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 (?)

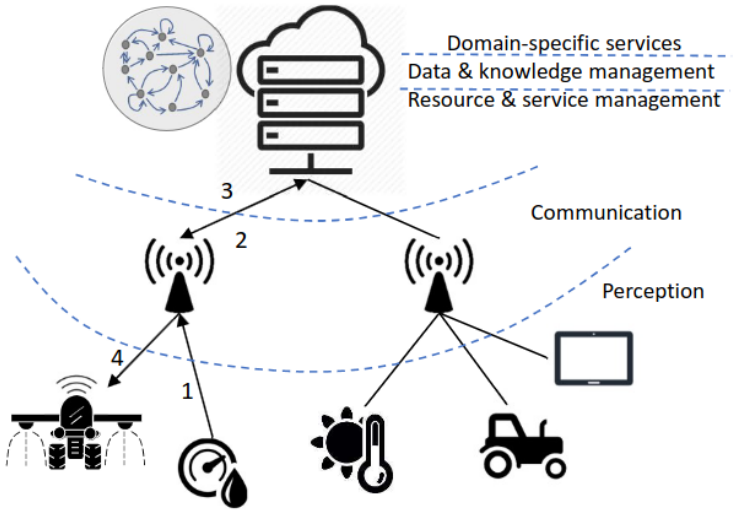


Figure 1.1: Communication outline in IoT
IoT systems are distributed, and servers may be dislocated around the globe, making room for **latency** and **reliability** issues.

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

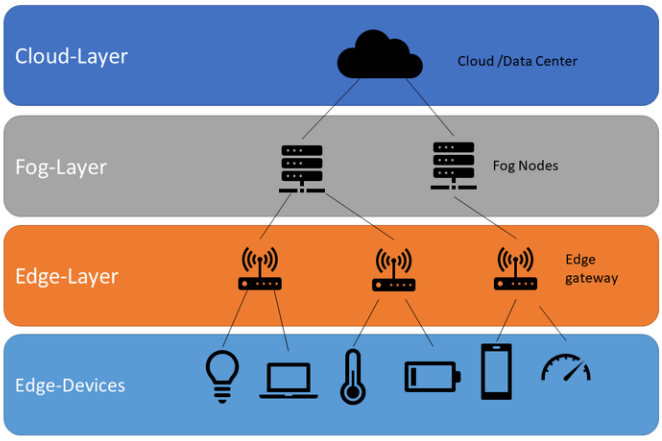


Figure 1.2: Layers scheme

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?

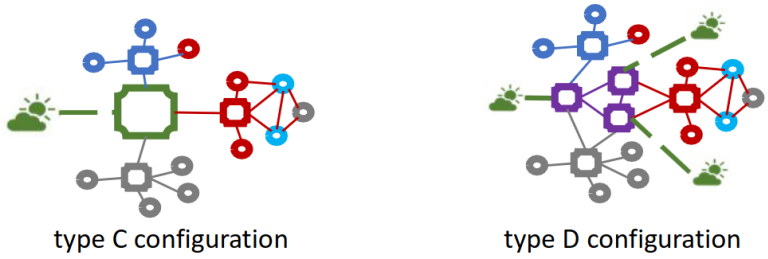


Figure 1.3: Gateway configs

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

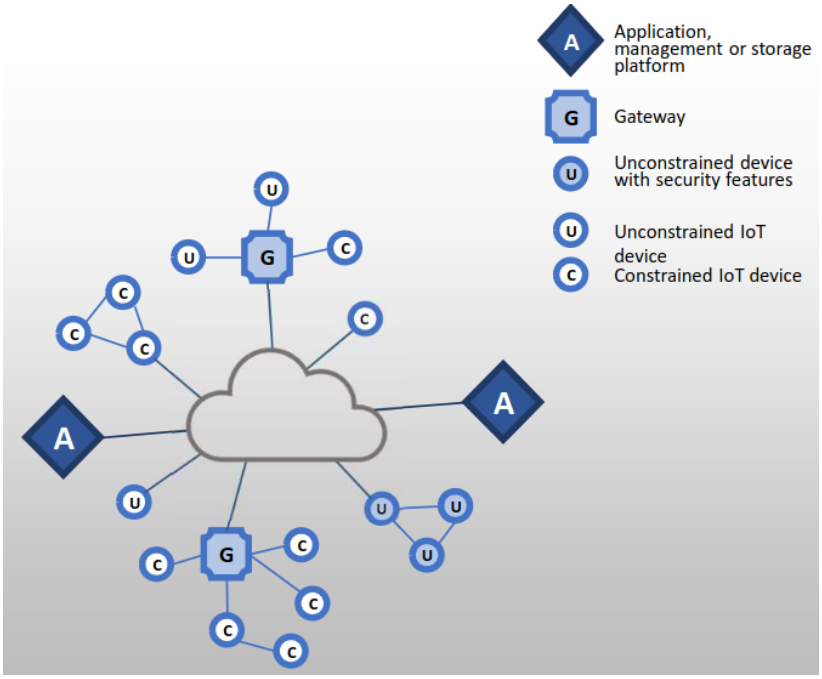


Figure 1.4: Security elements of interest

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.

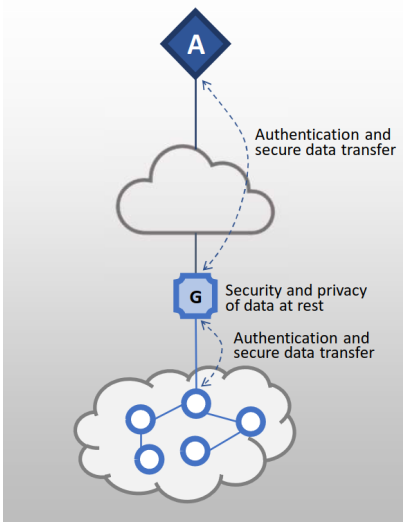


Figure 1.5: Gateways security functions

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*

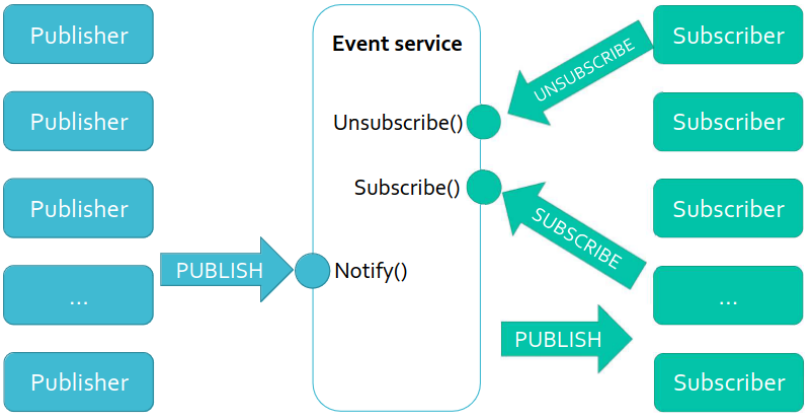


Figure 2.1: Broker management of events

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 `clientID` and does not keep a status for the client
    In this case *Clean Session* must be TRUE
- ◇ 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 keepalive 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

`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**

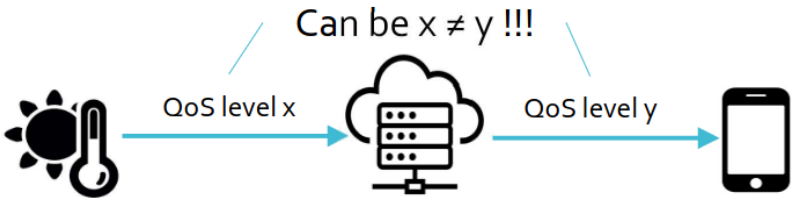
- ◇ Messages are numbered and stored by the broker until they are delivered to all subscribers with QoS level 1. Each message is delivered at least once to the subscribers with QoS, but possibly also more.

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

`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.

Part II

Federica Paganelli

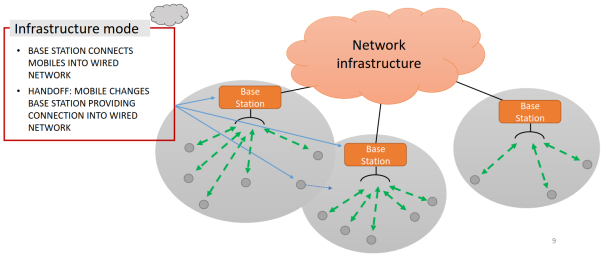
Chapter 3

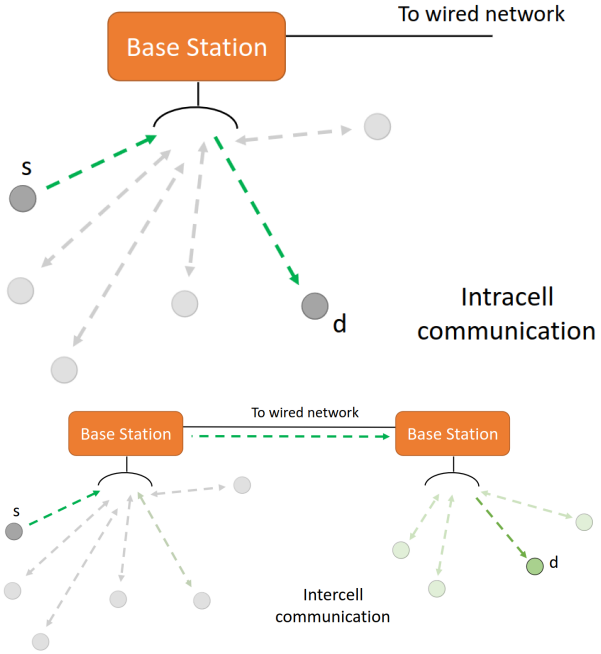
Wireless Networks

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*.





3.1 Link Layer

3.1.1 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

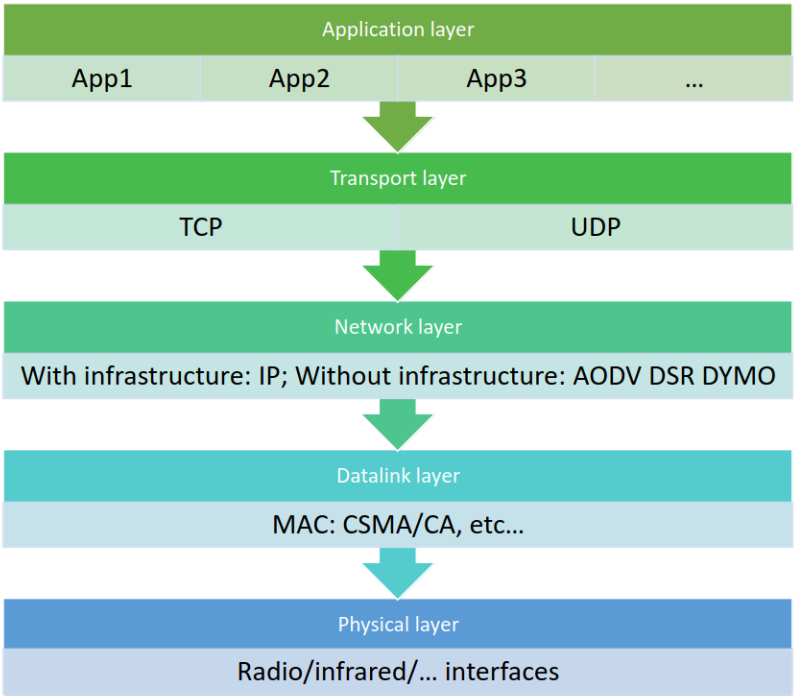


Figure 3.1: Protocol stack

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.

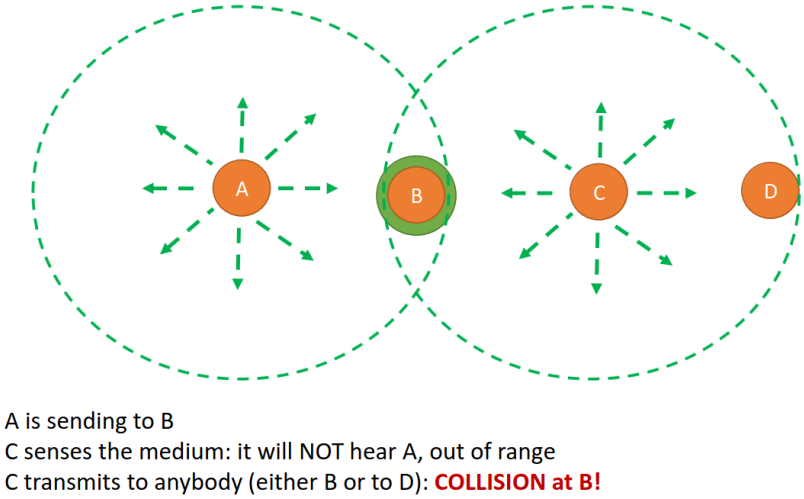


Figure 3.2: **Hidden Terminal** problem
Two or more stations which are out of range of each other transmit simultaneously to a common recipient

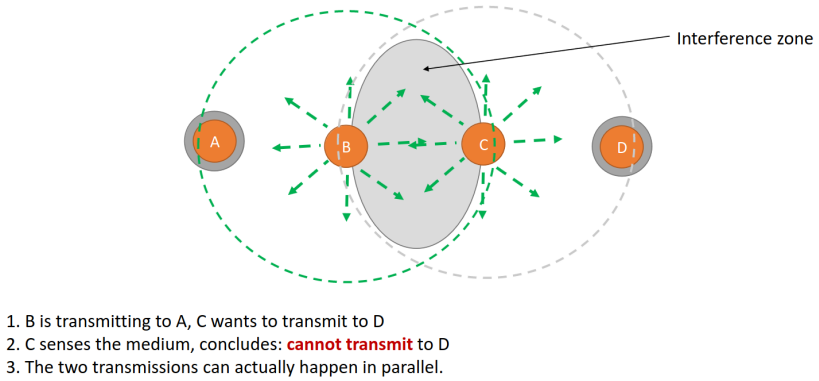


Figure 3.3: **Exposed Terminal** problem
A transmitting station is prevented from sending frames due to interference with another transmitting station

3.1.2 MACA

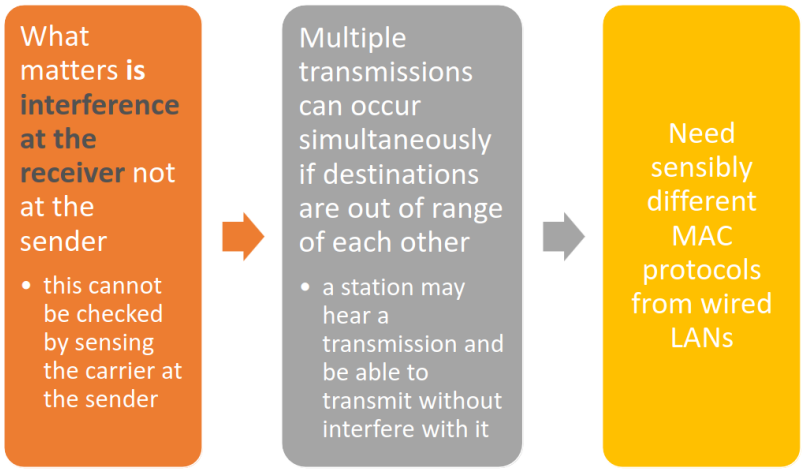


Figure 3.4: MACA Motivations

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 4

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 4.1: IEEE 802.11 standards

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