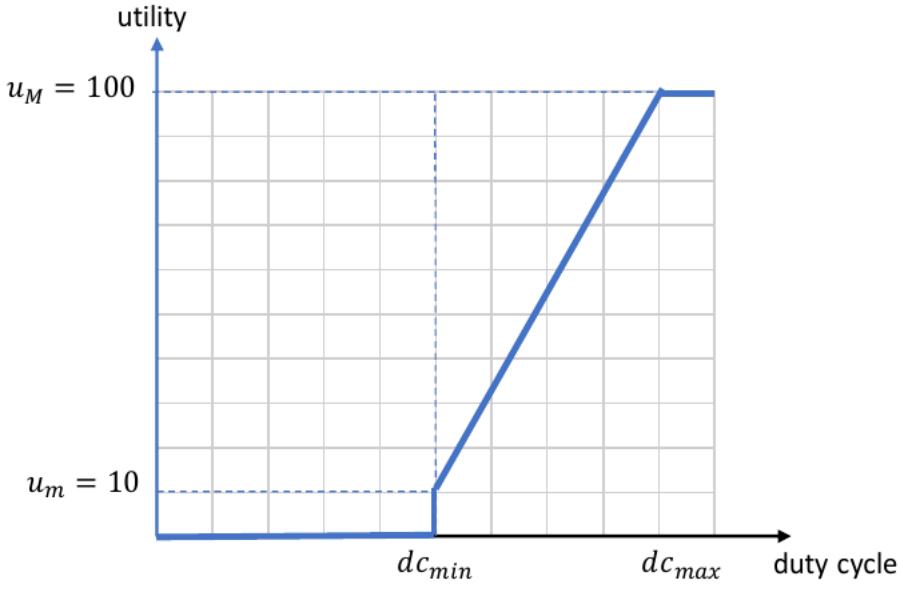


Figure 8.29: Duty cycle/utiltiy with respect to u_M, u_m

Assume that:

- $dc_{max} = 90\%$ corresponds to a consumption of $p_{max} = 5mA$
- $dc_{min} = 50\%$ corresponds to a consumption of $p_{min} = 1mA$

and that the power consumption is **linear with the duty cycle**. Hence:

$$p(dc) = \rho \times dc + \alpha \tag{8.18}$$

where:

$$\rho = \frac{p_{max} - p_{min}}{dc_{max} - dc_{min}} = \frac{4}{40} = 0.1 \tag{8.19}$$

$$\alpha = p_{min} - \rho \times dc_{min} = -4 \tag{8.20}$$

Indirectly, the model express a relation between *energy* and *duty cycle* passing through *utility* measure: the following image shows the linearity of the power consumption respect to the dc.

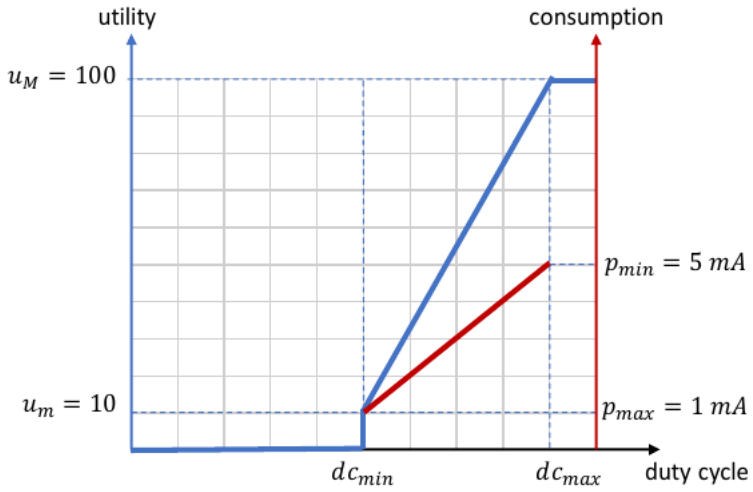


Figure 8.30: Linearity between energy, duty cycle and utility measure

8.5.1 Duty cycle modulation

This model of utility and duty cycle allows for **modulation of the load on the energy harvesting device** despite changing overtime the DC involves make *practical considerations*: the sampling sensor involves using the transducer that produce data points. The sampling work well (*for a theoretical result known as Sampling Theorem*) if the sampling if equally spaced, at regular time. Changing the DC **modify the sampling frequency indirectly**, possible messing up the data points sampled. To avoid this problem is needed to **sample at the same frequency for a given amount of time**: this allows to perform fourier analysis and obtain correctly the measurements.

Kansal allows to modulate the *duty cycle* by introducing the concept of ***slot***: in each slot the sampling frequency is guaranteed for a limited timeframe thus the disruption of the measurement is avoided. The scheduler, based on the previous input parameters, decide the schedule of the slots, trying to maximize the utility of each slot by mantaining the power consumption under the *oblique red line* in 8.29.

The optimization problem formulated by Kansal is now shifted towards the basic work unit of a slot: assume the time is *slotted* and in each time slot is setted **one duty cycle**. Consider also a timeframe of 24 hours so the optimization is daily based: use forecast to estimate the energy production of the entire day.

As previously seen in this chapter, the *power harvested, stored* and *consumed* in a generic slot i follow the assumption that the power **consumed** in idle mode is 0 and that the power **production** is constant in the same slot. Thus, formalizing the consumed/produced energy, we have:

- $p_s(i)$: **power harvested** in slot i
- $p_c(i)$: **power load** in slot i

The following two diagrams showed in **??** consider the energy consumption/production parameters: on the left, if the consumption is more than the production and the device is in *active* state then there is no storage because the harvested energy is used to perform the active duty cycle operations. Differently, if the device is in *idle* state, the harvested energy is stored, following the energy buffer model already seen.

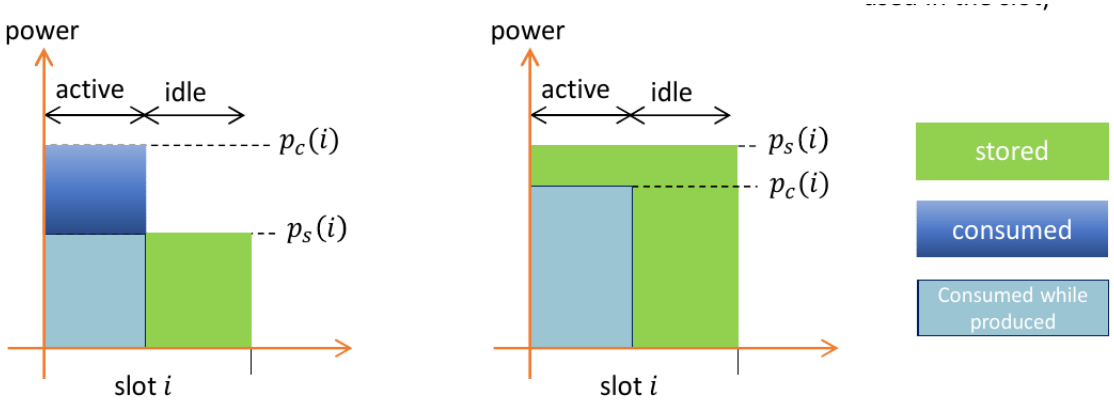


Figure 8.31: Energy storage diagram in *idle* and *active* phases of the device during slot i

The diagram on the right contemplate in case of the energy consumption is less than the energy production: in this case during the *active* mode we're able to store partially the harvested energy while in *idle* mode we're able to save almost all of the entire produced energy (*discarding real model parameters like battery efficiency and storage leakage*).

Kansal's optimization problem formalization

To formalize the optimization problem and solve it, the approach use three main components:

- *Forecast future power prediction* based on the production of the past, considering their relative weight using *EWMA*.
- Use a *polynomial-time* algorithm, optimal for low-powered devices, to solve the optimization problem
- Performs **reoptimization** by implementing dynamic adaptation of the duty cycle if the actual energy production significantly deviates from the expected one

The formalization involves defining those parameters:

- k be the *number of slots* in a day
- $B(i)$ be the battery charge at the beginning of slot i
- $B(k+1)$ be the battery level at the end of slot k (*e.g. end of the day*)

Based on the **expected power production** $\tilde{\rho}_s(i)$ at *each* $i \in [1,k]$ assign a *duty cycle* $dc(i)$ and hence a utility $u(i)$ to each slot such that:

$$B(k+1) \geq B(1) \tag{8.21}$$

so that the system must be *energy neutral*.

The model based on *EWMA - Exponentially Weighted Moving Average* catch well long timeframe forecast, while outlier can disrupt the forecast (*e.g. during cloudy days*). The EWMA for the forecast is made upon assumption that the production in a slot in a day will be similar to that of the day before. To this aim, define:

- $p_s^j(i)$ be the **actual power production** in slot i in day j (*measured*)
- $\tilde{p}_s^j(i)$ be the **estimated power production** in slot i in day j

The estimated power production in the same slot i in the next day $j+1$ is given by:

$$\tilde{p}_s^{j+1}(i) = \alpha \tilde{p}_s^j(i) + (1-\alpha) p_s^j(i) \tag{8.22}$$

where $\alpha < 1$ is a parameter that can be costant or dynamically adjusted based on the estimation errors comparison with actual production.

8.5.2 Kansal's algorithm

Assume that the **actual power production** matches the *estimated power production* and let:

- p_{max} be the power consumption when operating at maximum dc dc_{max}
- p_{min} be the power consumption when operating at minimum dc dc_{min}

Consider the two sets:

- **Sun slots**: $S = \{i \in [1,k] : \tilde{p}_s^j(i) \geq p_{max}\}$ which is the set of slots with ***overproduction***
- **Dark slots**: $D = \{i \in [1,k] : \tilde{p}_s^j(i) < p_{max}\}$ which is the set of slots with ***underproduction***

The first solution, possibly *non-optimum* and *not even admissable* would be:

$$dc_i = dc_{max} \forall i \in S \tag{8.23}$$

$$dc_i = dc_{min} \forall i \in D \tag{8.24}$$

It assign the maximum duty cycle dc_{max} in the sun slots and the minimum duty cycle dc_{min} in the dark slots, thus, we can end in two different scenarios in order to optimize the overall *utility* we have two options:

1. Having a *surplus* of power production at the end of the day, that result in a **non-optimum solution** due to a waste of produced energy
2. Having a *underproduction* of power at the end of the day that result in a **non admissible solution** due to insufficient energy production

The **surplus of production** at the end of the day can be rsolved by distibuting it in some slots: as Kansal defined the problem, we can put the surplus in the dark slots but this will not change the utility of the slot.

```
// p_r is the surplus of energy:  p_r = B(k+1) − B_1 > 0
while p_r > p_max − p_min { // p_r can power up a slot to the max utility
  let i ∈ D be a slot with dc_i = dc_min   // let i be the minimum index
  dc_i = dc_max;
  p_r = p_r − (p_max−p_min);
}
if p_r > 0 {    // p_r is insufficient to maximize the utility of another slot
  let i ∈ D be a slot with dc_i = dc_min // let i be the minimum index
  dc_i = DutyCycle(p_r + p_min);   //increases the DC of the slot as possible
}
```

Figure 8.32: Case 1: *overproduction*

The basic idea is that while the **residual energy** p_r is above the difference of p_{max} and p_{min}, raise the dark slot i to the maximum dc by $dc_i = dc_{max}$. This can lead to the problem of remaining with a residue which is not sufficient to increase to the maximum: in this case (*bottom* `if` *check*) the increase is not able to reach dc_{max} but still the slot i will be increment by a quantity returned by the `DutyCcle(...)` function.

The second case address the **underproduction of energy** by democratically reducing each slot of the same quantity p_u, reducing the duty cycle to dc_{min}: if the underproduction is very high, thus the reduction is too drastic as checked by the condition $p_{max} - p_u/|S| < p_{min}$, the problem is not ammisible. In case of ammisibility, it bring drown the same quantity for each slot.

This model is valid under condition that the **enegy production is the same of the expected one**: if not true, the scheduler intervene at the start of each slot, adapting based on the updated threshold allowing to take advtange of an overproduction of making sure the system remains *energy neutral* in case of underproduction.

```
// p_u is negative, it's the underproduction: p_u = B(k+1) − B_1 < 0
if p_max − p_u/|S| < p_min
        return(-1) // no admissible solution, too much underproduction

else for each i ∈ S    //i.e for the slots with dc_i = dc_max
  //decreases the DC of the sun slot of the same quantity
  dc_i = DutyCycle(p_max − p_u/|S|);
```

Figure 8.33: Case 2: *underproduction*

Highlights and drawbacks The Kansal's algorithm is able to reaches the optimum and it's suitable even on Arduino: the memory usage can be lowered by expanding the slots and reducing the array of previous day weights.

The main drawback of this approach is the underlying assumption that the duty cycle **scale linear** hence the sampling activity of a sensor will, in general, have irregular sampling frequencies, which may not be the best thing for some applications. It's also difficult to model complex behavior, like choosing between alternative transducer to achieve sampling task or use different processing algorithms because the **duty cycle modulation** is able to adapt the load to the *real power production*.

8.6 Task-based model for energy neutrality

All of the steps implemented by an IoT device like *sensing, storing, processing and trasmitting* may have different alternative implementations with different duty cycles: what change is not the functional behavior but the non-functional one like *througtput, latency, etc.* Each different implementation can correspond to different energy profile: each implementation is called **task** and when the device switch the implementation, this must be notified to the server to adapt the server expectation (*for example during transmit*), maybe notifying also other devices in the same network.

As an example, an application can choose different transducers (*sampling sensors*), each with a different energy consumption and performance by *disabling data processing* and only store data when the battery level is too low, scale up or down the sampling frequency, use more trasnducers to increase the utility level or communicates more/less frequently and reliably.

8.6.1 Task-based model

This approach focuses on **modulating the load of the device** by using ***tasks***: it's achieved by *scheduling* tasks in execution, advertising the server which task is running to adapt the behavior. Based on the chosen type, for example, the application process all data and transmit only the result so the server may just store the received data or the application can transmits all the sensed data so the server has to process and then store the data. The number of tasks are limited by the device constraints and scheduling but the overall *system functionality* does not change, what is changed is only the implementation that have effect on the device's behavior.

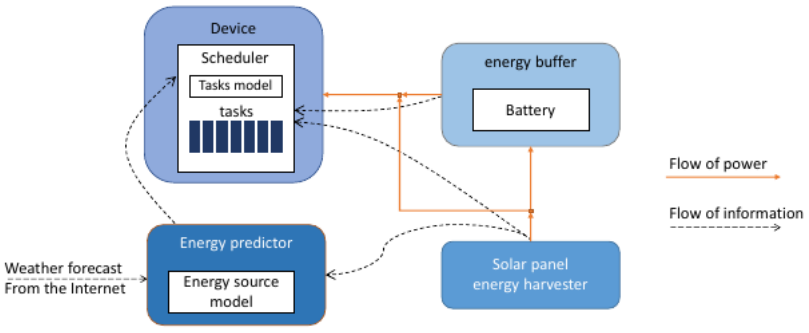


Figure 8.34: Schema of task-based model scenario

The **task model** is discrete, definite task by task, differently from Kansal which exploited the linear relationship between utility, duty cycle and power consumption. A single task is charaterized by a

power consumption per unit of time and the **utility gained** when it is executed. As in Kansal, switching from a task to another have its own overhead: as in sampling, the frequency of sampling must be guaranteed in the same slot unit. Here for each task, considering k slots of time (*e.g. 24 slots a day, each of 1 hour*), we have a straight association with energy cost and utility:

Tasks and their properties

Alternative tasks impl.	Energy cost	utility
T_0	c_0	u_0
T_1	c_1	u_1
T_2	c_2	u_2
...	...	...

n Tasks

Figure 8.35: Different tasks implementation with cost and utility value

A **scheduler** assigns one task per slot, fixing the schedule once per day (*e.g. at midnight*), obtaining the following table:

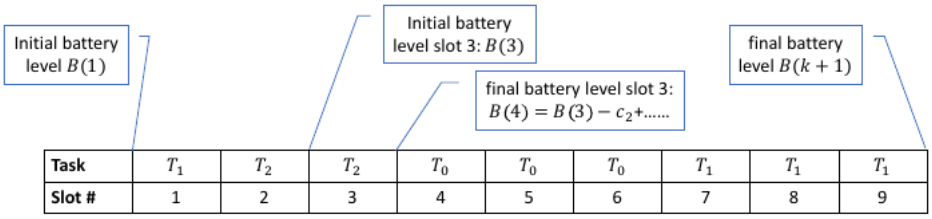


Figure 8.36: Schedule determined by the scheduler

The underlying assumption is that the power consumption during the same slot is constant, despite the effective consumption can be difficult to predict for the *forecaster*: as an example, a *weather forecast* allows to estimate the expected energy production in slot i , indicated by $p_s(i)$ by evaluating the task assigned to each slot.

Optimization problem formalization

The following relation compute the **expected power produced** and **not consumed** in slot i:

$$p_s^+(i) = [p_s(i) - p_c(i)]^+ \tag{8.25}$$

This quantity correspond to the *recharges of the battery*, while the following one express the **expected power consumed** from the battery in slot i:

$$p_c^-(i) = [p_c(i) - p_s(i)]^+ \tag{8.26}$$

It's noticeable to remember that that $p_c(i)$ express the power consumption in the slot i, while $p_s(i)$ express the power stored in the same slot. Overall, this formalization allows to express the **battery level at the end of the slot** i as:

$$B(i+1) = min\{B_{max}, B(i) + \eta \times p_s^+(i) - p_c^-(i)\} \tag{8.27}$$

where:

- η is the *charging efficiency of the battery*
- the *min* operator allows to bound the battery level, limiting it to B_{max}
- $\eta \times p_s^+(i)$ express the **battery charge in the slot** i
- $p_c^-(i)$ express the power of the **battery consumed** in the slot i

The optimization problem thus can be formalized as:

$$max \sum_{i=1}^{k} \sum_{j=0}^{n} x_{i,j} \times u_j \tag{8.28}$$

which is the **utility of the task assigned to the slots** (*i is the slot index, j the task index*). This formalization is valid under the following linear constraints:

1. $\sum_{i=1}^{k} x_{i,j} = 1$: only one task j for each slot i
2. $B(1) \leq B(k+1)$: define the admissibility of the solution, respecting the energy neutrality problem formalization
3. $B_{min} \leq B(i) \forall i \in [1,k]$: in a slot the battery charge cannot be below B_{min}, otherwise the device will stop working
4. $B(i+1) = min\{B_{max}, B(i) + \eta \times p_s^+(i) - p_c^-(i)\}, \forall i \in [1,k]$: gives the battery level at the end of the slot as a function of the battery level at the beginning, the scheduled task and the energy produced by the panel

The presented problem is *NP-Hard* but can be reduced to a version of knapsack. However exists a *pseudo-polynomial* solution based on dynamic programming that by providing realistic restriction can be run even on low-power devices.

8.6.2 Dynamic Programming formalization

The *pseudo-polynomial* solution can be described by defining the **state of the system** as a pair of integer values (i, b), and associate to each pair a subproblem defined as the optimal energy balanced schedule of the n **tasks** into slots in the range $[i, k]$, starting with an energy level of b.

The state parameters (i, b) indicates:

- i is the slot up to which the system is optimized
- b is the corresponding battery level at the beginning of that slot

The **optimal utility** $opt(i, b)$ can be found using a backward recursive Bellman's equation:

$$opt(k, b) = max_{j=1..n}\{u_j : b + \eta \times p_s^+(i) - p_c^-(i) \geq B(1)\} \tag{8.29}$$

where k is the last slot and $opt(k, b)$ is the *base of induction*. The *recursive step* is expressed as:

$$opt(i, b) = max_{j=1..n}\{u_j + opt(i+1, B^j(i+1) : B^j(i+1) \geq B_{min})\} \tag{8.30}$$

where $B^j(i+1)$ is the **residual battery at the end of slot** i if the task T_j is assigned to that slot. We also assume that the charging efficiency is $\eta = 1$.

The algorithm is *pseudo polynomial* because, starting from slot one, we don't know the initial battery level so we must test all the values of the initial battery level. Using an *Analog to Digital Converter (ADC)*, the initial battery level are discrete (*limited by a maximum and a minimum setted by the specific ADC*) thus the test are limited to a set of value. Arduino works with 10 bit, and excluding the battery level $< B_{min}$ remains only about 300 battery level values thus the problem is tractable. Posing also limits on the number of slots k, allows to obtain a *time complexity* of $O(k \times battery_{level})$: the solution obtainable is sub-optimal due to an approximation of initial battery level. The complexity can be improved disregarding the number of task k (*that is a small number*), obtaining $O(battery_{level})$.

8.6.3 Execution times[2]

Experimental evidences shows that the complexity remains in the order of $O(k \times Battery_{levels})$ but we can choose the number of slots k and the battery level to keep the *feasibility* for the target platform. The following section compare the execution time of C code of *Arduino Uno, Raspberry PI* and a PC linux with a *battery max charge* $B_{max} = 2000mAh$, *operative range* of $4.2V - 3.4V$, an ADC Of 10 bits and *battery levels* ranging from 828 and 1024.

[2]The following section contains a deeper explaination, summarized from the original source "A Dynamic Programming Algorithm for High-Level Task Scheduling in Energy Harvesting IoT"

In order to achieve a sustainable operaton of IoT platform considered, the battery level should always be within the range $[B_{min}, B_{max}]$. Additionally the **starting battery level** is fixed at $B_1 = B_{min} + ([B_{max} - B_{min}]/2)$. The following simulations are generated on a dataset of $A = 50$ applications, eah one composed of a set of $n \in [6, 10]$ task τ. Remind that each t_i performs a functionality with a certain degree of quality, thus, t_i is defined by the tuple $t_i = (q_i, c_i)$, where q_i is its quality ($q_i \in [1, 100]$) and c_i is its energy cost.
The *dynamic programming* algorithm used is not reported because it's outiside the scope of the following notes.

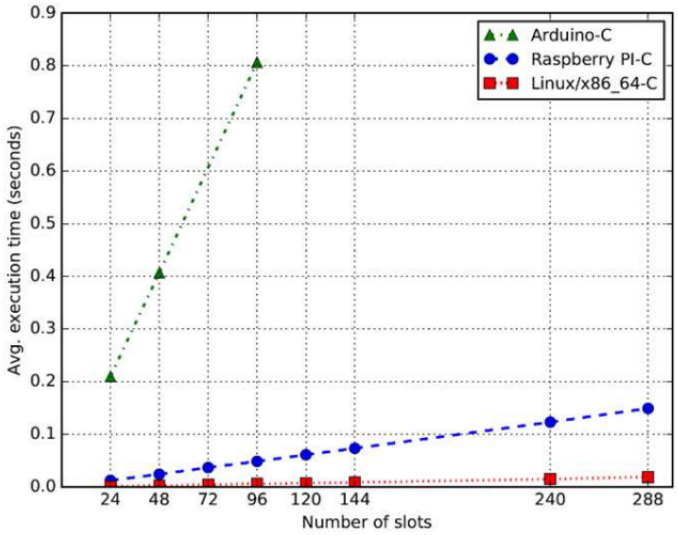


Figure 8.37: Average execution time of the optimization algorithm on the three platforms

The data shows that even for the maximum execution time, the overload due to the execution of the algorithm is very small: in the worst case, given by *Arduino* with 96 slots or 4 slots per hour of 15 minuts each, the execution time is $0.8s$; this means a negligibe overload of 0.08% with respect to the duration of the slot. The increase of the number of slots result into a larger execution times but contrasts with a *growing quality (as shown in the next picture)*, which pose a tradeoff between execution time and quality.

The following diagram show the *average quality* delivered for the simulation of A applications assuming *Raspberry PI* as target platform, working in all months of the year with different number of slosts per day $k \in \{24, 28, 72, 96, 120, 144, 240, 288\}$ with values ranging between $[70, 100]$.

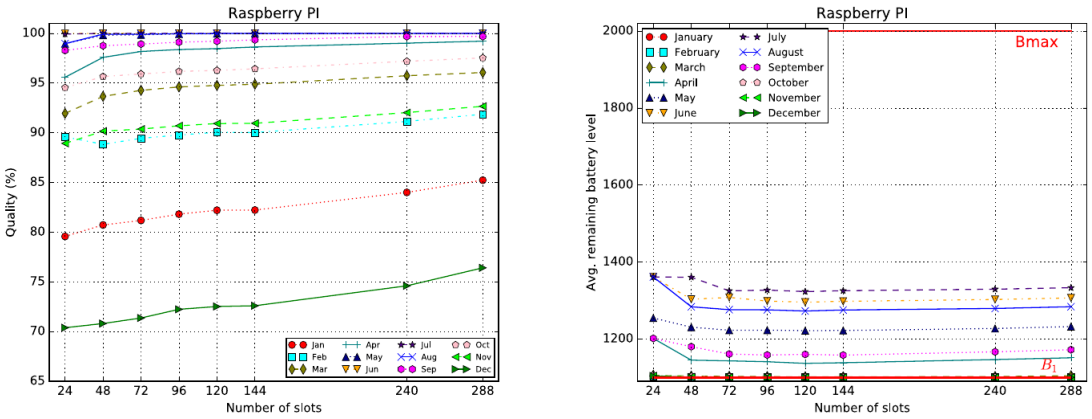


Figure 8.38: Simulation results of an RPI platform: average quality for a different number of slots k (on the left) and the corresponding average remaining battery level (on the right).

Regardless the number of slots, **the larger the energy production, the higher the quality**: June and July are the months with the largest production with the highest quality, while December have the lowest production, with minor quality provided.
Quality grows with the number of slsts up to converge into the overall maximum quality with the larger

value of k, regardless the solar production: in general a **larger number of slots result into more assignments of tasks-to-slots**, where the task assigned execute for less time and the production *per slot* will be less with regard to a larger number of slots. Thus, increasing the number of slots imply a reduction of the same proportion of the energy production per slot and the cost of the execution of the task, which depends on the duration of the slot.
Without going deeper into the algorithm, the energy neutrality condition is checked at the last slot ($B_{k+1} \geq B_1$), so in each intermediate assignment of task-to-slot the algorithm may select a *not energy-neutral application* with regard to the individual slot and with higher cost, which result into a larger *average quality*. The data on the right 8.38 shows the *average remaining battery level*, showing that is lower in the months of lower production, which measn that most of the budget is destined to the application execution. The residual battery tends to grow slightly with k: the tight deviation between B_1 and the average residul battery in December indicates how close is the solution to the *limit of energy neutrality* in this case.

Chapter 9

Wireless Sensor Networks

A **wireless sensor network (WSN)** is a collection of small, autonomous devices called sensors that are wirelessly interconnected to monitor physical or environmental conditions. These sensors are equipped with various types of sensors such as temperature, humidity, light, motion, and sound sensors, among others, to gather data from their surroundings.
The sensors in a WSN communicate with each other using wireless communication protocols such as Wi-Fi, Bluetooth, or Zigbee. They form a network where *data can be transmitted from one sensor to another* and eventually reach a central base station or **sink node**. This base station acts as a gateway to the external world and is responsible for processing and analyzing the collected data.

The sink nodes act as a output gateway to the outside networks: based on specific sensor purposes, the nodes sample the environmental parameters, producing a **stream of data** that can be pre-processed locally and then forwarded to the *sink* node.

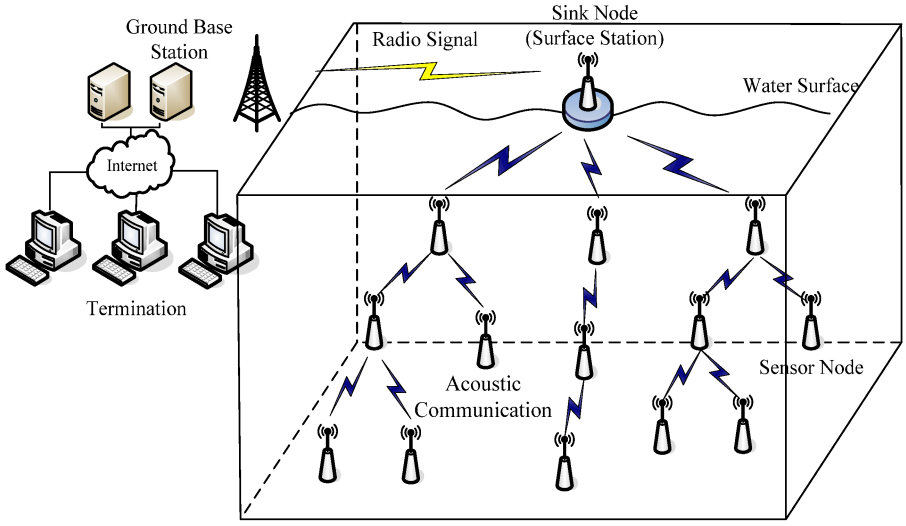


Figure 9.1: Overall structure of WSN

The sink node can also be temporary unavailable thus the network must operates autonomously by pre-processing and storing sensed data (*by logging partially of fully the data stream produced, considering also simple aggregation methods*). To optmize the overall mechanism, **data aggregation** can be performed by the all the nodes by coordinating themself to avoid energy consumption and optimize memory storage among the network's nodes.
The deployment of this type of networks is easy due to an *absence of cabling* and the feature offered by the devies to **self-configure themself** without manual intervention.
The main differences respect to *ad-hoc networks* (*a type of wireless decentralized network*) are that in *WSN* there can be an higher number of nodes and sensors are *strongly constrained* in power, computational capacities and memory. Due to the higher number of nodes, sensors networks are denser and are prone to failures.
This kind of network poses also the problem of routing data from internal node to the sink and then outside the network, considering the device's constraints in term of resources.

9.0.1 Data centric vs node centric

The main problem is to obtain information to route information to the specific devices and services. The association between the address of a device and its physical location must follow an approach that allows to **derive one from another**, allowing to simplify the routing process.

We can distinguish two main approches for network organization and communication between nodes:

- **Node centric**: the communication is mostly based on node addresses or identifier rather than the content it store or can provide to other nodes
- **Data centric**: the communication and node-to-node interaction is based on the node content. This behavior involves using a *query* or *subscription* communication model that involves identify a set of attributes of interest and notify the other nodes when a certain type of data to which some node expressed interet is produced/received.

The second paradigm, considering the network perspective, involves the source node that transmit data of interest for the destination node that expressed interest by broadcasting a query: some of the data will travel along the same path of another transmitted data. These data will travel independetly along the same path: to optmize the transmission it's possible to perform **data aggregation** to efficient the transmission (*e.g. in image this happen in 4 different points*).

For the example pictured above, we are interested in stream of data produced by nodes that respect the condition *temperature* < 5: the node 5 perform aggregation because is closer to node 2 and 4, alongside node 8 perform eggregation for the data received first from node 1 and 2 and then from the node 4. (*We explicitly mixed up temperature with node identifier, refer the image to get the overall point of the schema*).

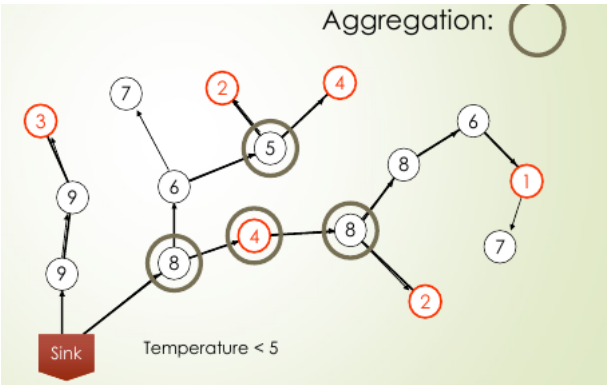


Figure 9.2: Data aggregation scenario

The specific requested query can be based on the **location** and not only based on a data condition, unrelated to the location: even in this case IP addresses and network adresses does not solve entirely the problem. The data-oriented communication process can also allows to identify different area based on common condition that implicitly produced data respect.

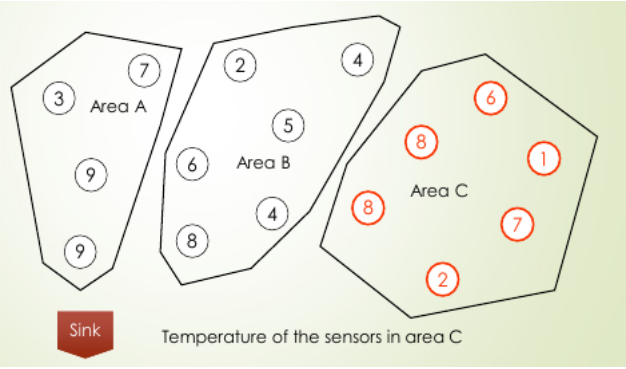


Figure 9.3: Location awareness sensors

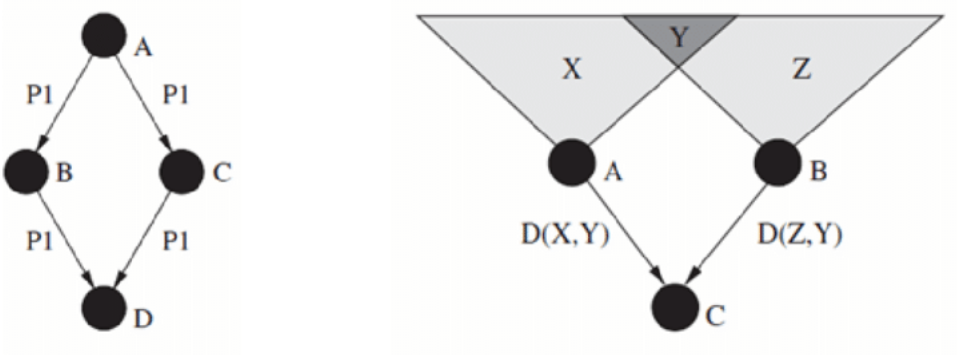
Differently from internet, the underlying devices are not full-resources devices and thus does not

support a layer only for discovering data based on keyword, for example (*as internet decouples addressing using DNS from the searching based on keyword using search engines*).

Implosion and overlap issues

The data-centric mechanism supported by the *query* communciation model, alongside with an overall absence of reliability mechanism for data transmission can lead to an *overflow* of packets towards network's nodes. Two main problem can arise from the presented scenario:

- **Implosion**: caused by the *flooding-based dissemination* of the data among nodes. In the figure, node A tart by flooding its data to all of its neighbors ant two copies of data eventually arrive at node D. This implies wasting resources for additional send/receive operation by the nodes.
- **Overlap**: two sensors cover an overlapping geographic region y so the sensors flood their data to neighbor's node but node C receives two copies of the data marked $D(node, Y)$. This scenario requires suitable *data identification and aggregation algorithms.*



(a) The implosion problem both of sensors B and C broadcasted event P1to D

(b) The overlap problem region Y sensed by both sensors A and B

Figure 9.4: Implosion and overlap scenarios

The implosion problem can be solved by applying some form of internal or network identification.

The overlap problem regards the *range of action of a sensor*: despite been placed in a fixed point, it also sense noise and accessory part, commonly shared by two different sensors. This poses the problem of differentiate between two different event that activate a sensor or a single event that is in the overlap area of two different sensor.

9.1 Directed Diffusion

Direct Diffusion is a *datacentric* protocol for WSN: basically it specify how organize the network, simplifying the routing operations. The data genered by sensors are named by **attribute-value pairs** so nodes requests data by sending interests for named data.
Allows to instruct the device on the fly by send a specific parametric request to the sink node, allowing to specialize or generalize a given **query**. The specific query can be posed as human-like questions: the sink broadcast (*or disseminate*) a packet to the entire network, specifying the parameters for all the device. The protocol can be identified in 4 steps:

1. *Interest-Based Communication*: the communication is based on **interests** expressed by sink nodes. Sink nodes indicate their interest in specific types of data or attributes by issuing **interest queries** to the network. These queries specify the data they want to receive.
2. *Gradient-Based Data Propagation*: Once a sensor node receives an interest query, it begins to **propagate** the data towards the sink node using a **gradient-based approach**. Each sensor node maintains a *local gradient* or preference for a specific data type. The **gradient** represents the desirability of the data and *guides the data propagation process (see later).*

3. *Data Collection and Reinforcement*: As the data flows towards the sink node, intermediate sensor nodes **collect and aggregate the data**. They use the information to refine their local gradients, reinforcing paths that are more efficient in delivering desired data types.
4. *Feedback-Based Adaptation*: Directed Diffusion incorporates *feedback mechanisms* to adapt the data *dissemination process*. Sink nodes provide feedback to sensor nodes by reinforcing or suppressing data flows based on their needs and quality requirements. This feedback helps optimize the routing paths and enhances energy efficiency in the network.

Interest query

Interest queries are periodically generated by the sink: usually the first query of a given type is **exploratory** while the next broadcast are **refreshes** of the interest. This refresh is necessary because dissemination of interest is not realiable. Nodes that receives an interest query may forward the interest selectively to a subset of neighbors.

Nodes usually caches the received interest query:

- Interests that differ only for *sampling rate* are aggregated
- Interests inc ache expire when the duration time expires

Each interest in cache has a **gradient** which expresses:

- a **direction**: thhe node from which the interesrt was received, used to route back to the sink node
- a **data rate**

The same interest may be received from different nodes hence relevant data may be sent along several gradients, following multiple paths. Here an example of a simple animal tracking task (*or interest query*):

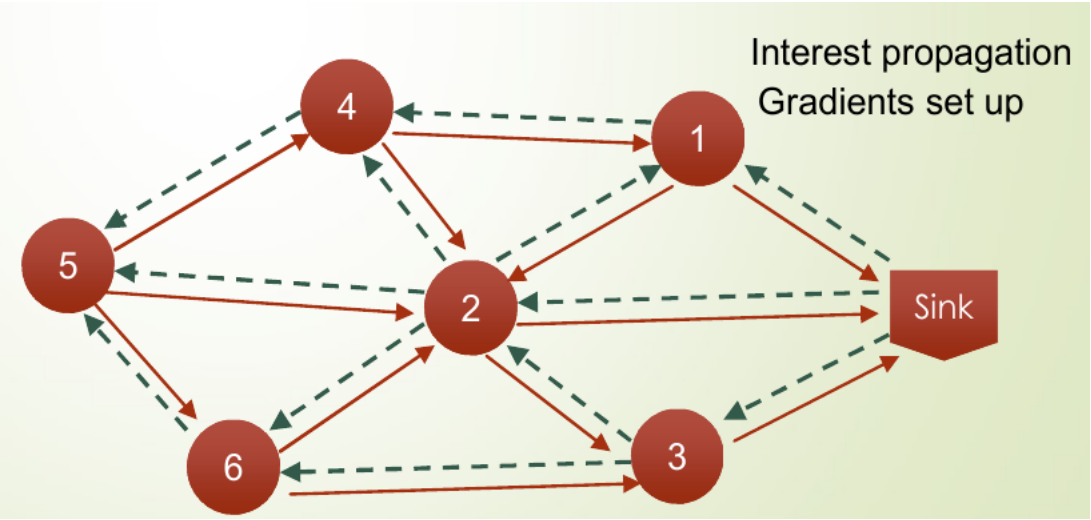


Figure 9.5: Interest propagation process

```
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
```

The `interval` field specify the sampling rate and the rate at which the information must be transmitted. This allows to determine the duty cycle of the device but also the transmission rate between the devices alongisde the path to the sink for transmission purpose.

State machine

When a sensor detects an *event* amtching with an interest in cache, it start sampling the event at the large *sampling rate* of the corresponding *gradients*. Then it sends sampled events through the gradients associated to the given interest in cache: these gradients correspond to the *neighbors interest in the event*, so dat through a gradient flow with the *rate* of that gradient.

Sampling does not stop **routing functionality**: this is possible because the state machine is implemented on *interrupt* thus to have a device act as a router and sensor by giving the possibility to manage interrupts[1] (*as in TinyOS*).

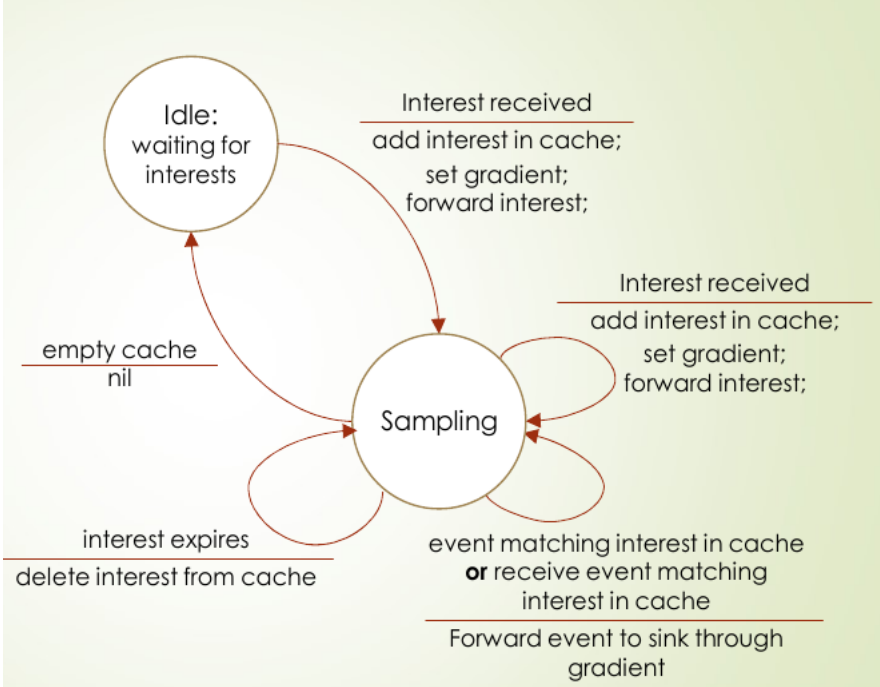


Figure 9.6: Direct Diffusion simplified state machine: the *forward interest* is directed to node's neighbors

The data propagation mechanism implicitly ensures reliability by duplicating the data for each path indicated by the gradient but this kind of reliability is not controllable, as shown here:

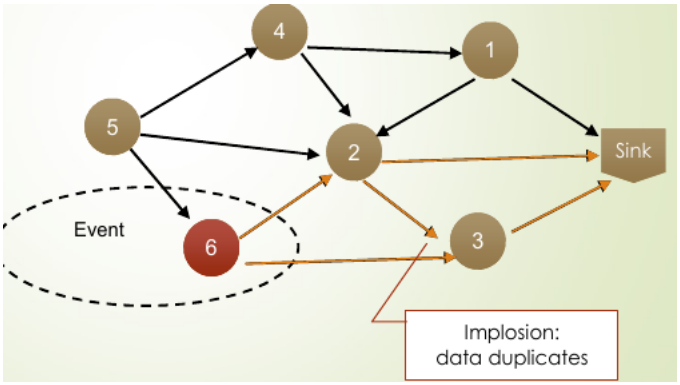


Figure 9.7: Implosion scenario

Reinforcement

The reinforcement process allows to enhance the quality of received data. Suppose to have a sink hat receives data that match a certain interest query from a given sensor n. The sink can *reinforce* n by incrementing the sampling rate along a given path, making the data propagation along a specific path

[1] This lead to more complex problem on a long chain of transmission that can result, among others, in the problem of managing the delay in a context where the duty cycle of the devices are not synchronized

thus making sure that only that specific route can survive among others directed to the node n. This is possible because on one hand, we preserve a single path, refining it's data rate, and on other hand we let other devices that previously received the *interest query* to discard the query as indicated by the query's *TTL* (*or timeout*).
The reinforcement behavior is usually subsequent to the first *exploratory request*, correlated to *refresh requests*. Here are pictured an example of *reinfrocement process* result:

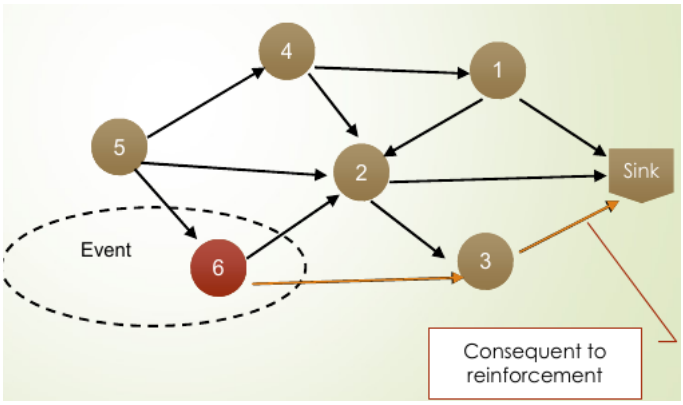


Figure 9.8: Data propagation exploiting delivery strongest gradients

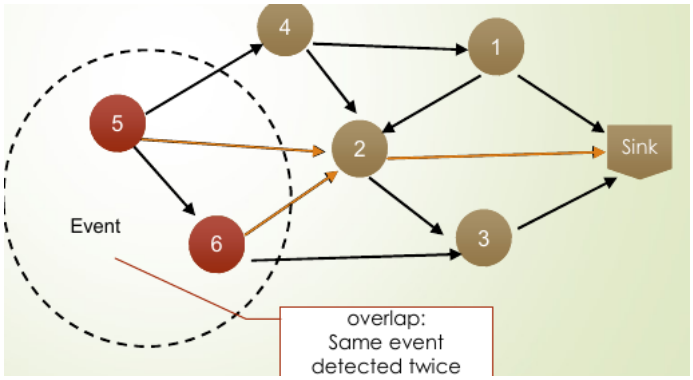


Figure 9.9: Data propagation with multiple sources

9.1.1 Scalability, pros and cons

Direct Diffusion protocol it's very scalable because the data to be stored are mainly the **gradient data structure** that grows with the number of node in the network. Surely, hundreds of devices cannot be in the same transmission range, it's optimal to have a low density number of devices on the same range.
It's suitable for application tha sense data but not require complex data aggregation or preprocessing. For complex event (*e.g. explosion*), the charaterization of the event must be done by *multiple sensors*, not only one. The data processing to identify that a given event happened can also be done outside the network, by transmitting the data and process it or it can be done in network by increase the **point-to-point exchange** of data between the internal nodes. Remember that transmit 1 bit correspond to process 1k of data: the ideal would be to push the computing power more locally to the devices, as the *fog and edge computing* mechanism try to deliver for cloud and data center solutions.

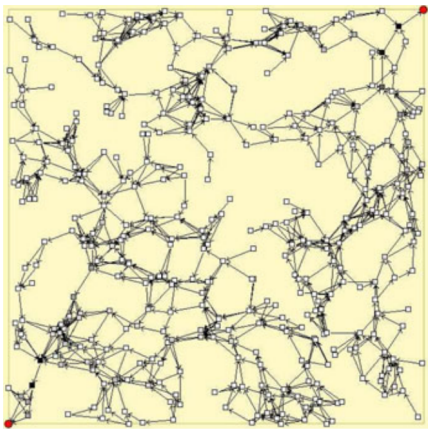
Directed Acyclic Graph Scalability

We mentioned before that a *tree-based network organization* allows to enhance scalability performances. Differently, for this scenario consider to have a network composed as a grid of nodes with two different sink, respectively located as *top-right* and *bottom-left* nodes, as pictured in 9.10b.

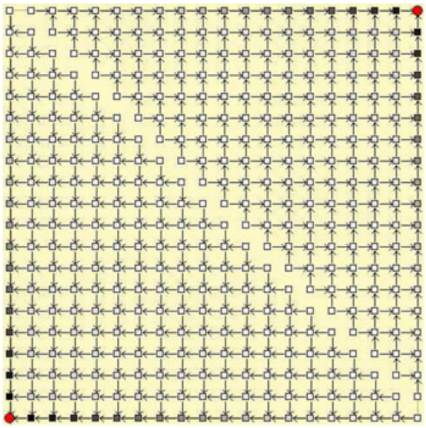
The nodes alongside the matrix border must carry out the burden of forwarding the data to the respective sink, also consume more energy respect to periferic nodes: to solve the problem we can introduce more sink nodes but overall the nodes in the range of a sink will still have an high burden of transmitting

data to this nodes. This special type of network topoology does not allows to process in a balanced way the data inside the network itself.

Even considering an arbitrary connected network 9.10a with two sink nodes (*in the same position as before*), as pictured below, the nodes closer to the sinks still represent a *bottleneck* for the overall network, consuming more in term of energy and managing the burden load of the network.



(a) Arbitrary connected network



(b) Grid hyper-connected network

In the previous image, darker nodes denotes larger power consumption.

The experimental results presented in "Coordinated and controlled mobility of multiple sinks for maximizing the lifetime of wireless sensor networks" for **data colelction strategies** shows that a *grid network deployment* with five sink nodes, one at the center and four at the corners, can ease the load on the nodes closer to the sink nodes.

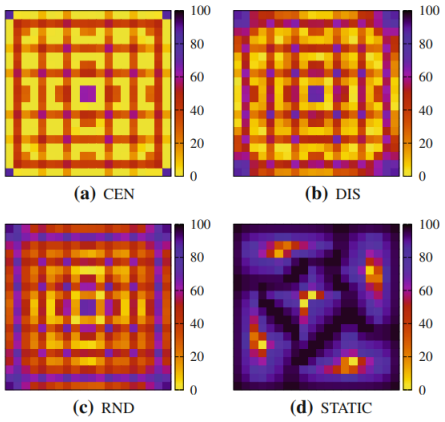


Fig. 2 Residual energy at lifetime for CEN (**a**), DIS (**b**), RND (**c**), and STATIC (**d**), 5 sinks in a 8 × 8 grid

Drawbacks When considering the event sampling of an entire WSN we can think also for *complex events* that is detected only by an high number of sensors in a very sparse geographic area.

Application:	Input bandwidth	Computation demand	output bandwidth	compression factor
Image (e.g. tracking)	80 Kbps	1 GOPS	0,16 Kpbs	500 x
voice/sound (e.g. speech control)	256 Kbps	100 MOPS	0,02 Kbps	12800 x
Inertial sensors	2 Kbps	10 MOPS	0,02 Kbps	100 x
Biometrics	16 Kbps	150 MOPS	0,08 Kbps	200 x

Figure 9.11: Tasks generic parameters

The task described in 9.11 implies the necessity of performing data pre-processing: if we work with images, the *input bandwidth* it's much higher respect to output because the pre-processing allows resize

the information that can be extracted from a image sensing activity. Overall, the differences between input and output in the table would suggest when can be computationally advantageous to pre-process data directly on the sensor itself.

9.2 Greedy Perimeter Stateless Routing (GPSR)

Considering Dijskstra and Distance Vector both have problem as other centralized routing algorithms: the first have the problem of matrix representation which represent an heavy storage burden, the second have problem of knowing the address of each consecutive node to be able to route the packet on the right path.

GPSR[2] it's an alternative to Direct Diffusion and similar protocols that allows to route packets node-to-node without mantaining a global state and having a low overhead. This is true under some assumptions:

- Nodes are located in a bi-dimensional space
- Nodes knows their location and the location of their neighbors
- Source node knows the destination node and its position

GRPS exploit *GPS (Global Positioning System)*, using geographical position to address individual nodes: you can communicate only with physical neighbors and the naming identification it's performed by a given geographical order. GPSR enable **arbitrary point-to-point communication** which Direct Diffusion does not support.

GPSR reduce also the burden on nodes: simplify the routing protocol respect, for example, to link protocols already mentioned, without having the wide knowledge of the network to forward data. The protocol is **strongly localized**, operating only on local knowledge thus reducing the overall storage of information to operate.

9.2.1 Protocol overview

GPSR work using two type of mode:

- **Greedy forwarding**: forward the packet to one of your neighbor so the neighbor pick up the next node closer to the destination.
- **Perimeter forwarding**: support the first mechanism to operate. When there is no point to progress further toward the destination D thus greedy mode fails and the protocol switch to perimeter mode. This mode allows to identify the *perimeter of the **void** region* by traversing the perimeter along, circumnavigating the perimeter to discover nodes connected to the network, allowing to advance towards D.

For the *Greedy Forwarding* mode, consider a packet with destination D (*as shown in 9.12*): the forwarding node x selects as next hop a neighbor y such that:

- y is closer to D than x (*guarantee optimality*)
- among neighbors, y is the closest to the destination D (*guarantee to advance distance to D, avoiding local optimum*). The greedy forwarding fails when falls in a region where there isn't a sensor to forward or a **void**, in an empty area, so this mode alone does not provide reliability.

Now to hunderstand how the perimeter mode works, identifying the *void region* broder, consider the scenario pictured in 9.12: here the node x switch to the *perimeter mode* to route to node w. Makes sense to keep in **perimeter mode** until we are in node that in greedy mode is capable of making progress towards D respect to the previous node. So we switch to *greedy mode* when the node x have'nt any neighbors closer to D more than the selected node. During the perimeter mode the node x identify the nodes that are on the border of the void region in which the destination node D is founded, forwarding the packet to them to circumnavigate the region.

Consider a different scenario, pictured in 9.13: the source node is x and the destination is still node D. We need to forward on the path:

[2] *For further considerations refer to GPSR: Greedy Perimeter Stateless Routing for Wireless Networks*

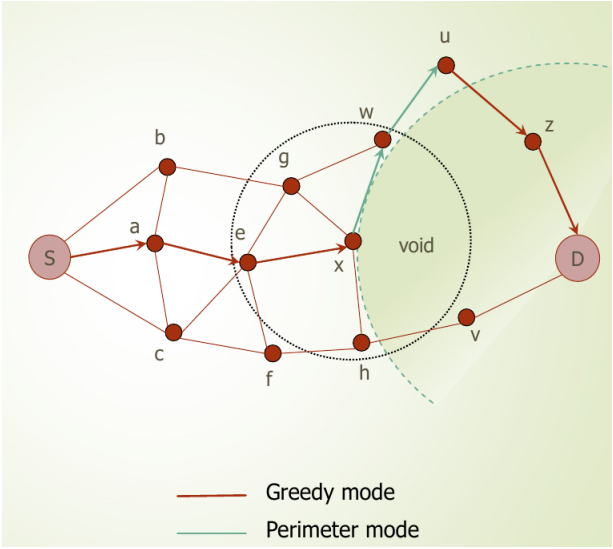


Figure 9.12: GPSR greedy and perimeter forwarding overview

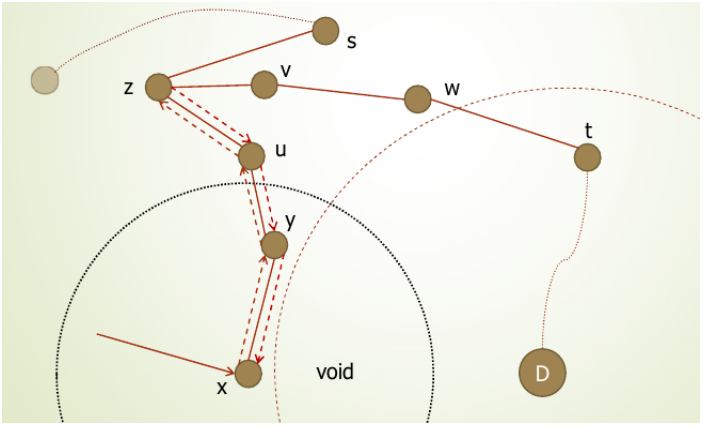


Figure 9.13: Switching from greedy mode to perimeter mode

$$y \rightarrow u \rightarrow z \rightarrow v \rightarrow w \rightarrow t \rightarrow D$$

It's necessary to understand when **switch back to greedy mode**, for example when the packet is forwarded to z: switch to greedy mode in z would mean creating a loop because packets would be forwarded back to u which is geographically closer to D respect to z. In this scenario, the perimeter mode is mantained until the node w is reached thus the node t is switched back to greedy node and D is reached.
The general rule to switch from the *perimeter mode* to the *greedy mode* is that: "**GPSR switch back to greedy mode when it found a node that is closer to destination D respect to a node** x **that firstly switched to perimeter mode**". Applied to the previous example, the node w is aware that the node t is closer to D respect to x, (*where* x *previously switched from greedy mode to perimeter mode*), thus w switch again to the greedy mode.
It's **implicitly safe** the switch back from perimeter mode to greedy mode because we ensure the progress with the perimeter mode exploration. When x began the perimeter mode add its location to the **header of the packet**, alongisde other information to keep properly the perimeter mode. This information are used to switch back from one mode to another (see later).

Perimeter mode

To implement the perimeter mode is necessary to identify the nodes alongside the *void area* by exploiting only the local information and routes around the void: the exchange with the local neighbors allows to identify the nodes on the perimeter.
The forwarding mode in *perimeter mode* it's based on **Right Hand Rule (RHR)** or equivalently **Left Hand Rule (LHR)**: based on figure 9.14, when arriving from y to x selects the first **counterclockwise**

edge from (x, y) and traverses the interior of a **closed polygonal region** (*or face*) in clockwise order. Imagine starting from y, using *RHR* arrive to node x and apply again the *RHR* from the outgoing edges of x to reach z.

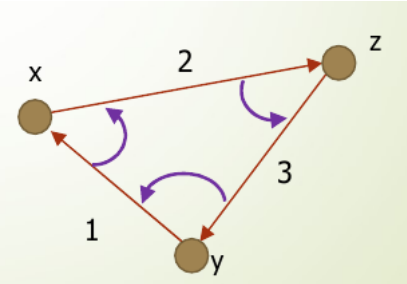


Figure 9.14: Perimeter mode Right Hand Rule

This methods allows to explores the edge of the polygon enclosing the void region.
The algorithm works only if the underlying graph of the network topology is **planar**: the WSN's graph are mostly not planar so edges may *cross* and the *RHR* may take a *degenerate tour* that does not trace the boundary of a *closed polygon*.
In figure 9.15, from x to v the *RHR* produce the path:

$$x - v - w - u - x$$

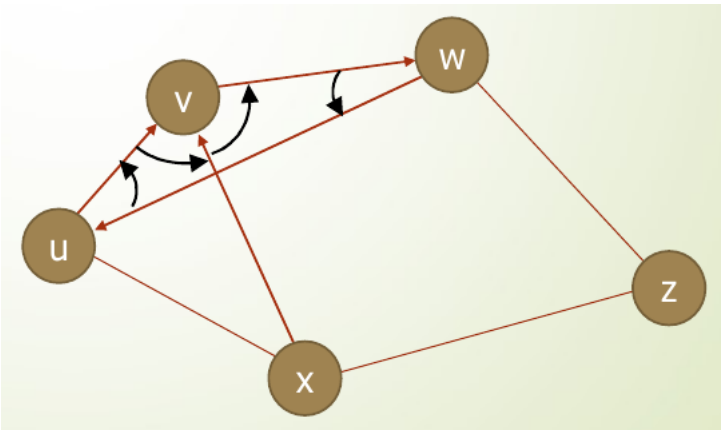


Figure 9.15: Degenerative loop

Graph Planarization

We cannot assume that the network topology is **planar**: the solution is to enforce the planarity to the network topology graph representation [3]. We can build a planar network topology exploiting local information by using:

- **Relative Neighborhood Graph of G (RNG)**
- **Gabriel Graph of G (GG)**

They both create a *subgraph of a given graph*, without adding link but only removing. They mantains some properties: if G is connected then the produced planarized graph P is connected and it's obtained by removing edges from G.
Secondly, they execute a **localized, distributed knowledge discover** to identify the edge of the *planarized graph*, this is possbile because every node have executed the planarization of the graph in advanced so minimizing the memory and computation overhead.
Starting from the obtained graph P, *GPSR* uses the *greedy mode* on *all links* of the **original graph** G but limit the *perimeter mode* only on the links *preserved in the* ***planarized graph*** P.

[3] *Differently, in Direct Diffusion the DAG is directed based on the sink position thus there are not constrained on the real network topology that leads to geometric-planarity property ploblem*

Relative Neighborhood Graph (RNG) The edge $(u, v) \in P \iff (u, v) \in G$ and

$$d(u, v) \leq \max_{\forall w \in N(u) \cup N(v)} (d(u, w), d(w, v)) \tag{9.1}$$

Consider node u: for each neighbor $v \in N(u)$, edge (u, v) is kept if and oly if the area between the node u and v is empty.

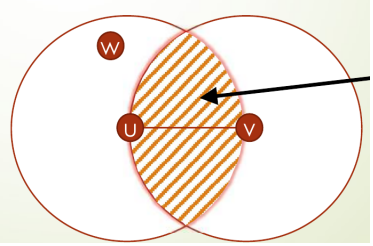


Figure 9.16: RNG *void area* for link removal in P

If we have a node in the *empty zone*, we can remove the ray (u, v) because the connectivity is preserved by this node that act as a form of *bridge* between u and v (*assuming circular transmission range to both* u *and* v). So RNG **cut long links**, preferring shorter links that preverse connectivity.

Gabriel Graph (GG) The edge $(u, v) \in P \iff (u, v) \in G$ and

$$d(u, v)^2 \leq d(u, w)^2 + d(w, v)^2, \forall w \in N(u) \cup N(v) \tag{9.2}$$

Consider node u: for each neighbor $v \in N(u)$, edge (u, v) is kept if and oly if the *circular area* between the node u and v is empty.

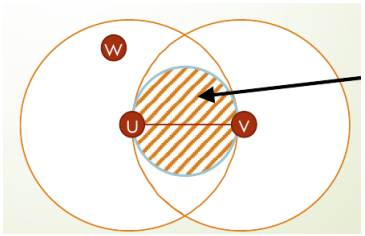


Figure 9.17: GG *void area* for link removal in P

The Gabriel Graph P is built with a distributed algorithms and keeps more links of the original graph respect to *RNG*, thus RNG is a subgraph of GG. Also, GG works better with GPRS because provide a graph with *shorter perimeters*.

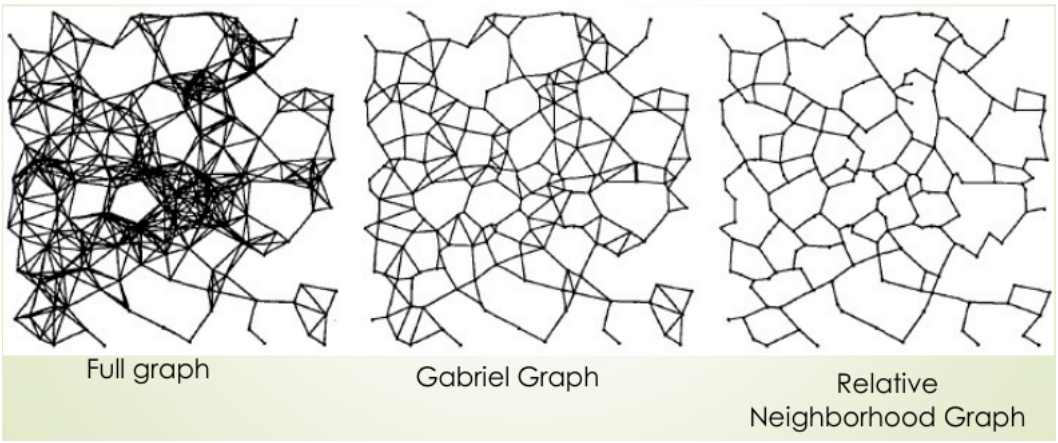


Figure 9.18: Graph G and output graph P with RNG and GG

GPSR with graph planarization A **planar graph** has two types of faces:

- **Interior faces**: closed polygonal regions bounded by the graph edges
- **One exterior face**: unbounded face outisde the outer boundary of the graph

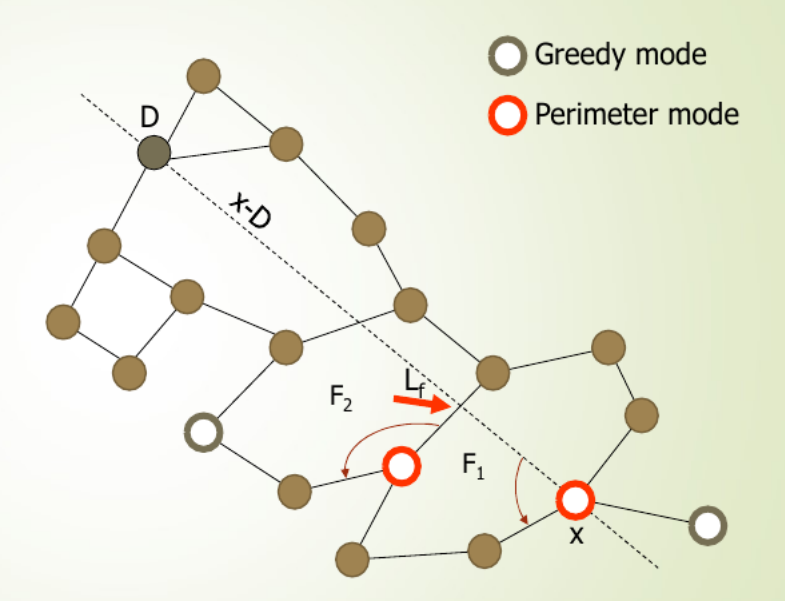


Figure 9.19: Planar graph with three faces

In figure 9.19, the line x to D allows to identify the faces F_1, F_2, F_3 of the graph, alongside the x's two neighbors along F_1 border. In the **perimeter mode** the packet will reach the second red node, at the intersection with $x - D$ line and L_f: from this node onward we switch the graph face from F_1 to F_2 thus the packet is sent to all nodes alongside F_2 border until it reaches a node that is closer to D respect to x, switching back to greedy mode.

So more formally:

- In each of the faces GPSR uses the right hand rule (RHR) to reach an edge that intersects with the $x - D$ line and that is closer to D than x.
- At that intersection edge GPSR moves to the adjacent face crossed by $x - D$ line
- Each time the packet enters a new face then we store both the intersection point L_f with the $x - D$ line and the edge that is currently being used, alongside e_0 that is the first edge crossed in the new face. GPSR returns to **greedy mode** if the current node is closer to D than x, so the **perimeter mode** is only intended to recover from a *local maximum*.
  In *perimeter mode* the packet header contains:

Field	Function
D	Destination Location
x	Location where packet entered in perimeter mode
L_f	Point on x-D where the packet entered current face
e_o	First edge traversed on current face
M	Packet mode: greedy or perimeter

Figure 9.20: Perimeter mode packet's header data

where x is the node where the packet enters in the *perimeter mode* and considering the segment $x - D$, GPSR forwards the packet on progressively closer faces on the planar graph, each of which intersects $x - D$.

The rule of finding the intersection and then moving on to the next face is not used too often, this is because there is an **aggressive policy** for switching from perimeter mode to greedy mode and then we switch to greedy mode when the node we reach is closer to the destination with respect to x.

The node forward the packet alongside the faces of the graph, switching back to greedy mode only **when the distance of the node that started the perimeter node is longer than the actual node distance.**

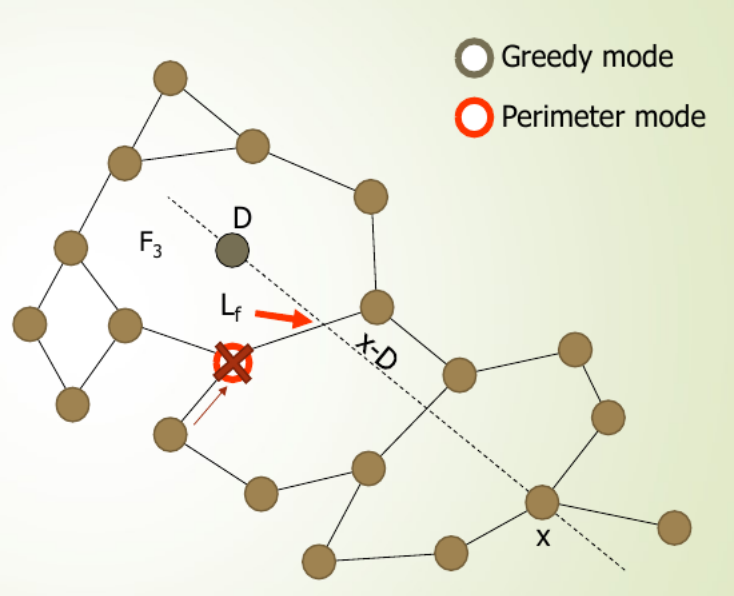


Figure 9.21: Delivery failure due to unconnected destination inside interior face F_3

Delivery failure Considering the graph faces, there are two cases in which the destination node D is not reachable:

- D lies inside an interior face F_i: referring 9.21, the packet tour aroud face F_3 and when crosses the *red node* picture it's dropped.
- D lies inside the *exterior face* F_e: the packet will reach the face by making a tour around the entire border of the graph, until it passes again through the node that started the *perimeter node* so at that point the packet is dropped and the delivery fail.

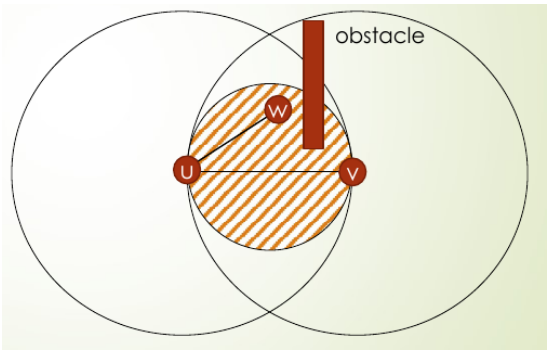
Simulations

The dynamic of the network (*node mobility, joining nodes, etc*) leads to *performance degradation*: performing planarization at each topology variation implies an high overhead of recomputing the graph P. An approach to avoid performance degradation consists in nodes that periodically communicate their position to neighbors (***beaconing***) to allows adjacent neighbors to mantain a *neighbor's list*.
The paper shows that when the performance degrade over a certain threshold the planazation must be re-executed, despite the *beaconing mechanism*: data shows that decreasing the beaconing interval improves the *delivery rate*, and that sparse network can represent a problem due to a shortage of allowed paths.

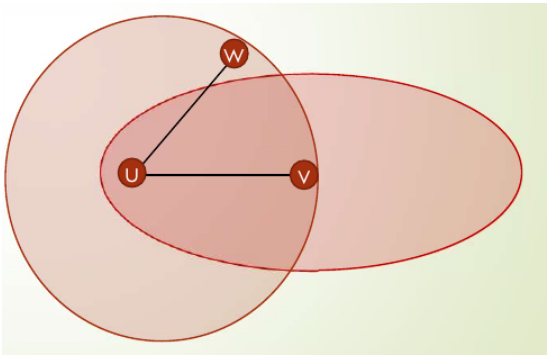
9.2.2 Drawback

Planarization can be unprecised or even fail in case of **unidirectional links obstacles** 9.22a or **non-circular trasmission ranges** 9.22b.
For the obstacle scenario, the node u is aware of node w so will not consider the edge (u, v) but v is not able to identify node w thus the **planarization** will only consider two links: $u - w$ and $v - u$, producing a graph with unidirectional links.

An incorrent graph planarization can lead GPSR to fail by triggering a loop in which the algorithm is applied to unidiretional link, as pictured in 9.23.
The graph pictured in 9.23 is the result of an incorrect planarization: the problem is the link between the



(a) Unidirectional links obstacle



(b) Non-circular trasmission range

node u and x because x will not see v in its neighbors list, while u will see v as neighboard and will not consider the link to x (*no outgoing edges, only incoming because it's unidirectional, while in the original graph is bidirectional*). The packet from the source, following *RHR* will flow through $u - v - w - x - u$, forming a loop because u is closer to D respect to x, under x local view.

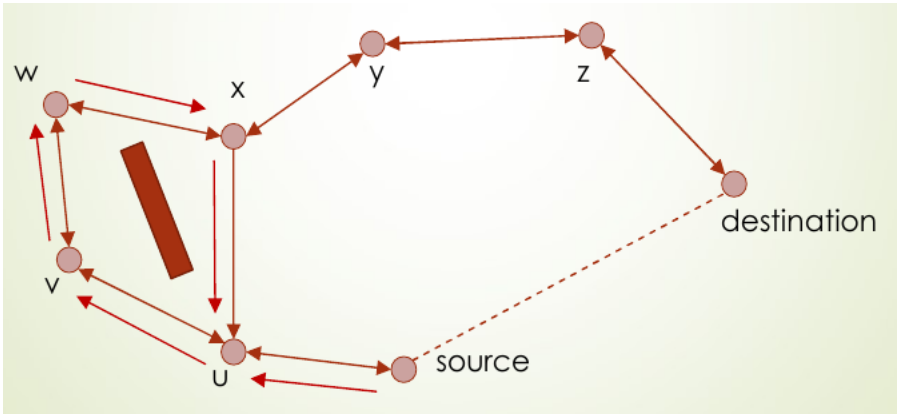


Figure 9.23: GPSR loop due to wrong graph planarization

The solution to this problem is known as *Mutual Witness* that represent an extension of the planarization algorithm. The description of this solution is outside the scope of this notes.

Chapter 10

Wireless Networks

10.0.1 Wireless networks

Wireless allows to replace cables in communications in cyber-physical systems that can embed computers in physical objects. Wireless networks are networks of **hosts** connected by **wireless links**: we refer to **host** as end-system devices that run application (battery-powered, small or mobile). There are two modes of operations for such networks:

- **Infrastrucuture**: offer an Access Point that can provide a link to a cabled infrastracture and offer various services
- **Ad hoc networking**: there is no central coordinations so they must somehow coordinate themself and manage properly shared resources

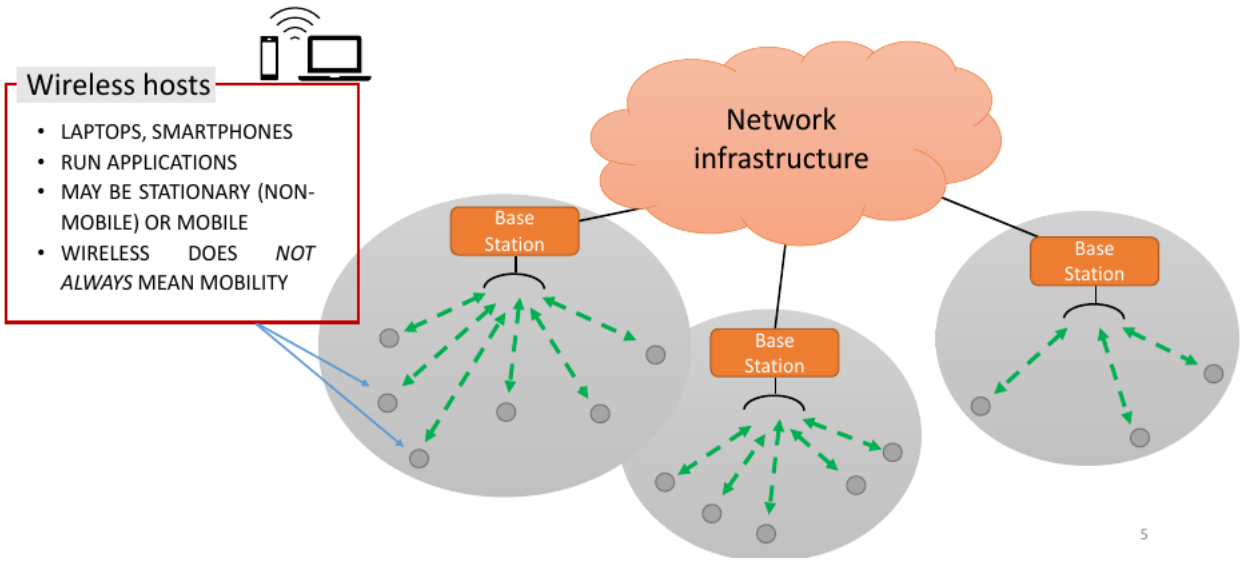


Figure 10.1: Wireless Network overview

The network infrastructure can be a mix of cabled and wireless infrastructure. The devices connected to the **base station** can be both mobile or not; the same can happen with the type of links that can be cabled or wireless, depending on the scenario.
The **base station** have a revelent role in structured networks: it's responsible to manage the resources of the *physical medium*. They usually are connected to a wired network and it's responsible for *sending packets* between wired network and wireless hosts in its area. They are represented by cell towers or Access Point.
The **wireless links** allows to connect mobile(s) to base stations and can be used as backbone links, coordinating multiple access to the link. In **infrastructure mode**, base station connect mobiles into wired networks. Mobile can changes base station providing connection into wired network.
In figure 10.2 are pictured the two scenario in which the base station is involved.

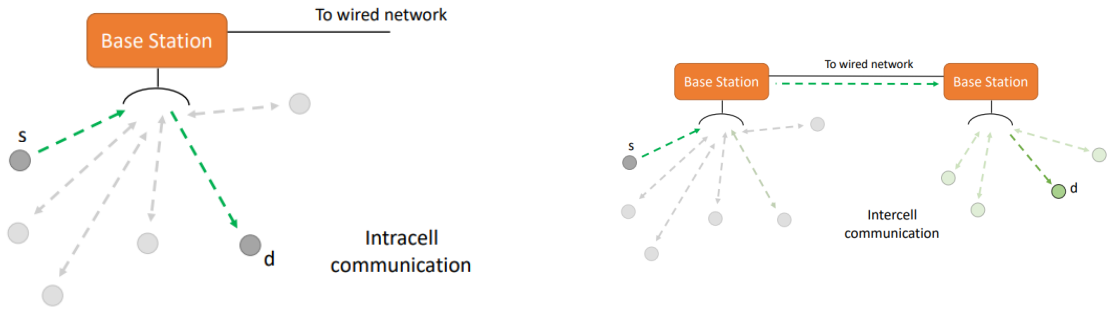


Figure 10.2: Base station role in intracell and intercell communications

Wireless Network taxonomy We distingish the network type based on two category, as shown in the table 10.3.

	Single hop	Multiple hops
infrastructure (e.g., APs)	host connects to base station (WiFi, WiMAX, cellular 3G,4G,5G) which connects to larger Internet	host may have to relay through several wireless nodes to connect to larger Internet: MESH networks
no infrastructure	no base station, not necessarily connection to larger Internet (e.g. Bluetooth)	no base station, no connection to larger Internet. May have to relay on other nodes to reach a given wireless node (ZigBee, ad hoc, VANET)

Figure 10.3: Taxonomy parameters and use cases

An example of *infrastructure, single hop* is the mobile connection or throught the WiFi. The schema behing *no infrastructure, multiple hosts* is used when different devices needs to exchange information and makes routing decisions among them, without relying on a fixed infrastructure.

10.0.2 Charateristics of wireless communications

There are foundamental differences between wired and wireless links:

- **Decreased signal length**: radio signals attenuates as it propagates trhought matter causing **path loss**
- **Interference**: wireless frequency can be shared by other wireless devices by specification but engines or appliances may interfere as well.
- **Multipath propagation**: the reflection of the wave on context object can be distorted and change the origianl signal that it's necessary to rebuild

Communication Environment metrics

Some important metrics in **communication environment** are:

- **SNR - Signal to Noise Ratio**: measured at the receiver side. The SNR *bit error rate* (**BER**) is the probability that a transmitted bit is received in error at the receiver. Based on different modulation technique we can achieve a lower or higher data rates, with different BER. Refer the results in 10.4.

BPSK have a lower data rate and if we increase the SNR by increasing the *energy of the signal*, we lower the BER. For QAM16: you need more energy to get the same BER than BPSK at a certain level, as you send more information.

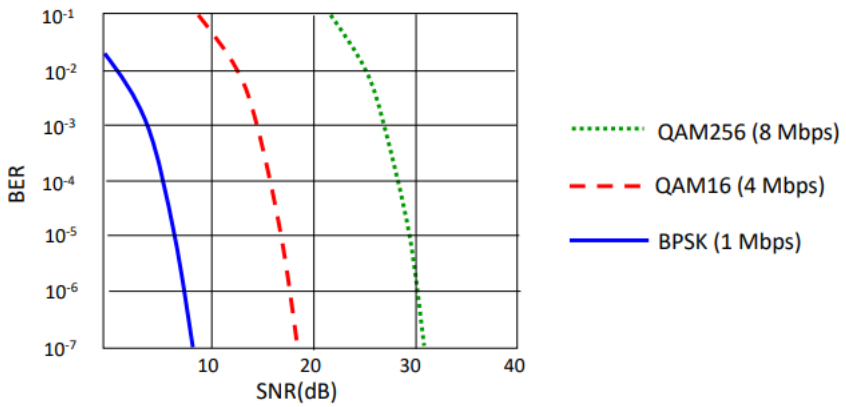


Figure 10.4: BER/SNR comparison between BPSK, QAM16 and QAM256

10.0.3 Obstacles

Signal attenuation or obstacles limit transmission ranges: the situation sensed by a node, given that there may be a wall, which prevents two devices to detect each other, while they can talk with a central host. According to how those host are positioned in distance:

- `A` has large signal strenght to `B`, but when we arrive at `B` the energy of the signal decrease and `B` is able to detect it.
- With higher distance `A` is not able to detect `C` signal. Also since `B` may receive two different signals at the same time, it may be difficult for it to reconstruct a signal. With the distance the amplitude of the signal decrease but despite that, the signal from C to B is received, the same from A to C but the intersection of receiving two signals from two different sources to B can be problematic.

The described scenario is pictured in 10.5

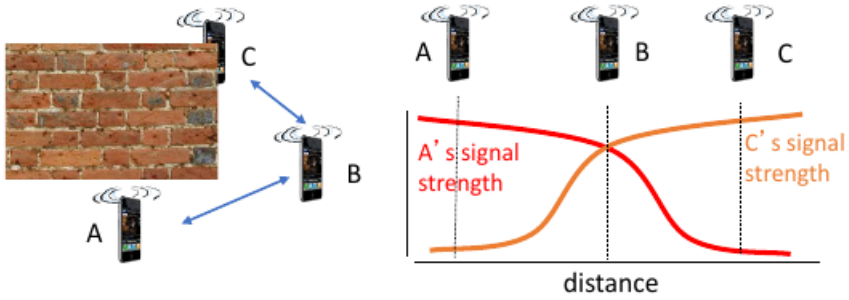


Figure 10.5: Obstacle scenario pictured

Some of wireless network challenges are that nodes have **limited knowledge** regarding a terminal that cannot hear all the others or **hidden/exposed terminal** problems (*see later*). Another main problem is **Mobility or Failure of terminals** in which they move in the range of different base station. **Limited terminals** regards the battery life, memory and processing. In 10.6 is pictured the protocol stack that we'll describe.

Let's describe the **MAC layer** for ***wired network***. We start from simple assumption like:

1. a **single channel** for all communications
2. all stations can transmit on it and receive
3. if frames are sent simultaneously on the channel the resulting signal is *garbled* and a **collision** is generated, generally all stations can detect collisions. Different protocol are at this level, like: *ALOHA, slotted ALOHA, CSMA, etc.*

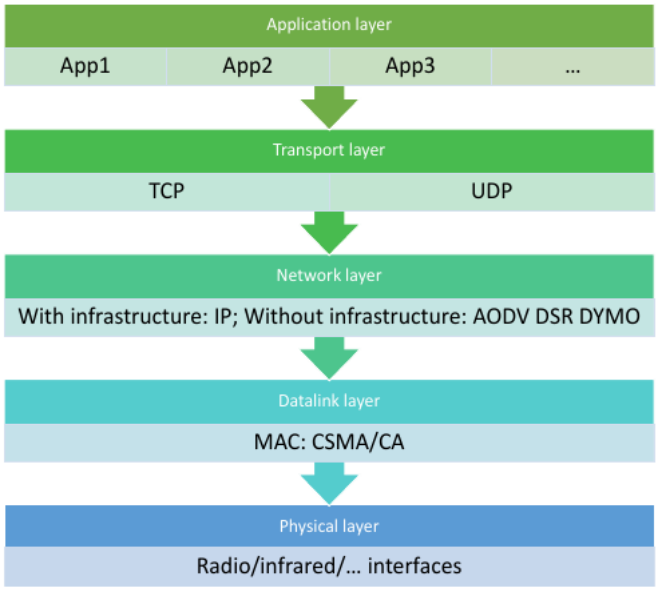


Figure 10.6: WiFi protocol stack

10.0.4 CSMA/CD - Carrier Sense Multiple Accesses with Collision Detection

The basic idea is that when a station has a frame to send, it first listens to the channel to see if anyone else is transmitting: if the channel is **busy**, the station waits until it becomes **idle**. When a channel is *idle*, the station transmits the frame: if a *collision* occurs the station waits a random amount of time and repeat the procedure.

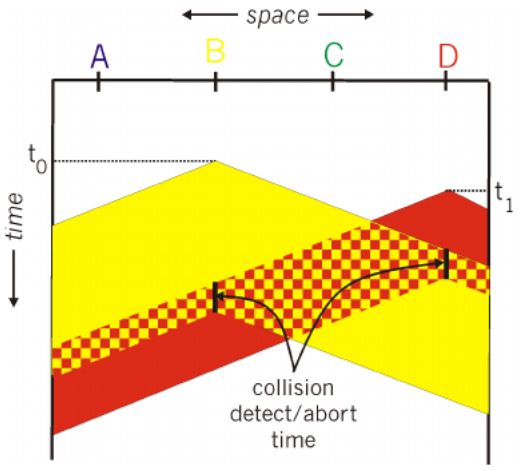


Figure 10.7: Collision Detection example

In the scenario shown in 10.7, the signal is transmitted at time t_0 because the channel was *idle* but at time t_1 another signal (*the red one*) is transmitted by generating a collision on the channel. So in CSMA/CD a station **abort its transmission as soon it detects a collision**: if two stations sense the channel idle simultaneously and start transmitting, they quickly abort the frame as soon collision is detected. It's widely used on LAN MAC sub-layer like *IEEE 802.3 Ethernet*.
The CSMA/CD behaviour can be descripted by defining:

- `T` is the *time required to reach the fartest station*
- It takes a minimum of RTT time (`2*T`) to detect a collision

So we can consider a **slotted system** 10.9 with *mini-slots* of duration 2τ, so if a node starts transmission at the beginning of a mini-slot, by the end on the slot either two things can happen:

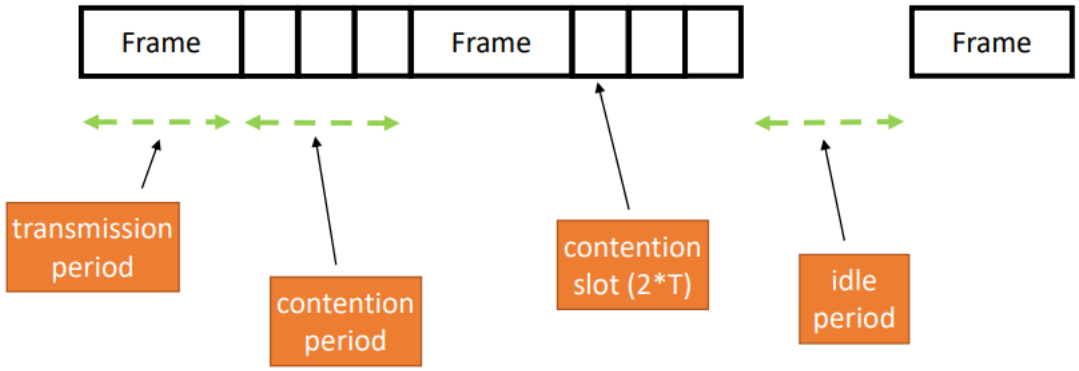


Figure 10.8: CSMA/CD periods

1. No collision occurred and the rest of the transmission will be *uninterrupted (because now the channel is occupied and this condition is detectable by other nodes)*
2. A collision occured but by the end of the mini-slot the channel would be **idle** again

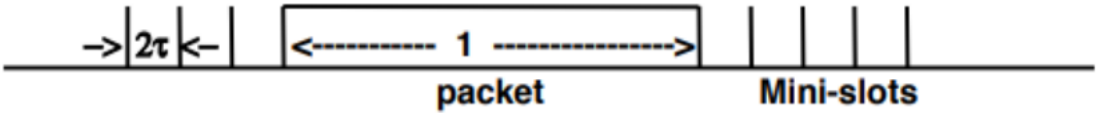


Figure 10.9: Slots of 2τ used in trasmission phase

So a collision at most affect *one* mini-slot.

Binary Exponential Backoff The algorithm used by CSMA/CD to retry retransmission is called **Binary Exponential Backoff** and defines how much time the station must wait before trying transmitting again.

The time after a collision is divided in **contention slots**: the length of a slot is equal to the *worst case round propagation time* (*so if T is the time to reach the most distant station, the length will be* $2T$). After the first collision each station waits 0 or 1 slot before trying again: so after collision i the nodes chooses x at random in the interval $[0, 2^i - 1]$ so that it skips x slots of time before retrying the trasmission. Number at hands, after:

- 10 collisions, interval is frozen at $[0, 1023]$
- 16 collisions, failure is reported to upper levels (*retrasmission is aborted*).

Why does this approach not work with wireless network? The algorithm described is suitable in **wired networks** but not in **wireless scenarios**. The main problems are:

1. The transmitted sides is sensing if there is another sending a signal in the channel. In wireless is not possible to both transmit and receive (*at least with 1 antenna*), the problem is that you transmit your signal with high energy, the one you receive has low energy, so it happens that you do not recognized the other. With wireless transreceiver you cannot detect collisions.
2. If a node is out of range, it may not note a collision, this problem is called **Hidden terminal problem** 10.0.5.
3. If two node overlap in the same transmission range, a third node does not transmit due to a wrong channel busyness perception: this is known as **Exposed terminal problem** *see 10.0.6*

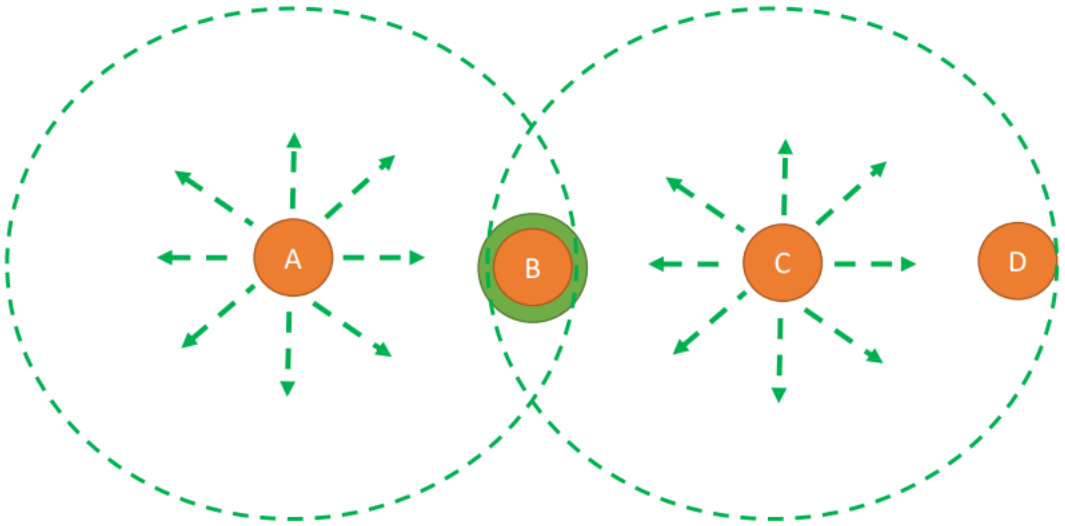


Figure 10.10: Node C as hidden terminal respect $A \rightarrow B$

10.0.5 Hidden terminal problem

The problem's scenario is sketched in 10.10. The node A can receive and transmit to B because B it's in his radio range. The same happen to C: can transmit and receive to/from B and D because they are in C's radio range. The problem arise when A is sending to B and C senses the medium: *it will not hear A's transmission because its out of C's range* so if C start to transmit to anybody (*either B or D*) this *generate a collision at B*.

- **Hidden terminal problem**: C is not able to detect a potential competitor because it is out of range: a collision happens at B (*the receiver*). For the same reason A does not detect the collision so **C is hidden** with respect to the communication from A to B. Usually this problem arises when *two or more stations which are out of range of each other transmit simultaneously to a common recipient.*

10.0.6 Exposed terminal problem

The **Exposed terminal problem** scenario is sketched in figure 10.11.

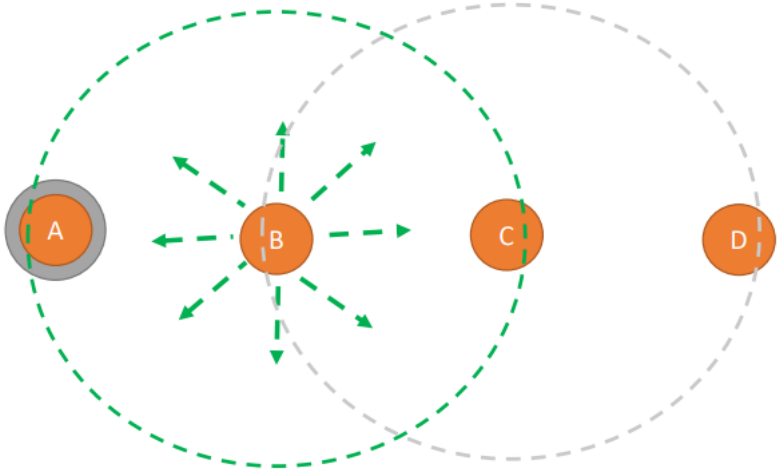


Figure 10.11: Node C as exposed terminal respect to $B \rightarrow A$

The node B is transmitting to A: the node C wants to transmit to D so C senses the medium and because detect the B $\rightarrow$ A transmission it concludes that cannot trasmit to D. But, **the two transmission can actually happen in parallel**.

- **Exposed terminal problem**: C hears a transmission. C does not send to D despite its transmission would be perceived without collisions. So **C is exposed** with respect to the communication from B to A. Usually this problem arises when *a transmitting station is prevented from sending frames due to interference with another trasmitting station.*

The solution on wireless network for this two types of problem is represented by **MACA Protocol**.

10.0.7 MACA - Multiple Access with Collision Avoidance

The basic idea is to stimulate the receiver by transmitting a **short frame** first and then transmit a long data frame. So stations hearing the short frame **refrain from transmitting** during the trasmission of the subsequent data frame.

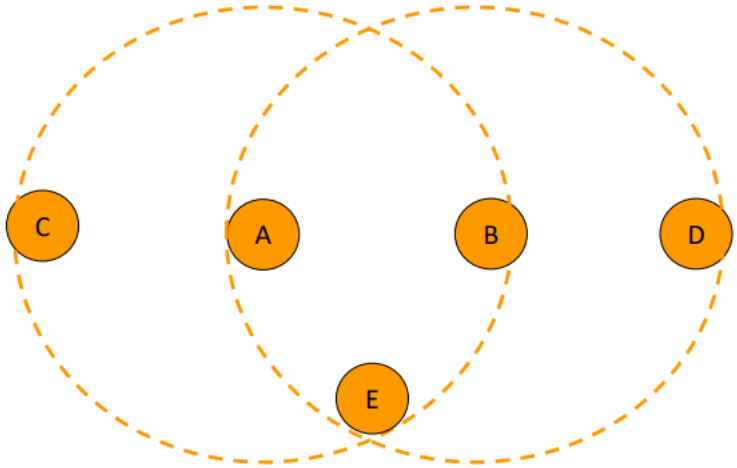


Figure 10.12: Initial configuration scenario

The scenario is pictured in 10.12 and it's here described:

- C is **within** range of A and **out** of range of B and D
- D is **within** range of B and **out** of range of A and C
- E is **within** range of both A and B

The protocol flows:

- **Step 1**: A wants to transmit to B so send a **RTS - Request To Send** packet to B: *the short frame* sended include also the length of the data frame that will eventually follow.
- **Step 2**: B, C and E receive the **RTS** from A (*because they're in A's radio range*). If B wants to receive the message, it replies with a **Clear To Send - CTS** that is a short frame with data length copied from RTS sended by A.
- **Step 3**: the **CTS** sended by B is received by A, D, E
- **Step 4**: upon reception of the **CTS** frame by B, A (*who is the only one interested in receiving the CTS because he want transmit the data frame*) start to transmit the data frame

Two main problems can arise from this process:

1. **C hears the RTS from A but not the CTS from B** so it is free to transmit (*may be interested in transmitting).* So **C is exposed**. Refer the figure 10.17.
2. **D hears CTS from B nut not RTS from A** so should stay silent until data framr transmission completes. So **D is hidden**. Refer the figure 10.18.

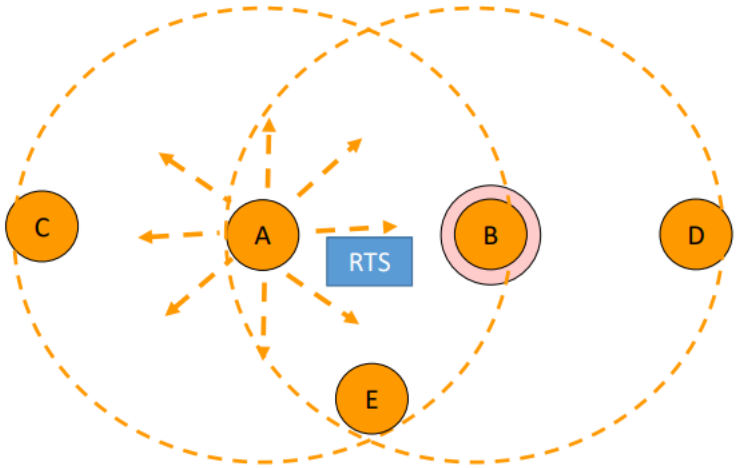


Figure 10.13: **Step 1**: A wants to transmit to B so send a **RTS - Request To Send** packet to B: *the short frame* sended include also the length of the data frame that will eventually follow.

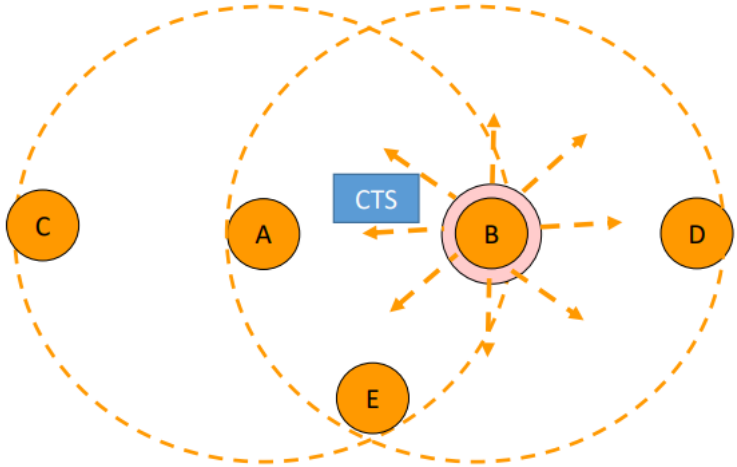


Figure 10.14: **Step 2**: B, C and E receive the **RTS** from A (*because they're in A's radio range*). If B wants to receive the message, it replies with a **Clear To Send - CTS** that is a short frame with data length copied from RTS sended by A.

MACA collisions Given the previous scenario, *what happen if anoter node out of the range of A and B wish to trasmit a message to node E?*.
The two problem are pictured in 10.17 and 10.18.
Anoter problem is presented by the scenario in which C and B sends *RTS* simultaneosly to A. **The two RTS messages collide so no CTS message is generated**. The adopted solution is that C and B use *Binary Exponential Backofff* to retry send the RTS.

10.0.8 MACAW: MACA for Wireless Network

Fine tunes MACA to improve performance:

- introduces an **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
- exponential backoff is run for each separate pair source/destination and not for the single station
- mechanisms to exchange information among stations and recognize temporary congestion problems
- CSMA/CA used in IEEE 802.11 is based on MACAW

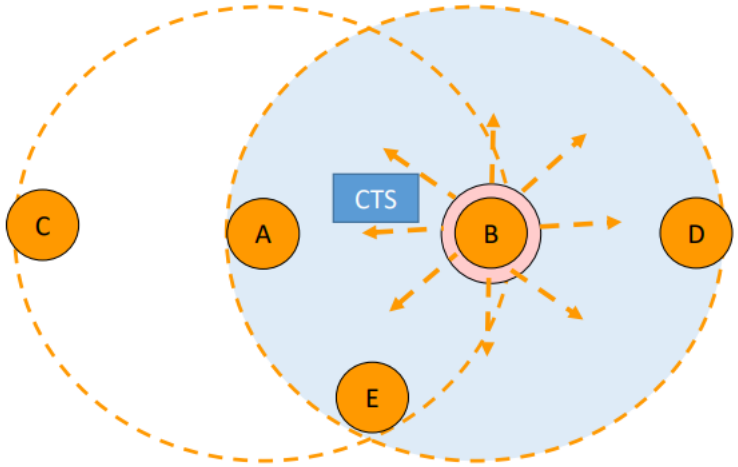


Figure 10.15: **Step 3**: the **CTS** sended by B is received by A, D, E

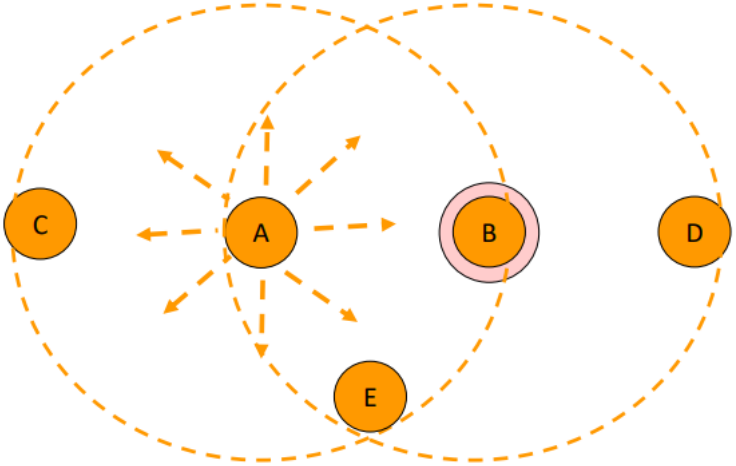


Figure 10.16: **Step 4**: upon reception of the **CTS** frame by B, A (*whose the only interested in transmit data*) start to transmit the data frame

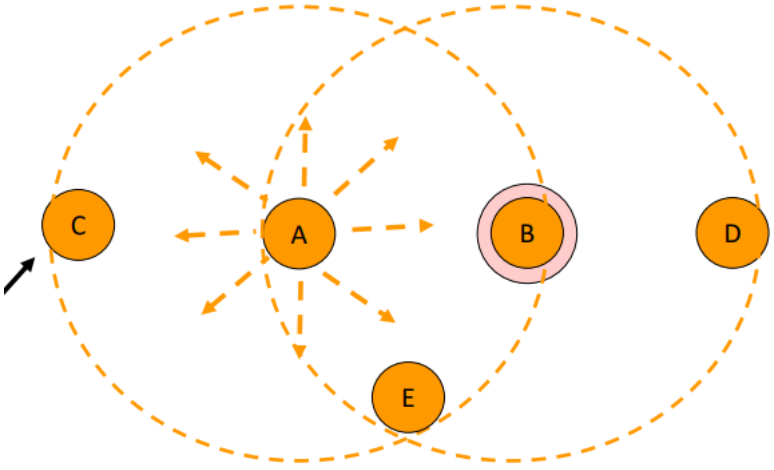


Figure 10.17: **Problem 1: C hears the RTS from A but not the CTS from B** so it is free to transmit (*may be interested in transmitting*). So **C is exposed**

Questions (1)

Given the network in the figure A.7, assume that the MAC protocol uses the RTS/CTS mechanism for the channel access.

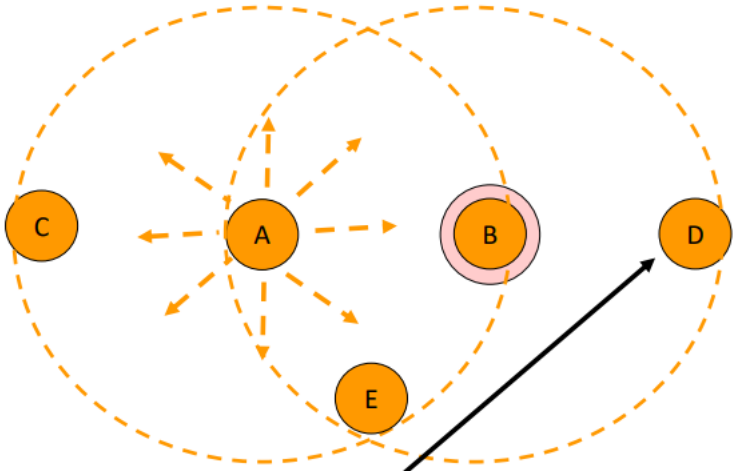


Figure 10.18: **Problem 2: D hears CTS from B nut not RTS from A** so should stay silent until data framr transmission completes. So **D is hidden**

Discuss which nodes detect themselves as hidden or exposed as consequence of the RTS/CTS handshake in the following cases:

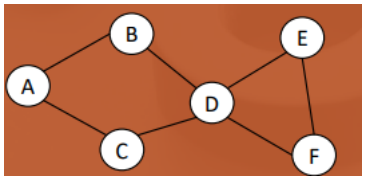


Figure 10.19: Exercise 1 network schema

- Hidden terminals with respect to a transmission from E to D
- Hidden terminals with respect to a transmission from D to C
- Exposed terminals with respect to a transmission from D to B
- Exposed terminals with respect to a transmission from B to A

Questions (2)

Given the network in the previous figure A.7:

- assume that D hears the RTS sent by E but it does not hear the corresponding CTS. What does D can do?
- assume that B hears the CTS sent by D but it does not hear the corresponding RTS. What does B can do?
- Assume D is receiving a communication from a node, and B did not receive the corresponding RTS & CTS and it does not hear the signal transmitted to D. If B wishes to transmit to D what happens?

The solution are:

- Exposed
- Hidden
- Collision

10.1 IEEE 802.11

The IEEE 802.11 identify a protocol family in which each release is identified by a subsequent letter. The first one was the legacy mode and was rarely used because supported only 1-2 Mbps data rate, implemented via Infrared signals with a radio frequencies in the `2.4 GHz` band (*as frequency range*). Subsequent releases were:

- `802.11a` : operate at 5GHz band with a throughtput of 23 Mbps and data rate of 54 Mbps.
- `802.11b` : operate at 2.4GHz but poses a problem of interference with other appliances (*cordless telephones, microwave, etc*). Had a throughput of 4.3 Mbps and data rate maximum of 11 Mbps.
- `802.11n` : operate at 2.4GHz band and 5GHz It support **MIMO** technologies for using multiple antennas at the transmitter and receiver
- `801.11ac - WiFi 5`
- `801.11ax - WiFi 6`

All of those use *CSMA/CS* for Multiple Access and have base-station (*infrastructure*) and ad-hoc network versions (*implementing function to access the medium, see previous).* With lower frequencies we have longer *wave length* so it's suitable for IoT sensors. Despite the **max data rate** is not huge remains suitable for transmitting data in an IoT context.
We can have also different implementation at **data link layer** of both **MAC sublayer** and *Logical Link control* as shown in 10.20.

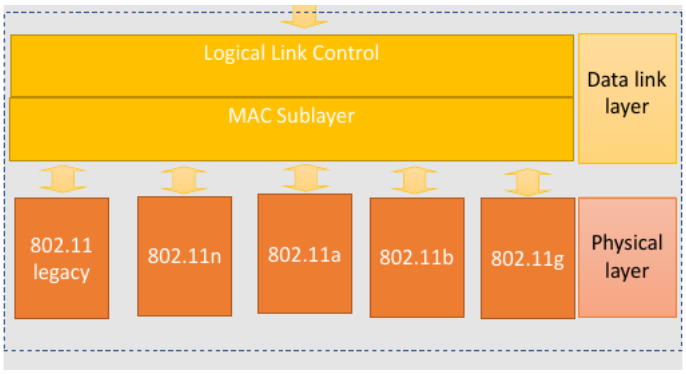


Figure 10.20: Decoupling of data link layer and physical layer

Architecture A group of stations operating under a coordination function may or may not use a *base station* or *Access Point (AP)*. If using AP a station communicates with another by channeling all the traffic through a centralized AP: so AP provide connectivity with other APs and other groups of stations via fixed infrastructure. It can also support *ad hoc networks* by introducing a *group of stations that are under the direct control of a single coordination function without the aid of an infrastructure network.*

10.1.1 Channel association

The **spectrum of a mean** is divided into ***channels*** at different frequencies: the AP administrator chooses the frequency for each AP. At this level the interference is possible because channels can be same as that chosen by neighbors AP.
A new host must be associated with an **AP**: first it scan the channel frequencies and listen for ***beacon frames*** (*like a sort of hello frame, advertise device on the device on the same frequency and sended by the AP*). The beacon frame contains the name of the *network (SSID) and MAC address of AP (BSSID).* The node select one AP to associate and perform authentication (if needed) and then run the DHCP protocol to get IP address in the AP's subnet.

Active/Passive scanning AP periodically sends beacons frames on their frequency so a joining node can detect the beacons frame and select the most suitable AP by sending the **association request**.
The choosing phase of the AP made by nodes is not fully deailed in the IEEE 802.11 RFC: usually is selected the AP with the best **signal strenght** that also is indicative of the number of communication opened by AP with other nodes so is a measure of *quality/availability* of resources.
The **passive scanning** flows as following:

1. Beacon frames sent from APs
2. Association request frame sent from *Host 1 (H1)* to the selected AP
3. Association Response frame sent from the selected AP to H1

The **Active scanning** flows:

1. Probe Request frame broadcast from H1
2. Probe Response frames sent from APs
3. Association request frame sent from H1 to the selected AP
4. Association Response frame sent from selected AP to H1

10.1.2 MAC Sublayer

A sketched version of the MAC sublayer is presented in 10.21. The MAC sublayer can operate in two

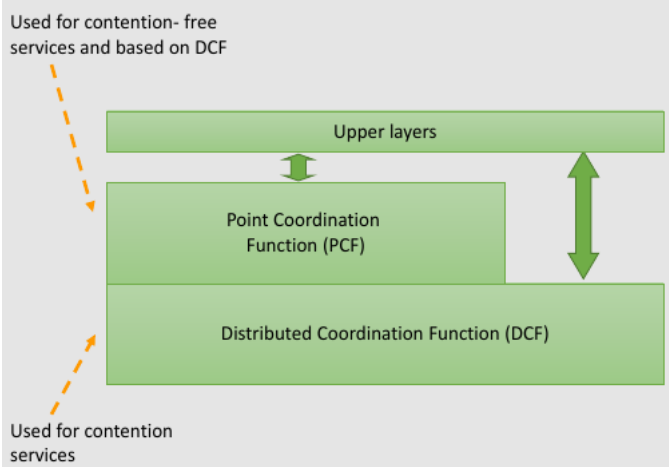


Figure 10.21: MAC sublayer components

different modes:

- **Point Coordination Function (PCF)** : uses a base station (*called PC - Point Coordinator*) to control all activity in its cell in a *contention-free manner* The PC assigns specific timeslots to different station for transmission: during these time slsots, the PC polls individual stations and grant them permission to transmit data.
- **Distributed Coordination Function (DCF)**: operates in *contention-based manner*, where stations contend for access to the wireless medium (*mediated by the CSMA/CA protocol*).

DCF must be implemented by all stations: it's completely decentralized and use a best affort asyncronous traffic. Both DCF and PCF can be active at the same time in the same cell.
Carrier sensing for **DCF** is performed at two levels:

- *Physical*: by checking the frequency to determine whether the medium is in use or not. It detect an incoming signal and any activty in the channels due to other sources.
- *Virtual*: is performed by seding a duration information in the header of an RTS, CTS and data frame. So keeps the channel *virtually busy* up to the end of data frame transmission. A channel is marked **busy** if either the physical or the virtual carrier sensing indicate busyness.

Priority access to the medium is controlled through the use of **interframe space (IFS)** time intervals. The IFS is a mandatory periods of idle time on the transmission medium.

Three IFS specified by the standard:

- *Short (SIFS)*: it's used for high-priority communications like `ACK` frames, minimizing delays for critical communications. After a successfull transmission, a station must wait for the SIFS duration before accessing the medium again.
- *Point Coordination Function (PIFS)*: is intended for medium-priority communications, such as data frames assigned by the Point Coordinator (PC).
  If the medium is idle for a duration longer than PIFS, a station with a frame assigned by the PC can access the medium.
- *Distributed Coordination Function IFS (DIFS)*: is used for low-priority or best-effort communications, such as regular data frames.
  Before attempting to access the medium, a station must wait for the DIFS duration after the medium becomes idle.
  If the medium remains idle for the DIFS duration, the station can initiate a transmission using the Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) protocol.

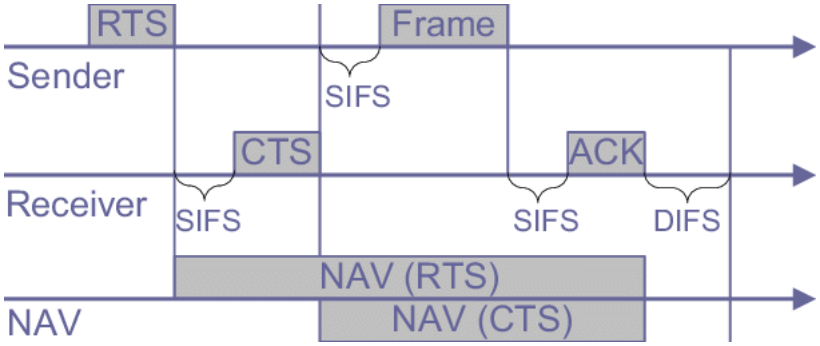


Figure 10.22: Exchange flow to determine the *NAV*

The hierarchical order based on priority and period length is given by:

```
SIFS < PIFS < DIFS
```

The exchange of RTS and CTS between sender and receiver allows the other nodes to calculate the *NAV (Network Access Vector)* that is an **estimation of duration** of communciation between sender and receiver (*a sort of period of silence to avoid illegal accesses to the mean*), as shown in 10.22.

10.2 Mobile Networks

Nowadays there are more mobile-broadband-connected deviced than fixed-broadband. The 4G-5G cellular network now embrace internet protocol stack, including SDN. There are two important different challenges:

- *Wireless*: communication is performed over a wireless link
- *Mobility*: handling the mobile user who changes point of attachment to network or by the provider that wants advertise a service. It also poses authentication and authorization problems towards user requested services.

Components of cellular network architecture The *MSC (Mobile Switching Center)* intermediate the connection betwwen different cells sets of networks, it also manage the billing information by tracking times and call setup. MSC can manage more than 4 cells or *base station*: some MSC can be also used as gateway to the *wired-telephone network*.

From 4G to 5G we have a change in the architecture, integrating software engineering choices in new components like supporting orchestration and function virtualization ([1]).

[1] *More at https://doi.org/10.3390/en12112140*

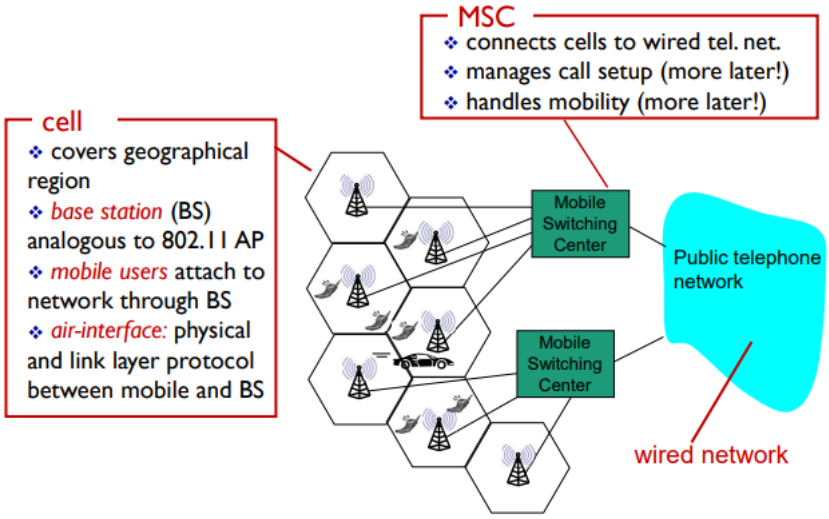


Figure 10.23: Cellular Network architecture

With the evolution of mobile network technology both evolves with the *access techniques*, as the time diagram show in 10.24.

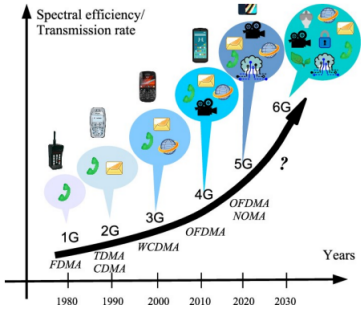


Figure 10.24: Access evolution

10.2.1 FDMA/TDMA and CDMA for first hop access

Frequency Division Multiplexing Access/Time Division Multiplexing Access are two techniques used for **sharing mobile-to-BS (*base station*) radio spectrum**. They can be combined in one by *dividing the spectrum in frequency channels and divide each channel into time slots* as pictured in 10.25

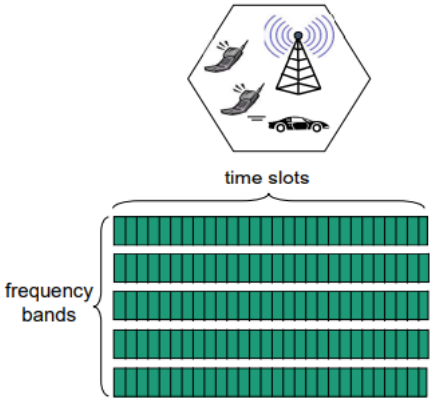


Figure 10.25: FDMA/TDMA: Frequency channel division in time slots

Another methods is *CDMA - Code Division Multiple Access*: is a digital cellular technology that allows multiple users to share the same frequency band simultaneously. It works by **assigning a unique code to each user, which is used to modulate the user's signal.**
CDMA uses spread spectrum technology, which spreads the user's signal across a wide bandwidth. This makes it possible for multiple users to transmit their signals at the same time and on the same frequency band without interfering with each other.
The **unique code assigned to each user acts as a key to unlock the signal at the receiving end**. The receiver is able to identify and separate the different users' signals based on their unique codes, allowing each user to receive their own signal without interference from other users, as pictured in 10.26

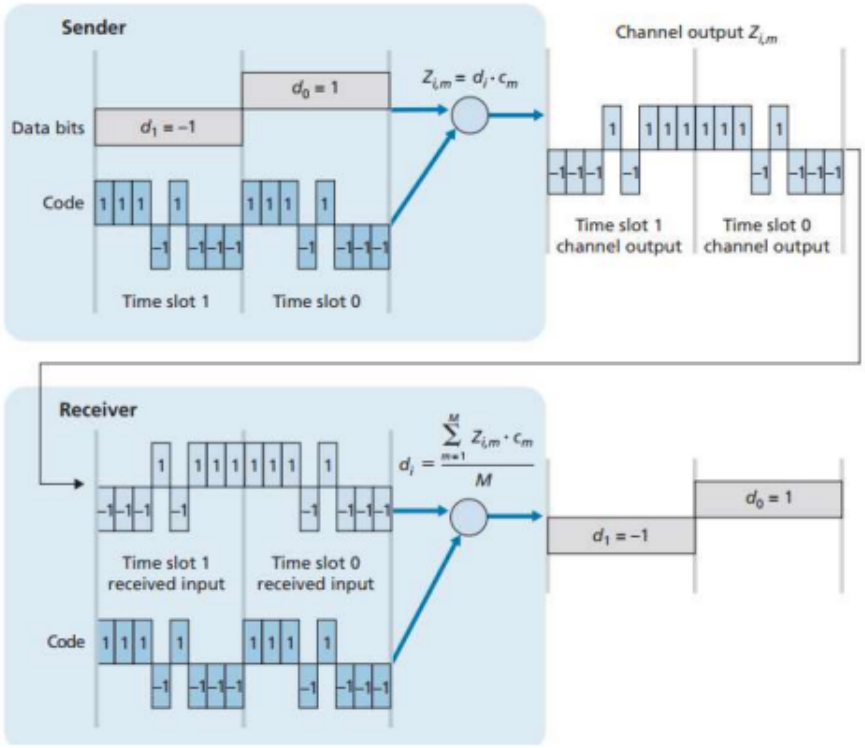


Figure 10.26: Unique code to modulate singular signals scenario

Even in case of different generated signal, if the receiver is able to known the code used by the sender, even if the received signal is the overlap of different codes of different sender the receiver will still be able to reconstruct the data. The specific code used by the sender is negoatiated beforehand with the receiver.

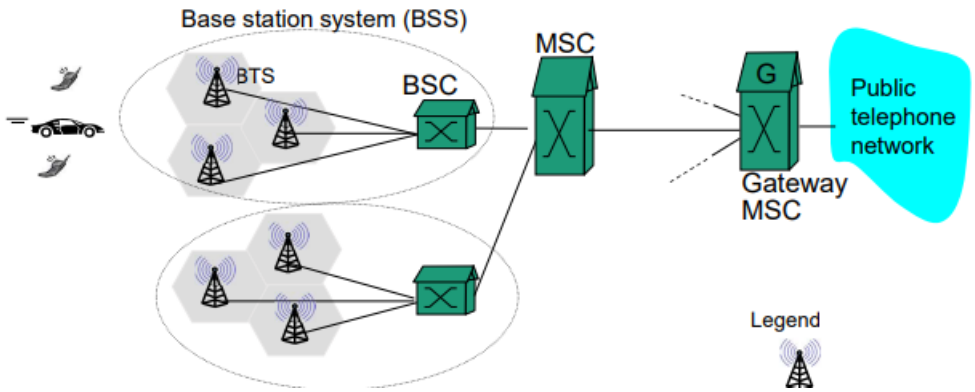


Figure 10.27: 2G network architecture for voice communication

2G Network Architecture (voice)

3G Network Architecture (voice+data) New cellular data network operate in parallel (*except at the edge*) with existing cellular voice network. The voice network is unchanged in core and operates *circuit switching* while *data network* operates in parallel by operating **packet switching**. The general schema is pictured in 10.29

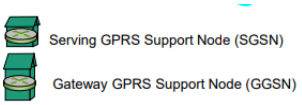


Figure 10.28: Symbol reference

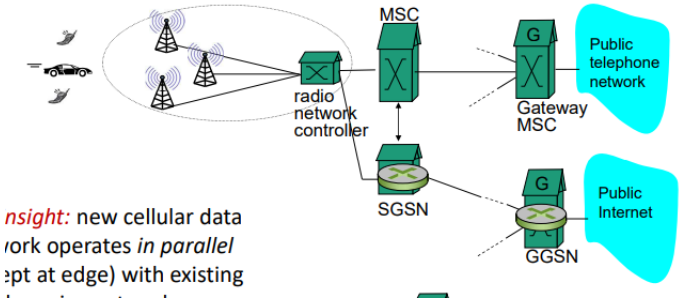


Figure 10.29: 3G Network Architecture

Inside the *core network*, the *Mobile Switching Center* manages the all setup, handles mobility and connects t public wired telephone network. The *Serving GPRS Support Node* and *Gateway GPSR Support Nodes* respectively delivery the data packet from and to the mobile stations within its geographical service area ad manage the inter-networking between GPRS network and external packet data networks.

10.2.2 3G vs. 4G LTE (Long Term Evolution) network architecture

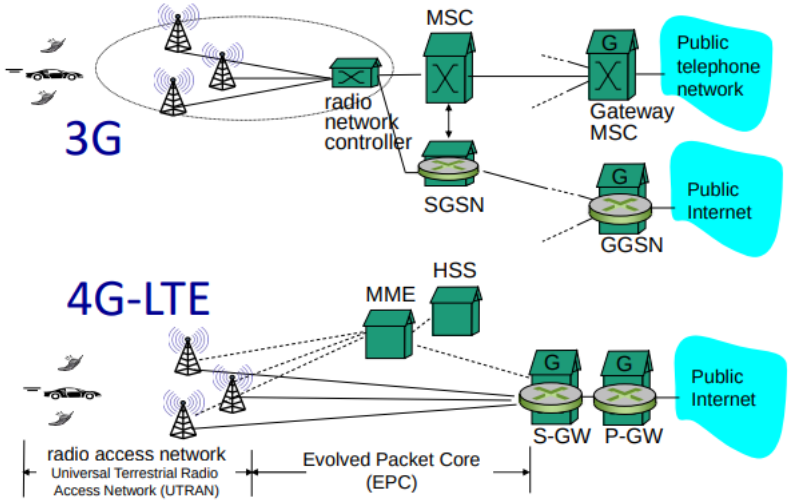


Figure 10.30: 3G vs 4G LTE Network Architecture

The changes between the 3G and 4G network architecture are in the core network, like:

- **All-IP core**: IP packets are tunneled from base station to gateway. The architecture is pictured in 10.31

10.3 4G/5G cellular networks

There are some similarities to *wired internet* like:

- *Edge/core* distinction where we have application services (*edge*) and core (*e.g. services to authenticate devices*)
- *Global cellular network*: operators agree on a network of networks among all geographical countries
- Use of protocols like *HTTP, DNS, TCP, UDP, DHCP*
- Mobile networks are *interconnected to internet* to be able to use services

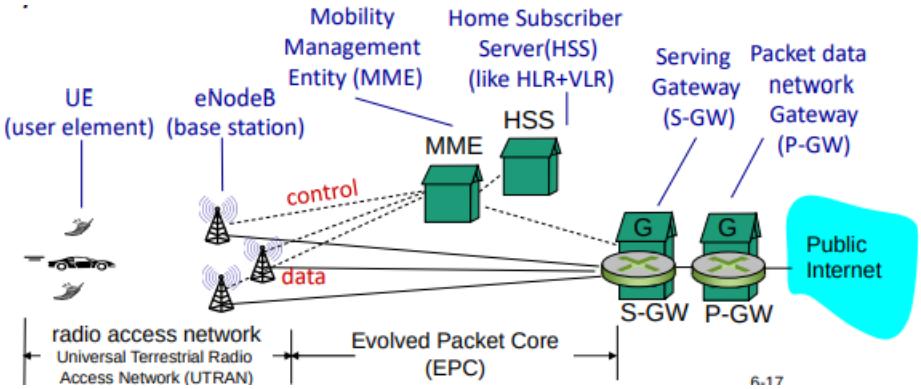


Figure 10.31: 4G-LTE Network Architecture

- No separation between voice and data: both traffic is carried over IP core to the gateway

But there are also main differences from wired internet, like:

- Mobility as a *first class service*
- User *identity* via SIM card
- *Business models*: user subscribe to a cellular provider called "*home network*" that works differently when roaming on visited nets. The global access is granted thanks to authentication infrastructure and inter-carrier settlements

Architecture The general model adopted by 4G/5G Network is pictured in 10.32, where:

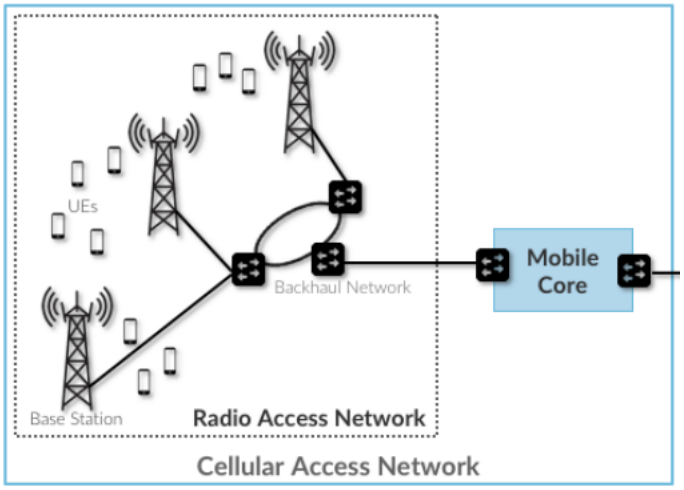


Figure 10.32: 4G/5G general architecture

- **Mobile Core**: provides IP (*internet*) connectivity for both data and voice services. Also ensure QoS requirements and allows to tracks user mobility to ensure uninterrupted service, providing *billing* and *charging*.
- **Blackhaul Network**: allows to connect base station to mobile core (*interconnecting the RAN*) by using optics or wireless solution (*IAB - Integrated Access Blackhaul*).
- **Radio Access Network (RAN)**: it's a distributed connection of Base Stations (BSs) and allows to manage the radio spectrum.

10.3.1 Elements of 4G architecture

In the **Radio Access Network (*RAN*)** the mobile device can be a large series of devices like *laptop, ioT* devices with 4G LTE radio: they are identified by 64 bit *International Mobile Subscriber Identity*

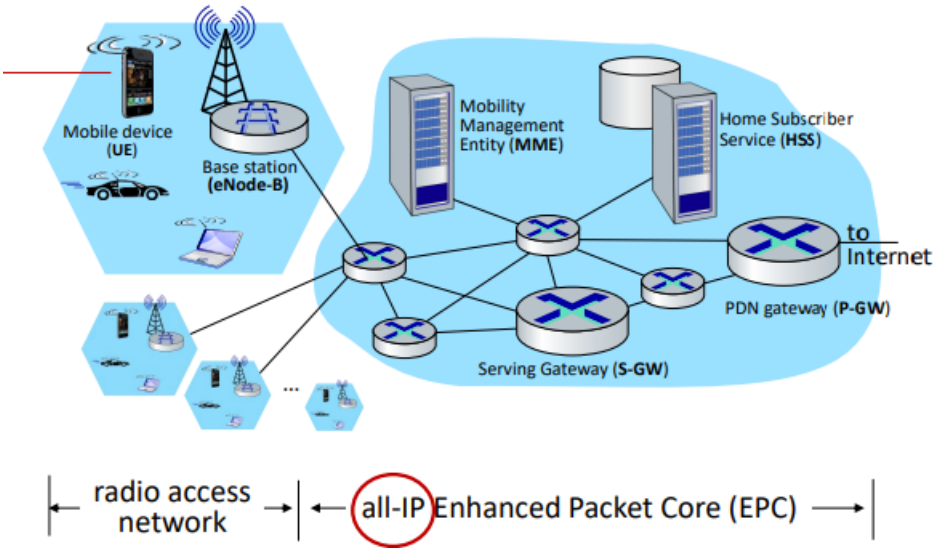


Figure 10.33: 4G Packet Core (EPC - Evolved Packet Core) interconnection

(IMSI) stored on the **SIM - Subscriber Identity Module**.

The **core network** is composed by ***Home Subscriber Service - HSS*** that is a **database** containing information about subscribers with their identifiers and allows to manage the authentication phase. It mantains the information about the device's "*home network*".To be able to authenticate, it works with ***Mobile Management Entity***.

The **Mobile Management Entity (MME)** perform the device authentication, coordinated with mobile home network (*HSS*). The *MME* is also involved when changing the base station and allows to track the device and page the device location (*ask the base station for the presence of this device*). For data packets transfer we need to establish a *data path* from the mobile device to the ***Serving Gateway (S-GW)*** (the *tunneling* and the data path is explained later).
Both the *Serving Gateway (S-GW)* and *PDN (Packet Data Network) Gateway (P-GW)* are on the path from mobile to/from internet network: the S-GW forwards IP packets to and from the RAN, the P-GW is a gateway to mobile cellular network that looks like any other internet gateway router. It provides *NAT services* and support access-related functions like *policy enforcement, traffic shaping and charging*. When devices move to one base station to another, also the S-GW gateway is involved in this process. The details of those processes will be detailed later in this chapter.
A simplified view of the *Mobile Core* architecture is pictured in 10.34

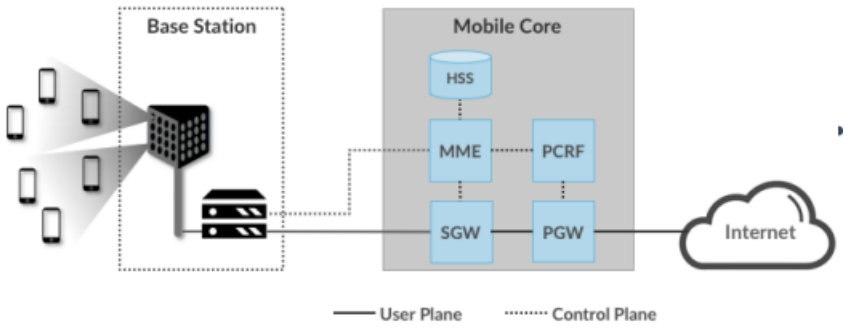


Figure 10.34: Mobile Core architecture

For example, a single **MME/P-GW** pair might serve a metropolitan area, with *S-GWs* deployed across 10 edge sites spread throughout the city, each of which serves 100 base stations.

In 4G there is a strong separation between **data plane** and **control plane** that allows the mobility management and the boostrapping of the data path, as shown in 10.35.

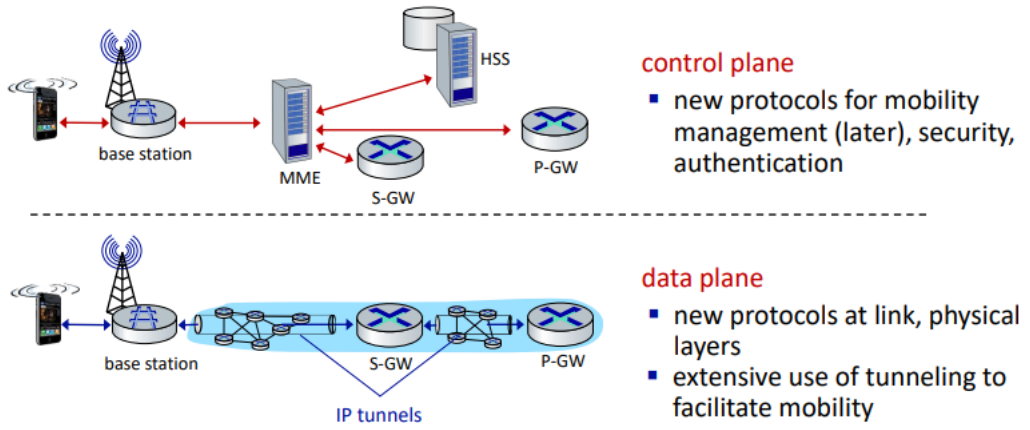


Figure 10.35: Data plane and control plane decoupling

The **base station** forwards both control and user plane packets between the mobile core and the user. The separations concerns:

- ***Control plane***: the packets are tunneled over **SCTP/IP (Stream Control Transport Protocol)** that is an altenative reliable transport to TCP, tailored to *carry signaling (control)*. It allows to provide user's device authentication, registration and mobility tracking.
- ***User plane***: packets are tunneled over **GTP (General Packet Radio Service) Tunneling Protocol**.

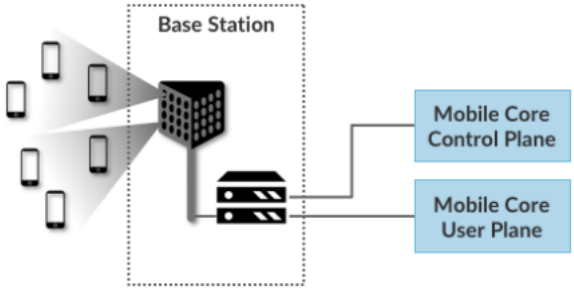


Figure 10.36

The Long Term Evolution (LTE) **data plane protocol stack** in the *core network* adopt, as already mentioned, the **tunneling mechanism** (*as pictured in 10.37*): mobile datagrams are encapsulated using GPT and sent inside a UDP packet to the Serving Gateway (S-GW). The S-GW re-tunnels the UDP datagrams to P-GW[2], allowing reaching internet networks.
The tunneling mechanism described allow supporting *mobility*: only tunneling endpoints change when mobile user moves. In this context the S-GW represent a **fixed point** both from the device to internet and viceversa so if the device change base station, the S-GW is able to *retrieve* the new base station of the device and change the IP address to deliver the packet to the correct base station in which the device is connected. This mechanism allows to implement *mobility across base stations*.

LTE: Link Layer Protocol For the **first hop**, the **LTE Link Layer protocols** (10.38) in the IP range defines:

- **Packet Data Convergence**: contains header for compression, decompression, encryption and decryption.
- **Radio Link Control (RLC)**: implement fragmentation/re assembly by ensuring reliable data trasnfer.
- **Medium Access**: requesting and use of radio *transmission slots*

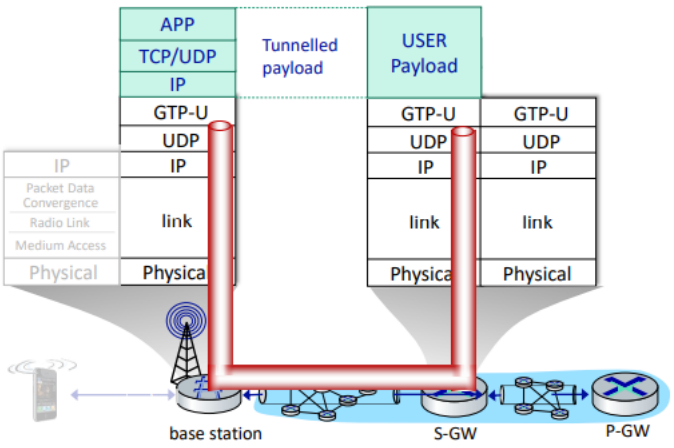


Figure 10.37: LTE tunneling across layers

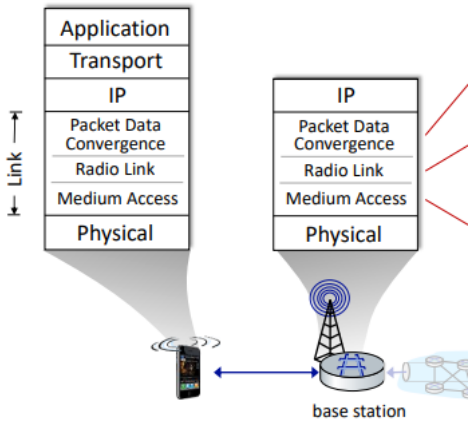


Figure 10.38: LTE link layer first hop

Between the *LTE radio access network* we have several mechanism like:

- **Downstream channel**: combination of FDM and TDM within frequency channel (*OFDM - Orthogonal Frequency Division Multiplexing: orthogonal refer to minimal interference between channels.*)
- **Upstream**: FDM, TDM similar to OFDM. Each **active mobile device allocated two or more 0.5 ms timeslots in or more channel frequencies**. The scheduling algorithms is not standardized so it's up to the operator and allows up to 100's Mbps per device.

10.3.2 Associating with a base station

The association process is briefly sketched in 10.39 with the following steps involved:

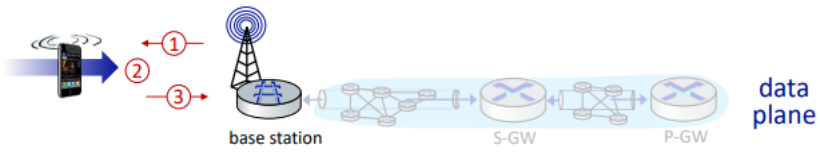


Figure 10.39: Process of association with a base station

1. *Base Station (BS)* broadcast primary synchronization signal every 5 ms on all frequency

[2]PDN-GW or P-GW both indicates the Packet Data Network Gateway

2. *Mobile finds a primary synch signal and locate the second synch signal on this frequency*: mobile then finds info broadcast by BS like channel bandwidth and configuration. Mobile may get informations from multiple base stations, multiple cellular networks.

3. Mobile selects which Base Station to associated with (*e.g. prefererence for home carrier*)

4. More steps till needed to authenticated, establish state and set up control plane.

The **sleep modes** of LTE allows to put radio to *sleep* to conserve battery:

1. *Light Sleep*: afet 100 millisecond of inactivity: need to wake up periodically to check downstrema transmissions.

2. *Deep Sleep*: after 5-10 seconds of inactivity then the mobile may change cells while deep sleeping so need to re-establish association

10.3.3 5G architecture

The main goal of 5G is to increase in peak bit rate, $10x$ decrease in latency and $100x$ increase in traffic capacity over 4G. The **5G New Radio Frequencies** have two frequency bands, respectively at $450MHz - 6GHz$ and $24GHz - 52GHz$.
The 5G is not backwards-compatible with 4G: the *millimiter wave frequencies* have much higher data rates but over shorter distances because uses **pico-cells** that have a diameter of 10 to $100m$. This implies a dense deployment of *new base station*.

The underlying idea is to realize the functions of authentication, tracing and so on by means of **microservices** implemented throught ***Network Function Virtualization*** *(refer to Chapter 12)*: this allows the deploy of those functions both close to the core network and base stations so the path between the mobile device and the functions/services is really close, reducing the latency and load balancing the traffic. This also avoid traffic directly flow into the core network and manage the traffic at the edge.

5G User Plane

The 5G user plane is sketched in 10.40

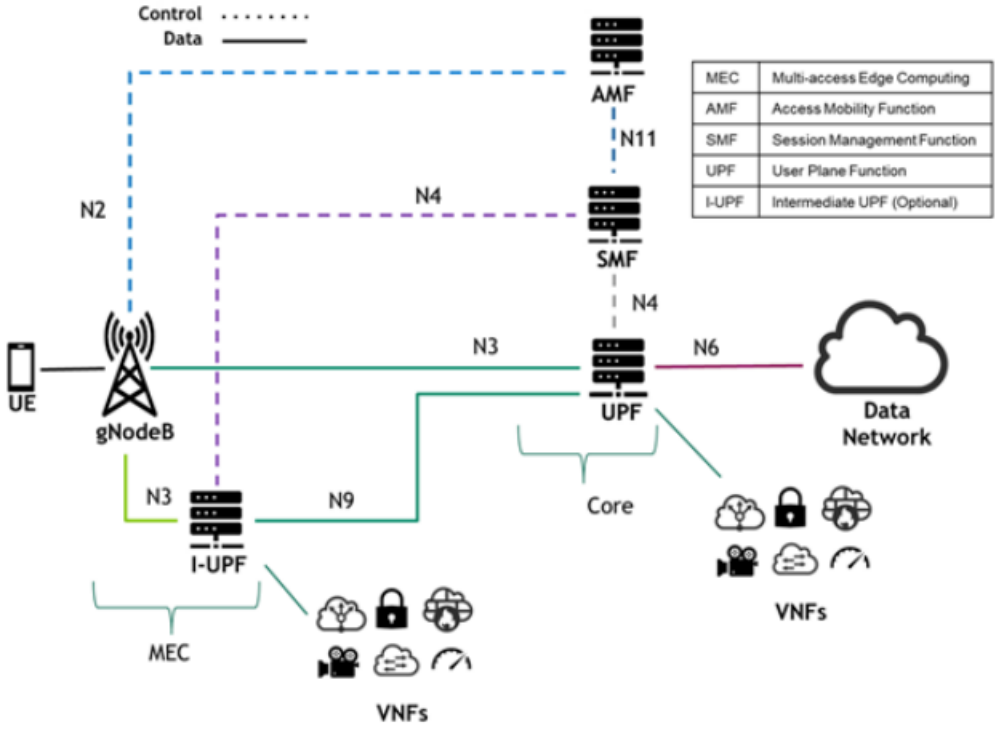


Figure 10.40: 5G user plane simplified

The **UPF - User Plane Function** allows to forward traffic between RAN and internet, corresponding to the S-GW/P-GW combination into the **EPC - Evolved Packet Core**. It's also responsible for: *packet inspection and application detection, QoS management, Traffic usage reporting.*
Those components allows flexibility to deliver user plane functionality at the edge as well as the network core because UPF can be co-located with edge and central data centers at both locations, implementing also **MEC - Multi Access Edge Computing**.

5G Control Plane

The basic 5G architecture schema is sketched in 10.41. The main components that provide the enhanced services are:

- ***AMF (Core Access and Mobility Management Function)***: connection and reachability management, mobility management, access authentication and authorization, and location services
- ***SMF (Session Management Function)***: manages each UE session, including IP address allocation, selection of associated UP function, control aspects of QoS, and control aspects of UP routing.
- ***PCF (Policy Control Function)***: manages the policy rules that other CP functions then enforce.
- ***UDM (Unified Data Management)***: Manages user identity, including the generation of authentication credentials.
- ***AUSF (Authentication Server Function)***: Essentially an authentication server.
- ***SDSF (Structured Data Storage Network Function)***: a "helper" service used to store structured data.
- ***UDSF (Unstructured Data Storage Network Function)***: A "helper" service used to store unstructured data.
- ***NEF (Network Exposure Function)***: A means to expose select capabilities to third-party services
- ***NRF (NF Repository Function)***: A means to discover available services.
- ***NSSF (Network Slicing Selector Function)***: A means to select a Network Slice to serve a given UE. Network slices are essentially a way to partition network resources in order to differentiate service given to different users

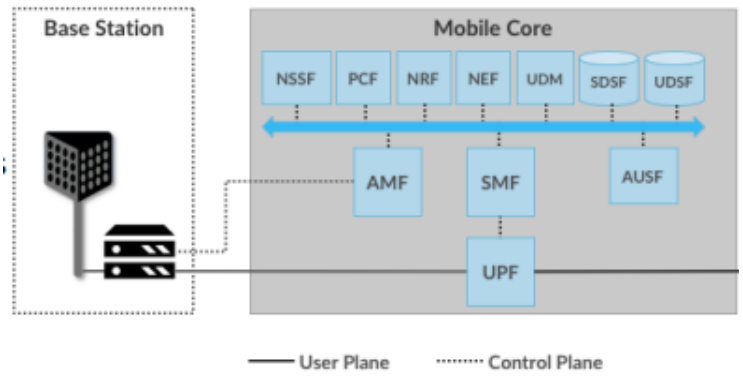


Figure 10.41: 5G Control Plane simplified

10.3.4 Deployment options

Three main deployment options are considered, the two most used are pictured in 10.42:

1. **Standalone (SA) 4G / SA 5G**: SA 4G operates independently using the LTE standard, while SA 5G uses the 5G NR standard and the 5G core network architecture. Deployment involves network planning, RAN and core network deployment, testing, and commercial launch. SA 4G and SA 5G provide significant benefits but require significant investment and infrastructure upgrades.
2. **Non standalone (NSA 4G+5G BSs) over 4G's EPC**: upper part of 10.42 in which 5G base stations are deployed alongside existing 4G base stations ina given geography to proide a data-rate and capacity boost. In NSA, the control plane traffic between user equipment and the 4G Mobile Core utilizess 4G base stations and the 5G base stations are usd only to carry user traffic.
3. **Non standalone (NSA 4G+5G BSs) over 5G's NG-Core**: combines 4G and 5G network technologies over a 5G New Radio (NR) infrastructure connected to the 5G Next-Generation Core (NG-Core) network. The NSA deployment model involves network planning, RAN and core network deployment, testing, and commercial launch. 5G NR is used for the data plane, while the control plane is handled by the 4G Evolved Packet Core (EPC). The NSA model provides a transitional approach for network operators to upgrade their infrastructure gradually and take advantage of the benefits of 5G technology while still supporting legacy 4G networks.

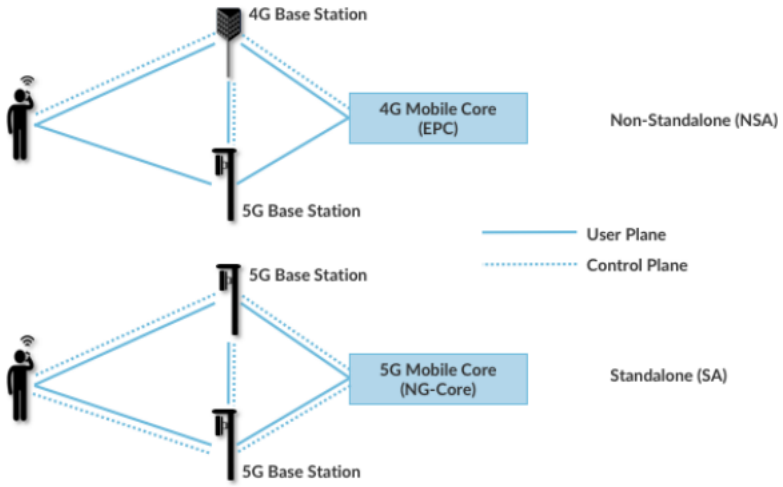


Figure 10.42: Non standalone (4G+5G) over 5G's EPC vs over 5Gs NG-Core

10.4 Mobility

From the network perspective, the spectrum of mobility have a wide area of scenarios and application, as shown in 10.43.

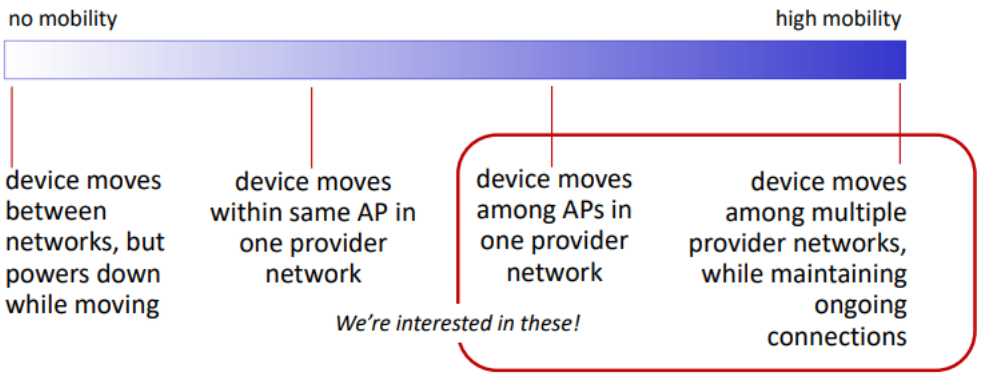


Figure 10.43: Mobility spectrum

The **home network** allows to have a service plan with a cellular provider (*Verizon, Orange*) where the **Home Subscriber Service (HSS)** stores identity and service informations of the device. The **visited network** is any network other than your home network and by service agreement with other networks allows to provide access to *visiting mobile*, implementing mobility. The described scenario is pictured in 10.44.

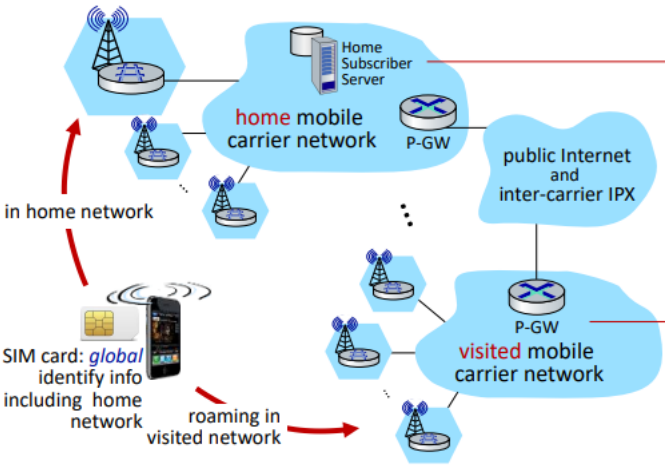


Figure 10.44: Flow between home network and host network for mobility purposes

Wifi/ISP does not have a concept of Home network where the user informations are stored but they are stored on the device or with the user (eg. username, password), different networks different credentials (*like academic network "eduroam"*).
To implement mobility and have a correspondence between the mobile device and the home network/visited network we have two different approaches:

1. **Network router handle it**: the network router handle the management of the mobility. The routers advertise well-known name, addresses (*e.g. permanent 32-bit IP address*) or number (*phone cell*) of visting mobile node via usual *routing table exchange*. Internet routing could do this already without changes: routing tables indicate where each mobile is located by using **Longest Prefix Match**. This approach is not scalable to a billions of mobile devices: we could have routing tables too large to expand and retrieve data (*considering 32 bit IP addresses*).

2. **End-system handle it**: the funcionality is provided at the *edge*. There are two types of routing:

   (a) ***Indirect routing***: communication from correspondent to mobile goes through home network and then forwarded to remote mobile (*connected to a thid-party visited network*). The correspondant, knowning the home network gateway, send the request and thanks to the **HSS - Home Subscriber Server** that contains the updated address of the device, is able to forward the request to the Serving Gateway of the *visited network*. Refer the schema in 10.45.

   (b) ***Direct routing***: correspondents get foreign address of mobile, sending directly to mobile device. The *IMSI - International Mobile Subscriber Identity* is used as a sort of *MAC address* of the device. Refer the schema in 10.46.

In the ***indirect routing*** a ***mobile-device-to-visited network association protocol*** is needed: the mobile device will need to be associated with the visited network and will similarly need to be disassociated when leaving the visited network. There is also the need of a ***visited-network-to-home-network-HSS registration*** to allows visited network to register mobile device's location with the HSS in the home network.

A **datagram tunneling protocol** between the home network gateway and the visited network gateway router is needed to allows home gateway performs encapsulation and forwarding of the correspondents original datagram and, on the receiving side, the gateway router performs decapsulation, NAT translation and forwarding of the original datagram to the mobile device.

In the ***direct routing*** the process is less transparent to the correspondent because it must get care of the address from home agent. In case the mobile changes visited network the request sending can be

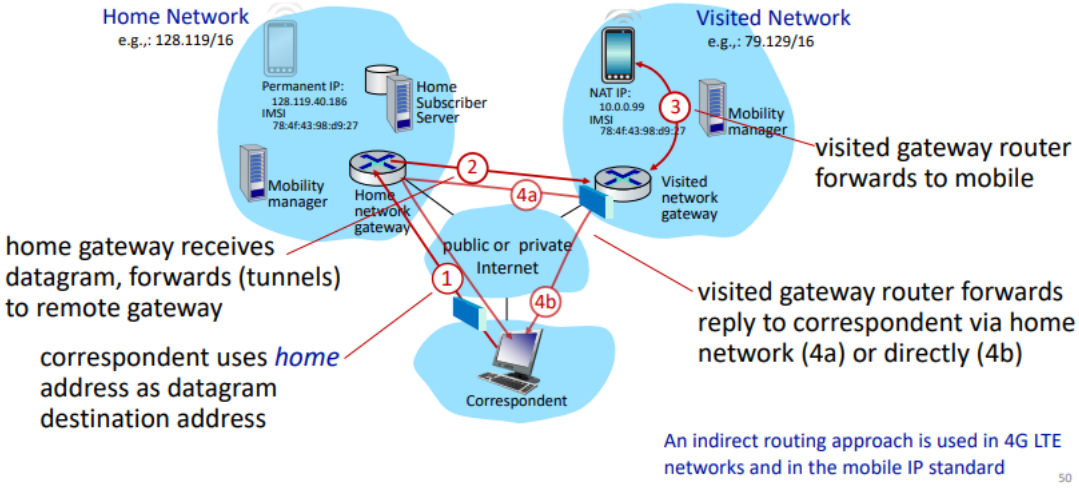


Figure 10.45: Indirect routing schema

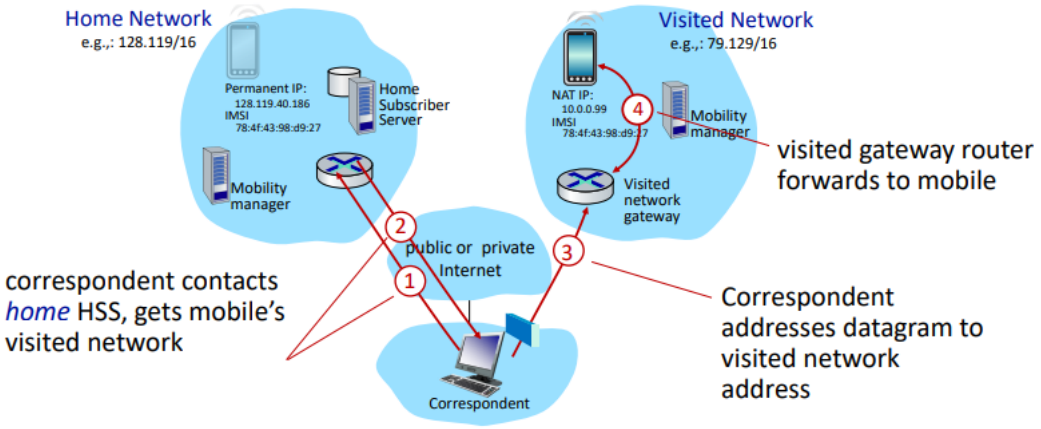


Figure 10.46: Direct routing schema

handled but with additional complexity by allowing the HSS to be queried by the correspondent only at the beginning of the session.
This last approach require a **registration** procedure to be sure that the home network know where the device is, as pictured in 10.47.

In **Direct routing** the mobile device needs an IP address in the *visited network*:

1. permanent address associated with the mobile device's home network
2. New address in the address range of the visited network
3. IP address via NAT

The end result is that the visited mobility manager known about the mobile and the home HSS knowns the location of mobile.

In case on **indirect routing**, the different scenario is pictured in 10.45:

1. The correspondent use `home` address as datagram destination address
2. The home gateway receives datagrams and forwards (*by tunneling*) to remote gateway
3. Visited gateway router forwards to mobile
4. The visited gateway router forwards reply to correspondent via home network **(a)** or directly **(b)**.

As mentioned, the main requiremenets for mobility in ***indirect routing*** are:

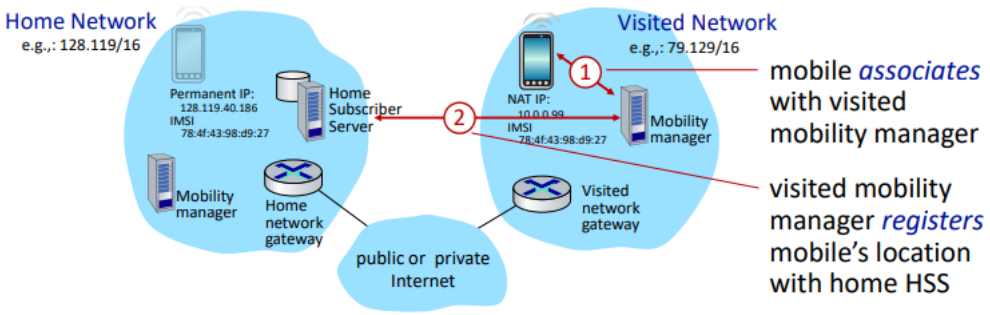


Figure 10.47: Direct routing registration process

1. *Mobile-device-to-visited-network association protocol*: the mobile device will need to associate with the visited network and will similarly need to dissociate when leaving the visited network.

2. *Visited-network-to-home-network HSS registration*: The visited network will need to registrer the mobile device's locatio with the HSS in the *mobile device home network*

3. *A datagram tunneling protocol between home network gateway and visisted network gateway router*: the home gateway perform encapslation 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. Some considerations about ***indirect routing*** evaluate the problem of **triangle routing** that results in an inefficient model when correspondent and mobile are in the same network: the forwarding of a messages to the home network and re-routing them to the correspondent's original network which correspond to the visisted network of the destination device result in high inefficies and avoidable transmission overhead.
The mobile device can see an interrupted flow of datagrams as it moves between network, losing some datagrams but the on-going TCP connection between correspondet and mobile can be mantained. The following section 10.4.1 describe the steps involves to allows an interrupted stream of datagrams. So if the mobile changes visited network it's necessary to update the HSS in the home network, changing the tunnel endpoint to terminate at the gateway router of the new visited network.

Regarding the mobility with ***direct routing***, it overcomes triangle routing ineffiencies at cost of be *non-trasparent to correspondent* because he need to get care of the address from home agent. For this type of routing, if the mobile changes visited network, the HSS is queried by the correspondent only at the beginning of the session, requiring additional complexity.

10.4.1 Mobility in 4G

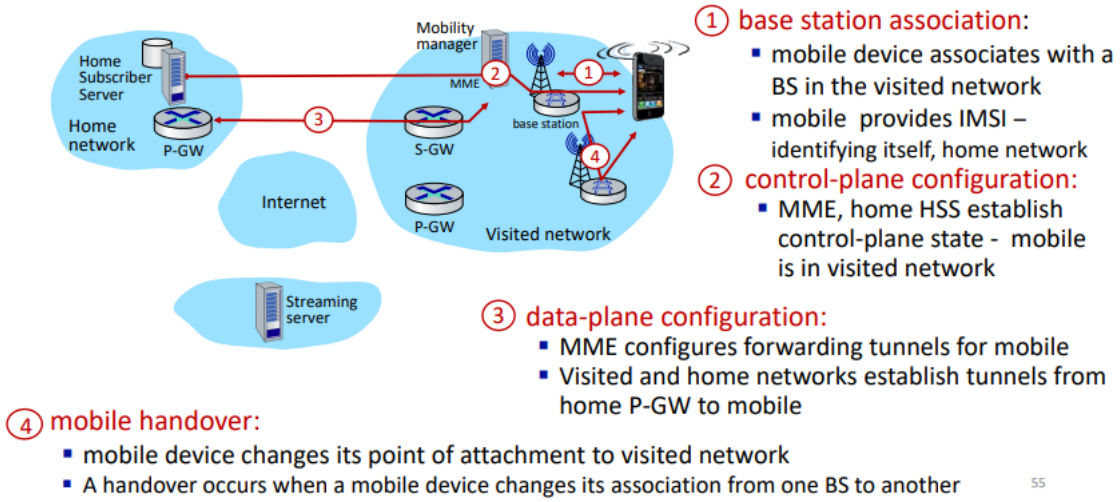


Figure 10.48: 4G Network's Mobility tasks

Mobility management over 4G network allows the mobile to communicate witht he local *MME - Mobile Management Entity* via a BS (*Base Station*) control-plane channel. The overall process is pictured in 10.48 and require the following steps:

1. Base Station association It's needed to perform the association with a base station in the *visited network*, providing also the IMSI to allow identify the source home network.

2. Configuring the LTE control-plane elements The mobile communciates with local MME via BS control-plane channel. The MME uses mobile's previously registered IMSI information to contact mobile's home HSS to retrieve authentication, encryption and network service information and, in return, the home HSS knowns the mobile now reside in the visited network. The BS and mobile select parameters for BS-mobile and data-plane radio channels.

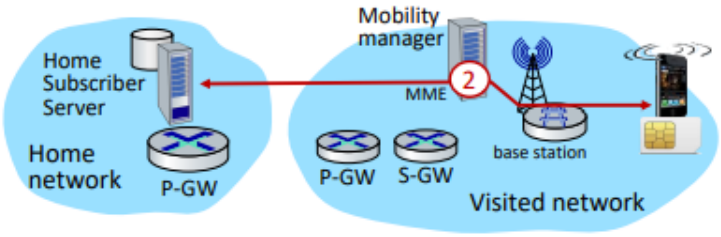


Figure 10.49: Step 2

3. Configuring data-plane tunnels for mobile The MME configures the data plane for the hosted mobile device so all traffic to/from the mobile device will be *tunneled* through the device's home network. Two different tunnels are established 10.50:

1. **S-GW to BS tunnel**: when the mobile changes base station, simply change the endpoint IP addres of tunnel
2. **S-GW to home P-GW tunnel**: to support the *indirect routing*

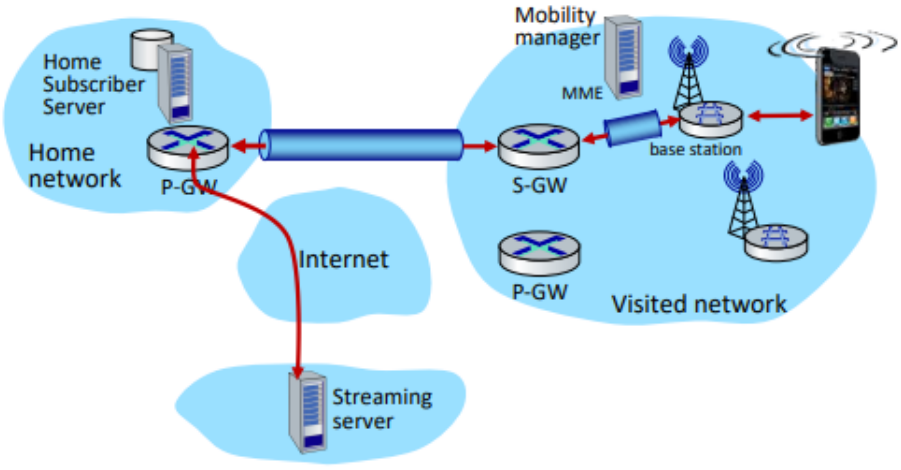


Figure 10.50: S-GW to P-GW tunnel across networks and S-GW to BS internally

The **tunneling via GTP (*GPRS - General Packet Radio Service tunneling protocol*)** allows the mobile's datagram directed to the streaming server to be encapsulated using GTP.

4. Handover between BSs in same cellular network The ***handover (also known as hand-off)*** is the process of transferring an ongoing call or data session from one base station (BS) to another base station while maintaining the continuity of the communication. Handovers are necessary when the

mobile device moves from one cell to another, and the signal strength of the current base station becomes too weak to maintain the connection.

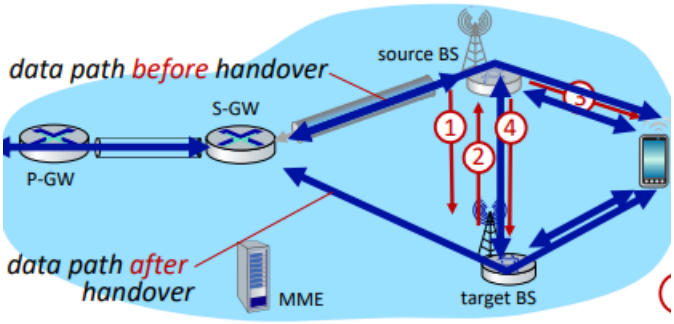


Figure 10.51: Handover workflow steps from 1 to 4

The steps are:

1. Current source BS chooses to select a handover *(e.g. due to cell overloading, signal deterioration between mobile and BS)*: selects target BS, sends `Handover Request message` to target BS
2. Target BS pre-allocates radio time slosts, responds with `HR ACK` with information for mobile
3. Source BS informs mobile of new BS: mobile can now send via new BS. The handover looks complete to the mobile
4. Source BS stops sending datagrams to mobile, instead forwards to new BS (*who forwards to mobile over radio channel*).

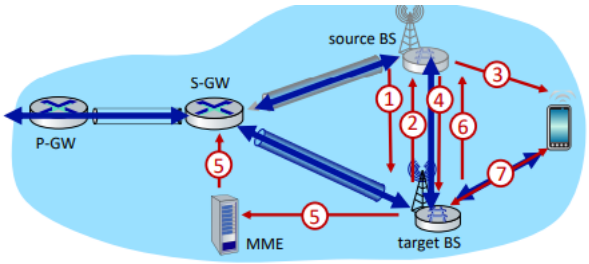


Figure 10.52: Handover workflow steps from 5 to 7

5. Target BS informs MME that it is new BS for mobile: MME instructs the S-GW to change tunnel endpoint to be new target BS
6. target BS ACKs back to source BS: handover compelte, source BS can release resourcees
7. Mobile's datagram now flow through new tunnel from target BS to S-GW

Mobile IP

The mobile IP architecture use *indirect routing* to node via home networks and using tunnels. Operates by two agents:

1. *Mobile IP Home Agent*: combined roles of 4G HSS and home P-GW
2. *Mobile IP Foreign Agent*: combined roles of 4G MME and S-GW The protocols for agent discovery in visited network, registration of visited location in home network via ICMP extensions.

Chapter 11

Software Defined Networking (SDN)

The **network layer** carried out two main functions:

- **Forwarding**: move packets from router's input to the appropriate router output
- **Routing**: determine route taken by packet from source to destination

The forwarding activity is implemented by the ***data plane***, the routing is implemented by the ***control plane***. Respectively, both planes operates on fast timescale (*per-packet in case of forwarding*) and slow time scale (*per control event in case of routing*).

Two main approaches exists to structuring the network control plane: the **per-router**, which is the classical one, and the **logically centralized control**, which is the basic building block of **Software Defined Networking**.

Per-router control plane

In this approach, there is an *individual routing algorithms* component in each router and every router interact with the control plane to compute the forwarding tables as pictured in 11.1. In traditional IP networks, the control and data plane are ***tightly coupled***, embedded in the same networking devices and the whole structure is highly *decentralized*.

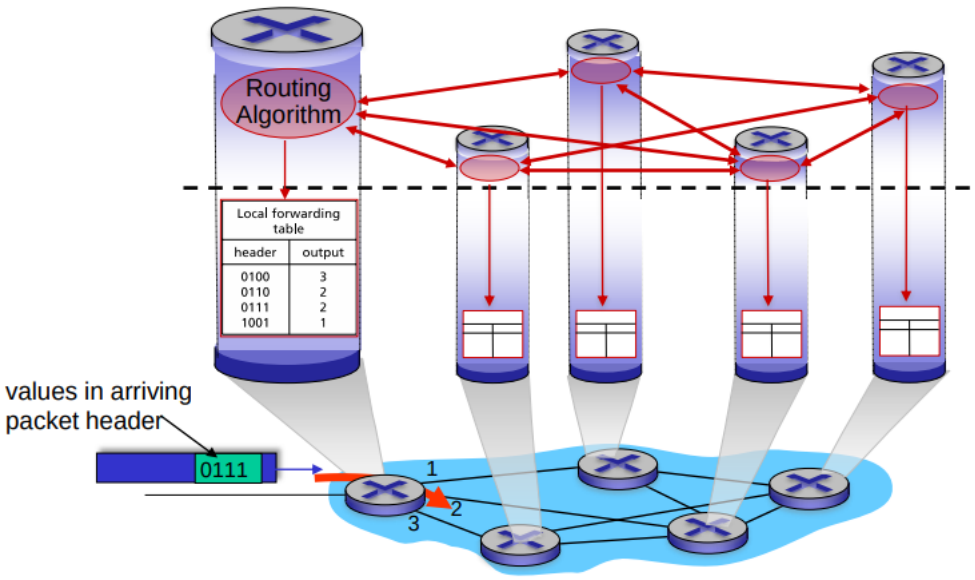


Figure 11.1: Per-router control plane

Software-Defined Networking control plane

In this approach, the ***remote controller*** computes and installs the forwarding tables in each router, as pictured in 11.2.

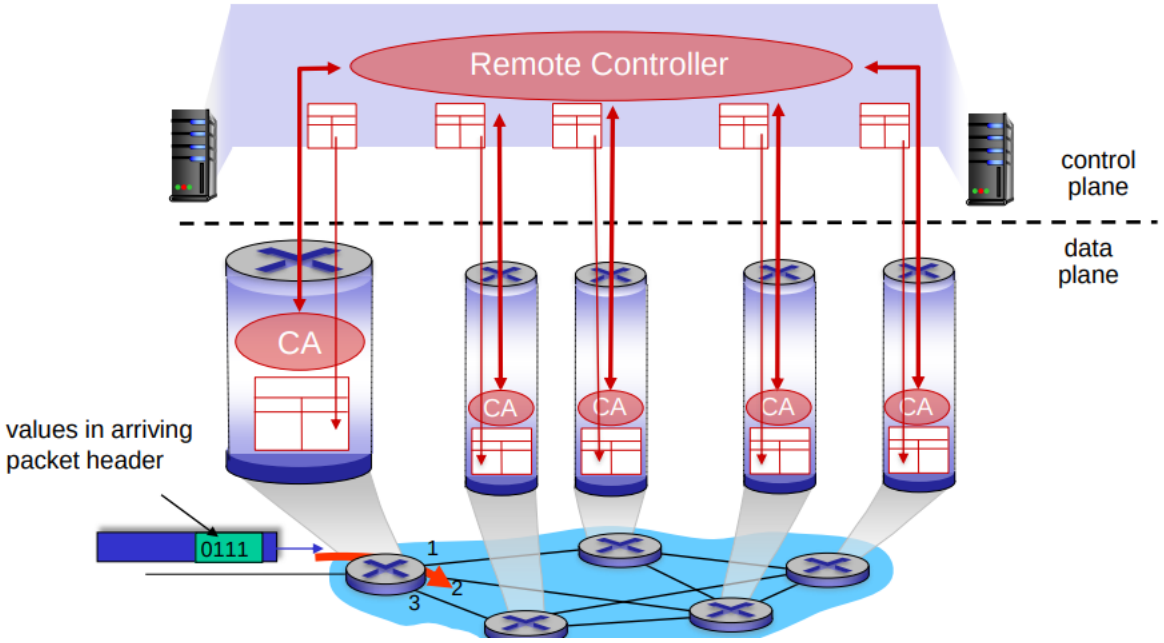


Figure 11.2: SDN Control plane

The rationale behind a **logically centralized** control plane reside in the fact that is easier to carry out management network functions by *avoiding router misconfiguration* and gaining greater flexibility of traffic flows. The so called ***table-based forwarding*** (*e..g OpenFlow API*) allows programming routers in different ways:

1. *Centralized programming easier*: allows to compute tables centrally and distribute them among routers
2. *Distributed programming more difficult*: compute tables as a result of distributed algorithm implemented in each and every router

There are some difficult regarding the **traffic engineering** with a traditional router. For example, let's consider those scenarios and relatives implications:

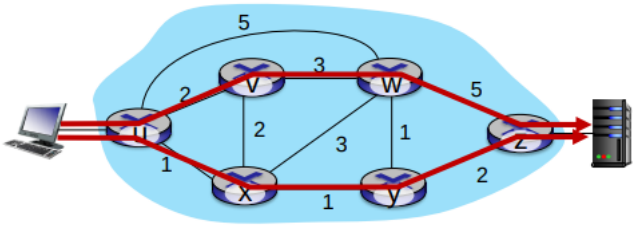


Figure 11.3: Scenario1: node `w` on the left and node `z` on the right

- Scenario 1: *What if the network operator want* `u-to-z` *traffic to flow along* `uvwz` *rather than* `uxyz`*?* There is the need to re-define link weights so traffic routing algorithm computes routes accordingly: the tuning of the weight does not provide enough control on the routing itself. Refer the diagram in 11.3

- Scenario 2: *What if network operator wants to split* `u-to-z` *traffic along* `uvwz` *and* `uxyz` *(load balancing)?* It's not possible or at lest require a new routing algorithm. Refer the diagram in 11.4.

- Scenario 3: *what if w wants to route blue and red traffic differently from w to z?* Not possible. Refer the diagram in 11.5.

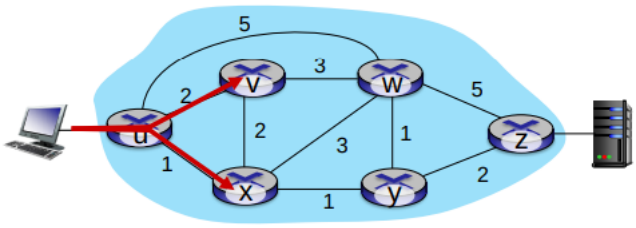


Figure 11.4: Scenario 2: load balacing the traffic among two different path

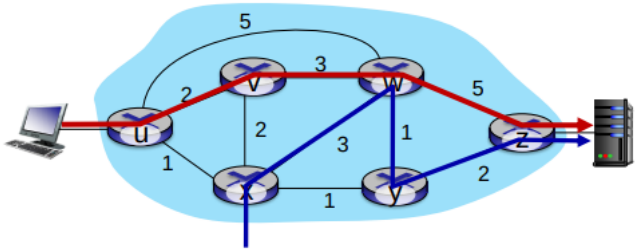


Figure 11.5: Scenario 3: routing red/blue traffic choosing different routes

11.0.1 SDN Architecture

The main three components are 11.7:

1. **Data plane switches**: they implement generalized data-plane forwarding in hardware. The forwarding table is already computed, installed under the supervision of the controller. They provide also API for table-based switch control (*like OpenFlow*) and provide a protocol for communicating with the controller. They are *dummy* devices, without added complex and routing mechanisms to be computed.

2. **SDN Controller**: mantain the network state infromation and interact with the network control application above via the *northbound* API and the network switches via the *southbound API*. It's implemeted as distributed system to guarantee scalability, performance and fault-tolerance. It separates the control and the data plane (*as opposed to classical routing*), representing the *Network Operating System* because perform routing decision based on different customizable criteria.

3. **Network control apps**: they implement the control function (*like routing, access control and load balancing*) using lower-level services, using the APIs provided by the SDN Controller. This layer can also be *unbundled* so provided by third-party supplier thus can have different routing vendor or SDN controller, customizable thanks to the northbound/southbound API exposure. Specifically, the northbound API provides a deeper abstraction respect to southbound API (*mainly OpenFlow*).

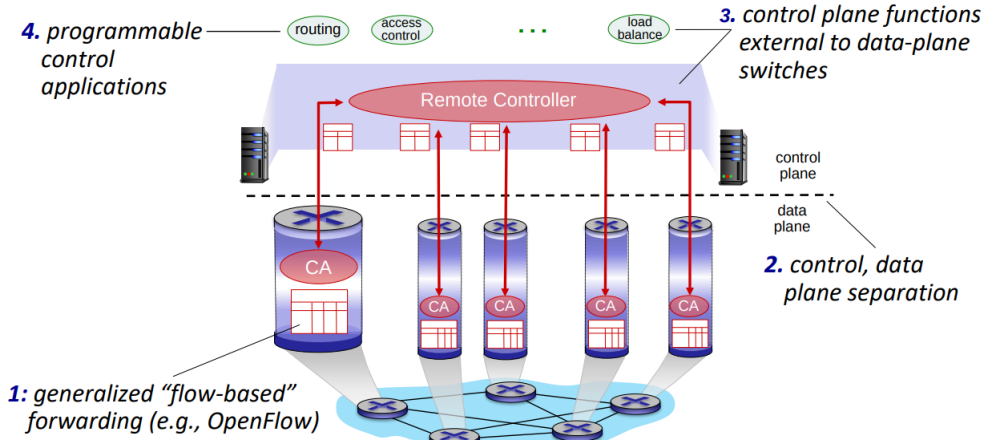


Figure 11.6: Overview of logically centralized control plane features

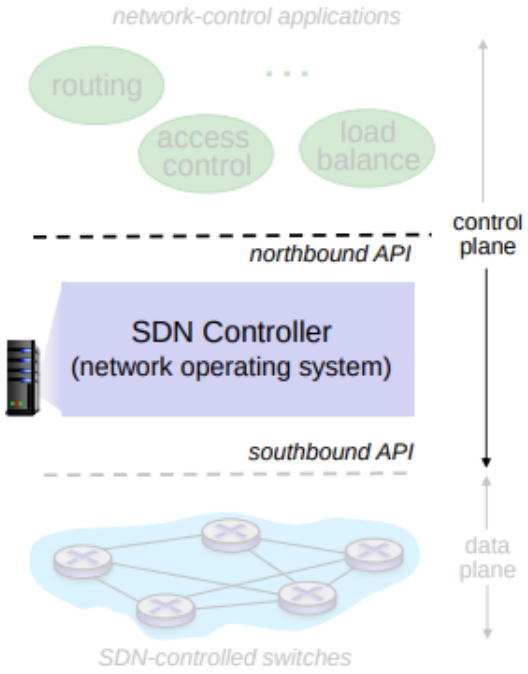


Figure 11.7: SDN three-tier architecture

11.1 Data Plane

The SDN data plane represent the resource or **infrastructure layer made by forwarding devices**: it allows to perform the transport and processing of data according to decisions made by the SDN control plane. It perform those functions without embedding software, implementing an autonomous decision mechanism. The data plane can have virtual switches or physical switches, as sketched in 11.8.
The main function of the SDN data plane is to forward packets between network devices based on instructions provided by the SDN controller. The controller communicates with the data plane switches using a standardized protocol called *OpenFlow*, which allows the controller to program the switches to perform specific actions based on network conditions.

Some of the key functions of the SDN data plane include:

1. **Packet forwarding**: The data plane switches are responsible for forwarding packets between network devices based on the instructions provided by the controller. They also need to be able to forward packets at high speeds, without causing bottlenecks or delays. It accepts incoming data flows from other network devices and end-systems: forwards them along the data forwarding paths computed and established according to the rules defined by the SDN applications. The *control support function* include the interaction with the SDN controller for management of forwarding rules (*the reference standard specification is mainly* ***OpenFlow Switch Protocol***).

2. **Traffic shaping**: The data plane can implement traffic shaping policies to manage the flow of traffic through the network. For example, it can prioritize certain types of traffic, such as voice or video, over other types of traffic to ensure that they are delivered without delay.

3. **Security**: The data plane can implement security policies to prevent unauthorized access to the network. For example, it can filter out packets that do not meet certain criteria, such as those coming from a known malicious source.

4. **Quality of Service (QoS)**: The data plane can implement QoS policies to ensure that certain types of traffic are given priority over others. For example, it can ensure that real-time applications like voice and video are given higher priority over other types of traffic.

5. **Load balancing**: The data plane can implement load balancing policies to distribute traffic across multiple network paths. This helps to prevent congestion and ensure that traffic is delivered efficiently.

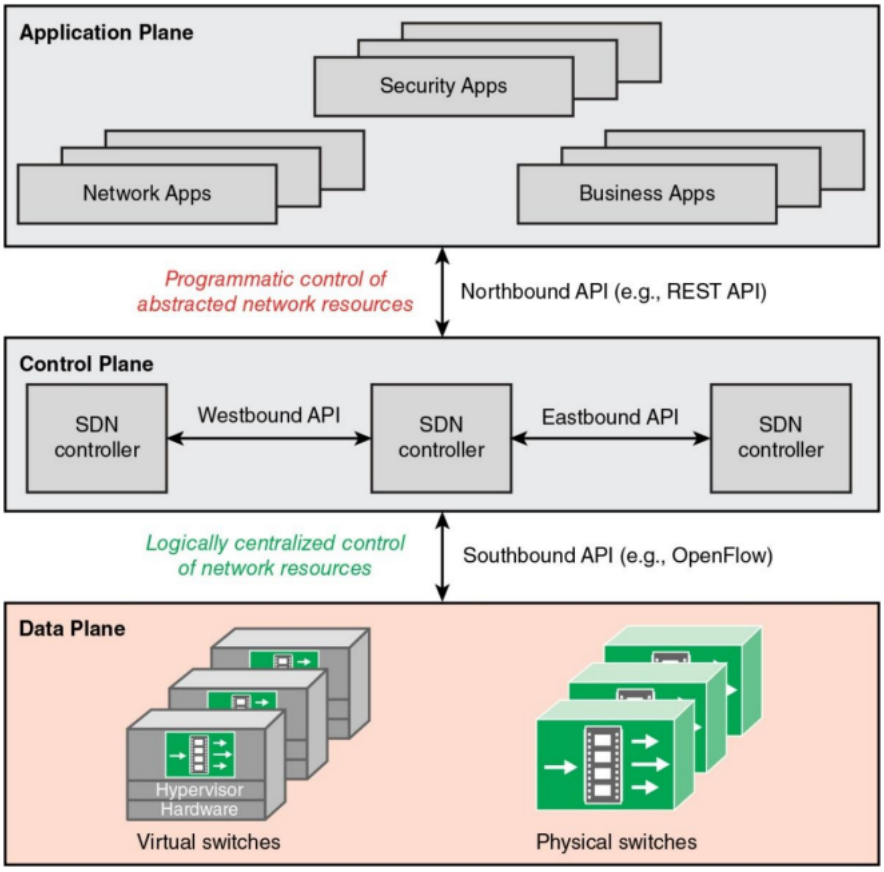


Figure 11.8: SDN layers interactions

Referring 11.8, the **data plane** consists of physical switches and virtual switches. In both cases, the switches are responsible for forwarding packets. The internal implementation of buffers, priority parameters, and other data structures related to forwarding can be vendor dependent. However, each switch must implement a model, or **abstraction**, of packet forwarding that is uniform and open to the SDN controllers.
This model is defined in terms of an open application programming interface (API) between the control plane and the data plane (**southbound API**). The most prominent example of such an open API is OpenFlow. As explained later, the OpenFlow specification defines both a protocol between the control and data planes and an API by which the control plane can invoke the OpenFlow protocol.

Controllers use information about *capacity* and *demand* obtained from the networking equipment through which the traffic flows. SDN controllers also expose **northbound APIs**, which allow developers and network managers to deploy a wide range of off-the-shelf and custom-built network applications, many of which were not feasible before the advent of SDN. As yet there is no standardized northbound API nor a consensus on an open northbound API. A number of vendors offer a *REpresentational State Transfer (REST)-based API* to provide a programmable interface to their SDN controller.

At the **application plane** are a variety of applications that interact with SDN controllers. SDN applications are programs that may use an *abstract view* of the network for their decision-making goals. These applications convey their network requirements and desired network behavior to the SDN controller via a northbound API. Examples of applications are energy-efficient networking, security monitoring, access control, and network management.
The **generalized forwarding** operated by the data plane consists in providing each router a **forwarding table (*flow table*)** that use the *"match plus abstraction" (see 11.1 for OpenFlow specific implementation of this abstracted model)*: match bits in arriving packets and take the corresponding action. Two main forwarding techniques are used:

1. **Destination-based forwarding**: forward based mostly on destination IP address
2. **Generalized forwarding (*followed by OpenFlow*)**: many header fields (*e.g. MAC Address,*

destination port, etc) can determine the action and many actions are possible on a given flow/packet like `drop/copy/log packet/modify`. It generalize both `matching` information and `action` to be taken.

The **flow** is defined by the header fields values (*respectively in link-,network-, transport-layer fields*). The generalized forwarding define simple packet handling rules (11.9):

- `match`: pattern values in packet header fields
- `actions`: for matched packet can `drop, forward, modify` or `send` matched packets to controller to decide the action to be performed.
- `priority`: disambiguate overlapping patterns (*flows can match more than one entry so needs disambigution*)
- `counters`: allows to count the number of bytes and packets, mainly for statistics. Those are helpful to identify patterns and derive rules based on traffic volumes and parameters, updating the controller knowledge on the network.

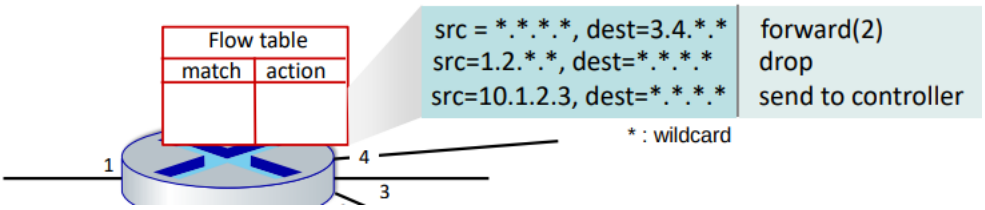


Figure 11.9: Example of flow table with `match/action` field specification

The packet can be sent to the controller only once: the first time a packet is sent back to the switches with informations for future patterns (*e.g. as in the last picture*).

OpenFlow flow table entries

The following figure in 11.10 present the information that can be used at different layer to set the **matching rules** in the flow tables.

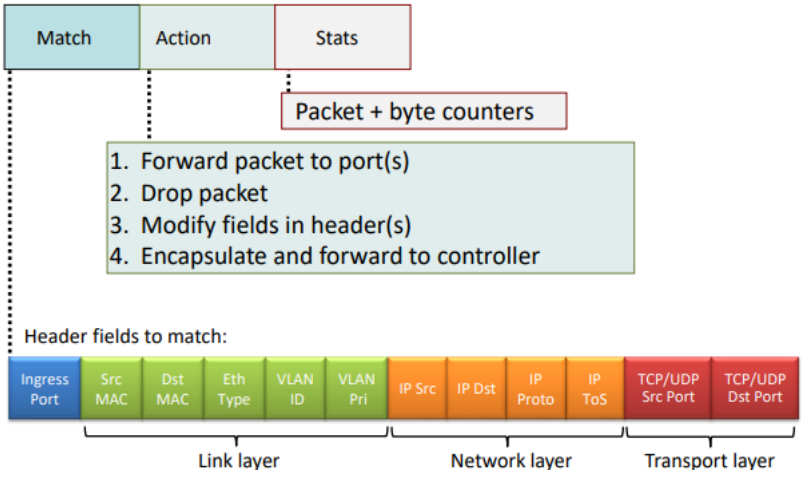


Figure 11.10: SDN Generalized forwarding

OpenFlow abstraction The *match plus action* abstraction allows to unifies different kinds of devices, based on their behavior:

- **Router**: the `match` is the longest IP prefix, the `action` forward out a link
- **Switch**: the `match` is the destination MAC address, the `action` forward or flood

- **Firewall**: the `match` is the IP address and the TCP/UDP port numbers, the `action` is permit or deny
- **NAT**: the `match` is IP address and port, the `action` is to substitute a private IP address with a public IP address. So the boundaries between switches and router's behaviour is blurring.

Example The idea is that the OpenFlow controller have a complete overview of the network, providing the forwarding tables to the switches without them implementing and re-compute the algorithms itself. In the example picture in 11.11, the datagram from host `h5` and `h6` are destined to `h3` or `h4` and should be forwarded via `s1` and from there to `s2`. The routing tables are configured to forward the traffic by passing from `s1`.

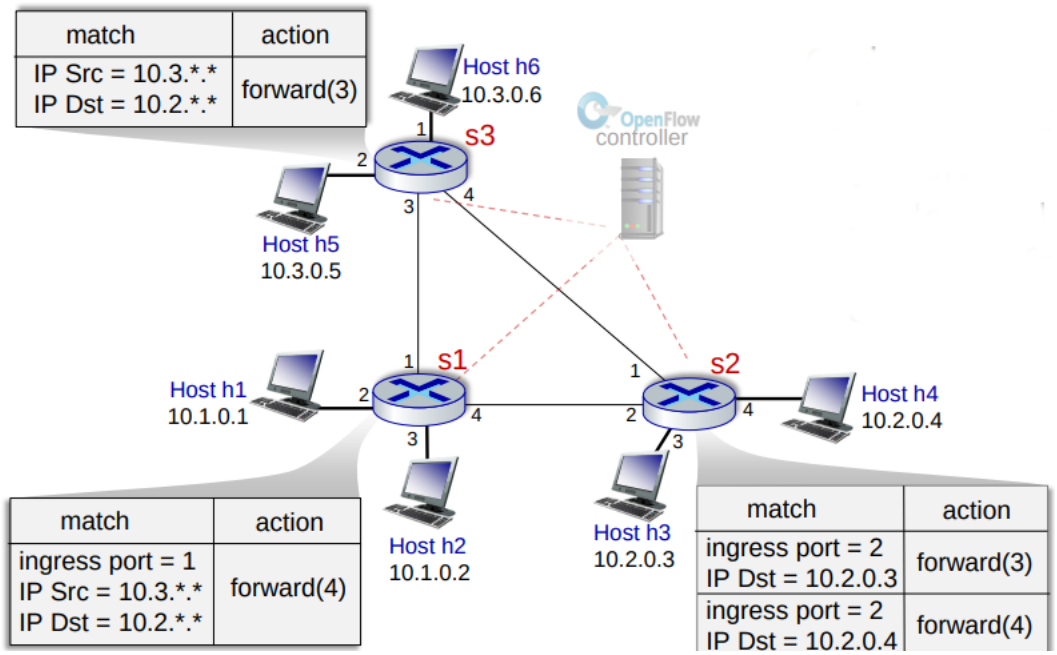


Figure 11.11: The datagram from host `h5` and `h6` are destined to `h3` or `h4` and should be forwarded via `s1` and from there to `s2`. The routing tables are configured to forward the traffic by passing from `s1`

OpenFlow Switch

In a OpenFlow switch there is more than a one flow table, usually the tables are multiple and organized in **pipeline** as showed in 11.12. Every switch can be seen as composed by different components. The **group table** in the *datapath* specify a more complex behavior that refer to a group of flows, specifing rules for group of flows based on their attribute. Referring the *flow table* and *group table*, both allows to specify the data plane behavior. The *control channel* allows to communcate to the control plane: typically can have a connection with multiple controllers as they are implemented in a distributed way. The simplest configuration is the **master-slave** with only one controller while the complex one can include **active-passive controller** for redundancy.[1].

The packet entering the switch are analyzed according to those mentioned tables as sketched in 11.13. The passage through different table accumulate the **actions** associated with a given `match` forming a **pipeline of actions** that will be executed only as final step before the packet is sent out as stated in 11.14. If there is a match on one or more entry in the table, the match is defined to be with the highest priority matching entry. If there is a match only on a table-miss entry, the table entry may contain instructions as with any other entry. In practice, thee **table-miss entry** specifies one of three actions:

1. **Send packet to controller**: this will enable the controller to define a new flow for this and similar packet or decide to drop the packet
2. **Direct packet to another flow table** further down the pipeline

[1]More complex assignation of *switch-controller* can be configured, as described in Scalable OpenFlow Controller Redundancy Tackling Local and Global Recoveries

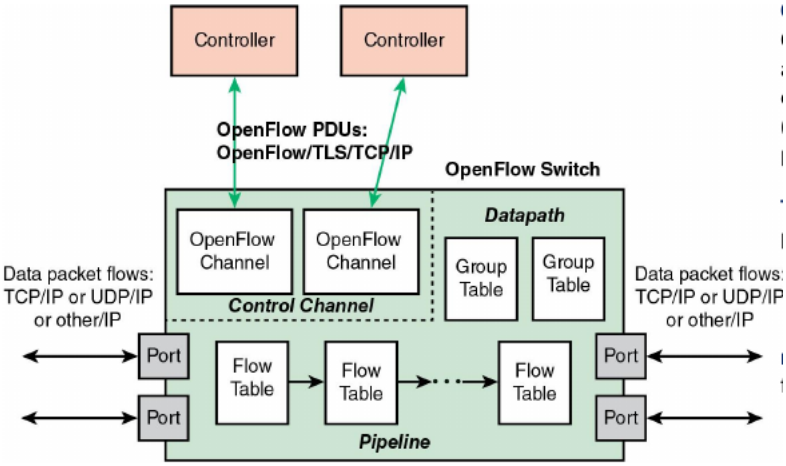


Figure 11.12: OpenFlow Switch Architecture

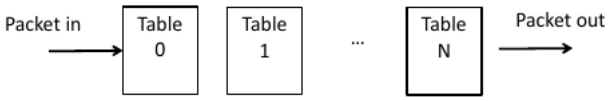


Figure 11.13: Flow table pipelining

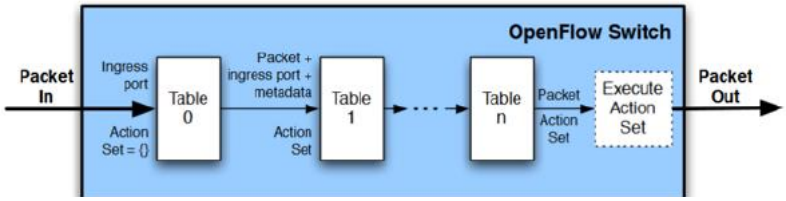


Figure 11.14: Packet forwarding through flow table pipeline, end-to-end process

3. **Drop the packet**: If there is no match on any entry and there is no table-miss entry, the packet is dropped.

11.2 Control Plane

The general structure of the SDN Control plane is sketched in 11.15.

We may distinguish three layers:

1. **Communciation layer**: in the bottom one there is the OpenFlow implementation and the SNMP protocol.

2. **Network-wide state management**: the central part is the core of the controller. It collect and manage the information about the underlying infrastructure (*host information, how many switches, how many ports are active, etc*) and according to those information build the general network overview and extract statistics and manage dynamically flow tables.

3. **Interface layer to network control apps**: In the upper layer there are components that handle the interaction with application like *network graph, RESTful API* to expose the network topology to the application or *intent* that, for example, state at high level *"drop the malicious traffic pattern"* without specifying how: the SDN controller elaborates this intent and traduce this intent in a set of rules to be installed based on the current network topology.

11.2.1 OpenFlow

The **OpenFlow protocol** operates between a *controller* and the *switches*: use TCP to exchange messages (*suggested with TLS*). There are three classes of messages:

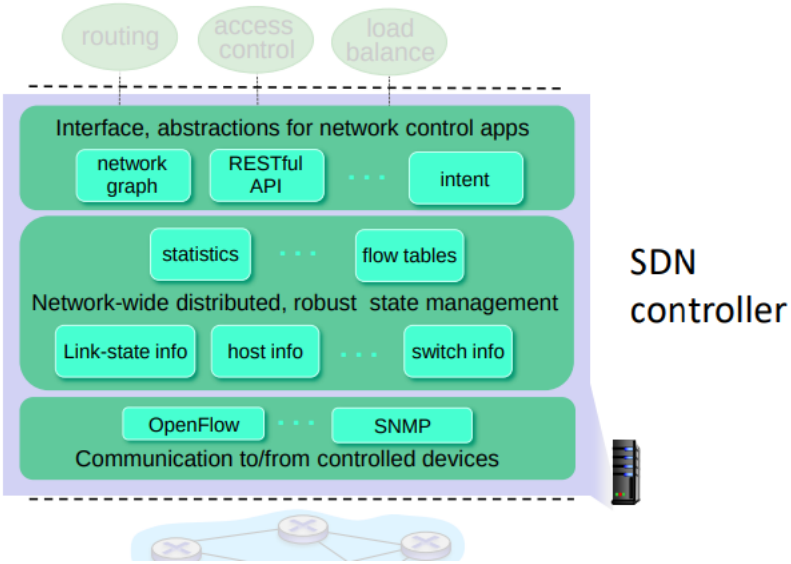


Figure 11.15: SDN Control Plane three-tier overview

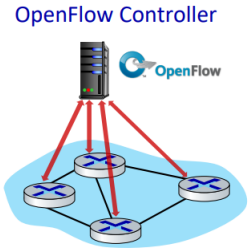


Figure 11.16: OpenFlow controller schema

1. **Controller to switch**: These OpenFlow messages are initiated by the controller and sent to the switch to manage its behavior. Examples of **controller to switch** messages include `packet-out` messages to send packets to a specific port or set of ports, `flow-mod` messages to add, modify or delete flow entries in the switch's flow table, and `barrier-request` messages to synchronize the switch's processing with the controller.

2. **Asynchronous (Switch to controller)**: These OpenFlow messages are initiated by the switch and sent to the controller to notify it of events that occur on the switch. Examples of asynchronous messages include `packet-in` messages that inform the controller of packets that match a flow entry with the *"send to controller"* action, `port-status` messages that notify the controller of changes in the state of switch ports, and `error` messages that report errors encountered by the switch.

3. **Symmetric (both directions)**: These OpenFlow messages can be initiated by either the controller or the switch and are sent in both directions to maintain the state of the OpenFlow channel. Examples of symmetric messages include `echo-request` and `echo-reply` messages to test the connectivity and responsiveness of the channel, and features-request and features-reply messages to exchange information about the switch's capabilities and configuration.

The main ***controller-to-switch*** messages can be of the following type:

- `features`: controller queries switch features, switch replies
- `configure`: controller queries/sets switch configuration parameters
- `modify-state` (*FlowMod*): the controller ask the switch to add, modify, delete an entry in the routing rule table
- `packet-out`: controller can send this packet out of specific switch port. The controller instruct the switch to implement a specific action for a specific pattern.

The main ***switch-to-controller*** messages can be:

- `packet-in`: transfer packet to controller. For this packet the switch delegate the control to the controller.
- `flow-removed`: the switch inform the controller that has deleted an entry in its flow table
- `port-status`: inform the controller of a change on a port

Usually network operators don't *program* switches by creating/sending OpenFlow message directly but use higher-level abstraction at the controller.

Control/data plane interaction example

The interaction between the SDN control plane and data plane consists of 6 steps, as pictured in 11.17.

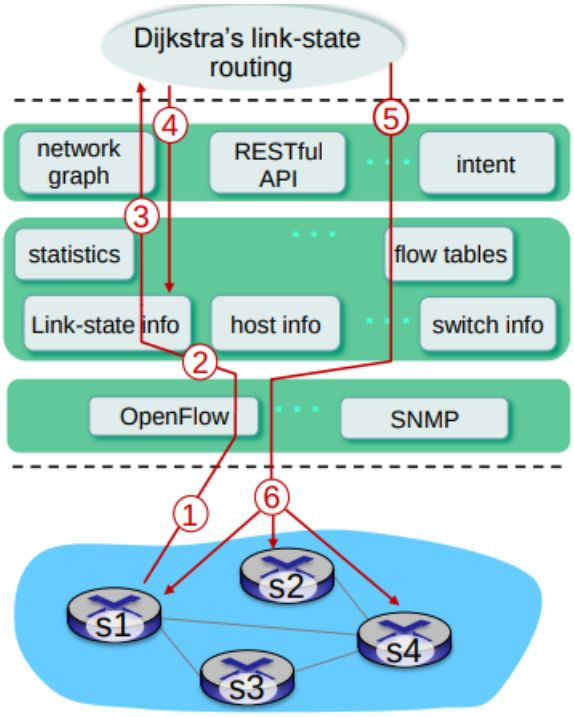


Figure 11.17: SDN Data Plane-Control Plane workflow

1. $S1$ is experiencing link failure so uses OpenFlow `port-status` message to notify the controller
2. SDN Controller receive the message and updates link status info
3. Dijkstra's routing algorithm application has previosuly registered to be called whenever link status change so it's called
4. Dijkstra's routing algorithm access the network graph information, link state information in the controller and computes new routes
5. Link state routing app interacts with flow-table-computation component in the SDN controller, which computes new flow tables needed
6. Controller uses Openflow to install new tables in switches that need updating by issuing a `flow-mod` message to switch

11.3 Topology discovery and forwarding

Traditonally the **routing** function is distributed among the routers in a network: in an **SDN controlled network** it make sense to centralize the routing action within the SDN controller. The controller can develop a *consistent view* of the network state for calculating shortest paths and can implement application-aware routing policies. **Data plane switches** are relieved of the processing and storage burden associated

with routing, leading to improved performances because **centralized routing** application performs two different functions:

1. ***Link/Topology discovery***: routing function needs to be aware of the links between data plane switches. The topology discovery in OpenFlow is not fully standardized.
2. ***Topology manager***: mantains the topology information for the network, computing routes in the network like the shortest path between two data plane nodes or between a data plane node and a host.

The **Topology Discovery** is implemented by exploiting two major *initial configuration* features of Openflow (*OF*) switches:

1. every OF switch has initially set the IP address and TCP port of a controller to establish a connection as soon the device is turned on
2. switches have pre-defined behavior which support by implementation the support of the topology discover. It's supported by preinstalled flow rules to route directly to the controler via a `Packet-in` message any message of the **Link Layer Discovery Protocol (LLDP)**.

11.3.1 LLDP - Link Layer Discovery Protocol

It's a **neighbor discovery protocol** of a single jump: it advertises its identity and capabilities and receive the same information from the adjacent switches. It's a vendor neutral protocol (*defined by IEEE*) ad operates at **layer 2** of OSI Model. In OpenFlow network, switches send the LLDP messages to discover the underlying topology as a direct request of the controller.

LLDP uses *Ethernet* as its transport protocol, thus the Ethernet type for LLDP Is `0x88cc`. An LLDP frame contains the following fields (11.18):

- **Chassis ID (Type 1)**: contains the identifier of the switch that sends the LLDP packet.
- **Port ID (Type 2)**: contains the identifier of the port through which the LLDP packet is sent.
- **Time to Live (Type 3)**: time in seconds during which the information received in the LLDP packet is going to be valid.
- **End of LLDPDU (Type 4)**: indicates the end of the payload in the LLDP frame

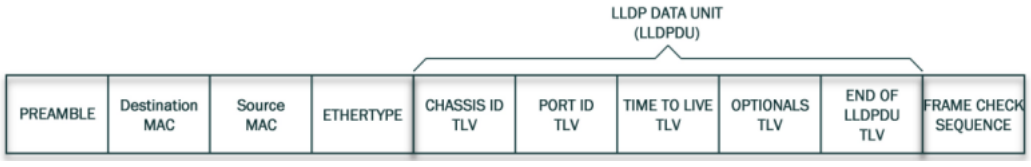


Figure 11.18: LLDP Fields

Switch initialization

The initialization under the *LLDP* protocol, for the pictured scenario in 11.19, implies the need of the controller to understand the **circular topology** of the switches.

When the switch is initialized:

- Establish a connection with the controller
- The controller send a message *FEATURE REQUEST MESSAGE* to the switch
- The switch responds with a message *FEATURE REPLY MESSAGE*
- It informs the controller of relevant parameter for the discovery of the links like the `Switch ID` and a list of `active ports` with their respective MAC associate, among other. At the end of the initial handshake, the controller knows the exact number of active ports on the OF switches **but** it doesn't know the physical connections between switches so it needs to build it by performing *Topology Discovery*

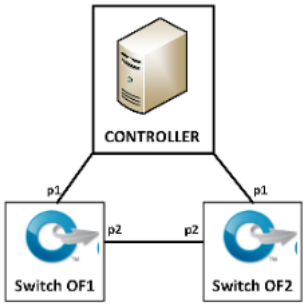


Figure 11.19: Controller-aware of network topology

The **topology discovery** it's based on the information obtained from the initial handshake so the controller known the exact number of active ports:

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 are 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

The controller at the end known the `Switch ID` and `Port ID` in the LLDP message and `Switch ID` and `Port ID` in metadata (*from which the switch received the LLDP message*). This process is repeated for every OF switch on the network: the entire process of discovery is performed periodically. In a network of S switches interconnected by a set of links L, the total of packet-out message that the controller sends out to the network to discover all existing link between OF switches with P active port is:

$$Total_{packet-out} = \sum_{i=1}^{S} P_i \tag{11.1}$$

Let's clarify with an example pictured in 11.20:

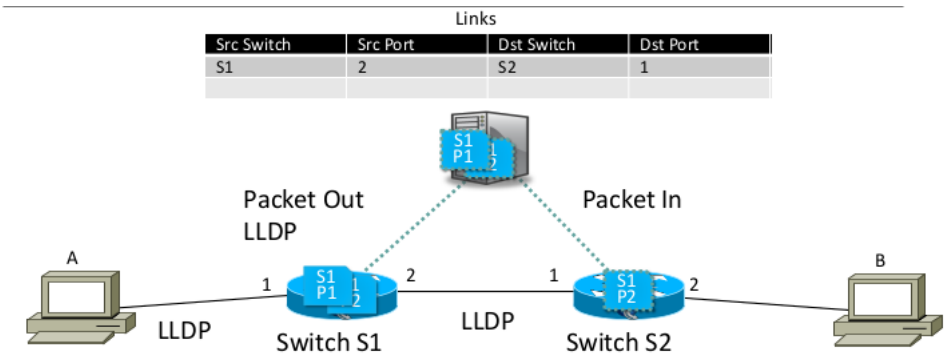


Figure 11.20: Topology Discovery Scenario

The controller send a **Packet-Out** to $S1$: the switch send the packet $S1-P1$ to host A which it's not an LLDP device so did not answer, while send the LLDP packet $S1-P2$ to switch $S2$. The switch $S2$ add its metadata information like `Switch ID, Port ID` and send the **Packet-In** message to the controller that now knows that $S1$ and $S2$ are adjacent because the packet sended by $S2$ contains the information that the initial LLDP packet was received from $S1$ via port 1_{S2} .

The topology discovery itself does not able both the controller and the switches to identify which hosts are reachable. The controller can learn that there is a **reachable host** in the network (*that is not an OF switch, so do not reply to OF packets*): the discovery is triggered by **unknown traffic entering** the controller's network domain either from an *attached host* or from a *neighboring router*. (*For unknown we mainly refer to not LLDP packets*).
For the **path computation**, initially the flow tables on all switches are *empty*, but assume that **Host table** and **Topology** at the controller level are fully populated with the necessary knowledge of the network and host.
Refer the scenario in picture 11.21: when the switch receive a packet from the host `A` send the packet to the controller so it can compute the path and send the `Flow Mod` to update the flow table of the switches along the path from `A` to `B`. Based on the assumption that the controller already have all the knowledge about the host table and topology, the $S1$ and $S2$ flow table are updated as in figure.

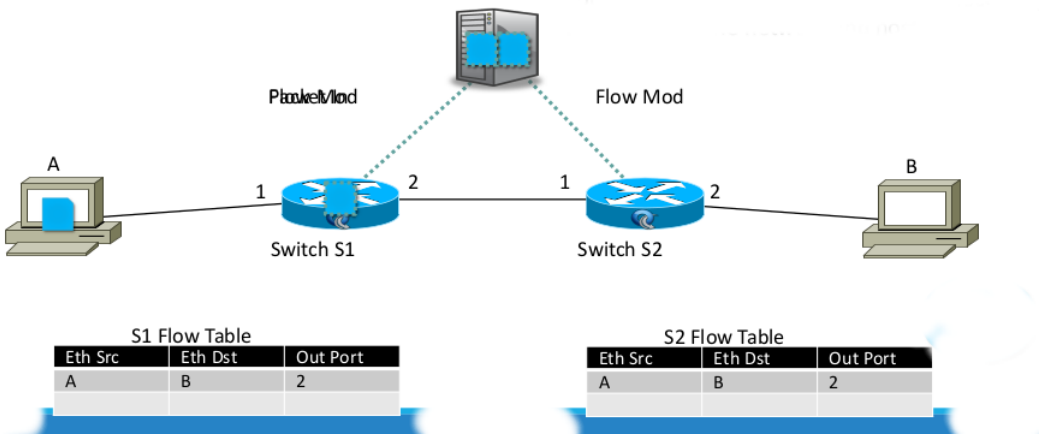


Figure 11.21: LLDP Path Computation

These Flow Mod packets (*both from the controller to $S2$ and $S1$*) will be sent to each switch in **reverse order**, from the switch closest to the destination and so on until the source switch: the rationale behind this order is to **avoid the scenario in which multiple switches trigger the path computation to the controller** while there is the flow table updating operation ongoing by the controller.

All future packets from this flow will match directly at each switch and no longer need to take a trip up to the controller.

Exercise Describe the source, destination and semantics of the following OF messages:

- packet-in
- packet-out
- FlowMod:

Chapter 12

Network Function Virtualization

Within SDN, **Network Function Virtualization (NFV)** allows *programmable networking*: it's mainly used in 5G and datacenter, allowing managing traffic with different requirements by performing load balancing, security, etc. The global traffic trend have increased within the user requirements: more data and rapidly changing services involves also an increase in *CapEx*. To give a context, in traditional telco industry this increase was based on deploying physical equipment for each function that is part of a given service: this has led to *long product cycles, very low service agility and depencencies on specialized hardware (invest money and time to confgiure that must last years to not sustain useless costs).*
A **typical enterprise network** is pictured in 12.1.

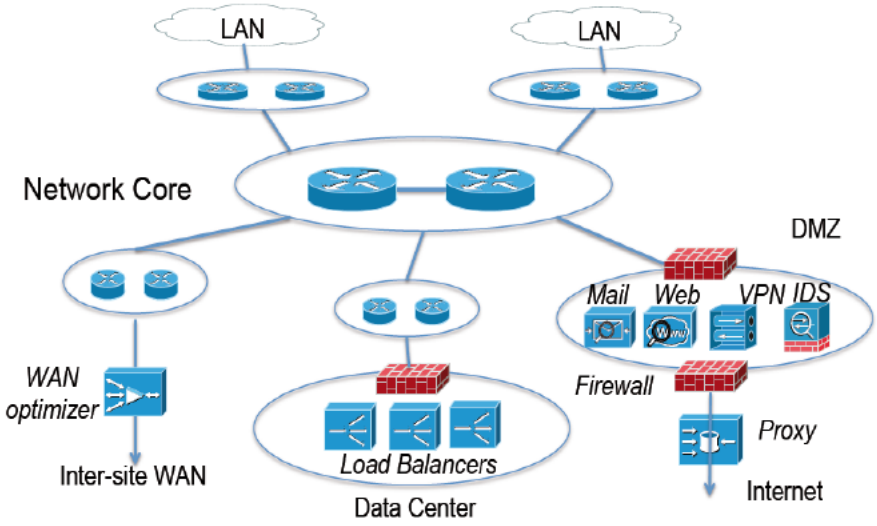


Figure 12.1: Generic enterprise architecture

This network contains beside all the routing gear, the *load balancer* connected to the data center or the *DMZ* that provide computing for third services inside the enterprise network: those function manage flows for internal purpose (*IDS, VPN, Mail, etc*). Those last functions are good candidates to be virtualized by NFV.
Virtualize this function represent an advantage because they're usually in high number inside an enterprise network [1].

The complexity of internet arises, shifting from end to end device to a multiple type of device as an intermediary between two end nodes. So we can state the problem as *increasing demand* from users and short-lived service with high data rates: this implies the need to purchase and operate physical equipment, obtaining a dense deployment of network equipment. The telco operator progressively shifted from hardware pure based solution to software-based solution.

[1] (*taken from https://doi.org/10.1145/2070562.2070583*)

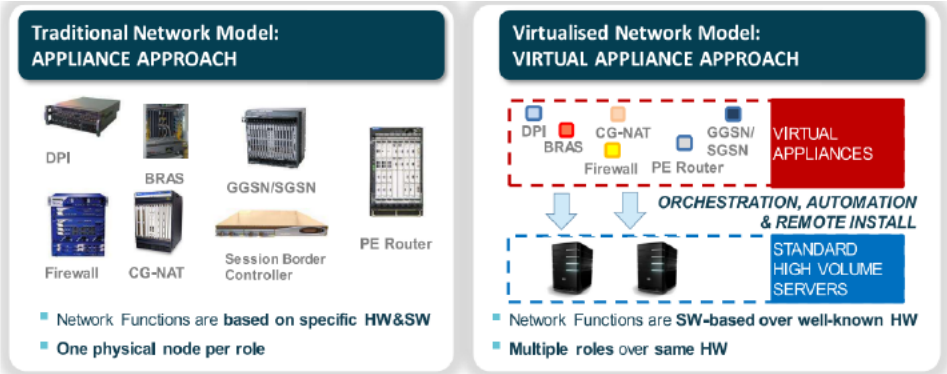


Figure 12.2: Approach evolution from physical devices to virtualized services

12.0.1 Overview

NFV address the previous problem by leveraging virtualization technologies: allows to **decouple hardware from sofware, allowing to migrate and execute network function where needed**. A **Network Service** can be decomposed into a set of **Virtual Network Functions (VNFs)**. The picture 12.2 shows the shift that the virtualization achieve.

The main **use cases** for NFV are the one where data plane packet processing and control plane in mobile and fixed network, like:

- **Switching elements**: BNG, CG-NAT, routers
- **Mobile network nodes**: HLR/HSS, MME, SGSN, GGSN/PDN-GW, RNC, Node B, eNodeB
- Functions contained in home routers and set top boxes to create virtualized home environments
- **Tunneling gateway elements**: IPsec/SSL VPN gateways
- **Traffic analysis**: DPI, QoE measurement
- **Service Assurance, SLA monitoring, Test and Diagnostics**
- **NGN signaling**: IMS
- **Converged and network-wide functions**: AAA servers, policy control and charging platforms
- **Application-level optimization**: CDNs, Cache Servers, Load Balancers, Application Accelerators
- **Security functions**: Firewalls, virus scanners, intrusion detection systems, spam protection

Use case: vEPS

A *virtual Evolved Packet System* is a third-generation (3G) broadband, **virtual** packet-based transmission of text, digitized voice, video, and multimedia at data rates up to 2 megabits per second (Mbps) which offers a consistent set of services to mobile computer and phone users, no matter where they are located in the world. It allows computers and phones to be constantly attached to the Internet wherever they travel (roam) with access through a combination of terrestrial wireless and satellite transmissions.

In 12.3, both the **core (*EPC - Evolved Packet Core*)** and the **RAN (Radio Access Network)** can be virtualized, this allows to improve the network usage efficiency and guarantee higher service resiliency thanks to the flexibility allocation of different NFs on the hardware pool and by providing dynamic network configuration. The following scheme shows how the shift from the hardware-based functions is performed by virtualize and splitting the function across different locations that can be connected by IP network, even if they're located in different datacenters, allowing a **virtualized RAN**.
In traditional networks, all devices are deployed on proprietary/closed platforms. All network elements are enclosed boxes, and hardware cannot be shared: each device requires additional hardware for increased capacity, but this hardware is idle when the system is running below capacity.

With NFV, however, network elements are **independent applications** that are flexibly deployed on

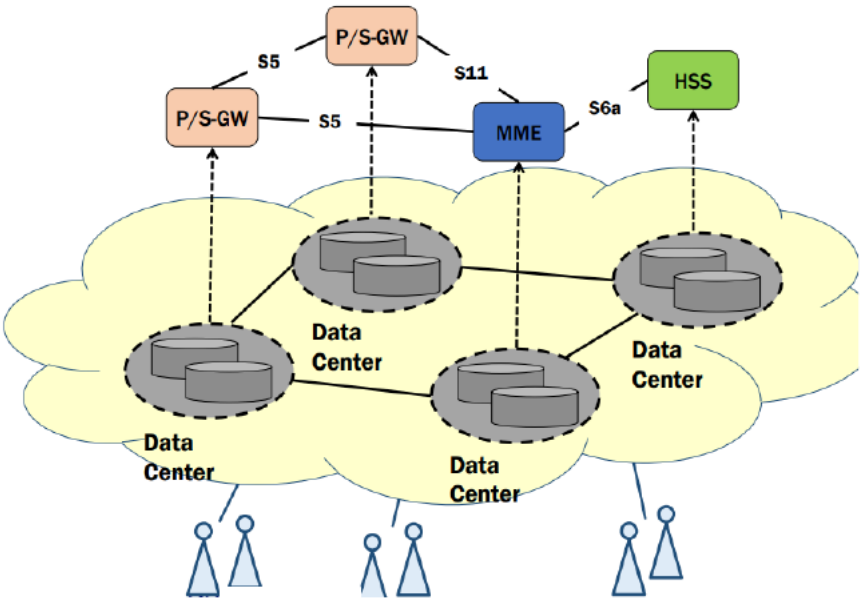


Figure 12.3: Coordination to access virtualized functions across sites

a unified platform comprising standard servers, storage devices, and switches. In this way, software and hardware are decoupled, and capacity for each application is increased or decreased by adding or reducing virtual resources.
The cons consists in having **overhead** respect to the specialized hardware that can have **optimized performance** respect to a general device server.

12.0.2 Network service

Usually a telco operator need a **chain of the network function**, being able to differentiate the treatment between each flow, as described in 12.4

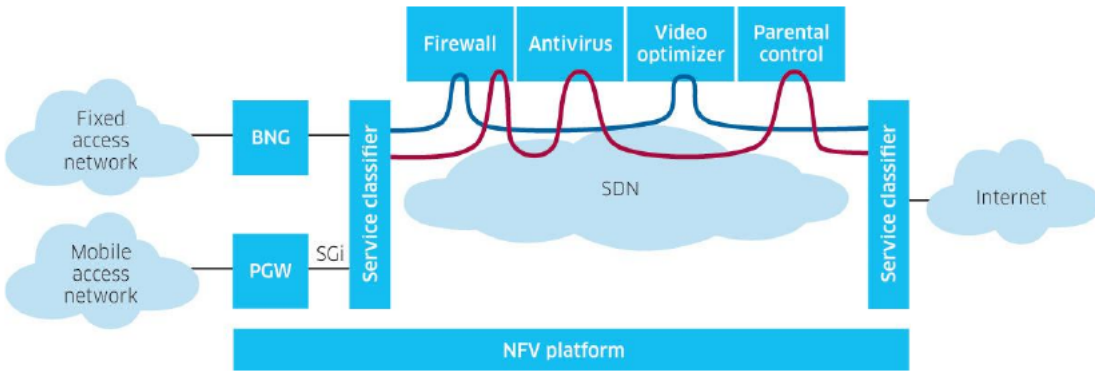


Figure 12.4: Network function chaining

The pictured feature is called **Network service chaining**, also known as **Service Function Chaining (SFC)** (*see IEEE RFC7665*) and uses SDNs capabilities to create a chain of connected network services, such as L4-7 services like firewall, NAT and intrusion protection.
Network operators can use network service chaining to set up suites or catalogs of pathways for traffic travel through the network. Any path can consist of any combination of connected services depending on the traffic's requirements. Different traffic may require diffent levels of *security, lower latency, or quality of service (QoS)*.
The primary advantage of network service chaining is to automate the way virtual network connections can be set up to *handle traffic flow* for connected services.

For example, an SDN controller could take a chain of services and apply them to different traffic flows depending on the source, destination, or type of traffic. The chaining capability automates what traditional network administrators do when they connect up a series of physical L4-7 devices to process incoming and outcoming network traffic, which traditionally may require a number of manual steps.

A *network service* in NFV is an **end-to-end network service** that can be defined as ***forwarding graph of network functions and endpoints/terminals***, basically *what* an operator provides to customers. A network service can be viewed architecturally as a *forwarding graph* of *Network Functions (NFs)* interconnected by the supporting network infrastructure: these network functions can be implemented in a *single operator network* or by *interwork between different operator networks*. The underlying network function behaviour contributes to the behaviour of the higher-level service. An example is pictured in 12.5.

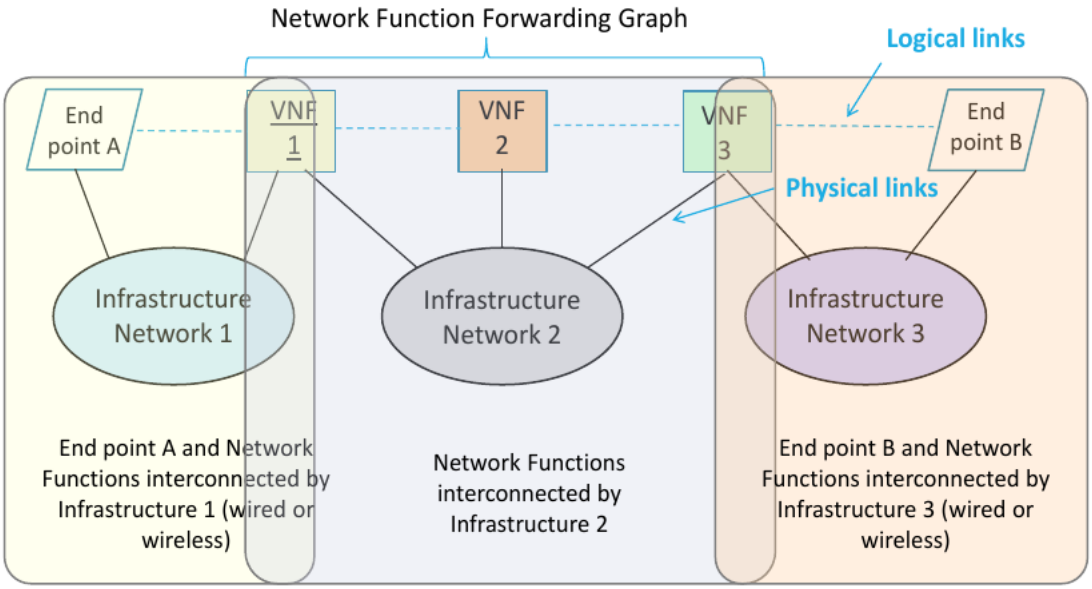


Figure 12.5: Network Function forwarding process

In the previous picture (12.5) there is a chain of network functions: the interconnections among the *NFs* and endpoints are depicted by *dashed lines*, representing logical links. These logical links are supported by **physical paths** through infrastructure networks (*wired or wireless*). Endpoints can be a network enterprise (`e.g site A`) or a datacenter (`e.g. site B`), depending on the specific scenarios.

The *topological view* of the previous scenario in 12.6 indicates also that VNF can be decomposed in different functions: the VNFs run as VMs on physical machine (*called* ***Points of Presence - PoPs***) so the placement it's generally located where it's function it's optimal. As an example, consider `VFN-1` as a cache service that is placed at the edge of the network. Consider also the pictured schema where the nested function *VNF-FG-2* is composed by three different *Virtualized Network Functions* (*respectively VNF-2A, VNF-2B, VNF-2C*). The virtualization allows to migrate and instantiate those functions among different PoPs, also automating those operations by **orchestrating** the setup, installation and execution of VFN autonomously.

12.0.3 NFV Architectural Framework

The **architectural framework** addresses the following:

- **NFVI (Network Function Virtualization Infrastructure)**: The resources required to setup a Network Service
- **VNF (Virtualized Network Function)**: The functionality that is required due to the decoupling Network Functions into software and hardware
- **MANO (Management and Orchestration)**: Allocate a set of virtual resources and support the management of orchestration of the functions

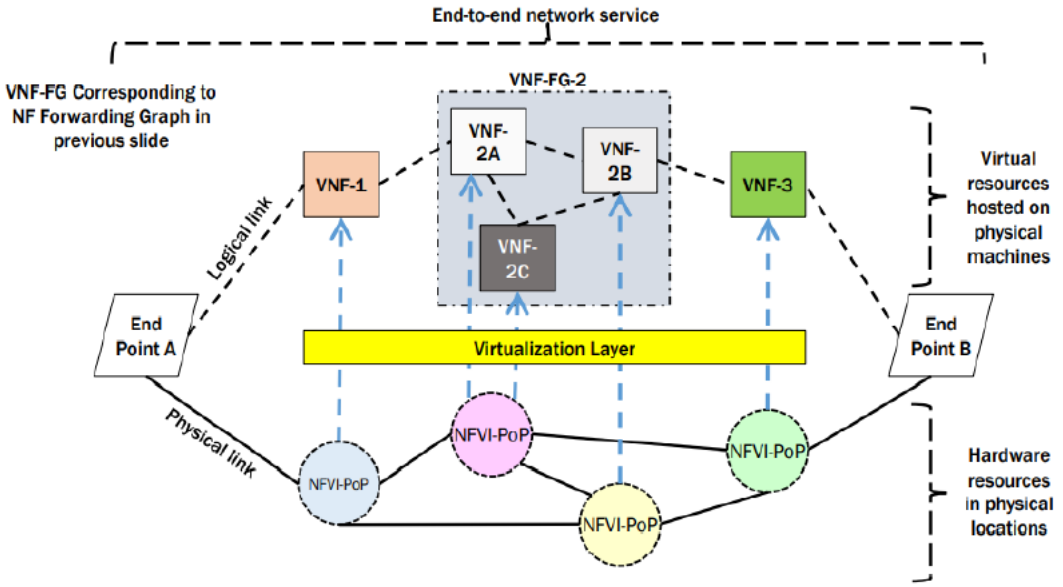


Figure 12.6: Forwarding graph of 12.5

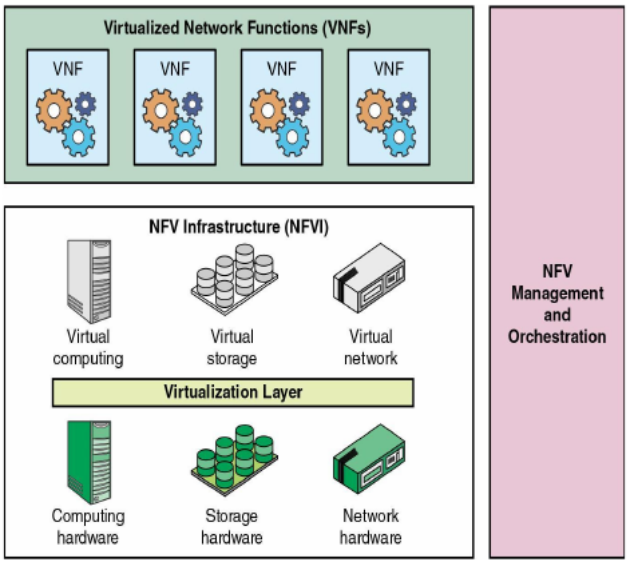


Figure 12.7: NFV Infrastructure schema

The general infrastructure schema is pictured in 12.7. The **NFV infrastructure (NFVI)** comprises the hardware and software resources that create the environment in which VNFs are deployed. Each **VNF** is implemented in software to run on virtual computing, storage, and networking resources. Finally, the ***NFV management and orchestration (NFV-MANO)*** is the component responsible for the management and orchestration of all resources in the NFV environment, ensuring the quality of service and the SLAs.

NFV Infrastructure It comprehends the totality of all **hardware and software components** which build up the environment in which VNFs are deployed, managed and executed by virtualizing physical computing, storage, and networking and places them into resource pools. The **physical hardware resources** include computing, storage and network that provide processing, storage and connectivity to VNFs through the virtualisation layer (*e.g. hypervisor*). The computing hardware, in this context, is assumed to be general-purpose and the storage resources can be differentiated between shared network attached storage (*NAS*) and storage that resides on the server itself. Overall, inside the *NFV Infrastructure* we can identify 3 different domains (*as shown in 12.8*):

- **Compute domain**: provides commercial off-theshelf (*COTS*) high-volume servers and storage.

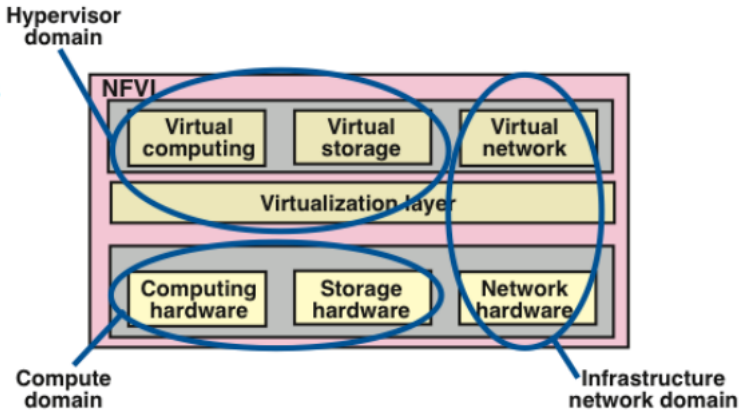


Figure 12.8: NFVI layers & domains

- **Hypervisor domain**: mediates the resources of the compute domain to the VMs of the software appliances, providing an abstraction of the hardware. (*e.g. VMs but now VNFs are also executed in containers, implementing Cloud Native Functions (CNF)*)
- **Infrastructure network domain**: comprises all the generic high volume switches interconnected into a network that can be configured to supply network services.

The NFVI can span across *several location*, so there are only **two main types of networks**:

- **NFVI-PoP network**: interconnects the computing and storage resources contained in an *NFVI-PoP*. It also includes specific switching and routing devices to allow external connectivity.
- **Transport network**: interconnects *NFVI-PoPs, NFVI-PoPs* to other networks owned by the same or different network operator, and *NFVI-PoPs* to other network appliances or terminals not contained within the *NFVI-PoPs*.

We focus on **Infrastructure network domain** by providing an example of **OpenStack Networking deployment** (pictured in 12.9):

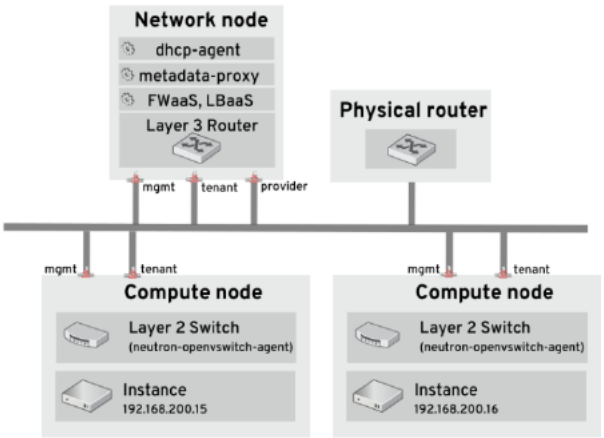


Figure 12.9: OpenStack Virtualized nodes interaction

In the previous picture (12.9), we have a dedicated *OpenStack Networking node* performing L3 routing and DHCP, and running advanced services like *FWaaS (Forwarding as a Service)* and *LBaaS (Load Balancing as a Service)*. There are also two *Compute nodes* run the **Open vSwitch (*openvswitch-agent*)** and have two physical network cards each, one for tenant traffic, and another for management connectivity. The OpenStack Networking node has a third network card specifically for provider traffic. The **vSwitch** provide connectivity between VIFs (*Virtual Interfaces*) and PIFs (*Physical Interfaces*) that allows to connects different Virtual Machines (as in 12.10). It also handle the traffic between VIFs colocated on the same physical host. The virtual switch usualy reside in the host and are tipically entirely software (*as an example, see Open vSwitch*).

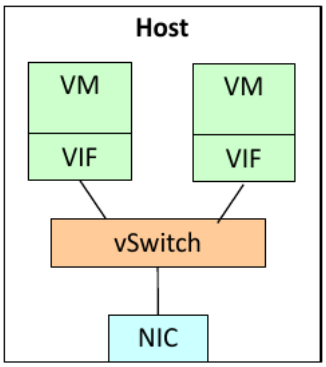


Figure 12.10: vSwitch simple schema

12.0.4 Management and Orchestration (MANO)

The schema pictured in 12.11 illustrates the *NFV Reference Architectural Framework*: the *MANO* component is sketched on the right (*gray area*).

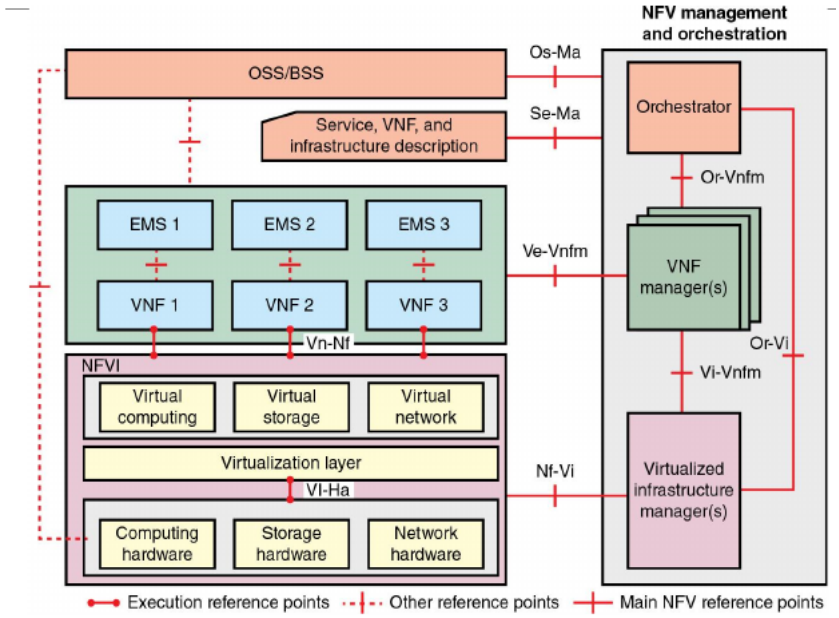


Figure 12.11: MANO general schema

Overall, the *Management and Orchestration* component provides the functionality required for the **provisioning of the VNFs** by detailing the VNFs configuration and the configuration of the infrastructure on which the VNFs run. It also 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*.
Going bottom-up, the first component is **VIM - Virtualized Infrastructure Management**: it comprises the functions that are used to control and manage the interaction of a VNF with computing, storage and network resources under its authority. A single instance of a VIM is responsible for controlling and managing the NFVI compute, storage, and network resources, usually within one operator's infrastructure domain. To deal with the overall networking environment, multiple VIMs within a single MANO may be needed.
The second component is the **Virtual Network Function Manager - VNFM**: It oversees the *lifecycle management* by instation, update, query, scaling up/down, terminate of VNF instance. It's also under its responsibility the *collection of NFVI performance measurement results* and faults/events information, and correlation to VNF instance -related events/faults.

Finally, the **NFV Orchestator - NFVO** is responsible for installing and configuring new Network Services (*NS*) and virtual network functions (*VNF*) packages alonside the management of the lifecycle of network service and global resources. The **Network services orchestration** it's responsible of manage/coordinate the creation of an end-to-end service that involves VNFs: it creates end-to-end service between different VNFs and can instantiate VNFMs, where applicable. Differently, the **Resource orchestration** manages and coordinates the resources under the management of different VIMs.
Overall, the *NFV MANO* internal schema is pictured in 12.12. The **Network Services Catalog** act as

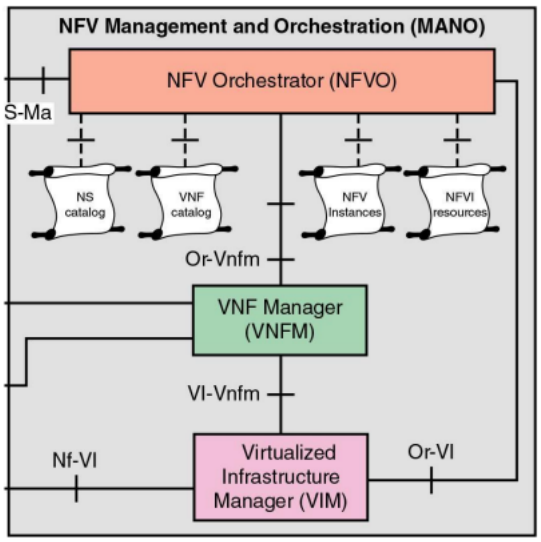


Figure 12.12: NFV management and orchestration process

a repositories of network services and have four main purposes:

1. **List of the usable network services**: a deployment template for a network service in terms of VNFs and description of their connectivity through virtual links is stored in NS catalog for future use.
2. **Database of all VNF descriptors (VNFD)**: a VNFD describes a VNF in terms of its deployment and operational behavior requirements.
3. List of **NFVI resources** utilized for the purpose of establishing NFV services.
4. List containing details about network services instances and related **VNF instances**.

12.1 Network Slicing

The 5G networks are intended to provide three types of services:

- *mMTC - massive Machine Type Communications*: massive number of devices communicating with each other and requires low cost along with long battery backup time
- *URLLC - Ultra-Reliable Low Latency Communications*: needs simultaneos low-latency and increased reliability mechanisms
- *eMBB - Enhanced Mobile Broadband*: higher data rates along with large coverage area

Those services are not uniquely determined, but are correlated to the specific use case. The metrics can varies as the specific technical requirements. The table 12.13 report the guarenteed services with a relative example and the ensured requirements.
Network slicing idea is to create different logical network on top of the physical one. The resources are pooled together, on the physical resources: this allows to shape and address the requirements of the previous use cases. Each subnetwork will perform slicing of the physical network resources to yield an *independent network* for its applications.

Basically network slices allows to create *customized networks* realized by creating multiple virtual and end-to-end networks. This can be done by exploiting the two technical element already presented:

5G use cases	Example	Requirements
Mobile Broadband (eMBB)	4K/8K UHD, hologram, AR/VR	High capacity, video cache
Massive Iot (mMTC)	Sensor network (metering, agriculture, building, logistics, home, etc.)	Massive connection (200000/km^2), mostly immobile devices, high energy efficiency
Mission-critical IoT (URLLC)	Motion control, autonomous driving, automated factory, smart-grid	Low latency (ITS 5 ms, motion control 1 ms), high reliability

Figure 12.13: 5G Use cases table

- **NFV**: implemenets the NFs in a *network slice*. The virtualization alllows the isolation of each network slice from all other slices: this assures QoS and security requirements for that slice, independently from other network slices.
- **SDN**: once a network slice is defined, SDN operates to minitor and enforce Qos requirements by controlling and enforcing the behavior of the traffic flow for each slice.

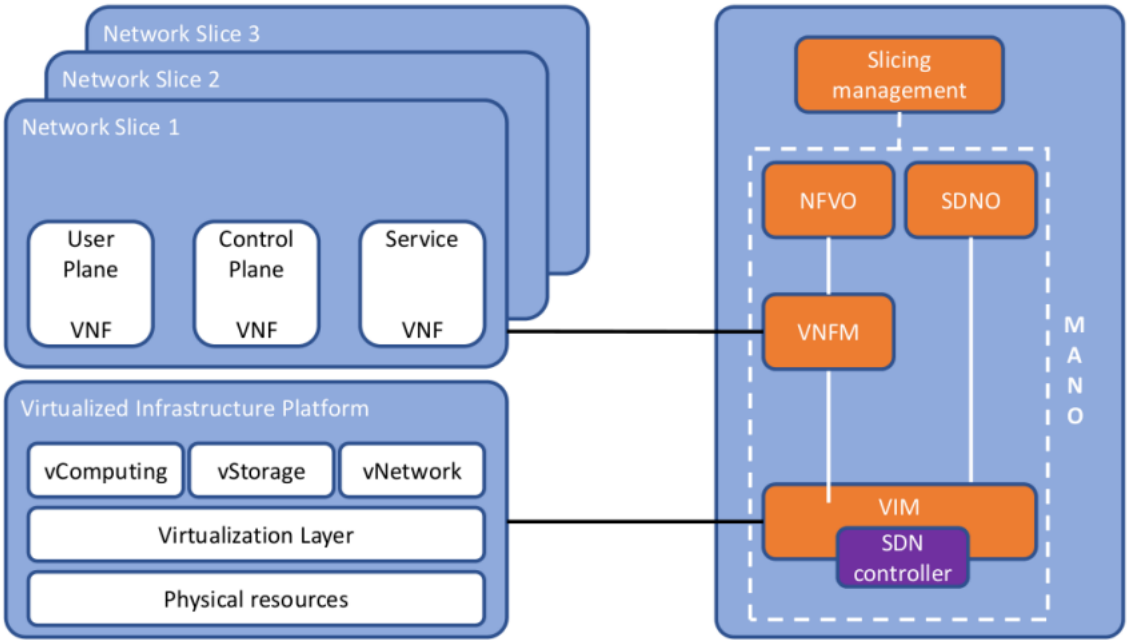


Figure 12.14: Network Slicing: component interaction flow

There are several ways to combine NFV with SDN: the SDN controller can be used with the *VIM (Virtual Infrastructure Manager)* to program and route the traffic flows, providing traffic isolations and balancing the resources to some flows respect to other.

5G Use case

The general scenario is sketched in 12.15 , without great detail.

Two locations are considered in 12.15: the *edge* is closer to antennas and the *core cloud* is the remote one. The underlying hypothesis is that the first component have few resources compared with the core cloud DC: if you need to support low-latency requirements, balancing resources respect to the total utilization and flow heavy computation to the core cloud.

Some terminology:

- *v2x*: for application services
- *Radio Unit (RU)*: radio hw unit

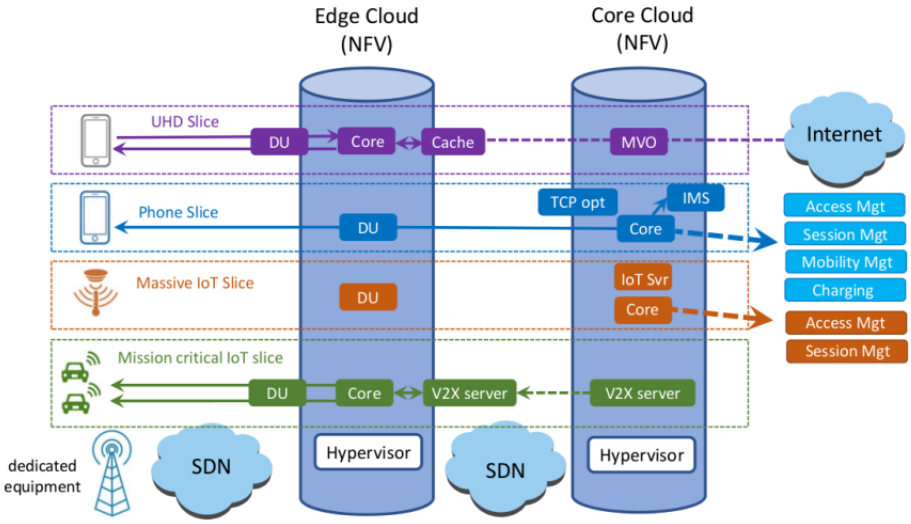


Figure 12.15: Example scenarios: distribution of services among different data center locations

- *Distributed Unit (DU)*: typically deployed close to the RU
- *Centralized Unit (CU):* can be placed in the cloud (*radio resource control, packet data converge protocol*)

NFV allows to move those functions where needed, based on specific requirements and provied resources.
For the *phone slice*, the core is distributed in the core cloud, moving the workload to the more resource-powered location.

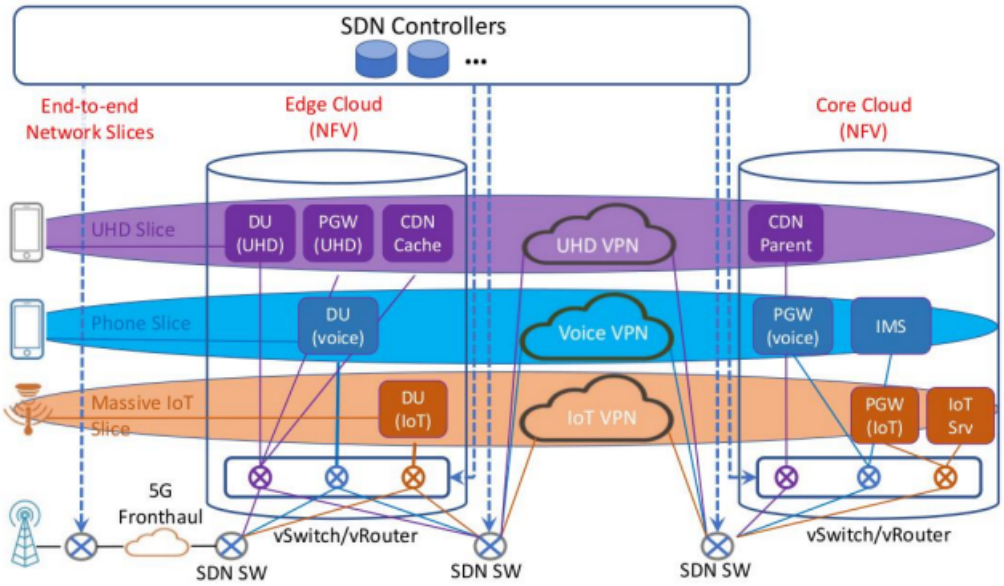


Figure 12.16: Flow filtering, different view of 12.15

The isolation of different flow, as pictured in 12.16, is usually prformed by the *vSwitch* or *vRouter* within a node. The `SDN SW` (*ingress*) classify the traffic incoming from the fronthaul: flow filtering enforces isolation and implement the forwarding rules based on the classification provided. The traffic flows through the *egress* router and flows in the specific VPN. SDN Controllers allows to define how to handle the traffic, defining the flow rules to be applied by the router/switches along the path chain.

12.1.1 Network Slicing Example

Here we present an example of network slicing, implemented in `comnetsemu`. As mentioned, slicing network topology resources is done by two criteria:

- *Connectivity*
- *Bandwidth*

In the following example, pictured in 12.17, two slices are defined to interconnect hosts `h1` and `h3`, the same for `h2` and `h4`. The two resulting **overlay topologies** are isolated, so they do not share any link (*h1 and h2 cannot comunciate*). Each host has its own *view of the network*; for the topology in 12.17 we have:

- **Upper slice**: `h1 -> s1 ->s2 ->s4 ->h3, 10 Mb/s`
- **Lower slice**: `h2 -> s1 ->s3 ->s4 ->h3, 1 Mb/s`

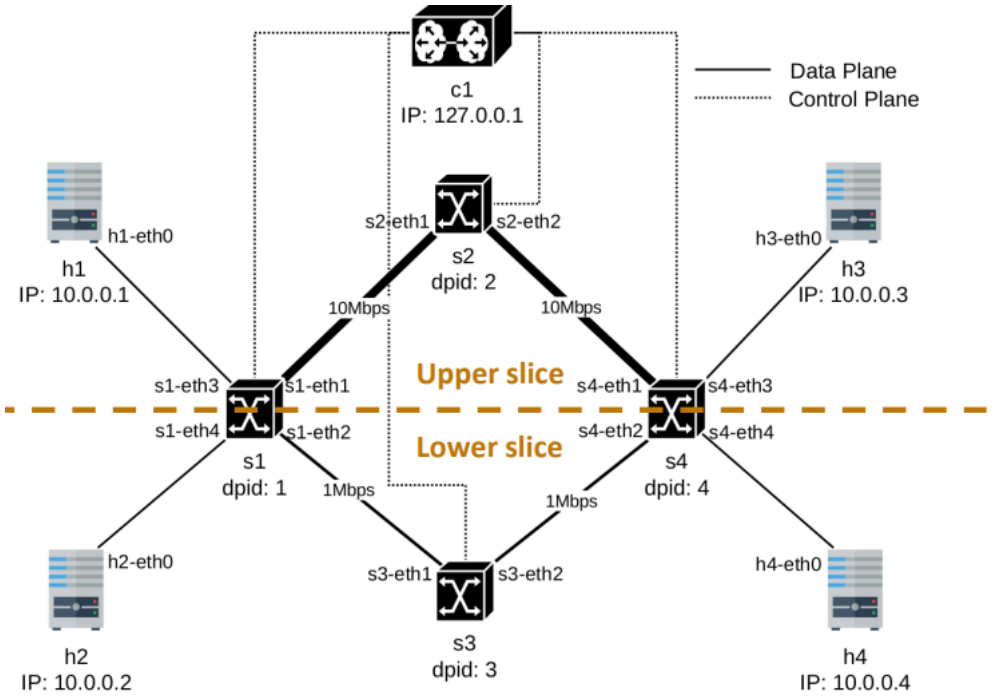


Figure 12.17: Sliced topology: division in upper and lower slice

The following code is contained in `topology_slicing.py`. The code is a part of a traffic slicing application using the **Ryu framework** for *software-defined networking (SDN)*. Let's go through the code and understand its functionality.

1. The code begins with importing necessary modules and classes from the Ryu framework.

```
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
```

1. The code defines a Ryu application class called `TrafficSlicing`, which extends `RyuApp`. This class will handle events and implement the traffic slicing functionality.

```
class TrafficSlicing(app_manager.RyuApp):
OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
```

1. Inside the class, there's an `__init__` method that initializes the `TrafficSlicing` object. It also defines a dictionary called `slice_to_port`, which maps the input port of a switch to the corresponding output port for traffic slicing. The dictionary represents the **traffic slicing rules**.

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

  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},
  }
```

---

1. The `switch_features_handler` method is a decorator function that handles the `EventOFPSwitchFeatures` event in the `CONFIG_DISPATCHER` state. This method is triggered when a switch connects to the controller. Inside this method, a flow entry is installed on the switch to send **all unmatched packets to the controller** for further processing.

```
  @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

  match = parser.OFPMatch()
  actions = [
  parser.OFPActionOutput(ofproto.OFPP_CONTROLLER, ofproto.OFPCML_NO_BUFFER)
  ]
  self.add_flow(datapath, 0, match, actions)
```

---

1. The `add_flow` method is used to **add flow entries to the switch**. It constructs an `OFPFlowMod` message and sends it to the switch to install a flow entry.

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

  inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS, actions)]
  mod = parser.OFPFlowMod(
  datapath=datapath, priority=priority, match=match, instructions=inst
  )
  datapath.send_msg(mod)
```

---

1. The `_send_package` method is a helper function used to send a packet out through a specific output port of a switch. It creates an `OFPPacketOut` message and sends it to the switch.

```
  def _send_package(self, msg, datapath, in_port, actions):
  data = None
  ofproto = datapath.ofproto
  if msg.buffer_id == ofproto.OFP
```

Chapter 13

Theory of signals

Theory of signals allows *sensors* to sample real signals or physical quantities in a finite set of time (*like temperatures*) and transform them into *discrete, digital signals* (*in a computable format of finite set values*), processing them and transmitting the obtained *discrete result*. Under *wireless transmission*, the theory of signals introduce the tools to transform a data packet into an *analog signal* (*often radio waves*) and receive analog radio signals, transofrming them into *digital signals*.

To understand these steps, it's necessary to understand the nature of signals, deriving how they can be *decomposed, reconstruncted and sampled*. All of this activities passes through the understanding of **Fourier series**.

13.1 Classification of signals

We consider only **deterministic signals** because are known before they are produced, differently from *random* signals that are analyzed using probabilsitic methods. A **deterministic signal** can be represented by a function $f(t)$ of a real value (*generally the time*):

$$f(t) : \mathbb{D} \to \mathbb{E}$$

The domain and codomain can be either the *set of real number* $\mathbb{R}$ or a *discrete set* $\mathbb{Z}$, or even the *set of complex numbers* $\mathbb{C}$. The set of complex number $\mathbb{C}$ allows to represent two *independent signals* combined together. We mainly distinguish in two types of signals, based on their definition *domain*:

- **Continuous-time signals**: the definition domain $\mathbb{D}$ is $\mathbb{R}$
- **Discrete-time signals**: the definition domain $\mathbb{D}$ is $\mathbb{Z}(\mathbb{T}) = \{nT, \forall n \in \mathbb{Z}, T \in \mathbb{R}\}$

So, different codomain $\mathbb{E}$ allows to represent a *signal* as a real function of time like:

- **Time-continuous and amplitude-continuous** (*analog signals*): if the domain $\mathbb{D}$ and the codomain $\mathbb{E}$ are the set of real numbers $\mathbb{R}$ the signal is charaterized by ***continous amplitude*** .

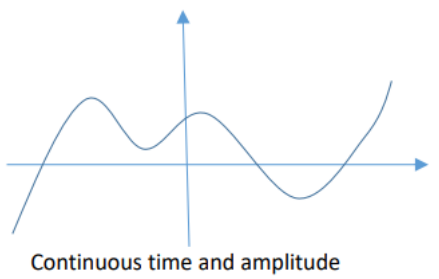


Figure 13.1: Continuous time and amplitude signal

- **Time-continuos and quantized** (*quantized signals*): if if the domain $\mathbb{D}$ is the set of real numbers $\mathbb{R}$ and the co-domain $\mathbb{E}$ is the discrete set $\mathbb{Z}$, the signal is charaterized by ***discrete amplitude*** (*or quantized signal 13.2*).
- **Time-discrete and amplitude-continuous** (*discrete signals*): as said, the domain is the *set of integers* $\mathbb{Z}(\mathbb{T}) = \{nT, \forall n \in \mathbb{Z}, T \in \mathbb{R}\}$. In picture 13.3 example, $\mathbb{Z}(2) = \{.., -4, -2, 0, 2, 4\}$.

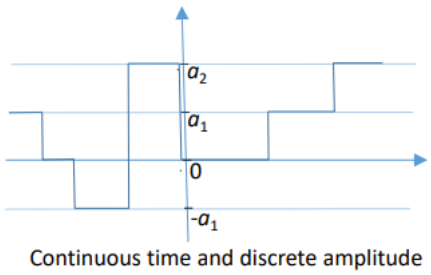


Figure 13.2: Continuous time and *discrete* amplitude signal

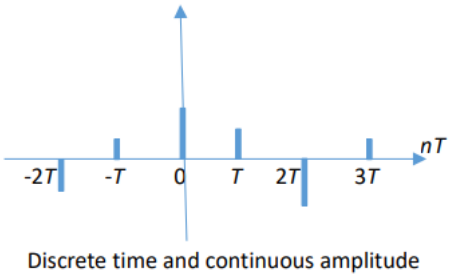


Figure 13.3: Discrete time and continuous amplitude signal

- **Time-discrete and quantized** (*digital signals*): a discrete signal is known as **digital signal** (*or symbolic sequence*) when the codomain is a *finite set of symbols*. For example, a text is an example of symbolic signal 13.4.

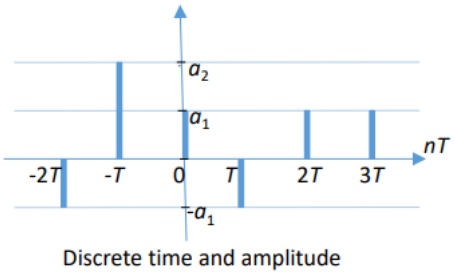


Figure 13.4: Discrete time and amplitude signal

Digital signals example Consider the alphabet of 26 symbols $A = \{a, b, c, .., x, y, z\}$. A symbolic signal is any sequenc eof such symbol like "$ababfxje$".
Differently, if consider the alphabet of two symbols $B = \{0, 1\}$, we can represent any symbol in A with a sequence of symbols in B like $a = 0000, b = 0001$, etc. Thus, the digital signal $ababfxje$ would become

$$00000000010000000001001011011101001001 00.$$

Now, assume a source able to transmit f symbols/second (*symbol frequency*), having:

- 8 symbols with the alphabet A: the transmission last $\frac{8}{f}$ seconds.
- 40 symbols with alphabet B: the transmission last $\frac{40}{f}$ seconds. .

The term *throughput* usually refers to the *binary frequency*. Assume a source that samples an analog signal with f_c samples/seconds, each sample is represented (***quantized***) with M bit/sample thus the source has a **throughput** of $f_c \times M$ bit/second.

Question Considering a source that emits a digital signal encoded in an alphabet of 8 symbols, with a symbol frequency equal to 10 symbols per second. What is the throughput of the source?
Answer: $2^3 = 8$

13.1.1 Periodic continous signals

A *continous signal* $s(t) : \mathbb{R} \to \mathbb{R}$ is **periodic** with period T if:

$$s(t) = s(t+T) \forall T \in \mathbb{R} \tag{13.1}$$

An example of periodic signal is $s(t) = sin(nt)$ and $s(t) = cos(nt)$ with period $T = \frac{2\pi}{n} (\forall n \in \mathbb{Z})$. 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** also holds the property:

$$s(t+T) = s(t+2T) = s(t+nT) \forall t \in \mathbb{R}, n \in \mathbb{Z} \tag{13.2}$$

Here an example of periodic continous signal (*pictured in 13.5*):

$$s(t) = cos(2t) + 5 \times sin(2t) - cos(5t) + 4 \times sin(6t)$$

which has period $T = 2\pi$.

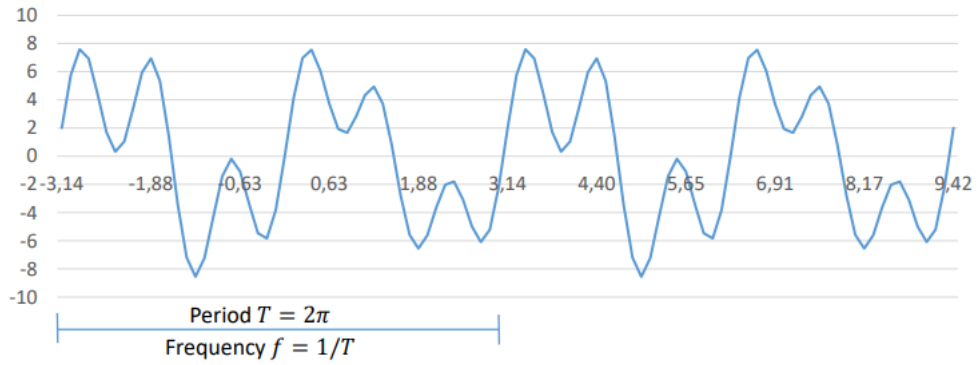


Figure 13.5: Example of a periodic continous signal

By knowing the signal in a specific period T, we can use **periodic extension of aperiodic signals** to obtain the signal in the subsequent periods. This can be applied to an **aperiodic signal** s with ***support limited*** to the interval $[a, b)$ such thath $s(t) = 0, \forall t \notin [a, b)$. The ***periodic extension*** s^* of s is defined as:

$$s^*(t) = \sum_{n=-\infty}^{\infty} s(t - nT) \tag{13.3}$$

where $T = b - a$, obtaining the following pictured extension:

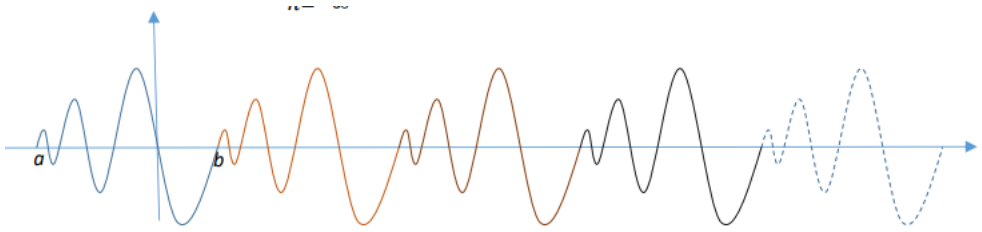


Figure 13.6: Periodic extesion of aperiodic signal defined in $[a, b)$

13.1.2 Transmission

Signals may have different nature, depending on the type of transmission channel: they can be *electromagnetic, sound, optical, waves, etc.* At the source, a **transducer** converts a message into a signal, while at the destination another transducer converts a signal into a message (*e.g. an antenna is a tansducer for electromagnetic signals*). During the transmission from source to destination, the signal can be **distorted** or turbated by the **noises** that charaterize the specific *trasmissive medium*. The *distortion* of a signal is common when dealing with a signal composed by *different waves* that are attenuated during the transmission, obtaining at destination a different signal respect to the source one. A common metric for

signals is **SNR - Signal to Noise Ratio** express the index quality of a channel, allowing to measure the intrinsic property of the transmissive medium, not of a message itself.

The transmission process can be **analogic** or **digital** based on the type of signal to be transmitted. The *analog transission* 13.7 use devices called **adaptors** that reduce the effect of noise and increase the efficiency, improving the quality of the signal. A **transducer** can be imagined as the *antenna* that emits the *electromagnetic waves* on the sender side or on the receiver-one used during signal reception, transforming it into an **electrical signal**, as shown in the following figure.

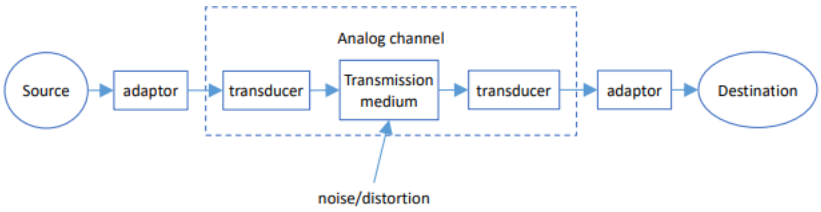


Figure 13.7: Analog transmission

Differently, in the **digital transmission** 13.8, there is an exchange of *discrete (sequence of symbols) messages* and can be performed also over an *analog channel*, as pictured here:

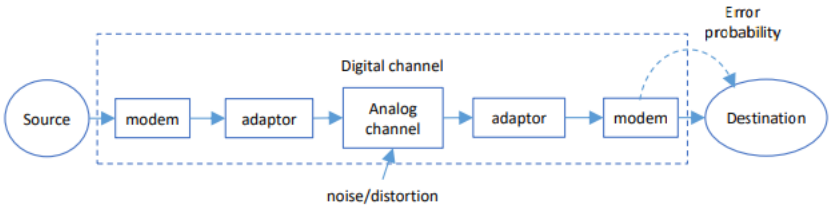


Figure 13.8: Digital transmission

The previous schema made use of the *analog channel* transmission component already described, using at both the transmission and reception phase a component called **modem**: it allows to perform *modulation* and *demodulation*, transforming the *discrete signal* in an *analog signal* (***modulation***) and, at receiver side, extracting from the received *analog signal* the correspondent *digital message* transmitted by the source.

The error probability is computed based on probability of *noise/distortion* happened during the analog transmission phase, combined with error in the *demodulation phase*. An optimal solution is to provide a stable or predictable limit to this probability, allowing **channel encoding** to use also *redundancy mechanism* to correct errors in *transmission and demodulation phase*.

Sampling and quantization Referring to the **digital transmission**, there are preliminar operations to be carried out both by the transmitter and receiver before starting transmitting the signal. The **transmitter** first sample the *analog signal* at *discrete intervals* (***quantization operation***) by means of an **Analog to Digital Converter (ADC)**, allowing the analog signal to be represented into a sequence of computable digital symbols. At the **receiver** side, the received symbols must be converted again into an *analog signal* by means of a **Digital to Analog Converter (DAC)**.

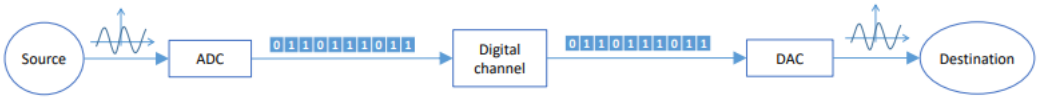


Figure 13.9: ADC/DAC in digital transmission

13.2 Fourier series

The cornestor of signal and system analysis are both the **Frequency Domain Analysis (FDA)** and the **Fourier transforms**. The general expression for a *sinudoid* at frequency ω (*or frequency* f *in Hertz*) is:

$$x(t) = \alpha sin(\omega t + \varphi) = \alpha sin(2\pi f t + \varphi)$$

We can combine the two sinudoids in (*as sketched at bottom in 13.10*):

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

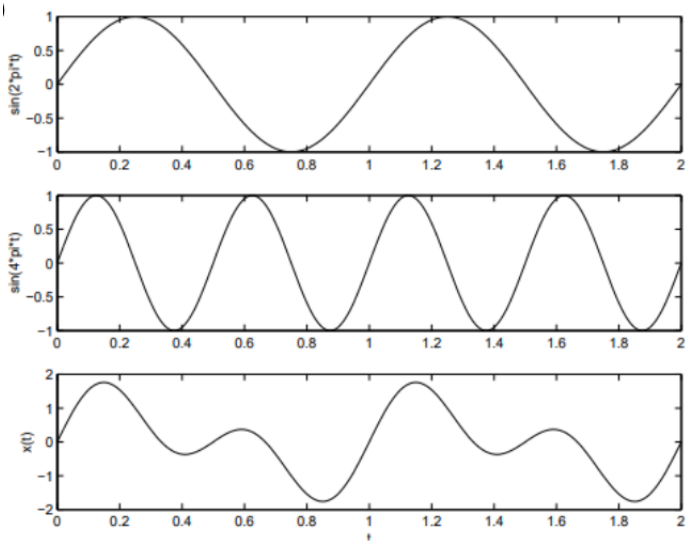


Figure 13.10: Combination of two sinusoids

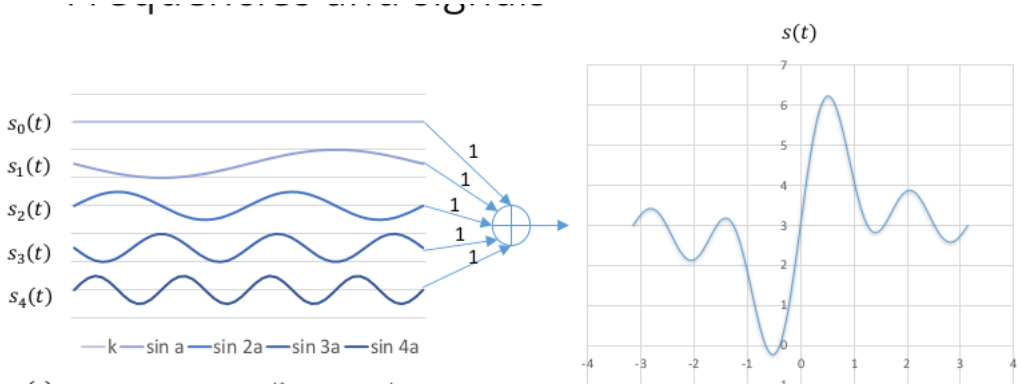


Figure 13.11: Example of different harmonic with their amplitude factor

Based on the example pictured in 13.11, we can merge the different components of the signal with their frequencies that indicates:

- $s_0(t)$: **constant component** (*frequency* 0)
- $s_1(t)$: **fundamental harmonic** (frequency $f_0 = 1/T$)
- $s_2(t)$: **second harmonic** (*frequency* $f_1 = 2/T = 2f_0$)
- $s_3(t)$: **third harmonic** (*frequency* $f_2 = 3/T = 3f_0$)
- $s_4(t)$: **fourth harmonic** (*frequency* $f_1 = 4/T = 4f_0$)

The **Fourier series** and **FDA** allows to *revert the process*, identifying the basic building block functions and their frequencies from a single signal, even composed by different waves at different frequencies.
The key idea is to **decompose the signal** in its basic building blocks, expressing it as the **sum of an infinite number of continous functions**, oscillating at *different frequencies*. This represents a change of coordinates: from *time* domain to *frequency* domain.
This set of continuous functions defines the ***base of decomposition***: the Fourier series has a base represented by a set of functions $\varphi_n(t), n \in \mathbb{Z}$ in which the functions must be **orthogonal** (*as in the case of decomposition of vectors in a vector space*).

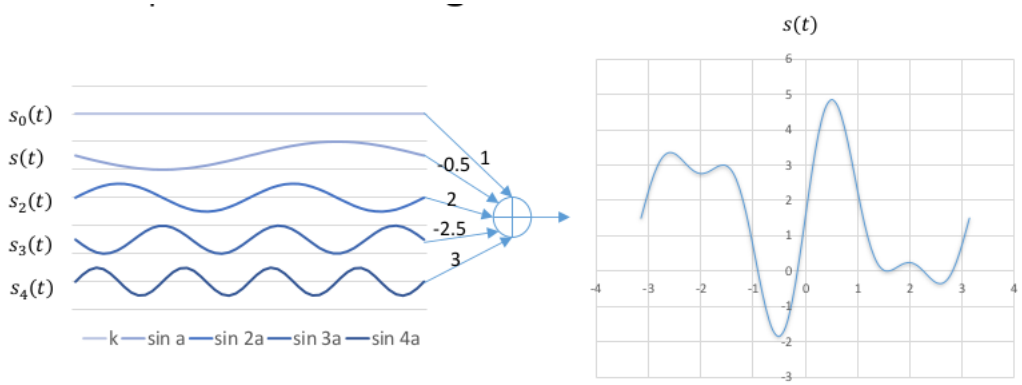


Figure 13.12: Modulation of the coefficient factor for each harmonic

Definition 13.2.1 (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:

$$s(t) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty} a_n cos(nt) + b_n sin(nt) \tag{13.4}$$

where:

$$a_0 = \frac{1}{\pi}\int_{-\pi}^{\pi} s(t)dt \tag{13.5}$$

$$a_n = \int_{-\pi}^{\pi} s(t)cos(nt)dt \tag{13.6}$$

$$b_n = \int_{-\pi}^{\pi} s(t)sin(nt)dt \tag{13.7}$$

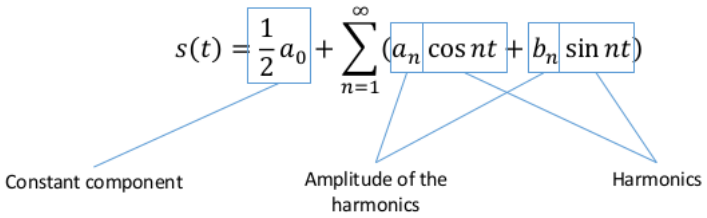


Figure 13.13: Fourier series structure

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 but the **sufficient conditions**, given by the *Dirichlet theorem*, are:

- $s(t)$ is **periodic**
- $s(t)$ is **piecewise continuous** (*finite number of discontinuities in the same period*)

Under these conditions the Fourier series of $s(t)$ *exists* and *converges* in $\mathbb{R}$.

Example

Consider:

$$s(t) = \begin{cases} 2 & \text{if } -\pi < t < 0 \\ 1 & 0 \leq t \leq \pi \end{cases} \tag{13.8}$$

With $s(t)$ as periodic function, with period 2π. The coefficients of the Fourier series are:

$$a_0 = \frac{1}{\pi}\int_{-\pi}^{\pi} s(t)dt = \frac{1}{\pi}\left(\int_{-\pi}^{0} 2dt + \int_{0}^{\pi} 1dt\right) = 3 \tag{13.9}$$

$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} s(t)cos(nt)dt = \frac{1}{\pi}\left(\int_{-\pi}^{0} 2cos(nt)dt + \int_{0}^{\pi} 1cos(nt)dt\right) = 0 \tag{13.10}$$

$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} s(t)sin(nt)dt = \frac{1}{\pi}\left(\int_{-\pi}^{0} 2sin(nt)dt + \int_{0}^{\pi} 1sin(nt)dt\right) = \begin{cases} 0 & \text{if } n \text{ is even} \\ \frac{-1}{nm} & n \text{ is odd} \end{cases} \tag{13.11}$$

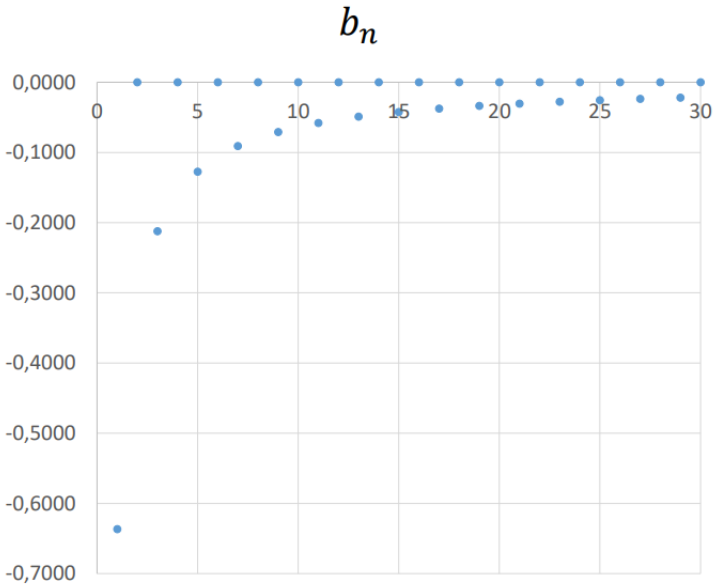


Figure 13.14: b_n coefficient plotted with n odd and even

Hence,letting $n = 2k - 1$ for all $k > 0$ (*the function is plotted in 13.15*):

$$s(t) = \frac{3}{2} - \sum_{k=1}^{\infty} \frac{2}{(2k-1)\pi} sin((2k-1)t) \tag{13.12}$$

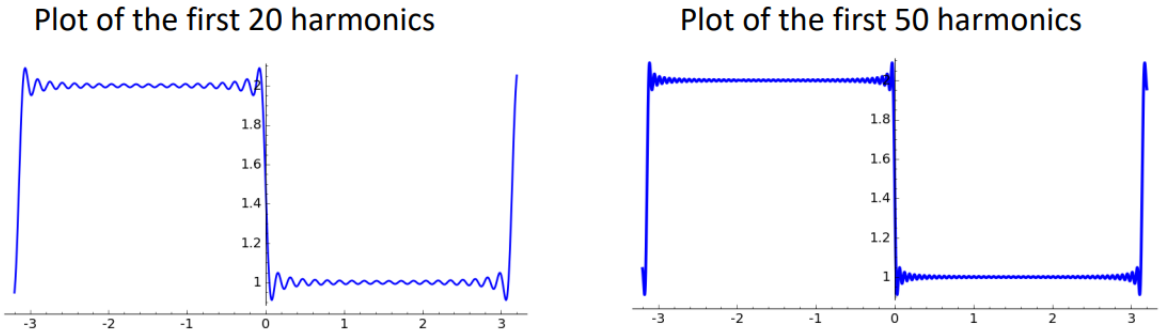


Figure 13.15: Signal approximation with Fourier series over $k = 20$ and $k = 50$ harmonics

As shown in 13.15, the approximation improves with an high number of harmonics.

The Fourier series is defined also for signals with **arbitrary period**. Given a continuous signal $s(t) : \mathbb{R} \to \mathbb{R}$, periodic in the interval $[-\frac{T}{2}, \frac{T}{2}]$.
Use the substitution $y = \frac{2\pi t}{T}$ obtaining:

$$f(y) = s(\frac{2\pi t}{T}) \tag{13.13}$$

which is periodic in the interval $[-\pi, \pi]$:

$$f(y) = s(\frac{2\pi t}{T}) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty}(a_n cos(ny) + b_n sin(ny) \tag{13.14}$$

Returning to the initial variable $y = \frac{2\pi t}{T}$ we obtain:

$$s(t) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty}(a_n cos(\frac{2\pi n}{T}t) + b_n sin(\frac{2\pi n}{T}t) \tag{13.15}$$

obtaining the coefficiencies already seen, computed in the interval $[-\frac{T}{2}, \frac{T}{2}]$:

$$a_0 = \frac{2}{T}\int_{-T/2}^{T/2} s(t)dt \tag{13.16}$$

$$a_n = \frac{2}{T}\int_{-T/2}^{T/2} s(t)cos(nt)dt \tag{13.17}$$

$$b_n = \frac{2}{T}\int_{-T/2}^{T/2} s(t)sin(nt)dt \tag{13.18}$$

Continuous signal energy

Given a signal $s(t)$ defined in the interval $[-\frac{T}{2}, \frac{T}{2}]$, the ***energy*** of the signal is defined as:

$$E_s(t) \doteq \int_{-\frac{T}{2}}^{\frac{T}{2}} |s(t)|^2 dt \tag{13.19}$$

where the relative phyisic interpretation regards $s(t)$ as *voltage* applied to 1ω resistor, so $E_s(T)$ is the **energy dissipated** in the period T.
A signal is with **finite energy** (*or energy signal*) if the limit:

$$E_s = \lim_{T\to\infty} E_s(T) = \int_{-\infty}^{\infty} |s(t)|^2 dt > 0 \tag{13.20}$$

and

$$E_s < +\infty \tag{13.21}$$

If the integral is greather than zero but does not go to ∞ we can conclude that it's a signal with **finite energy**. Even signals $s(t)$ with *infinite duration* **may (*depending on the signal*) have finite energy**. The two previous conditions includes signal with **finite or infinite duration**: if the signal has infinite duration then $E_s \to 0$ as fast as $1/t$. In the physical world, all signals have *finite energy.*

Power of a signal

Given a signal $s(t)$ defined in the interval $[-\frac{T}{2}, \frac{T}{2}]$, the **average power** of the signal is defined as:

$$P_f(T) \doteq \frac{1}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} |s(t)|^2 dt = \frac{E_s(T)}{T} \tag{13.22}$$

A signal is with **finite power** (*or power signal*) if the limit:

$$P_s = \lim_{T\to\infty} \frac{1}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} |s(t)|^2 dt > 0 \tag{13.23}$$

and, as previously:

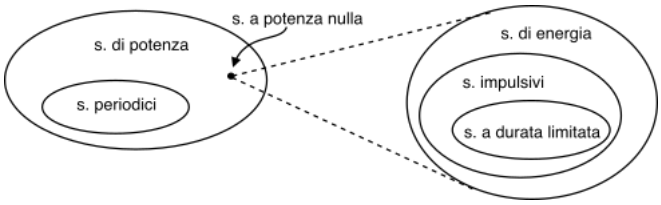
$$P_s < +\infty \tag{13.24}$$

Periodic signals are an important class of signals with **finite power**: they have *infinite energy* so their *average power* equals the average power computed in a period.

Overall, if a signal has infinite energy, its **average** power is zero hence the classes of signals can be structured as:

- with *finite enegy* $\to$ *energy signal*
- with **finite average power** $> 0 \to$ *power signals*

The two set are **disjointed**, as shown here:



Example The exponential signal:

$$s(t) = \begin{cases} 0 & \forall t < 0 \\ ae^{-bt} & \forall t \geq 0 \end{cases} \tag{13.25}$$

has **finite energy** if:

$$E_s = \int_0^{\infty} |s(t)|^2 dt = \frac{a^2}{2b} < \infty \tag{13.26}$$

and has **null average power**:

$$P_s = \lim_{T\to\infty} \frac{1}{T} \int_0^{\frac{T}{2}} |s(t)|^2 dt = \lim_{T\to\infty} \frac{a^2}{2bT} = 0 \tag{13.27}$$

The *periodic signal* $s(t) = cos(t) \forall t$ has **infinite energy and finite average power** so:

$$P_s = \lim_{T\to\infty} \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} \cos(t)^2 dt = \frac{1}{2\pi} \int_0^{2\pi} cos^2(t) dt = \frac{1}{2\pi} \int_0^{2\pi} \frac{1}{2} dt + \frac{1}{4\pi} \int_0^{2\pi} \frac{1}{2} cos(2t) dt = \frac{1}{2} \tag{13.28}$$

Question Classify the following signals as *Energy* and *Power* signals and explain why:

- $$s(t) = cos(t) \tag{13.29}$$
- $$s(t) = \begin{cases} 3 & -2 < t < 1 \\ 0 & \text{otherwise} \end{cases} \tag{13.30}$$
- $$s(t) = \begin{cases} \frac{3}{t} & 1 < t < 10 \\ 0 & \text{otherwise} \end{cases} \tag{13.31}$$

13.3 Fourier Transform

In this section the **complex numbers** and different form of its representation are used to present a different expression of the Fourier series. This notaton allows to express the Series compactly and explain the Discrete Fourier Transform with the same mathematical tools.

Euler's exponential The exponential $e^{j\phi}$ is a complex number (*not in the canonical form* $\tilde{x} = a + jb$) defined by Euler as:

$$e^{j\varphi} = \cos\varphi \pm j \sin\varphi \tag{13.32}$$

from which we can derive the Euler's formulas:

$$\cos\varphi = \frac{e^{j\varphi} + e^{-j\varphi}}{2} \tag{13.33}$$

$$\sin\varphi = \frac{e^{j\varphi} - e^{-j\varphi}}{2j} \tag{13.34}$$

As a complex number property, we can rewrite $\tilde{x} = |x|e^{j\varphi}$ where:

$$|x| = \sqrt{a^2 + b^2} \tag{13.35}$$

$$\varphi = tan^{-1}(\frac{b}{a}) \tag{13.36}$$

Or, by expressing them as a coffiecient for real and imaginary part:

$$a = |x|, b = |x| \sin \varphi \tag{13.37}$$

By complex number property, we can reformulate as:

$$|x|e^{j\varphi} = |x| \cos \varphi + j|x| \sin \varphi \tag{13.38}$$

This notation allows to express the Fourier series with a different *exponential base*. Usually, the Fourier series is expressed in the complex domain, representing signals $s(t) : \mathbb{R} \to \mathbb{C}$ that also includes signals defined in $\mathbb{R} \to \mathbb{R}$.
This different representation is possible due to a different **base of Fourier**, that is the set of functions:

$$e^{j2\pi nFt} \forall n \in \mathbb{Z} \tag{13.39}$$

where the complex exponetial e^{x+jy} follow the property:

$$e^{x+jy} = e^x e^{jy} = e^x(\cos y + j \sin y) \tag{13.40}$$

hence, we can express the Fourier set of functions as:

$$e^{j2\pi nFt} = \cos 2\pi nFt + j \sin 2\pi nFt \tag{13.41}$$

We now can redefine the **periodic signal** $s(t) = s(t+T), \forall t$ as a **power signal** $s(t)$ with the frequency $F = 1/T$ indicated as the *first (fundamental) harmonic* expressed in Hertz. The *linear combination* that express the signal is:

$$s(t) = \sum_{n=-\infty}^{\infty} S_n e^{j2\pi nFt} \tag{13.42}$$

where S_n is the series:

$$S_n = \frac{1}{T} \int_0^T s(t) e^{-j2\pi nFt} dt \tag{13.43}$$

$s(t)$ is represented by S_n but, for the *Dirichlet theorem*, the Fourier series gives the same values of $s(t)$ where $s(t)$ is continuous but not in **discontinuities**. This representation allows to thought $s(t)$ as composed by *infintie number of periodic signals*.
The functions in the Fourier base have frequency nF multiple of the fundamental harmonic $F = 1/T$: the terms $S_n e^{j2\pi nFt}$ are called *harmonic componets* of $s(t)$, each one with frequency nF that allows to compute different coefficent value by tuning the value of n, obtaining different harmonics approximation. The coeffient $S_0 = \frac{1}{T} \int_0^T s(t)dt$ is the average value of $s(t)$, called *continuous compoent*.

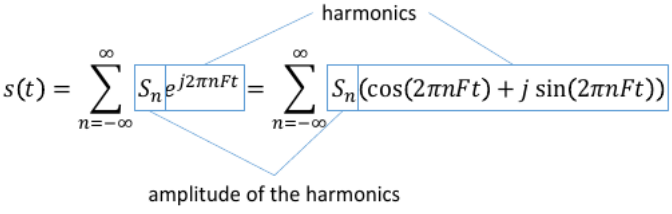


Figure 13.16: Geomtric interpretation of Fourier series

Referring 13.16, for $n = 0$ we have the continuous signal S_0, for $|n| = 1$ we obtain the fundamental frequency $f_0 = F$ with period $T = \frac{1}{F}$, lastly for $|n| > 1$ we obtain the harmonics of frequency $f_n = nF$ with periodi T/n.

Signals with limited support The Fourier series can also be derived for an **aperiodic signal** with limited support (*n.b.: null out of interval* $[a, b]$): the series assume that the signal is periodic anyway, taking as period the entire duration of the signal. The main consequences is that, if you revert from S_n to $s(t)$ you obtain the periodic extension of the signal, as seen in 13.6.

.

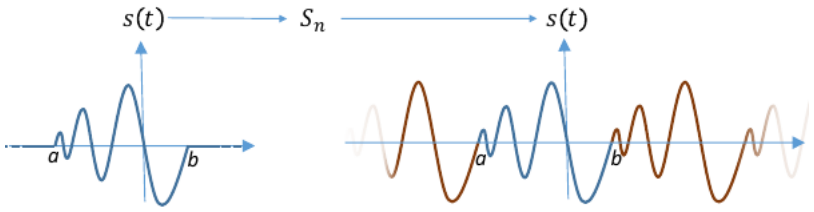


Figure 13.17: Fourier transform periodic extension

13.3.1 Fourier Transform

The transition from the continuous signal $s(t)$ to its spectrum or *discrete signal* S_n is called **Fourier Transform (FT)** and it's denoted by:

$$s(t) \longleftrightarrow S_n \tag{13.44}$$

or even with:

$$S_n = \mathbb{F}(s(t)) \tag{13.45}$$

and with the following notation for the *inverse transform*:

$$s(t) = \mathbb{F}^{-1}(S_n) \tag{13.46}$$

The transform passes from the **time domain** of $s(t)$ to the **discrete frequency domain** of S_n (*discrete is the set of frequencies*): this is a function of n that identifies a harmonic and thus a *frequency*. Recall that S_n is the amplitude of the harmonic frequenciy nF and period T/nF. Here pictured the shift from the time domain to the discrete one:

Example Consider the signal with period T and frequency $F = \frac{1}{T}$:

$$s(t) = \begin{cases} 1 & |t| \leq T/4 \\ 0 & T/4 < |t| \leq T/2 \end{cases} \tag{13.47}$$

The coefficients of the Fourier series are:

$$S_n = \frac{1}{T}\int_0^T s(t)e^{-j2\pi nFt}dt = \frac{1}{T}\int_{-T/4}^{T/4} e^{-j2\pi nFt}dt = \begin{cases} \frac{1}{2} & n = 0 \\ \frac{sin(n\pi/2)}{n\pi} & n \neq 0 \end{cases} = \begin{cases} \frac{1}{2} & n = 0 \\ 0 & n \text{ even} \\ -\frac{(-1)^k}{(2k-1)\pi} & n = 2k-1 \end{cases} \tag{13.48}$$

Hence, the Fourier series can be express in terms of:

$$s(t) = \frac{1}{2} + \sum_{k=-\infty, k\neq 0}^{k=\infty} (-\frac{(-1)^k}{(2k-1)\pi}) \times (cos(2\pi(2k-1)Ft)) + j \times sin(2\pi(2k-1)Ft) \tag{13.49}$$

and by recalling that $sin(a) = -sin(-a)$ we have:

$$-\sum_{k=-1}^{\infty} j \times sin(2\pi(2k-1)Ft) \tag{13.50}$$

Remeber that in this example the *complex terms* are null.
hence, we can simplify $s(t)$ as:

$$s(t) = \frac{1}{2} + \sum_{k=-\infty}^{k=\infty} (-\frac{(-1)^k}{(2k-1)\pi}) \times (cos(2\pi(2k-1)Ft)) \tag{13.51}$$

and by letting $T = 4$ (*the same of having* $F = 1/4$) we have 13.18:

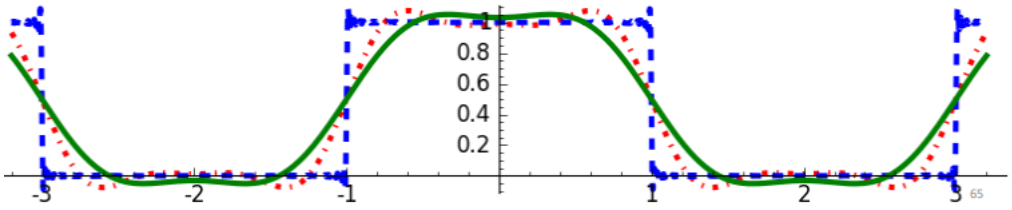


Figure 13.18: Green $k \in [-1, 1]$, Red $k \in [-2, 2]$, Blue $k \in [-100, 100]$

Spectrum of a signal Given a signal $s(t)$, by Fourier we have a mean to find the coefficients of the series S_n: on the other hand, given the coefficient S_n we can reconstruct the signal $s(t)$. The **ordered coeffiecients** S_n from $-\infty$ to ∞ is called **spectrum of** $s(t)$ (*we can also define the spectrum as a discrete signal*).
It's not always possible to represent the spectrum with one 2D diagram but we can represent the complex number S_n by using the Euler's exponential:

$$S_n = |S_n|e^{j\varphi_n} \tag{13.52}$$

Where the factor $|S_n|$ indicates the **amplitude of the hamornic**, and $e^{j\varphi_n}$ is the **harmonic's phase**. The representation in frequency of $s(t)$ has hence two diagrams:

1. the *amplitude spectrum diagram*
2. the *phase spectrum diagram*

Referring the previous example 13.3.1, the spectrum of $s(t)$ is pictured in 13.19.

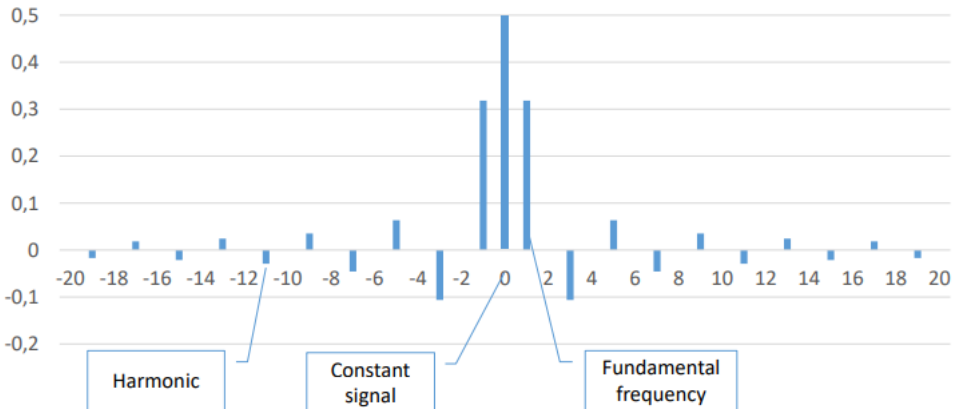


Figure 13.19: Spectrum of $s(t)$ in 13.3.1

13.3.2 Fourier Transform for non-periodic signals

The transform so far applies only to periodic signals but can be applied also to **finite signals** (*in a timeframe of length* T). In this case, the counter-transform returns a periodic signal with period T: the key idea is that we consider a periodic signal but with an **increasing period** T **to infinity** thus the fundamental frequency is decreasing so the signal became less and less periodic, obtaining an aperiodic or non-periodic signal because the frequency are closer to eacher other, obtaining a continuous set of frequencies thus the periodicity progressively disappear.
A **non-periodic signal** $s(t)$ can be expressed as:

$$s(t) = \int_{-\infty}^{\infty} S(f)e^{j2\pi ft}df \tag{13.53}$$

where f is the *continuous frequency*, ranging in $[-\infty, \infty]$ and $S(f)$ is the amplitude of the frequency f, given by:

$$S(f) = \int_{-\infty}^{\infty} s(t)e^{-j2\pi ft}dt \tag{13.54}$$

In this last case, the *spectrum of* $S(f)$ is a *continuous signal*.
The passage from the non-periodic signal $s(t)$ to its spectrum $S(f)$ is called **Continuous Fourier Transform (CFT)** and it's denoted by:

$$s(t) \longleftrightarrow_{CTF} S(f) \tag{13.55}$$

or by

$$S(f) = \mathbb{F}(s(t)) \tag{13.56}$$

or, for the *inverse transform*:

$$s(t) = \mathbb{F}^{-1}(S(f)) \tag{13.57}$$

Band of signal In general the spectrum of a signal spans all frequency in $(-\infty, \infty)$ while, if the signal is periodic, the spectrum contains only the harmonics of the main frequency (*which is the fundamental harmonic*). Filtering some frequencies of the spectrum allows to obtain a *signal limited in bandwidth*. Usually the filter can:

- exclude the *high frequencies*: it's a **low-pass filter**
- exlcude the *low frequencies*: it's a **high-pass filter**
- exlcude low and high frequencies, keeping the intermediate frequencies: it's a **bandpass filter**

Question Considering the amplitude spectrum $S(f) = \frac{sin(5\pi f)}{\pi f}$ of a signal $s(t)$ as shown in 13.20. Explain whether this is limited in band and, in case, to which band. Then, depict the spectrum filtered with a low-pass filer threshold frequency set at $1Hz$.

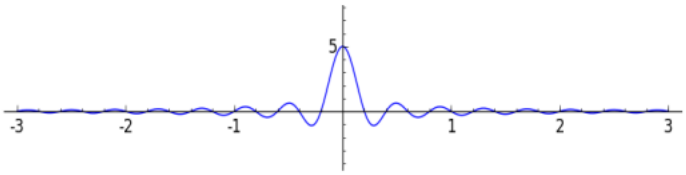


Figure 13.20: Limited signal

13.3.3 From FT to DFT

The **Discrete Fourier Transform** is a type of Fourier Transform for **discrete time signals** or signals known only at N instant separated by sample time T; those are an example of *finite sequence of data*.
An example is given by a discrete time signal obtained from a continuous signal with a domain restriction from $\mathbb{R}$ into $\mathbb{Z}(T)$. This operation, called **sampling**, is stated by the simple relationship:

$$sc(nT) = s(nT), nt \in \mathbb{Z}(T) \tag{13.58}$$

where $s(t), t \in \mathbb{R}$ is the reference **continuous signal** and $sc(nT), nT \in \mathbb{Z}(t)$ is the **discrete signal** obtained by sampling operation.
A **discrete-time signal** is a complex function of a discrete variable:

$$f(t) : \mathbb{Z}(T) \to \mathbb{E} \tag{13.59}$$

where the domain $\mathbb{Z}(T)$ is the set of multiples of $\mathbb{Z}(T) = \{.., -T, 0, T, T2, ...\}$ such that $T > 0$. The signal will be denoted in the forms $s(nT), nT \in \mathbb{Z}(T)$ or $s(t), t \in \mathbb{Z}(T)$.
Let $s(t), t \in \mathbb{R}$ be the continuous signal which is the source of data and let N samples be denoted by $s(0), s(1), ..., s(N-1)$. The Fourier Transform of the original signal would be:

$$s(t) = \int_{-\infty}^{\infty} S(f) e^{j2\pi ft} df \tag{13.60}$$

and $S(f)$ is the **amplitude of the frequency** f, given by:

$$S(f) = \int_{-\infty}^{\infty} s(t) e^{-j2\pi ft} dt \tag{13.61}$$

We could consider each sample $s(t)$ as an *impulse* having area $s(t)$. Since the integrand exists only at the sample points, we obtain:

$$S(f) = \int_{-\infty}^{\infty} s(t)e^{-j2\pi ft}dt = s(0)e^{-j2\pi f0} + s(1)e^{-j2\pi f1} + ... + s(N-1)e^{-j2\pi f(N-1)} \tag{13.62}$$

or, generalizing:

$$S(f) = \sum_{0}^{N-1} s(k)e^{-2j\pi fk}, \forall k \in \{0, ..., N-1\} \tag{13.63}$$

Becase the **input data point set** is a limited set, the DFT threats the data as if it were periodic, thus $s(N)$ to $s(2N-1)$ is the same as $s(0)$ to $s(N-1)$.
Since the operation threat the data as *periodic data*, we evaluate the DFT equation for the fundamental frequency $1/NT$ and its harmonics, like:

$$f = 0, \frac{1}{NT}, \frac{2}{NT}, ..., \frac{N-1}{NT} \tag{13.64}$$

so the DFT computes a finite number of DFT coefficients, in the number of N.
Given a **finite length sequence** composed of N values S_n, with $n \in \{0, N-1\}$, we can calculate the DFT as:

$$S_f = \sum_{n=0}^{N-1} S_n e^{-j\frac{2\pi f}{N}n} \tag{13.65}$$

with $f \in \{0, N-1\}$. The *anti-transform operation* is capable of returning temporal samples s_n:

$$s_n = \frac{1}{N}\sum_{0}^{N-1} S_f e^{j2\pi f_N^n} \tag{13.66}$$

The **Fast Fourier Transform (FFT)** is an algorithm for efficiently computing the discrete Fourier transform (DFT) of a sequence of numbers. The basic idea behind the FFT is to exploit the symmetry properties of the Fourier transform to reduce the number of computations required. Instead of computing the DFT directly using the definition, which requires $O(N^2)$ computations for a sequence of length N, the FFT algorithm can compute the DFT using only $O(NlogN)$ computations.

13.4 Analog to digital conversion

The sampling activity allows to transform a continous signal in a time-discrete value, allowing to convert it in a digital value, allowing to perform computation without modify the signal structure and without using special hardware purpose: the original signal is converted in a **sequence of integers** that can be stored and computed by an electronic device. The *conversion* activity is performed by the *ADC - Analog to Digital Converter* and includes two main operations:

- **Sampling**: the sampling allows to obtain a sequence $s(t)$ and from it extract the integer sequence at **predefined intervals** (*uniform in space*). The interval between two sequential samples is called **sampling period**.
- **Quantization**: transform the stream of *real numbers* produced by the sampling into a *stream of numbers* tht can be of *integer* or *float* type. The different type depends on the specific task to perform on the signal. The *sampling activity* allows to represent the original signal under some hypothesis (*detailed later in Sampling theorem*), while the *quantization* express how much information is **lost**.

The entire process, considering a signal $s(t)$ is sketched in 13.21.

Given $s(t)$, first we convert from analogic to digital and then we obtain a sampled signal readed at multiple values of T_c. Receiver side the signal is converted from digital to analogic, obtaining a signal more similar to the real one, before sampling activity.

The sampling activity for **continuous signals** can be described as: a *discrete signal* $g(nT), n \in \mathbb{Z}$, can be obtained with a *uniform sampling of continuous signal* $s(t)$ where $g(nT) = s(nT)$ with T is the sampling period, and $\frac{1}{T}$ is the sampling frequency.

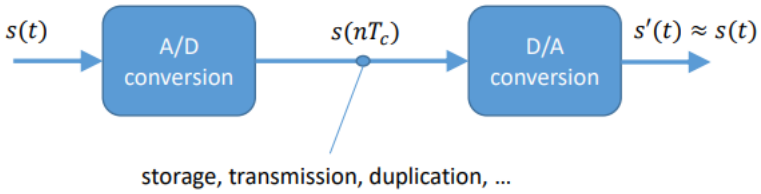


Figure 13.21: Analog to digital conversion process

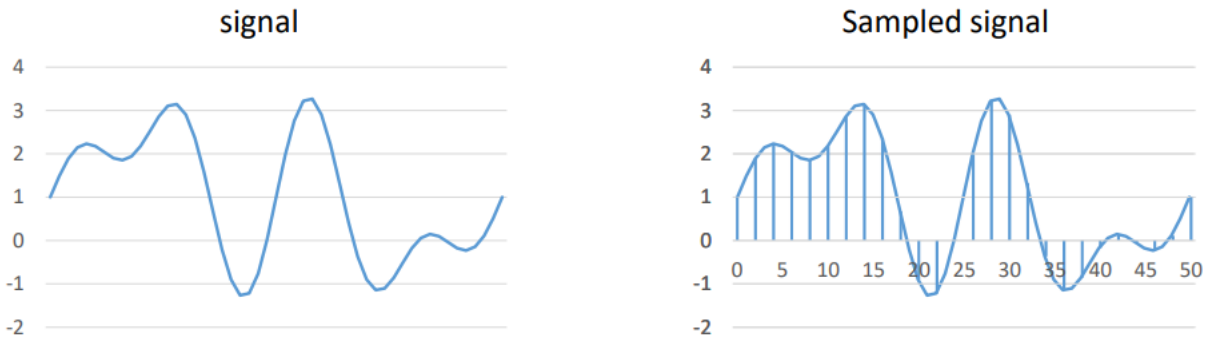


Figure 13.22: Comparison between real signal and sampled one

Example

Imagine to have the signal $s(t) = \sin(t)$ with frequency $f = \frac{1}{2\pi} = 0.159$ and assume to sample the signal with period $T_c = 3.09$, thus the *sampling frequency* is greater than the frequency of the original signal ($T_c = 3.09 \rightarrow f_c = 0.234 > 2f$). Refer the image in 13.23.

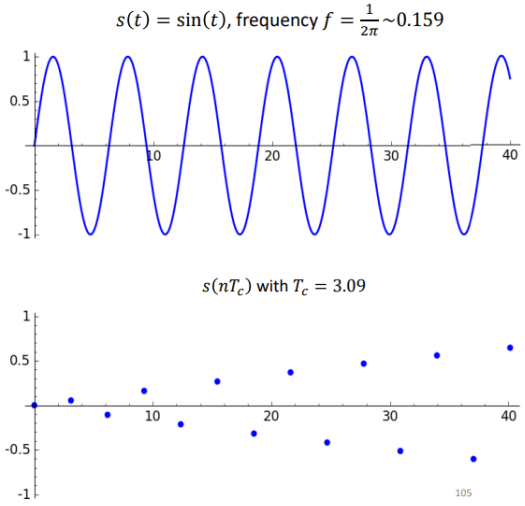


Figure 13.23: Example scenario

The sampling is a signal itself, $s(nT_c)$, which takes a value for every multiple of the sampling period T_c. The original signal is a *power signal* while the signal we obtain with the sampling it's composed by values taken at distnace T_c thus it's not a *power signal*. The sampling frequency is little more than $2f$. To reconstruct a continuous signal from a given set of samples, several methods can be applied, like **interpolation**. There are several way to reconstruct the signal, based on it's type:

- *Piecewise-costant signal*: methods like *zero-order hold* or *nearest neighbor*
- *Piecewise linear*: like *first-order hold reconstruction*

To apply the *zero-hold method*, consider a discrete-time signal with known samples at equally spaced

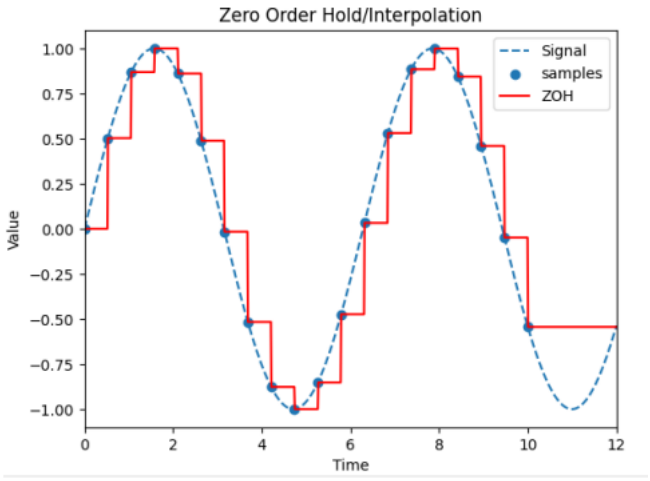


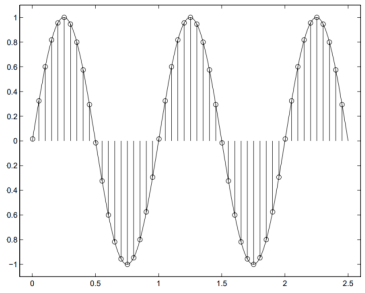
Figure 13.24: Zero Order Interpolation

intervals. Let's say we have a signal $x[n]$, where n represents the sample index. The zero-hold interpolation estimates the value of the signal $x(t)$ at any continuous-time point t.

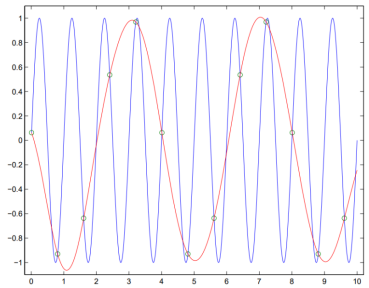
13.4.1 Sampling theorem

The sampling theorem give the **minimum sampling frequency** to be able to have enough information to reconstruct the signal from the given set of samples. It defines ideal conditions that not always can be applied to real world signals.

Here we compare two different sampling frequency for signal $x(t) = \sin(\frac{2\pi}{T}t)$. For the first case, the sampling frequency is higher than the signal frequency ($f_c >> \frac{1}{T}$) while for the second one is minor.



(a) Oversampling



(b) Undersampling

For the minor case, the original signal is the blue one but the sampled signal, due to minor frequency, is the red one: this does not allows to reconstruct correctly the real signal. This problem is also known as **aliasing** (*see later*).

Theorem definition Consider a signal $s(t)$ with *spectrum null* at frequencies above f_M (*the* ***maximum frequency of the signal***), thus the band of frequencies of $s(t)$ is limited. The signal $s(t)$ is **completely represented** by its samples taken:

- at regular intervals $t_n = nT_c, n \in \mathbb{Z}$
- with a sampling period $T_c \leq \frac{1}{2f_m}$

thus from the samples it's possible to reconstruct $s(t)$ for any t by:

$$f_{C_{min}} = \frac{1}{T_{C_{max}}} = 2f_m \tag{13.67}$$

This $f_{c_{min}}$ is called **Nyquist frequency**: it's the minimum frequency for which it's possible to sample the signal $s(t)$ limited in band. This implies that we can sample the signal only at higher frequency. Under

this conditions we can reconstruct the signal using interpolation, for example (*based on cardinal sin*):

$$s(t) = \sum_{n=-\infty}^{\infty} s(nT_c) \times \frac{\sin(\pi f_c \times (t - nT_c))}{\pi f_c \times (t - nT_c)} \tag{13.68}$$

The **Fourier Transform** of a band limited signal sampled $X(f)$ as described by the Sampling Theorem consists of shifted and scaled copies of $X(f)$, as pictured in 13.26.

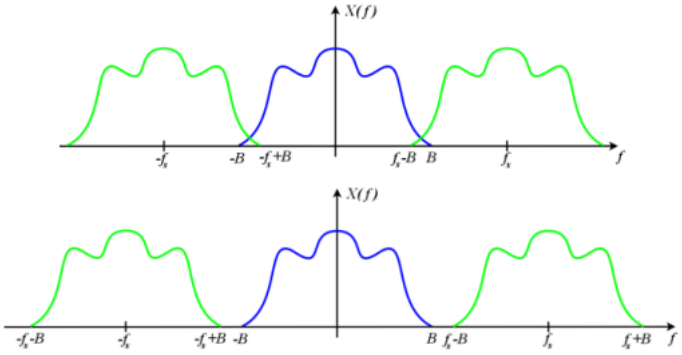


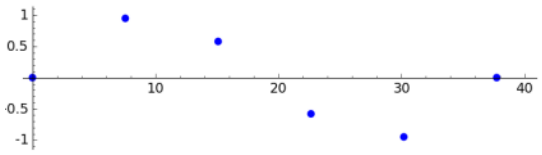
Figure 13.26: Shifted and scaled copies of $X(f)$

The spacing between the replicas of $X(f)$ depends on the *sampling rate*: the faster we sample, the further apart the replicas of $X(f)$. Sampling every T seconds implies that the spacing between the replicas in the **frequency domain** is equal to the sampling frequency $f_s = 1/T$.

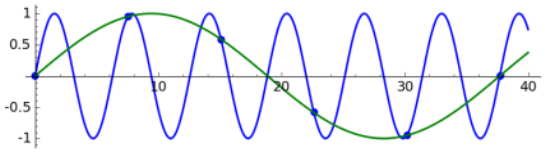
Drawbacks The hypothesis under which Nyquist frequency is valid are not always applicable to real world signals: there is no signal that is limited in band because a signal of this type would not have energy under a certain band of frequency and thus it must extend to infinity. For **periodic signals** the transform is performed by assuming a period $T \to \infty$. The hypothesis involves a number of infinite samples but in limited time we must limit the number of sample to a finite number, increasing the sampling frequency to better approximate the signal.

13.4.2 Aliasing

The *aliasing problem* arises when the *Sampling theorem* result it's not applied, thus we sample at a lower frequency. Suppose have a signal $s(t)$: the set of samples itself does not contains all the information to reconstruct the original signal, leading to an *incorrect frequency estimation*. We could have taken the following set of samples, shown in 13.27a. From this set we can reconstruct different signals, all different from the original one, with different frequencies, like the green one picture in 13.27b.



(a) Set of sample



(b) Wrong signal reconstruction (*blue is the real one, green is reconstructed by the samples*)

Starting from the given set of sample we could reconstruct an **infinite number of armonics**, undistinguishable from the armonics of the original signal, like the one pictured in 13.28:

In general, the *spectrum* of the **sampled signal** $S_c(t)$ consists of **infinite replicas** of the spectrum of $s(t)$, centered at frequencies multiple of $f_c = 3$, as shown in 13.29

This is the rationale behind the Nyquist condition to have $s(t)$ limited band: if we limit the signal, as in figure 13.30, at frequency $f_M = 1.5$ we can disregard all the replicas (*which have higher frequency*).

Hence, if $s(t)$ is **strictly limited in band** with null spectrum at frequencies above f_M, we can choose f_c to esnure that the replicas of the spectrum of $s(t)$ **do not overlap**, as shown in 13.31.

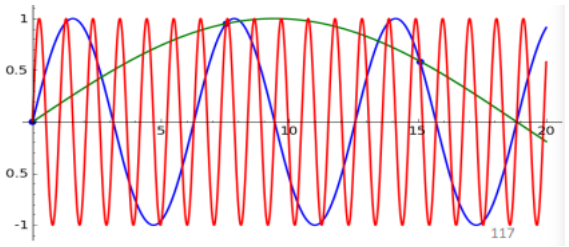


Figure 13.28: Infinite number of armonics from the given set of sample

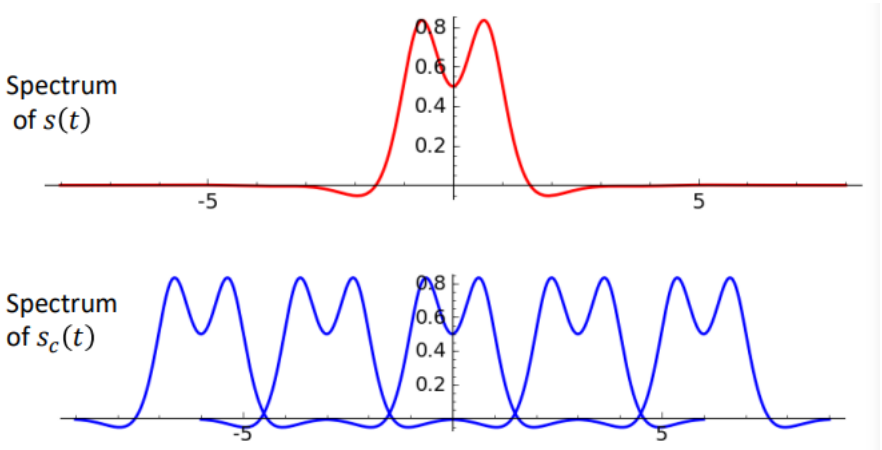


Figure 13.29: Infinite replcias of $s(t)$ centered at $f_c = 3$

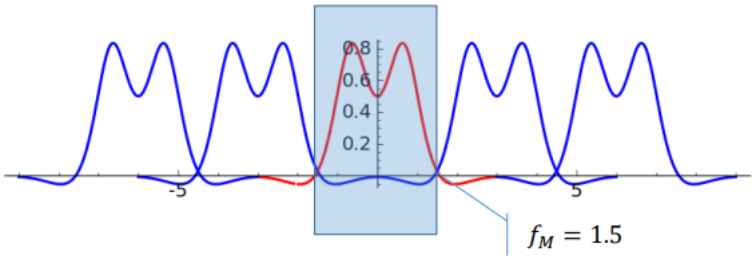


Figure 13.30: Limited signal at frequency $f_M = 1.5$

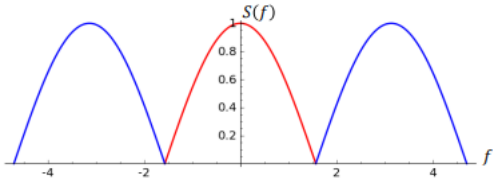


Figure 13.31: Non-overlapping spectrum replicas

Nyquist implication

No real-world signal is *perfectly limited in bandwidth*: such a signal shoud not have energy beyond a frequency band but for this must extended *infinitely in time*.
However, even if the signal is bounded in frequency there is no mean to sample a signal from the real-world and to rebuild the signal perfectly, unless we take an **infinite number of samples**: the bound of Nyquist is on the frequency of sampling, not on the number of samples required.

In practice, many signals are almost limited in a band, meaning that *their spectrum beyond their band is small* (*i.e. their energy beyond the band is small*). In any sampling you can expect a **little distorsion** due to this fact, but with a "*good*"" sampling this distortion is negligible. Furthermore, it is also possible to use a ***cutoff filter*** that cuts out frequencies above a threshold from a signal in order so to meet the

Nyquist requirements

Nyquist real-world application sometimes are based on wrong assumption or interpretation of the Nyquist Shannon theorem, as largely discussed in Sampling: What Nyquist Didn't Say, and What to Do About It - Tim Wescott. Here we summarize some mistakenly use cited by *Tim Wescott*'s work.

Mistake 1

- *"I am going to sample at 8kHz, so I need to use a filter with a 4kHz cutoff"*

Nyquist did not say that if you are sampling at rate N samples per second you can use an *anti-aliasing* filter with a *cutoff frequency* $f = N/2$. No analog filter is perfect, and there is a significant amount of frequencies above $4KHz$ that pass, despite can be attenuated. The result is that the aliased signal is not negligible and seriously disturbs your sampling. A possible solution is to **oversample**: samples are taken per unit of time than what is strictly necessary to satisfy the Nyquist criterion. An example is given by the old, analog telephone *powerline* that used a stronger *cutoff* of $3KHz$ and they oversampled at $8KHz$.

Mistake 2

- *"I need to monitor the 60Hz power line, so I need to sample at 120Hz"*
- *"We have a signal at 1kHz we need to detect, so I need to sample at 2kHz"*

Nyquist did not say that a signal that repeats N times a second has a bandwidht of N hertz. In fact, the powerline is nominally at $60Hz$ but in fact you can see a lot of distortion, meaning that there are many other frequencies involved.

If you dont' use *anti-aliasing filters*, you need to figure out the **maximum frequency of the signal**: if it come out that, as in this case, up to the *fifth harmonic* ($300Hz$) is meaningful, you should sample at least **twice that frequency** ($600Hz$).

So the quest should be rephrased: "*I need to monitor the 60Hz power line, so I guess need to sample at* ***well over 120Hz***". This is still *not entirely true*: in some cases we could even **downsample**. In this specific case, because the signal **repeat continuously, it's steady and with a known and steady frequency**,so downsampling allows still to reconstruct the signal well enough. Here an example:

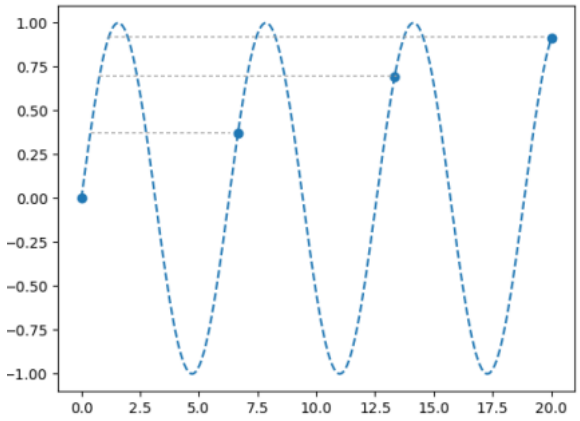


Figure 13.32: Downsampling still allows to reconstruct the signal

A hint is that the **signal phase** within a **cycle** will advance a little bit with each new sample. If we ***sample long enough***, we can have enough information to reconstruct the original signal, but we need to know that it is a **steady, repetitive signal with a steady frequency.**

13.5 Quantization

As mentioned, the after the sampling activity the obtained number are usually real $\mathbb{R}$, because is some cases the floating point representation it's costly. The **quantization** indicates an *approximation of the sampled value* so that the value can be represented in the interval $[0, 2^R - 1]$.

Consider a signal $s(t)$ with a **sample frequency** f_c: the value x_k is the readed sample in the time interval I_k. The quantization activity allows to encode x_k in a **integer value** $y_k \in [0, 2^R - 1]$, with R the **number of bit per sample**. This definition allows to define the **bit rate** for a signal sampled at frequency f_c as:

$$R_b = R \times f_c \text{ bit/sec}$$

The error can be quantified as **quantized error** expressed as $s(t) - y_k$ where y_k is the quantized value.

The *quantization noise* can be of two different type:

- **Overload**: each adjacent point in the set value is not capable of represent a value of $s(t)$ outiside the representation range, so $s(t) > 2^R - 1$. The value of the signal at time t exceeds the representation used by the quantization, leading to the scenario in 13.33. This can be solved by using some *flag bits* to represent higher or lower points
- **Granular noise**: occurs beause all values in interval I_k are represented with a unique y_k. The representation of the sampled value is not able to properly represent the minimal variations of the signal is a smaller value range, as shown in 13.34.

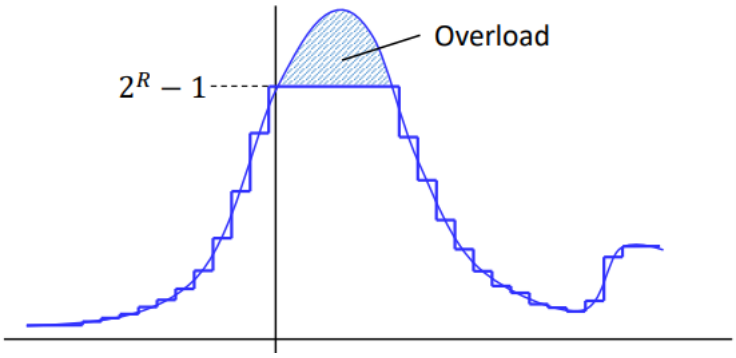


Figure 13.33: Overload

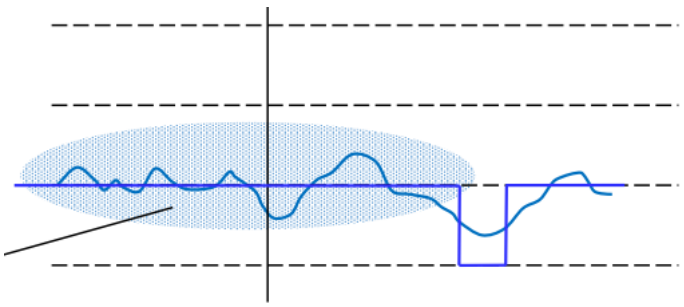


Figure 13.34: Granular noise: blu area variations are represented with the same y_k value.

Example The human ear can hear between 20 and 20k Hz. With an acceptable degradation in quality, it can be cut at around $4KHz$, hence the *sampling rate* $f_c = 8KHz$, each sample quantized on $R = 8$ bits and thus the bit rate $R_b = 64Kbps$.

Appendix A

Exercises

A.0.1 Exercise

Consider the following program and the table A.1 of energy consumption in the different states. Compute:

- the energy consumption of the device per single hour
- the expected lifetime of the device

```
1   void loop(){
2
3     //4 milliseconds
4     turnOn(analogSensor);
5     int sensorValue = analogRead(A0);
6     turnOff(analogSensor);
7
8     // 1 milliseconds
9     float voltaage = sensorValue*(5.0/1023.0);
10
11    //15 milliseconds
12    turnOn(radioInterface);
13    Serial.println(voltage);
14    turnOff(radioInterface);
15
16    //380 milliseconds
17    idle(380);
18
19  }
```

	value	units
Micro Processor (Atmega128L)		
current (full operation)	8	mA
current sleep	15	µA
Radio		
current xmit	1	mA
current sleep	20	µA
Sensor Board		
current (full operation)	5	mA
current sleep	5	µA
Battery Specifcations		
Capacity	2000	mAh

Figure A.1: Exercise 1 parameters

Answer

Consider the energy consumption per hour:

1. The processor has a duty cycle of: always active so $20ms/20ms$
2. The radio has a duty cycle of: $15ms/20ms$
3. The sensor has a duty cycle of: $4ms/20ms$

1. Sensor power consumption: $E_\eta = 5mAh * 0.2 + 5\mu Ah * 0,98$
2. Processor power consumption: $E_\rho = 1 * 8mAh + 0 * 15\mu Ah$
3. Radio power compsution: $E_\lambda = 0,75 * 1mA + 0,25 * 20\mu A$
4. The energy consumption in one hour is: $E = E_\eta + E_\rho + E_\lambda$
5. Battery specification (*in table*) $B_0 = 2000mAh$
   Power consumption: $2000mAh/E$

Exercise 2

Consider the sensor specification in the table pictured in A.2

	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 Specifications		
Capacity	2000	mAh

Figure A.2: Exercise 2 - Parameters specification

The device measures the hearth-rate (HR) of a person:

- Samples a photo-diode on the wrist at 20 Hz
  - sampling the sensor takes 0.5 ms
  - it requires both the processor and the sensor active
- HR is computed every 2 s (based on 40 samples)
- Transmit (from time to time... see below) a data packet to the server:
  - The average time required to transit is 2 ms
  - Requires both processor and radio active

Compute the energy consumption and the lifetime of the device if it sends all the samples to a server:

- Stores 5 consecutive samples from the photodiode
- Transmits the stored 5 samples to the server
- The server computes HR (hence the device does not compute HR)

Disregard battery leaks.

	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 Specifications		
Capacity	2000	mAh

Figure A.3: Exercise 3 - Parameters specification

Solution 2

Duty cycle of sampling: DC of processor + sensors: 0.5 milliseconds (sampling time) / 0.05 seconds = (sampling period)= 0.01

Duty cycle of transmitting: DC of radio + processor: 2 milliseconds (transmit time) / 0,25 seconds (transmission period) = 0.008

- Sensor power consumption: $E_\eta = 5mAh * 0,01 + 5uAh * 0,99 = 0,05 + 0,005mAh = 0,055mAh$
- Processor power consumption: $E_\rho = 0,018 * 8mAh + 0,982 * 15uAh = 0,144 + 0,0147mAh = 0,1587mAh$
- Radio power compsution: $E_\lambda = 0,008 * 20mA + 0,99992 * 20\eta A = 0.16mA + 0.0198mA = 0.1799mAh$

$$E = E_\eta + E_\rho + E_\lambda$$
Battery specification (*in table*) $B_0 = 2000mAh$
Power consumption: $2000mAh/0,3935mAh$ ($\frac{B_0}{E}$)
obtaining **Lifetime**: $5082h$

A.0.2 Exercise 3

Consider the sensor specification in the table pictured in A.3.
The device measures the hearth-rate (HR) of a person:

- Samples a photodiode on the wrist at 20 Hz
  - sampling the sensor takes 0.5 ms
  - it requires both the processor and the sensor active
- HR is computed every 2 s (based on 40 samples)
  - Computing HR in the device takes 5 ms
- Transmit a data packet to the server:
  - The average time required to transmit is 2 ms
  - Requires both processor and radio active Compute the energy consumption and the lifetime of the device if it computes HR itself:
- Transmits every 5 values of HR computed (1 packet every 10 seconds)

Disregard battery leaks.

	value	units
Micro Processor (Atmega128L)		
current (full operation)	8	mA
current sleep	0,015	mA
Radio		
current xmit	1	mA
current sleep	0,02	mA
Sensor Board		
current (full operation)	5	mA
current sleep	0,005	mA
Battery Specifications		
Capacity	2000	mAh

Figure A.4: Exercise 4 - Parameters specification

Solution 3

- Duty cycle of sampling: $0,5ms$ (sampling time) / $0,05s$ (sampling period)= $0,01$
- Duty cycle of processing: 5 milliseconds / 2 seconds = 0,0025
- Duty cycle of transmitting: 2 milliseconds (transmit time) / 10 seconds (transmission period) = 0,0002
- Power consumption of sensor (in 1h):
  - $E_\eta = 5mAh * 0,01 + 5uAh * 0,99 = 0,05 + 0,005mAh = 0,055mAh$
- Power consumption of processor (1h):
  - $E_\rho = 8mAh * 0,0127 + 15uAh * 0,9873 = 0,1016 + 0,0148mAh = 0,1164mAh$
- Power consumtpion of radio (1h):
  - $E_\theta = 0,0002 * 20mAh + 0,9998 * 20uAh = 0,004 + 0,02mAh = 0,024mAh$

$$E = E_\eta + E_\rho + E_\theta = 0.1954mAh$$
Battery specification (*in table*) $B_0 = 2000mAh$
Power consumption: $2000mAh/0,1954mAh$ ($\frac{B_0}{E}$)
Lifetime: $2000mAh/0,1954mAh = 10.235h$

A.0.3 Exercise 4

Consider a Mote-class sensor with the parameters pictured in table A.4.
Assume that the device performs a sensing task with the following parameters:

- The sensor board is activated with a rate of 0,1 Hz to perform the sampling; this operation takes 0.5 milliseconds. At the end the sensor board is put in sleep mode. During each sensing operation the processor is always active.
- After each sampling the processor performs a computation that takes 2 milliseconds.
- Then the processor activates the radio and transmits the data. The transmission takes 1 millisecond and, during it, the processor is active. At the end the radio and the processor are both set in sleep mode.

Compute the duty cycle of each component (*sensor board, radio and processor*), and the lifetime of the device (*assuming that the sensor stops working when its battery charge becomes 0*).

Solution 4

- Sampling takes $0.5ms$ with a rate of $0,1Hz$
- Processing: $2ms$
- Transmitting: 1 ms
- **Duty cycle of sampling (processor and sensors)**: $0.5ms/10s = 0,00005$
- **Duty cycle of processing (only processor)**: $2ms/10s$ $(10000ms) = 0.0002$
- **Duty cycle of transmissions (radio&processor)**: $1ms/10s = 0.0001$
- Power consumption of sensor (in 1h):
  - $E_\eta = 5mA * 0,00005 + 0,005mA * (1 - 0,00005) = 2.5 * 10^{-4}mA + 0.05mA = 0.00525mA$
- Power consumption of processor (1h):
  - $E_\rho = 8mA * 0.00035 + 0,015mA * (1 - 0.00035) = 0.0028mA + 0,015mA = 0,0178mAh$
- Power consumtpion of radio (1h):
  - $E_\theta = 0,0001 * 1mA + (1 - 0,0001) * 0,02mA = 100nA + 20\eta A = 0.0201mA$

- $E = E_\eta + E_\rho + E_\theta = 0.04315mAh$

Battery specification $(in\ table)$ $B_0 = 2000mAh$
Power consumption: $2000mAh/0.04315mAh$ $(\frac{B_0}{E})$
Lifetime: $2000mAh/0,04315mAh = 46.349h$

A.0.4 Exercise 5

Consider the following fragment of Arduino code:

```
void loop() {
  int sensorValue = analogRead(A0);
  Serial.println(sensorValue);
  delay(100);
}
```

Compute its duty cycle assuming that:

- Reading an analog value takes 2 milliseconds
- Transmitting along the serial line takes 5 milliseconds

A.0.5 Exercise 6

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

- In the interval $[0, 10sec]$ the energy production is constant and produces $P_s(t) = 80mA$
- In the interval $[0, 4sec]$ the load is $P_c(t) = 150mA$
- In the interval $[4sec, 10sec]$ the load is $P_c(t) = 20mA$
- The charging efficiency is $\eta = 95\%$
- The battery charge at time 0 is $400mAh$
- The energy leak of the battery is negligible

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

A.0.6 Exercise 7

Consider a device that samples the battery output voltage with an analog to digital converter (ADC) at 10 bits. The battery has a maximum charge of $2000mAh$, and its maximum voltage (*when fully charged*) is 10 Volts. When the battery reaches a voltage of 8 Volts the battery charge is $200mAh$ and it becomes insufficient to power the device.
Compute the **battery charge** B when the ADC outputs $920, 830, 1023$.

Solution Summarize the data we already known:

- d : 10 bits
- B_{min} : $200mAh$
- B_{max} : $2000mAh$
- v_{min} : $8V$
- v_{max} : $10V$

Now compute:

- $x_{max} = 2^d - 1 = 1023$
- $x_{min} = ROUND[\frac{v_{min}}{v_{max}} \times (2^d - 1)] = 818$

Hence, when the ADC outputs:

- $x = 920 \rightarrow B = 1096mAh$ (*half the charge*)
- $x = 830 \rightarrow B = 305mAh$
- $x = 1023 \rightarrow B = 2000mAh$ (*full charge*)

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

A.0.7 Exercise 8

The following table reports, for each slot in a day, the expected energy production of an energy-harvesting IoT device. Assuming that:

- $dc_{max} = 90\%; p_{max} = 5mA; u(dc_{max}) = 100$
- $dc_{min} = 50\%; p_{min} = 1mA; u(dc_{min}) = 10$

Complete the table A.5 by assigning a duty cycle, an utility, and a power consumption to each slot, according to the Kansal's algorithm.

Slot	1	2	3	4	5	6	7	8	9	10	11	12
$\tilde{p}_s(i)$	0	0	0	2	6	11	10	7	3	0	0	0
dc_i												
$u(\cdot)$												
$p(\cdot)$												

Figure A.5: Exercise 8 table

A.0.8 Exercise 9

In the Zigbee network in the figure A.6, built using parameters $Rm = 2, Dm = 2, Lm = 3$, where purple shaded nodes are routers and white nodes are end devices, what range of addresses are assigned to new router that joins the network in the following cases:

- The new router joins as a child of router 1
- The new router joins as a child of router 20
- The new router joins at node 7

Figure A.6: Exercise 9 Network

A.0.9 Questions 1

Given the network in the figure A.7, assume that the MAC protocol uses the RTS/CTS mechanism for the channel access.
Discuss which nodes detect themselves as hidden or exposed as consequence of the RTS/CTS handshake in the following cases:

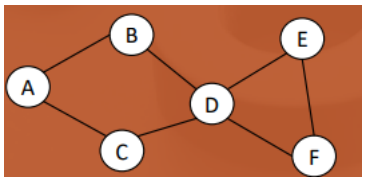


Figure A.7: Question 1 network schema

- Hidden terminals with respect to a transmission from E to D
- Hidden terminals with respect to a transmission from D to C
- Exposed terminals with respect to a transmission from D to B
- Exposed terminals with respect to a transmission from B to A

A.0.10 Questions 2

Given the network in the previous figure A.7:

- assume that D hears the RTS sent by E but it does not hear the corresponding CTS. What does D can do?
- assume that B hears the CTS sent by D but it does not hear the corresponding RTS. What does B can do?
- Assume D is receiving a communication from a node, and B did not receive the corresponding RTS & CTS and it does not hear the signal transmitted to D. If B wishes to transmit to D what happens?

The solution are:

- Exposed
- Hidden
- Collision

A.0.11 Question 3

- In type C configuration A.8, 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?

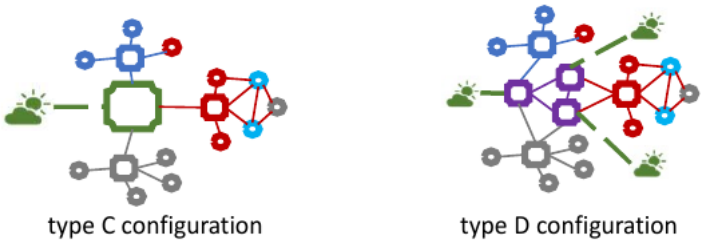


Figure A.8: Question 3 schema

A.0.12 Question 4

Explain the meaning of the following topics used in a MQTT-based system:

- Pisa/neighborhood_Cisanello/lamppost/TrafficSensor
- Pisa/neighborhood_Cisanello/+/TrafficSensor
- Pisa/+/lamppost/#
- Pisa/neighborhood_Cisanello/#
- $SYS/broker/clients/total

A.0.13 Question 5

Consider the binding table and the address maps shown below A.9 that represent the state of the ZigBee network at time t. Assume device 0X0022 disconnects from the network at time $t' > t$ and it reconnects again at time $t'' > t'$, obtaining network address 0X0003. Discuss to what network address and endpoint are delivered the following messages:

Src Addr (64 bits)	Src EP	Cluster ID	Dest Addr (64 bits)	Addr/Grp	Dest EP
0x00...02	5	0x0006	0x22...91	A	12
0x10...A3	6	0x0006	0x22...91	A	240
0x10...A3	5	0x0006	0x00...02	A	34
0x22...91	4	0x0006	0x10...A3	A	44

IEEE Addr	NWK Addr
0x0030D237B0230102	0x0000
0x1030B237B0235CA3	0x0001
0x2231C237b023A291	0x0022

Figure A.9: Question 5 table

- Time $h < t'$: message of cluster 0X0006 generated by device 0X0001 from endpoint 5
- Time $h \in [t', t'']$: message of cluster 0X0006 generated by device 0X0000 from endpoint 5
- Time $h > t''$: message of cluster 0X0006 generated by device 0X0022 from endpoint 4
- Time $h > t''$: message of cluster 0X0006 generated by device 0X0001 from endpoint 6

A.0.14 Question 6

The radio of a device using B-MAC takes 4.0^{-04} seconds to check for the preamble. Assuming that the frequency of the preamble sampling is $5/sec$ (*that is, preamble sampling is performed* 5 *times in a second*). What is the duty cycle of the preamble sampling activity?

A.0.15 Question 7

Referring the chapter on Direct Diffusion 9.1, answer the following questions:

- What is a gradient?
- Why does an interest is repeated periodically? Isn't the first one sufficient?
- How does the mechanism of directed diffusion may be used to optimize the underlying MAC layer?? (cross-layer optimization)
- In GPSR 9.2, when does a packet can switch back from perimeter mode to greedy mode?