Mobile and Cyber Physical Systems - Appunti

Francesco Lorenzoni

February 2024

Contents

Course info

...

Part I

Stefano Chessa

Chapter 1

Internet of Things

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

1.1 IoT introduction

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

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

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

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

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

1.2 Platforms for IoT

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

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

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

1.3 No-SQL Databases

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

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

1.4 IoT Issues

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

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

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

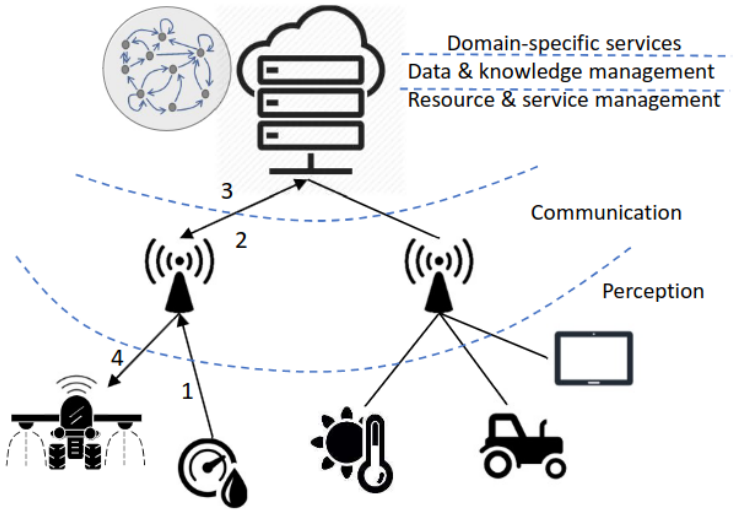


Figure 1.1: Communication outline in IoT
IoT systems are distributed, and servers may be dislocated around the globe, making room for **latency** and **reliability** issues.

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

Key Issues

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

1.4.1 Edge and Fog computing

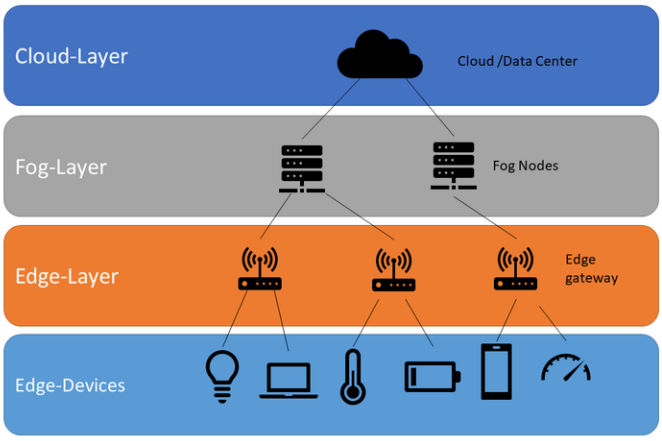


Figure 1.2: Layers scheme

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

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

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

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

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

1.4.2 Artificial Intelligence

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

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

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

Supervised and Reinforcement learning are more promising than

1.4.3 Blockchain & IoT

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

1.4.4 Interoperability

Vertical Silos

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

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

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

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

⇒ Without these conditions the standards will most likely fall

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

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

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

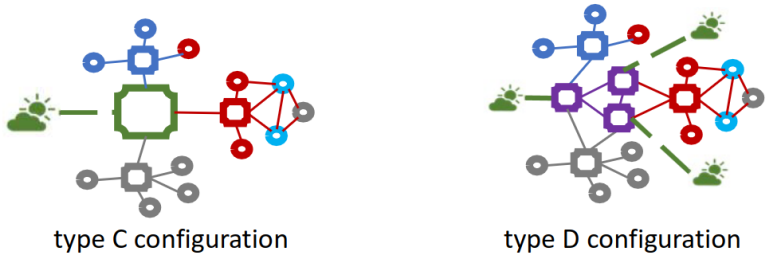


Figure 1.3: Gateway configs

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

1.5 Security in IoT

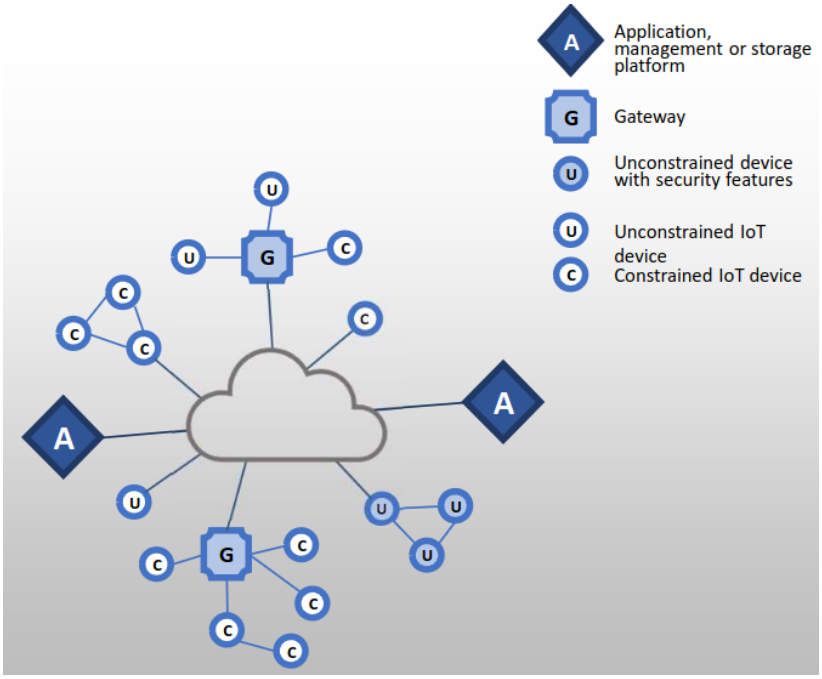


Figure 1.4: Security elements of interest

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

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

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

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

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

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

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

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

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

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

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

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

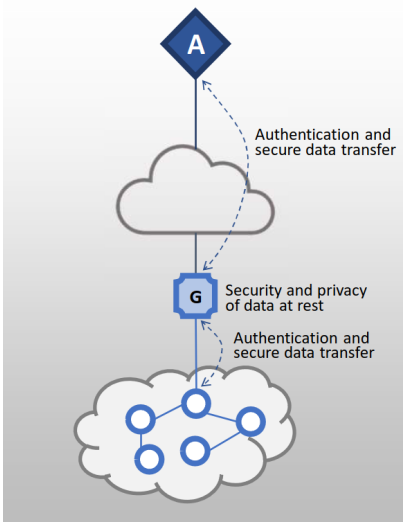


Figure 1.5: Gateways security functions

Chapter 2

MQTT

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

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

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

Lightweight

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

2.1 Publish-Subscribe recalls

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

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

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

The **Broker**:

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

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

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

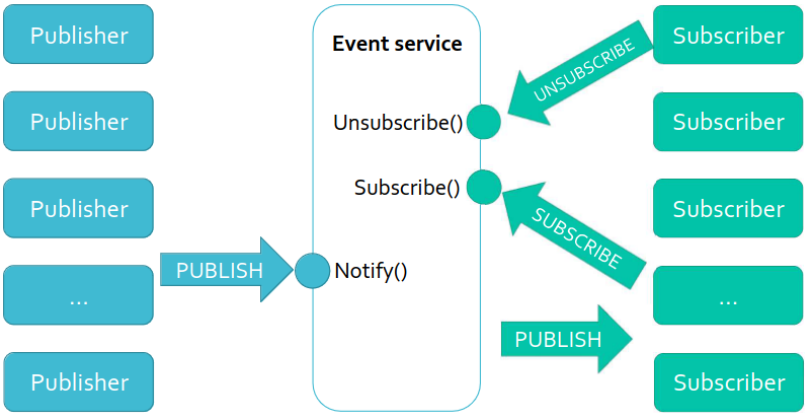


Figure 2.1: Broker management of events

2.1.1 Properties

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

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

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

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

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

2.2 MQTT and Publish-Subscribe

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

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

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

2.3 Messages

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

- ◇ Client ID

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

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

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

optional

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

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

`PUBLISH`

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

`SUBSCRIBE`

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

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

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

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

2.4 Topics

TODO

2.5 QoS

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

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

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

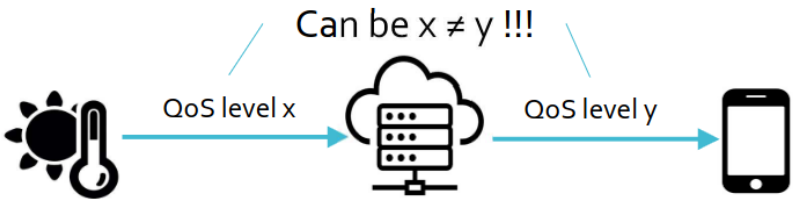
`level 0` **At most once**
- ◇ It is a "best effort" delivery and messages are *not* acknowledged by the receiver

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

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

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

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



2.5.1 Choosing the right QoS

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

2.6 Persistent Sessions

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

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

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

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

2.7 Retained messages

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

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

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

2.8 Last will & testament

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

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

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

2.9 Packet Format

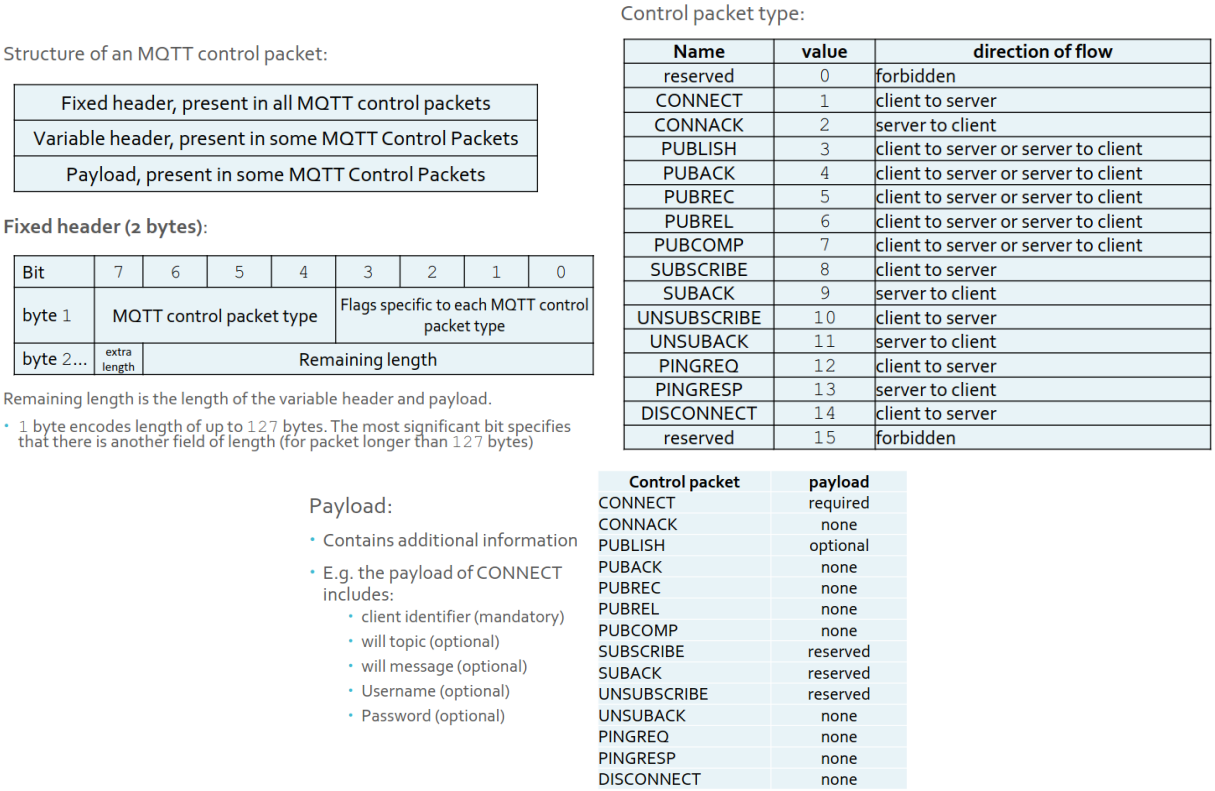


Figure 2.2: MQTT Packet headers

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

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

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

Chapter 3

ZigBee

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

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

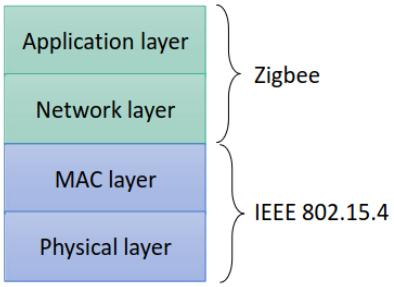


Figure 3.1: ZigBee layers

Key Features

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

[a] 250m outdoors in ideal conditions

3.1 Architecture

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

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

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

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

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

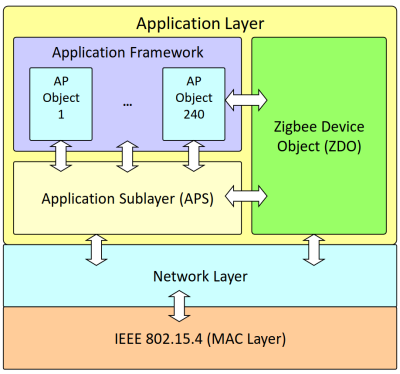


Figure 3.2: Zigbee architecture layers

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

3.2 Primitives

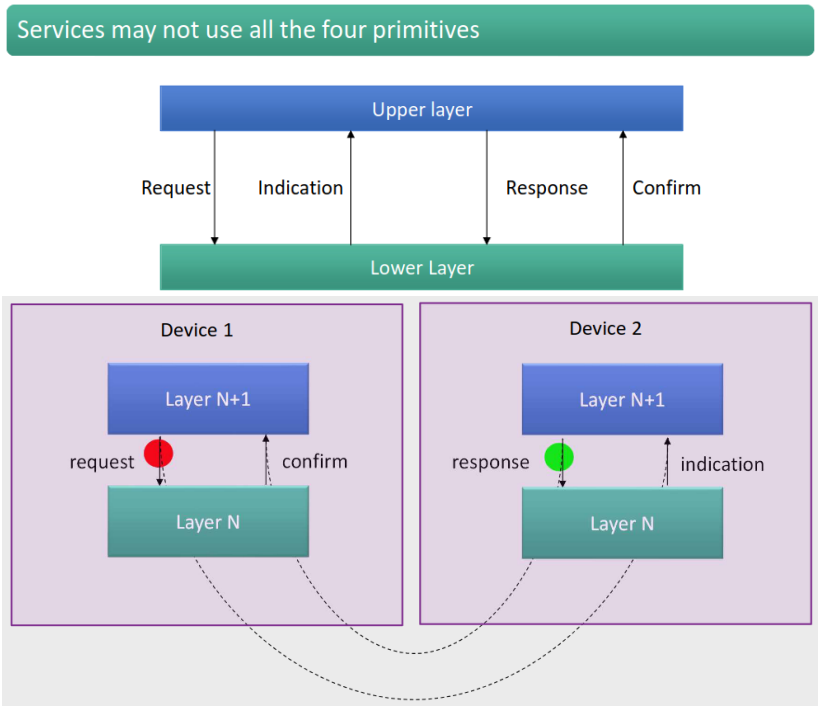


Figure 3.1: Mapping between zigbee primitives

Primitives

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

3.3 Network Layer

The ZigBee network layer provides services for:

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

In a ZigBee network there are three kinds of devices:

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

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

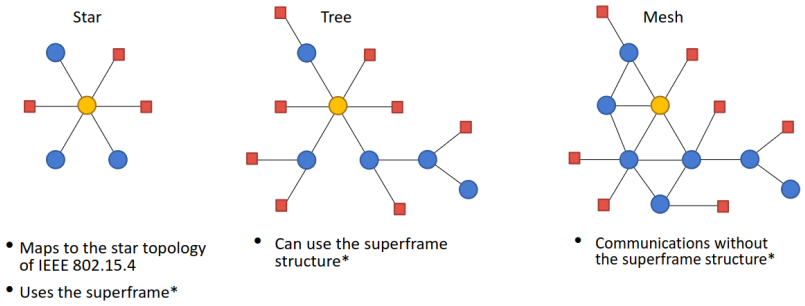


Figure 3.2: ZigBee Network topologies outline

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

3.3.1 Network formation and joining

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

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

The role of the device is chosen at compile-time

Formation

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

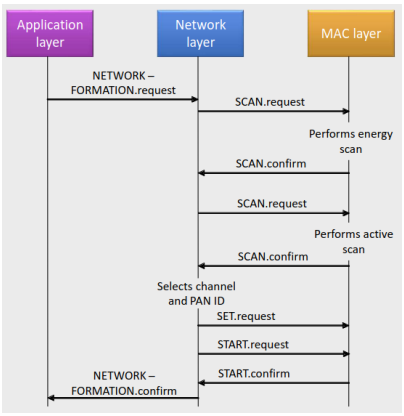


Figure 3.3: Network formation messages

Joining

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

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

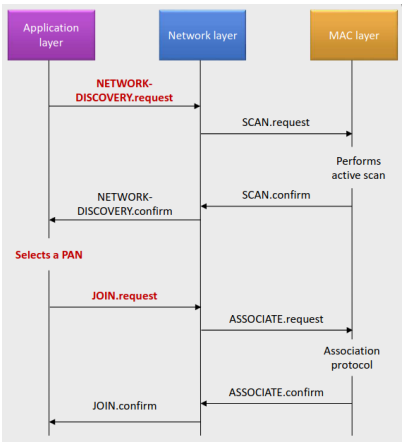


Figure 3.4: Network joining messages

3.4 Application Layer

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

3.4.1 APS - Application Support Sublayer

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

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

Concepts and related IDs

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

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

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

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

Using this schema, 16 bits become sufficient.

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

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

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

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

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

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

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

Figure 3.5: ZigBee General Domain clusters and common Profile IDs

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

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

Back to APS Services

APS Provides:

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

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

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

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

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

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

3.5 Binding

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

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

3.5.1 APS - Address Map

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

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

Figure 3.3: Address Map

3.5.2 APS - Binding

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

Primitives

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

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

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

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

Figure 3.4: Binding table

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

3.6 ZDO - ZigBee Device Object

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

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

ZDO services

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

3.6.1 Device and service discovery

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

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

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

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

3.6.2 Binding management

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

3.6.3 Network and Node Management

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

3.7 ZigBee Cluster Library

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

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

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

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

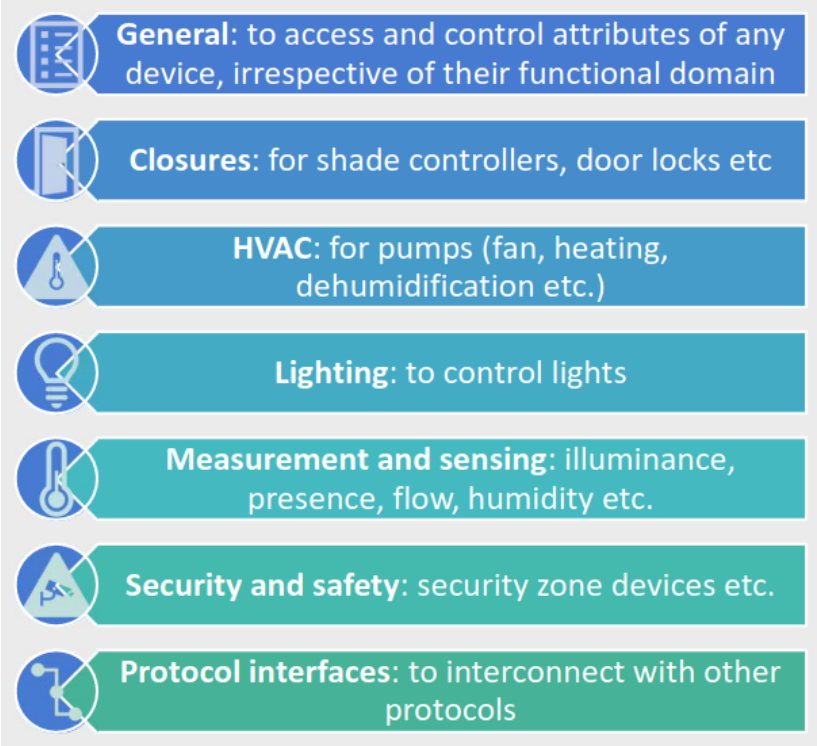


Figure 3.5: Functional domains

TODO integrate some slides

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

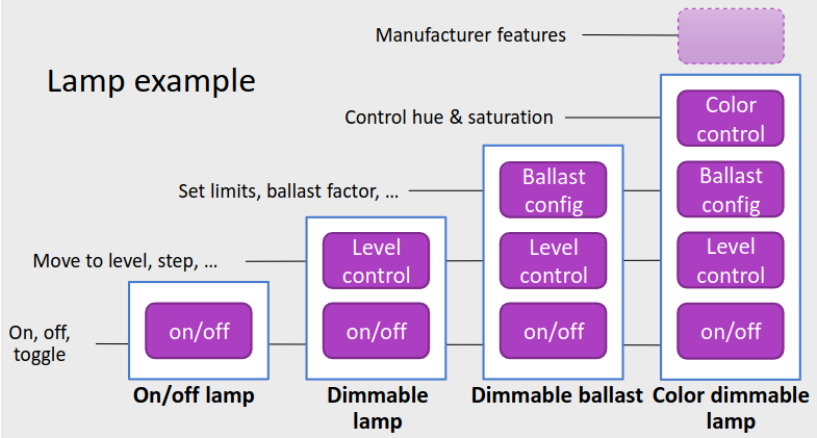


Figure 3.6: ZCL Hierarchical approach

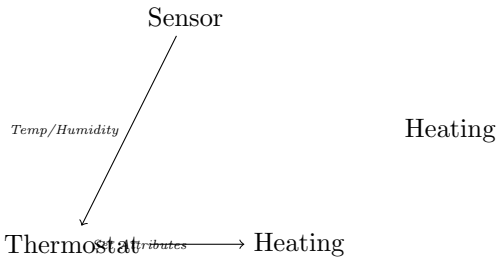


Figure 3.7: Heating System - exercise 2 Schema

Part II

Federica Paganelli

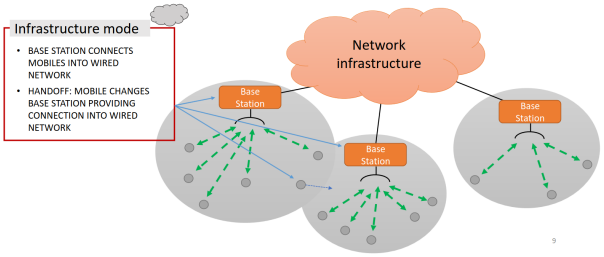
Chapter 4

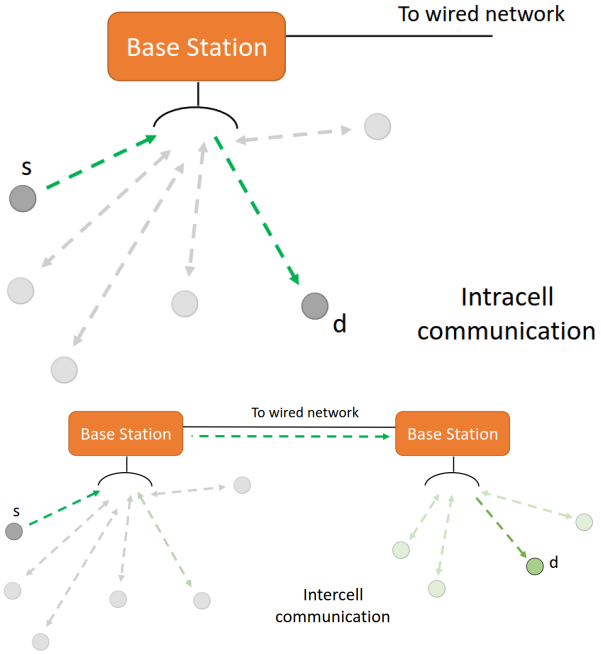
Wireless Networks

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

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

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





4.1 Link Layer

4.1.1 CSMA/CD

Basic idea of CSMA/CD:

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

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

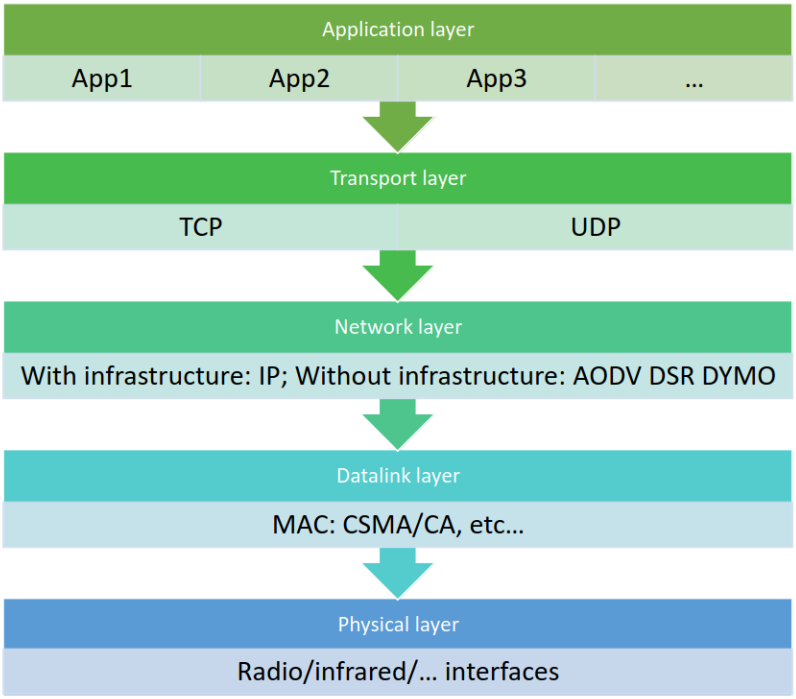


Figure 4.1: Protocol stack

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

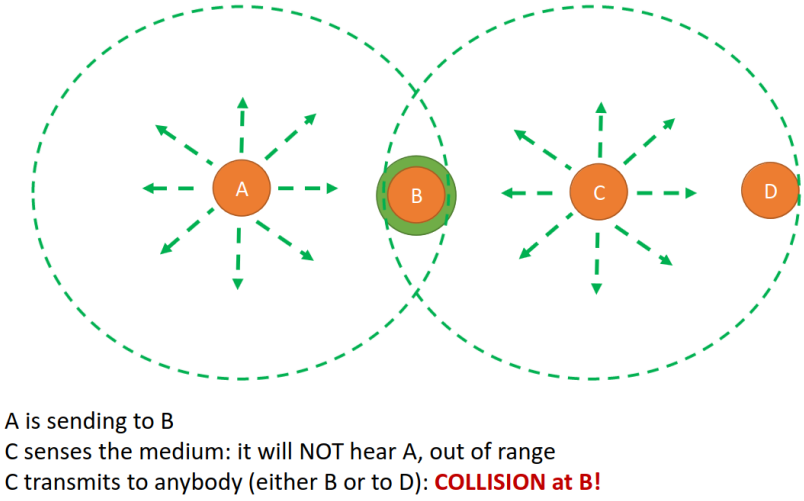


Figure 4.2: **Hidden Terminal** problem
Two or more stations which are out of range of each other transmit simultaneously to a common recipient

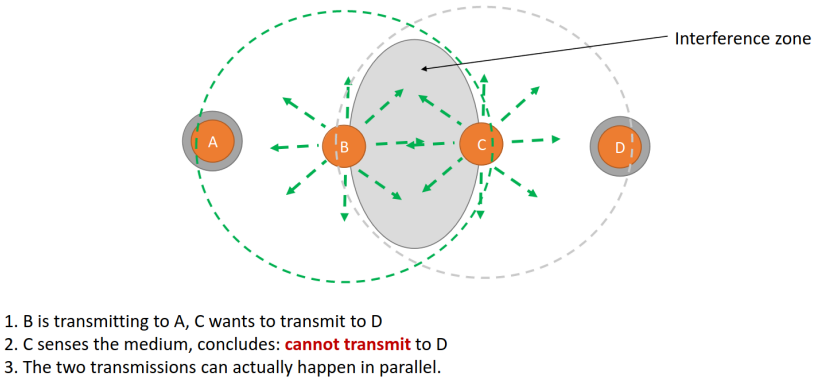


Figure 4.3: **Exposed Terminal** problem
A transmitting station is prevented from sending frames due to interference with another transmitting station

4.1.2 MACA

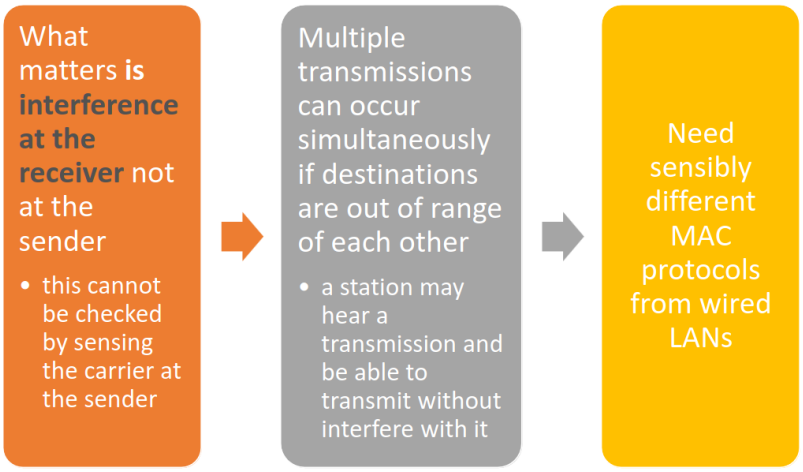


Figure 4.4: MACA Motivations

MACA stands for *Multiple Access with Collision Avoidance*

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

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

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

MACAW implements some improvements to MACA:

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

Chapter 5

IEEE 802.11

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

Figure 5.1: IEEE 802.11 standards

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

TODO

Chapter 6

Mobile Networks

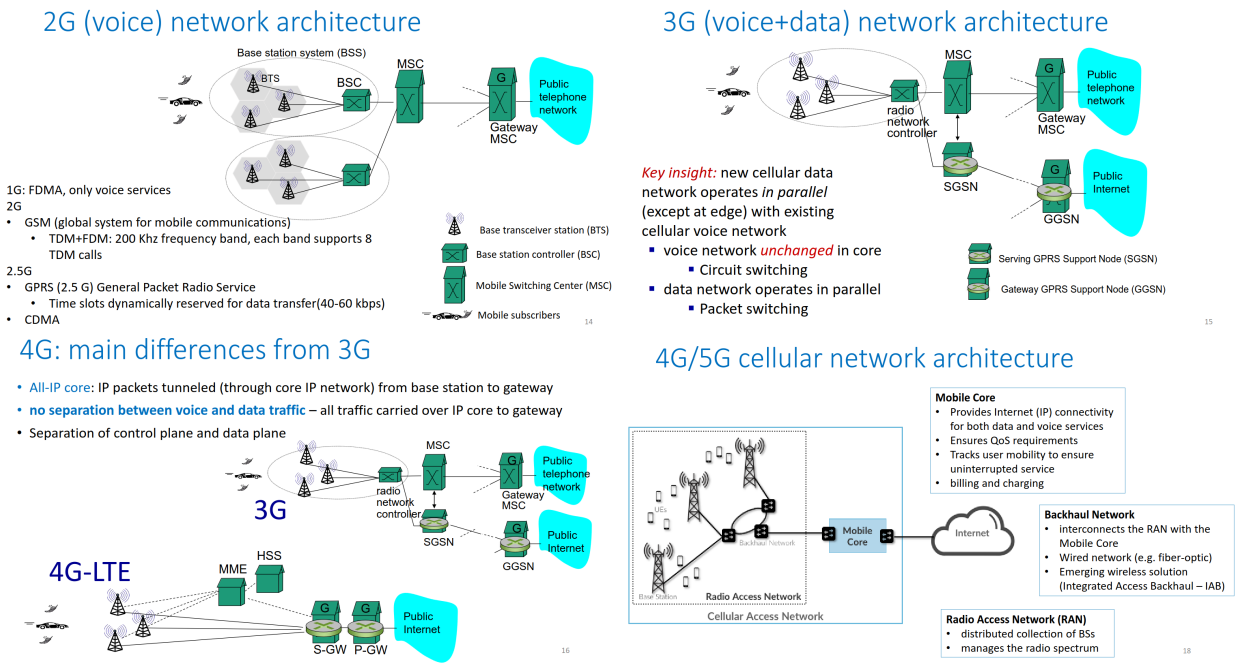


Figure 6.1: Mobile Networks architectures

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

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

Control vs Data plane

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

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

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