INTERNET OF THINGS

A.Y. 2022-23

Anastasi, Righetti, Vallati

(theoretical part)

Luca Arduini

last update: 2023-02-29

Disclaimer

These notes were taken during lectures by professors Anastasi, Righetti and Vallati for the "Internet of Things" course in the Master's Degree program in "Artificial Intelligence and Data Engineering" at the University of Pisa, during the academic year 2022-2023. The notes have not been reviewed by any instructor and do not constitute official course material.

Please use these notes as a supplementary resource for your studies and avoid relying solely on them. All images included are copyrighted and have been obtained from the professors' slides.

I hope you find these notes helpful, and I wish you the best of luck in your studies!

Special thanks

I would like to express my gratitude to my colleague Davide Bruni for sharing his course notes with me, which I have then expanded upon to create this final version of the notes.

Further insights into my notes

Did you find any errors or paragraphs that could have been written more clearly while reading these notes? Feel free to contact me on Telegram, and I'll be happy to hear your suggestions.

Have you found these notes helpful? Visit my GitHub page for additional notes and explore all the projects I've completed during my academic journey.

Sommario

Course segment held by Professor Vallati

1- Preliminary concepts

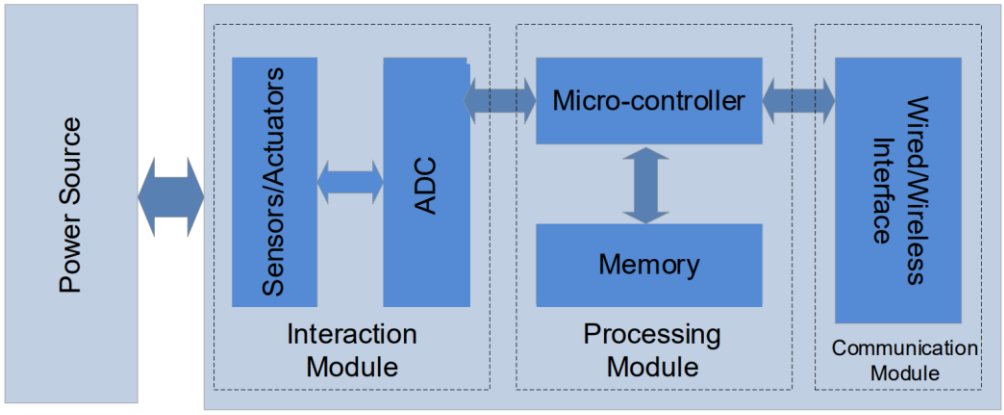
What is IoT? Any object can be connected: ***anytime, anywhere, by anyone and anything.***
Anything means literally anything, any real objects (cars, food, drugs, ecc.)

But how can we connect real object in order to make them "smart objects"?
We need to ***empower*** with the following capabilities: ***actuating, connectivity, sensing and computing capabilities***.
Of course ***they need power source*** in order to feed electronic circuits.

Consumer IoT devices

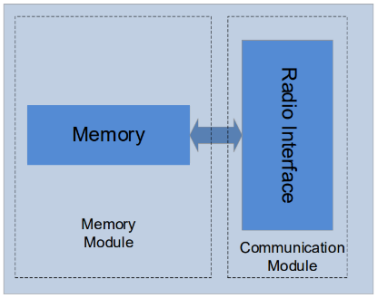
Industrial IoT devices

Devices architecture:



- Power source: we could have problems, for example if we use battery, after a certain amount of time we have to replace it. So we can use a solar panel or a DC source of electricity. It depends on the domain application and the specific situation.
- Interaction module: we can have one or more sensors/actuators which collect data from the environment and can interact with it.
- Processing module: processors with limited capabilities are presents and also the memory is very limited.
- Communication module: usually wireless because is flexible but it depends on the specific of the application

In some case, only a subset of these capabilities are important and needed to be implemented: for example in NFC tags. The reader is necessary for communication.

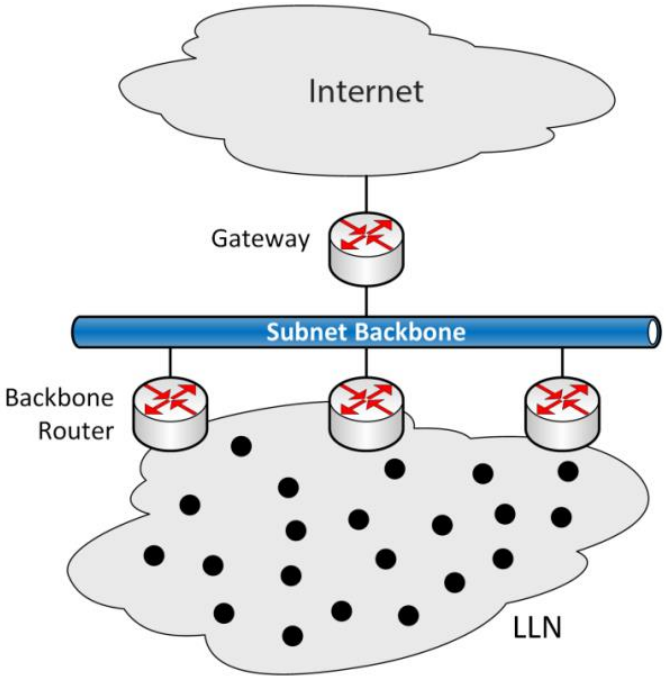


This is the tag architecture (not the reader).

The processing module is into the reader, which read and extract information.

Smart Objects typically are part of a distributed system, where different smart objects cooperate to perform a specific task. Each smart nodes performs a specific task in order to perform the global task.
They form a Network of smart objects: smart networks are the bricks of smart environments.
Smart environments are places where human activities are assisted and supported by ICT, like smart cities, smart buildings, smart lighting...

IETF Reference Architecture for IoT



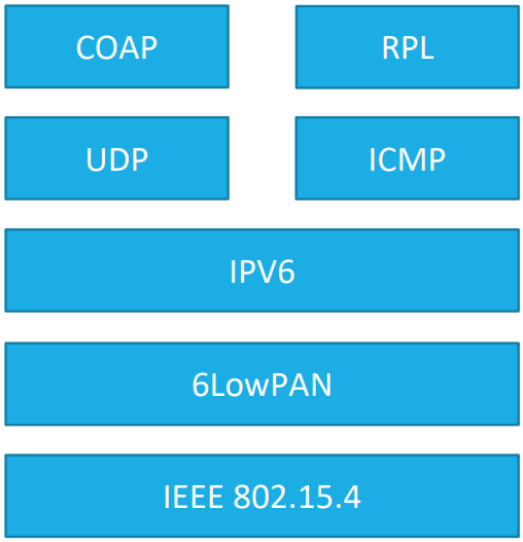
LLN: Low-power and Lossy Network

We have to remember that devices (black dots) have to save as much as possible energy (since the majority of them use battery).

The **backbone router** is the device able to communicate with both the Internet and the LLN.

Protocol Stack for IoT (IETF)

Now let's see which protocols have been chosen by the IETF (Internet Engineering Task Force) to enable communication between constrained devices in IoT systems.



- **Physical layer: IEE 802.15.4 PHY**
- **Data Link layer:** the IETF assumes the **IEE 802.15.4 MAC protocol**, which has been designed for low-power communications. (It was the best options, but there are better ones if we want some level of latency and reliability). Small frames, the max payload is 127 bytes.
- Here we have a new Adaptation Layer to allow the transmission of IPv6 datagram on a IEEE 802.15.4 frame. The **6LOWPAN** layer allow the transmission of IPv6 datagram on a IEEE 802.15.4 frame.
  6LowPAN defines the operations to be performed to transmit IPv6 packets in such networks: how compress/translate the header (since the max payload is 127 bytes, we want to compress to the max possible); how fragmentation can be performed (IPv6 standard payload is 65.535 byte).
- **Network layer**: **IPv6** in order to have a protocol that guarantees interoperability, scalability and QoS.
- **Transport layer: UDP** → TCP it requires a lot of resources (like memory and computational complexity)
- **Application layer**: The ***Constrained Application Protocol (CoAP)*** fulfill IoT devices needs. It's a simplified version of HTTP with specific features for the IoT (since they have lack of resources to use HTTP).
  RPL instead is the Routing Protocol for multi-hop communication. It collects information on the network topology, computes the multi-hop routes and populates the routing tables of each node.

Cyber-physical vs IoT

In ***Cyber-physical systems*** we have a Cyber component + physical component:
The cyber component receives data from the physical world, processes the received data and takes intelligent decisions that are communicated to actuators.
Smart object interact with the physical world: border between the cyber and physical world.

In ***IoT*** smart objects are connected to the Internet and communicate through IoT protocols.

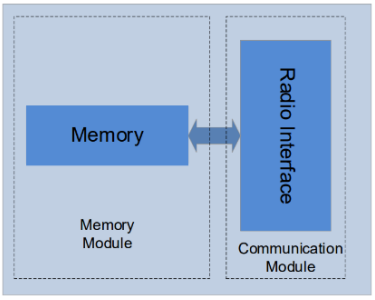
2- Smart objects

Real-world object + instrumenting device:
We need to ***empower*** with the following capabilities: ***actuating, connectivity, sensing and computing capabilities***.
Of course ***they need power source*** in order to feed electronic circuits.

Sensors

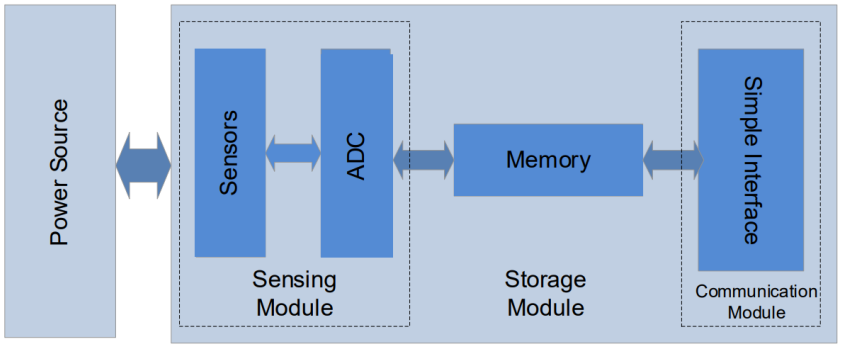
There are different type of sensors:

- **Passive sensors** are sensors that stores some information, reachable by some readers. They don't need any source power.
    - Architecture:



    - Application: Baggage/Animal tagging, Parcel tracking (postal services), Smart cities, Access control, Ticketing (public transport, parking, etc.)…

- **Semi-passive sensors**: also called data loggers, they need limited amount of energy and memory to measure and store some physical information, so they are powered by a battery.
    - Architecture:

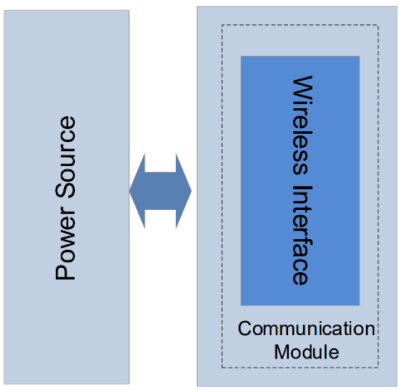


    - Application: Temperature monitoring of goods, bag identification, …

- **Active sensors**: sensors battery powered, with computational and communication capabilities. The active sensor communicates following the logic of the application even without something want to read it (like in passive sensors).
  The "Sensor nodes" and "Beacons" we will discuss below are examples of Active Sensors.

Beacons – Active Tags

iBeacon is a technology originally developed by Apple
iBeacon-compatible hardware transmitters - typically called Beacons - available also for non-Apple devices enables devices to perform actions when in close proximity to a Beacon: Beacons emit periodic signals and mobile devices understand their location based on the received signal and perform a specific action.



Beacon uses Bluetooth Low Energy (BLE) to transmit Advertisement (ADV) messages.

The ADV message is received by a smartphone and forwarded to a server (e.g., on the cloud). The server localizes the device (user) and sends appropriate information to the smartphone.

BLE Main goal it to provide considerably reduced power consumption and cost while maintaining a similar communication range. (available only for devices supporting Bluetooth 4.0 and above).

BLE vs Traditional Bluetooth

- BLE has lower power consumption than traditional Bluetooth.
- Lifetime up to 3 years with a single coin cell battery.
- BLE is 60-80% cheaper than traditional Bluetooth.
- BLE is ideal for simple applications requiring periodic transfers of small data, while traditional Bluetooth is better for more complex applications that requires consistent communication and higher throughput.

ADV messages:

Advertisement messages consist of *four* main *pieces* of information:

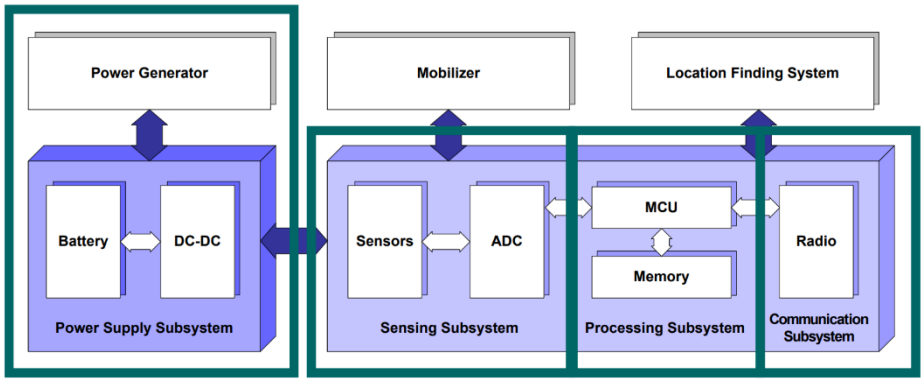
- ***UUID (Universal Unique IDentifier)***: *16 byte string used* to differentiate a large group of related Beacons. In its canonical form, a UUID is represented by 32 lowercase hexadecimal digits, displayed in five groups separated by hyphens, in the form *8-4-4-4-12 for a total of 36 characters*.
  E.g., ebefd083-70a2-47c8-9837-e7b5634df524
- ***Major***: *2-byte string distinguishing a subset of Beacons* within the larger group
- ***Minor***: *2-byte string used to identify individual Beacons*
- ***Tx Power***: Used to determine proximity (distance) from the beacon.
  TX power is defined as the strength of the signal at exactly 1 meter from the device.
  This has to be calibrated and hardcoded in advance.
  Devices can then use this as a baseline to give a rough distance estimate

Major and minor used if we have a hierarchy of beacons.

Application areas: Localization in Buildings, Tracking, Indoor Navigation…

Sensor Nodes (or Active Sensor)

Unlike "regular" sensors, which are devices that measure physical quantities in the external environment, sensor nodes are more complex devices. They not only measure these quantities, but also process them and are capable of transmitting the collected information.
In detail, sensor nodes are device that includes some sensors, but they are also equipped with micro-controllers to process data and wireless/wired interfaces for communication



In the class of sensor nodes are included also actuator nodes (the communication part remains the same, but the way they interact with the environment is different).

- First of all we need to store the information sensed (and parse to digital through the ADC converter), so a memory is present in processing subsystem: typically the memory is volatile and flash memory is present too (to store OS).
- Communication subsystem is present and it is typically the most power hungry component.

Examples: the smartphone is a very special sensor node since it has a lot of sensors, but it's easy to recharge so power management is not a big issue

Some challenges to be addressed:

- PCs, smartphones are driven by the user, instead *sensor nodes are driven by the external environment*.
- **Most important thing: limitation of resources.** The cost of sensor nodes must be very cheap, moreover the size must be very small.

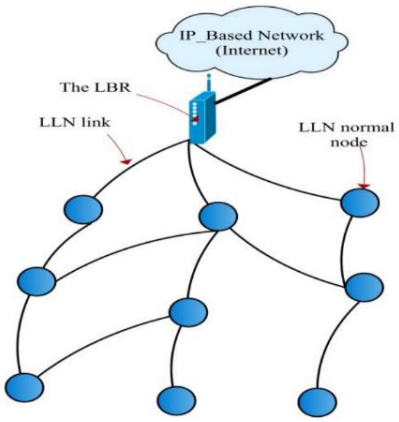
- **Reliability**: Although an individual node may fail, we require long-lasting applications. We do not have an on-field recovery mechanism, except for automatic rebooting.

3- Low power and Lossy Networks

A Low power and Lossy Network is a network composed by ***many smart object*** (embedded devices) with limited power, memory and processing resources, interconnected through wired/wireless link.
The communication protocol used usually are IEEE 802.15.4, BLE, Low-Power WiFi, etc.
Mobile communication is not suitable for Smart object due to the high energy consuming, but 5G, is moving in the direction of reduce the consume of energy.

LLNs often exhibit ***considerable loss*** (not received or received with errors), significant variability in the delivery rate, and ***some short-term unreliability***.
LLNs are typically characterized ***by limited and unpredictable bandwidth*** (order of hundreds Kb/s, but it's not a problem cause they don't generate a huge amount of traffic)

Low Power and Lossy Networks := constrained network of constrained nodes



For this reason we have many challenges to solve.

On the left we have an example of LLN: the final communication is to the Border Router (then or this is the final destination or data can be transmitted over traditional communication systems).
Directly communicating with the border router is not always an available option.
This because the distance are very large: we can't neither increase the power of the antenna to extend the communication range because we would have energy problems.

Since we cannot reduce the distance, so how can we transmit to it? By using multi-hop communication.

Multi-hop communication

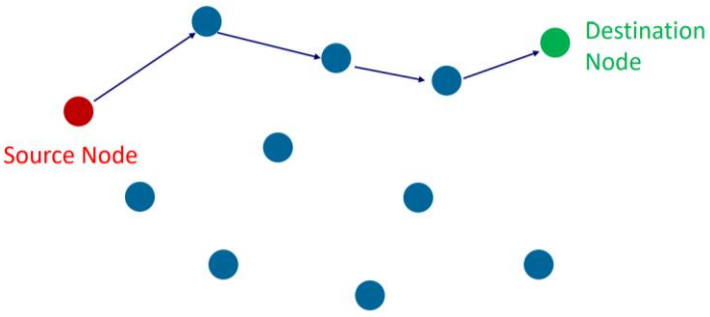
It is less energy-consuming to transmit the same message multiple times over a short range than to transmit the message once over a larger distance.
This is the concept of multi-hop communication: one node transmits the message to another node, which in turn transmits it to another node, and so on, until the message reaches its destination.
By doing this, we save energy for each single node and for the entire network.

But there are challenges to be addressed: unreliable links, shared communication medium (collisions), dynamic network topology (due to meteorological conditions, power management, obstacles) and of course energy-efficient communication.

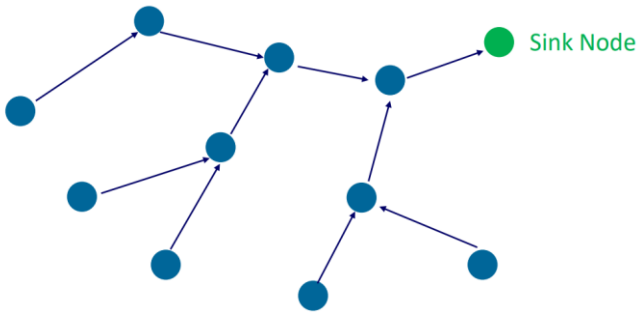
Communication patterns

The communication pattern depends on the application domain.

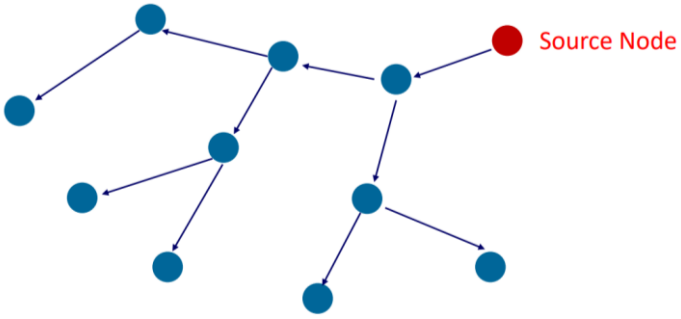
- **One-to-One Communication**
  Network very simple, one source node that send messages to the destination node.
  It's not very common



- **Many-to-One Communication**
  It arises if we had a sensor network. Nodes send information to a ***sink node***, which is the *common destination for all the node in the sensor network*.

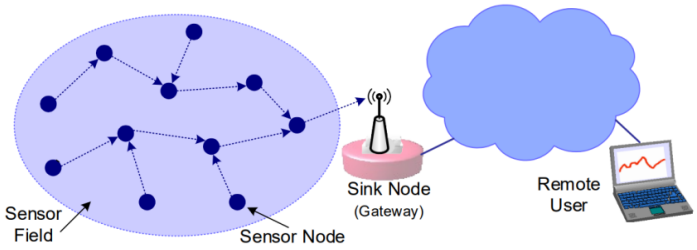


- **One-to-many communication**
  This is the case when we have to send a command to all the node (for example to exchange configuration)



Wireless Sensor/Actuator Networks

Wireless sensor networks are special case of LLNs. This type of networks consist of a large number of small sensor nodes deployed across a geographical area (for example a network of sensors in a city to monitoring the pollution). These sensor nodes possess three key capabilities: sensing, computing, and communication.



The network is built upon a distributed sensing infrastructure, where each sensor node is responsible for sensing various types of physical information such as temperature, pressure, vibrations, and pollution levels. The nodes can then process this data locally within their own computational capabilities, or send these collected data to one or more collection points. These collection points can take the form of a sink node, base station, gateway, or border router. These entities act as central hubs for receiving and aggregating the data from multiple sensor nodes.

In summary, Sensor Networks leverage the integration of sensing, computing, and communication capabilities of sensor nodes to enable distributed sensing of physical information, local data processing, and transmission to collection points for further analysis or storage.

Sensor and actuator nodes primarily communicate with each other through wireless links. However, in certain cases, wired communication is also utilized, particularly for emergency and critical applications.

WSN Classification

WSN can be classified based on:

Topology

- ***Static WSN***: all sensor nodes are stationary
- ***Quasi-static WSN***: (some) sensor nodes have limited mobility. (for example for strategic reasons or for communication quality issue)
- ***Mobile***: Some (or even all) sensor nodes are mobile: mobile relays/mobile sinks or mobile peers

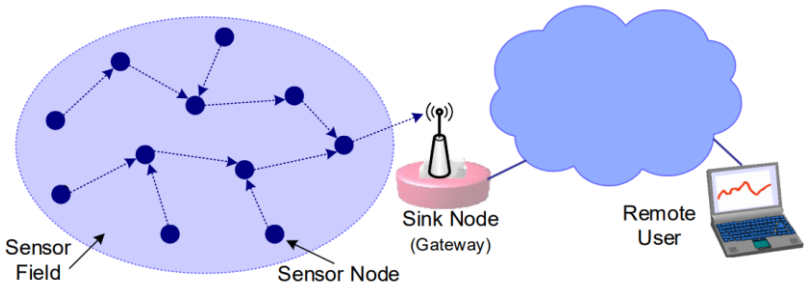
Density

- ***Dense***: the distance between sensor neighboring nodes is below the transmission range. So *multi-hop communication is possible*. High density, large number of sensor nodes.
- ***Sparse***: *the distance between sensor neighboring nodes is much larger than the transmission range*. For example if we have a network of sensors to monitor air quality, it doesn't have sense to install sensor every 100 m since the air quality is the same.
  The distance between neighbors is very large, so sensors are not able to communicate with each other. *So multi-hop communication are not possible*. So how can we communicate data after we collect them if the distance is very large? We can use direct communication, like 5G or other long range communication method, but this consume a lot of energy and it's possible only when the sensor nodes is connected to the electrical grid (or attach to some power source, like solar panel, etc.)
  The number of deployed sensor depends on the application needs.

Data collection paradigm:

- **Multi-hop communication**:
  - *Flat*: dense network with static topology. We need a lot of redundancy, with randomly deployment.
    Problems:
    - There are **long paths** from sensors to the sink: of course there is a *large sensor-to-sink delay.*
      In many application domain the delay is not a problem.
      (If there is no internal organization, the number of hops may be very large).
    - What about reliability? Assume that the probability to transmit without errors is very high like 0.9. But the reliability decreasing as the number of node increase. But in general the reliability is in the order of 0.5, 0.6 and so on. So only a small percentage of packets sent arrives correctly → **low reliability**.
      In principle we can retransmit, but retransmission consume energy → **high energy consumption**.
    - ***Funneling effect:*** almost all the data pass through to the same node, so it has more work than the other ones. So one node consume much more energy than other nodes. So if one node consume more energy than other, after a certain amount of time it's not able to work anymore. So the network is out of work. (They have the same quantity of energy available).
      Take the example where we have the one node nearest to the sink node.

    Application: military application, environnemental monitoring



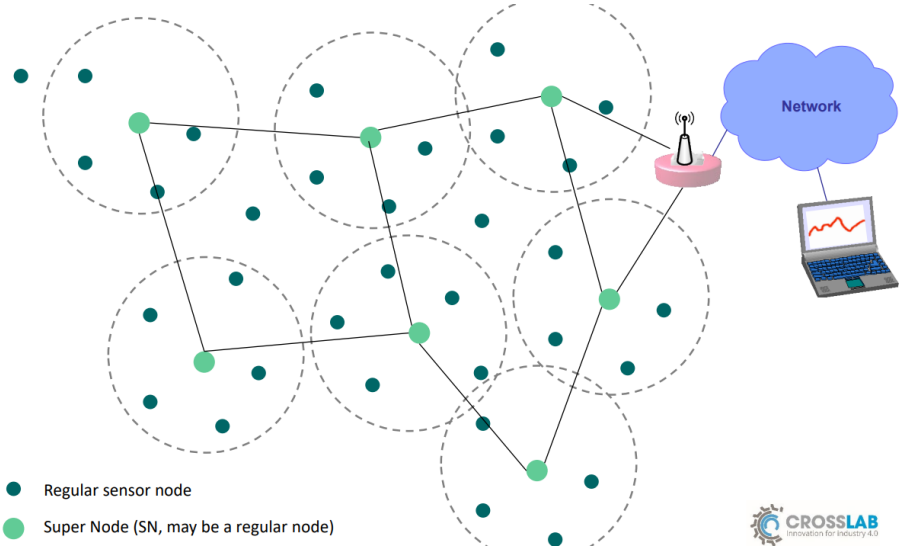
  - *Hierarchical*: dense network with static topology
    We have regular nodes and super nodes SN. Special node may be normal nodes that perform special tasks or special nodes with augmented capabilities.
    So we have clusters of nodes with a super node as cluster head. A communication network is then created between only the special nodes.
    Normal node transmit data to cluster head, and cluster head transmit them to the sink node through the SN network.
    Advantages:
    - Good approach if we can deploy nodes in strategical locations
    - We have short paths from sensors to sink, typically 1 hop from sensors to the sink
    - Limited sensor to sink delay
    - Increased reliability

- No funneling effect

Is there some cost to pay? **We have to introduce special nodes.**

Typically special nodes are more powerful nodes, more power communication capabilities, more energy consumed à more expensive.

Good when we have to reach some performance, for example in industrial applications, critical application and so on.

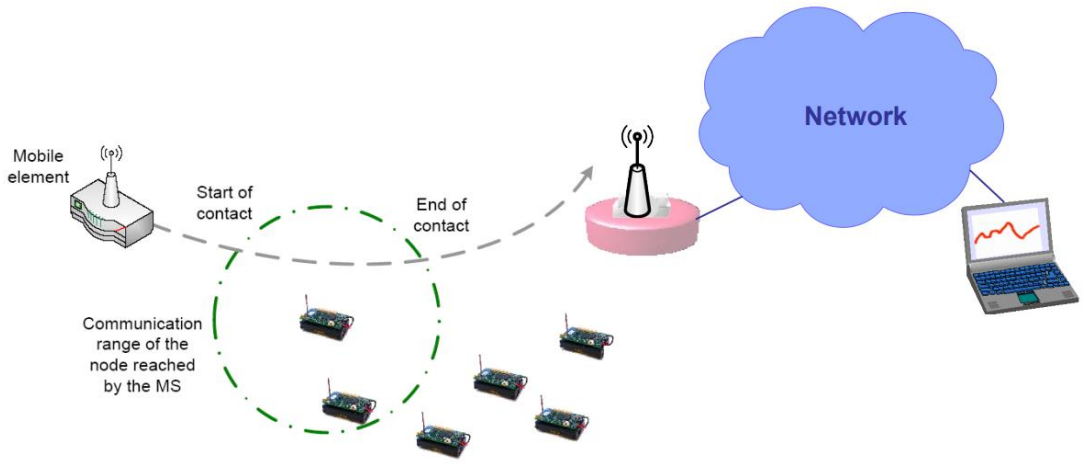


➢ **Mobile Data collectors**

We have deployed sensors in a sparse manner, along with a mobile element that is equipped with a device similar to a sensor node, allowing it to communicate with other sensor nodes. The mobile element moves along a path, and when it enters the communication range of a stationary node, communication becomes possible. The stationary node then transmits all the accumulated data since the previous visit by the mobile element. The mobile element carries the received data towards the sink node until it is close enough to transmit the data to the sink node.

So, it's the mobile device that enables the other nodes to share information with the sink node. It's important to keep in mind that the time for communication is limited.

Application: Military Applications, Environmental Monitoring, Precision Agriculture



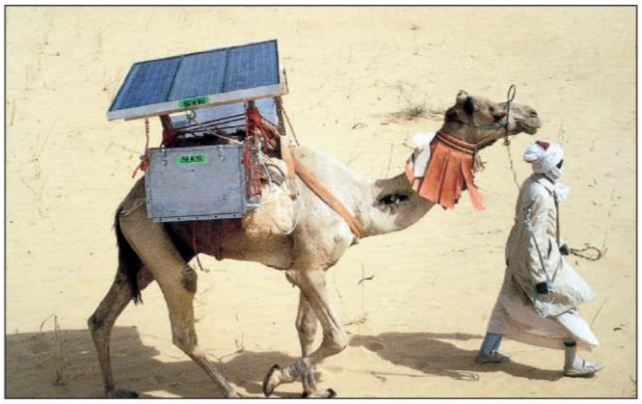
4- Energy Managment

Energy problem

Energy is the key issue in the LLN design: applications may require a network lifetime in the order of many months or, even, several years. If always active, nodes generally deplete their energy in less than a week.
Possible approaches to extend the lifetime of the devices are:

- *choose low-power devices*, if it's possible, depending on the domain application
- *energy harvesting*, for example device can harvest energy from the environment, with a solar panel for example and store it in battery (but even in that case battery capacity is limited) and external sources are not always available (like solar energy)
- *energy conservation*, it's always necessary, moreover in addition to energy harvesting.
- *Use energy efficient protocols/applications*.

Energy harvesting



The "Camel Fridge." Camels wearing solar-powered refrigeration units helped deliver vaccines to remote African villages in the 1980s. (photo courtesy Naps Systems)

How to scavenge energy?

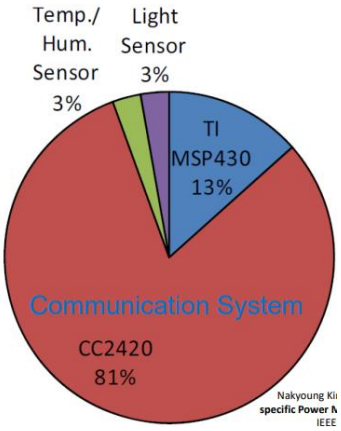
- *Thermal gradients*, it's based on the Carnot cycle. Of course when the gradient is low, the consumption of energy is low. It can be used for wearable sensor nodes
- *Thermoelectric generators*: they are able to convert heat into electrical power.
  Used for example for recovering of residual waste heat in industrial processes or for Spatial applications. But efficiency is limited to few percent (3-4 %), due to the high thermal conductivity of conventional materials
- *Radioactivity*: has been proposed as a source of energy for small devices. It's particularly suitable for devices operating with very limited power (i.e., tens of μW) for very long times
- *Radio Frequency (RF) signals*: RFID exploit this approach (very short distance required). The RFID tag receive energy form the reader, and it transmit the information. It can be used for feeding any kind of sensor nodes. For example it can be used to recharge cars in parking lot with this approach.
  For example we can use a mobile device to recharge nodes and collect data from them.
- *Vibration energy*, very suitable in environments where there are many vibrations, like bridges but also cars.
- *Solar cells,* current technology allows conversion efficiency just between 10% and 30%.

Energy harvesting is a very promising approach, but, currently the conversion process is not efficient enough. Can be used to power very simple devices or as a complementary power source, e.g. to replenish a battery in the background.

Energy conservation

The first thing to do is to determine the energy consumption for each task. By doing so, we can identify the areas where energy is primarily utilized and try to improve the energy efficiency of the most power-hungry system.

One important point to remember is that when we talk about power consumption during transmission and reception, we are referring to the energy used by the electronic components of the devices, not the antenna. The energy consumed by the antenna is very small and can be ignored for practical purposes.



Taking the example (but is a general approach), in order to conserve energy, our goal is to try to reduce as much as possible the radio activity, possibly performing local computations.

We have different approaches:

- ***Data driven approach:*** reduce as much as possible the amount of data to transmit to the final destination (so you need to do less transmission and reception and consequently the energy consumption).
- ***Mobility-based approach*** (will consider it later)
- ***Duty-cycling approach*** when the device is inactive you put it in sleep mode! Percentage of activity respect to the whole cycle. So the **duty cycle** is the amount of time the device is active respect to the total time, it should be smaller possible, close to zero.
  (idea: alternate activity and sleep mode, in order to minimize the duty cycle, i.e. the percentage of time in which the sensor is active)

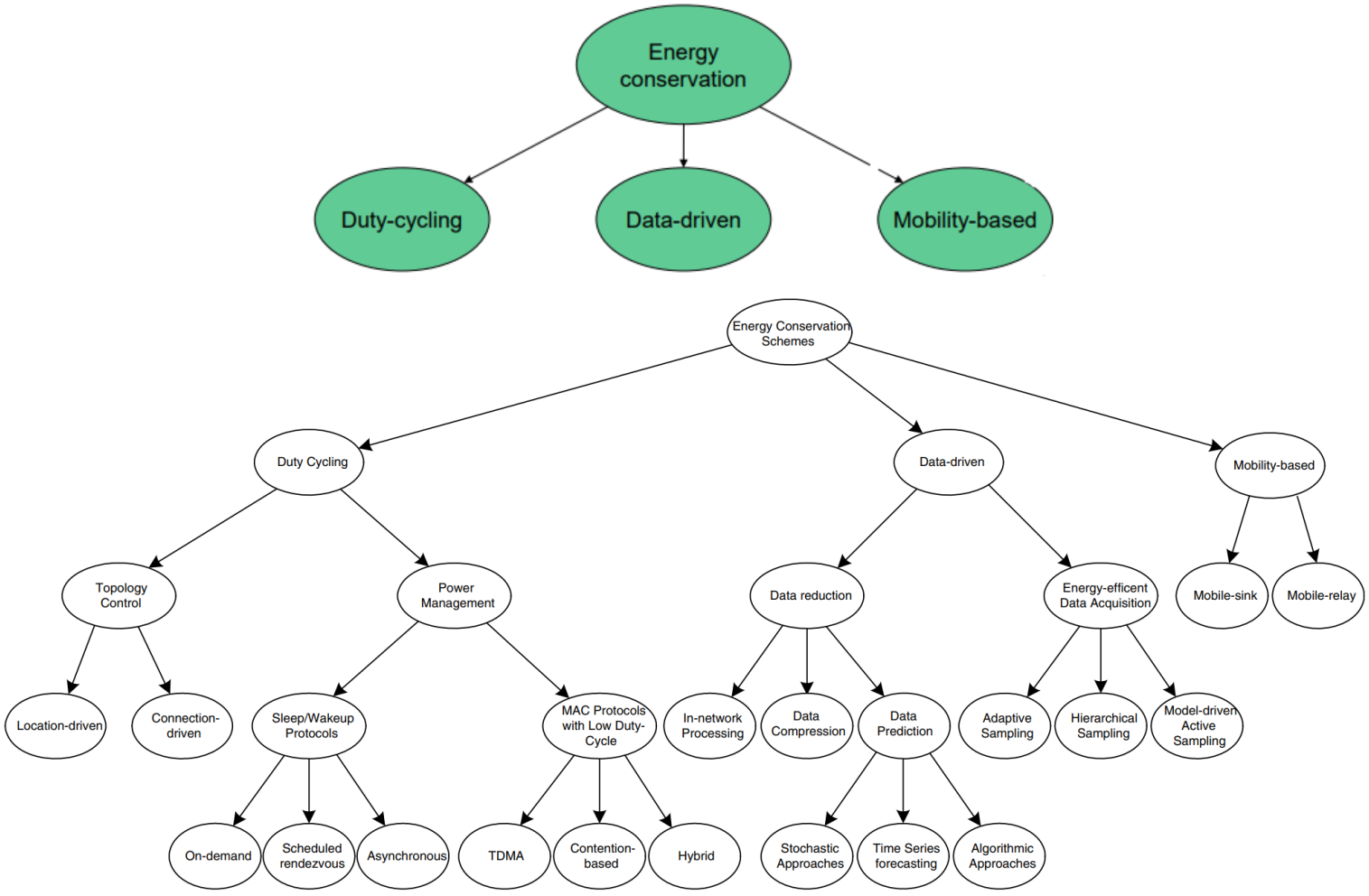


Fig. 3. Taxonomy of approaches to energy saving in sensor networks.

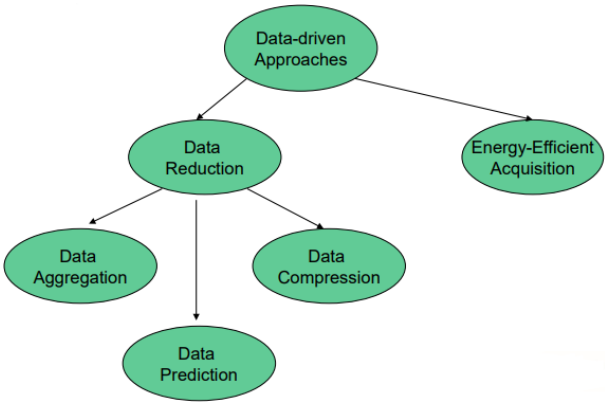
Data driven approach

Data sensing impacts on sensor nodes' energy consumption in two ways:

- ***Unneeded samples:*** Data collected at regular intervals often exhibits spatial or temporal patterns or similarities. Therefore, it becomes unnecessary to transmit this repetitive information to the central data collection point. Even though the cost of collecting the data is insignificant, it would be wasteful to consume extra energy by sending redundant information.
- ***Power consumption of the sensing subsystem***. Reducing communication is not enough when the sensor itself is power hungry.

Data driven techniques presented in the following are designed to **reduce the amount of sampled data** by **keeping the sensing accuracy within an acceptable level** for the application**.**

How can we do it? Using Data reduction (Data aggregation, Data compression or data prediction) and Energy-Efficient Acquisition.



Data driven approach – Data Reduction

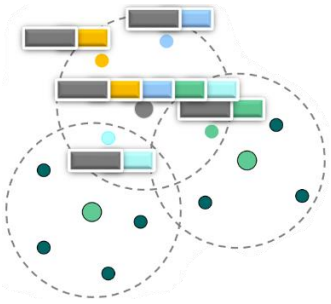
Data compression

Data compression aims to reduce the amount of information sent by source nodes. This scheme involves encoding information at nodes that generate data and decoding it at the sink. There are different methods to compress data.

Data aggregation

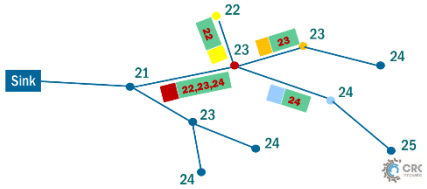
It consist into combine multiple network packets into a single network packet frame.
It can be performed in different ways:

- **MAC-Layer Data Aggregation:** it combines *multiple network packets into a single MAC frame* (so it reduced overhead since we transmit one single header, trailer, RTS, CTS).
  This solution is independent from other layers, we have no knowledge about network topology and it is application independent.



- **Cluster-based Data Aggregation:** similar approach can be used when the network is organized into clusters. Each node transmit its data to the cluster head, and then the cluster head aggregate all the packet received in a single packet.

- **Tree-based Data Aggregation:** each node send data to its parent and it accumulate information from all the children in order to send one unique message to its parent, until the root (which is generally the sink)



Aggregation factor: is the ratio between the number of bits to be transmitted with aggregation and the number of bits to be transmitted without aggregation.
Gives us a metric for how much we are actually reducing the load.

$$\alpha = \frac{D_{agg}}{D_{no-agg}} = \frac{H + N \cdot P}{N(H+P)} = 1 - \frac{H}{H+P} \cdot \frac{N-1}{N}$$

$$h = \frac{H}{H+P}$$

$$\alpha = 1 - h \cdot \frac{N-1}{N}$$

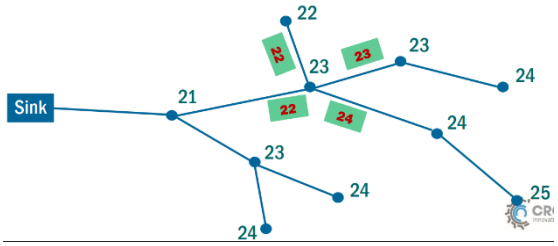
- Assuming H = 9 bytes (IEEE 802.15.4 MAC Frame)

Aggregated packets	Payload Size (bytes)							
	1	2	4	8	16	32	64	118
1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
2	0.55	0.59	0.65	0.74	0.82	0.89		
3	0.40	0.45	0.54	0.65	0.76	0.85		
4	0.33	0.39	0.48	0.60	0.73			
5	0.28	0.35	0.45	0.58	0.71			
6	0.25	0.32	0.42	0.56	0.70			
7	0.23	0.30	0.41	0.55				
8	0.21	0.28	0.39	0.54				
9	0.20	0.27	0.38	0.53				
10	0.19	0.26	0.38	0.52				

As we can see from the example above, more packets we aggregate, less bits we have to transmit.

Application-aware Data aggregation

There's also another technique to aggregate data: assuming that the sink is interesting in the minimum temperature, data reduction can be obtained in this way: assume that a node receive one or more data, a node can compare the values he received with the one generated by itself and transmit only the minimum.



In this way we reduce the data to be transmitted. This technique can be exploit both in tree-based and cluster-based. Of course is application domain based.

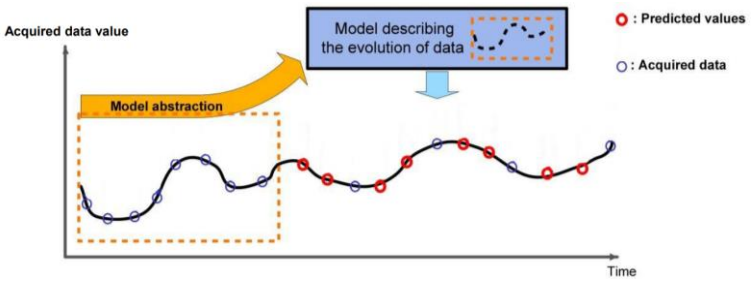
Data prediction

Instead of reporting all data to sink, only sends the trend. It is transmitted ***only if and when it changes.***

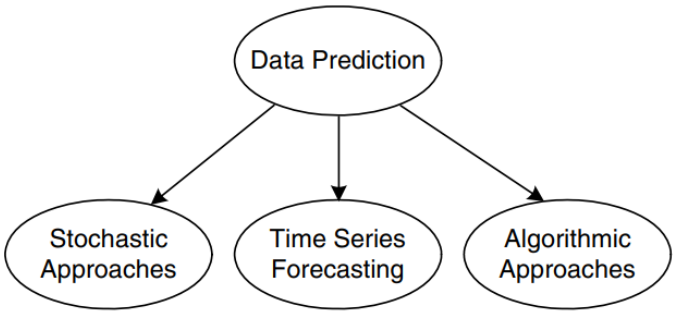
How it can be done? There are ***two instances of a model in the network***, **one** residing **at the sink** and the **other at source nodes** (there will be as many pairs of models as there are sources).

The model at the sink can be used to answer queries without requiring any communication, thus reducing the energy consumption. Clearly, this operation can be performed only if the model is a valid representation of the phenomenon at a given instant.

Here comes into play the model residing at source nodes, which is used to ensure the model effectiveness. ***To this end, sensor nodes just sample data as usual and compare the actual data against the prediction***. If the sensed value falls within an application-dependent tolerance, then the model is considered valid. Otherwise, the source node may transmit the sampled data and/or start a model update procedure involving the sink as well.



The destination can derive values using its own model, which will be updated by the sensor if it detects a value that deviates from what the model expects.



The model can be constructed in three different ways:

- ***Stochastic approaches,*** *map data into a random process described in terms of a probability density function (pdf).* Data prediction is then obtained by combining the computed pdfs with the observed samples.

This technique provides means to perform high-level operations such as aggregation.
The main drawback of this class of techniques is their rather *high computational cost*, which may be too heavy for current off-the-shelf sensor devices.
Stochastic approach seems to be convenient when a number of powerful sensors are available.

- ***Time series forecasting,*** where a set of *historical values* (the time series) obtained by periodical samplings are used to predict a future value in the same series. (e.g. Moving Average (MA), Auto-Regressive (AR), …)
  It can provide satisfactory accuracy even when simple models are used.
- ***Algorithmic approaches***, it relies on a heuristic or a state-transition model describing the sensed phenomenon.

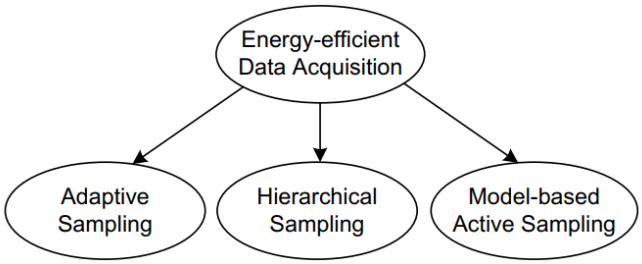
Data Driven approach - Energy-efficient data acquisition (Sensor energy management)

The idea is to reduce the number of data collected by sensors while still maintaining coherence with the measured quantity. By doing so, energy is conserved due to fewer samples being taken, resulting in a reduced amount of data that needs to be transmitted (and energy consumed).

We started making the assumption that the communication system is the most energy consumer.
Since this condition could not be true, we have to consider other solution to reduce the energy consume by the sensor.
In fact, the *energy consumption of the sensing subsystem not only may be relevant, but it can also be greater than the energy consumption of the radio* or even greater than the energy consumption of the rest of the sensor node.
How can the sensing part have an energy consumption not negligible? It could be possible due to *power-hungry transducers, power-hungry A/D converters, active sensors and long acquisition time*.

In this case reducing communications may be not enough, but energy conservation schemes have to actually reduce the number of acquisitions (i.e. data samples). It should also be pointed out that *energy-efficient data acquisition techniques decrease also the number of communications as well (so the energy consumption of the communication system)*.

Note: energy consumption = power consumption * time during that power is consuming



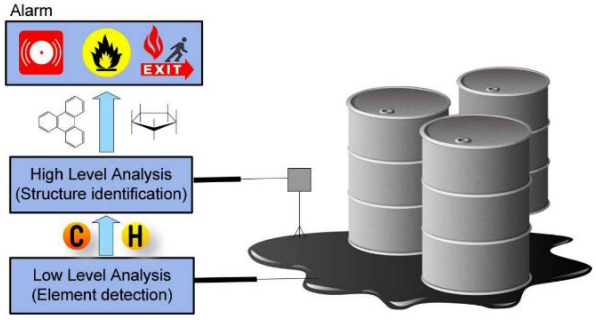
Hierarchical sensing

The hierarchical sampling approach consists in using sensor nodes equipped with *different types of sensors*, with different sensor accuracy and different power consumption for example. We can use different sensors to sense different phenomena if they are correlate.

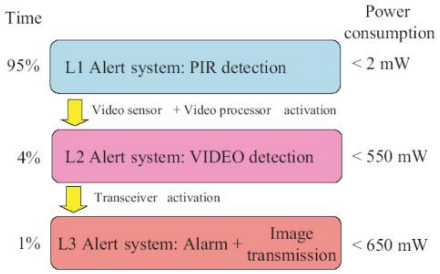
Accuracy can be traded off for energy efficiency by using the low-power sensors to get a coarse-grained information about the sensing field. Then, when an event is detected or a region has to be observed with greater detail, the accurate power hungry sensors can be activated.
For example we can use low accuracy sensors always active and if something is detected then the high accuracy sensor is activate to monitor the situation (if low accuracy sensor give us a false positive, we active the high accuracy sensor only for a short time)

Different ways to do this:

- ***Triggered sensing:*** Low-power low resolution sensors trigger high-power and high-accuracy sensors.



- ***Multi-scale sensing:*** Low-resolution wide area sensors are used to identify areas of interests, High resolution sensors are, then, switched on for more accurate measurements.
  Example on the right:



Adaptive sampling

Adaptive sampling can ***reduce the number of samples by exploiting spatio-temporal correlations*** between data. The idea is to changing the sampling rate depending on the dynamic of the phenomenon that is observed.

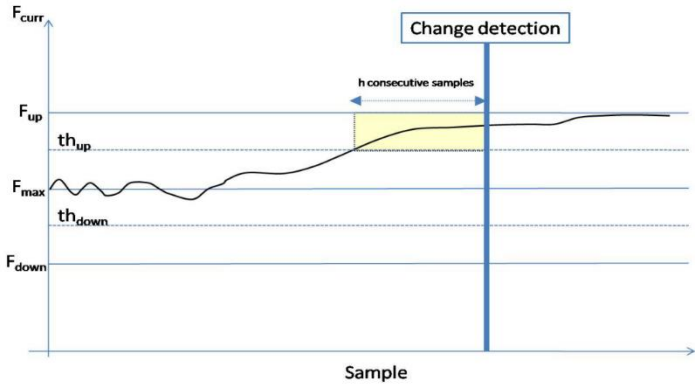
From the *Nyquist theorem it is known that the sampling frequency needed for the correct reconstruction of the original signal should be* $F_s \geq 2\ F_{max}$ where F_{max} is the maximum frequency in the power spectrum of the considered signal.
Unfortunately, choosing Fmax is not trivial because (i) it cannot be known a priori and (ii) it may vary over time

This led us to ask ourself: When to change the sampling rate? How to change it?

To overcome these problems the authors propose an adaptive algorithm that dynamically *estimates the current maximum frequency* F_{max}.
The estimated sampling rates obtained by the sink are then notified to sensor nodes.

Algorithm idea:

1. *Estimate the maximum frequency of the signal* by using a training sequence (W samples)
2. *Define* two alternative hypothesis F_{up}, F_{down} for the maximum frequency of the signal during the operational life
3. ***If the current maximum frequency*** F_{curr} ***of the signal*** (W samples) ***during the operational life is closer to*** F_{up} ***or*** F_{down} ***than for h consecutive samples, a change is detected in the maximum frequency of the signal***
4. *A new sampling frequency is defined*

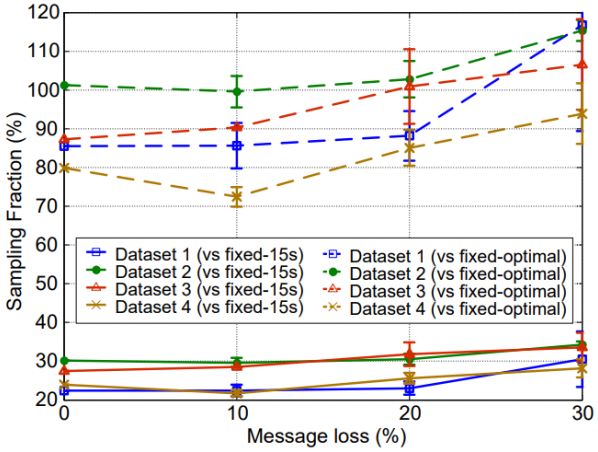


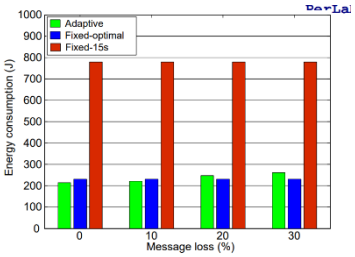
There are some parameters to be set:

- C: ***confidence parameter*** for the *maximum frequency detection* (c > 2, Nyquist)
- H: critical to the robustness of the algorithm
  - *low values (e.g., 1 or 2): quick detection but possible false positives*
  - *high values (e.g., 1000): few false positives but less promptness in detecting the changes*
- W: critical to the accuracy of the algorithm
  - *low values: not accurate estimation but low energy consumption*
  - *high values: accurate estimation of Fc but energy consumption*

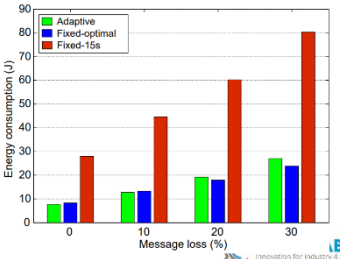
To evaluate the performance of we use the sampling fraction: *number of samples with ASA / number of samples with FS (oversampling)*

$$\text{Sampling Fraction} = \frac{\text{Number of Samples with } \textit{Adaptive} \text{ Sampling}}{\text{Number of Samples with } \textit{Fixed} \text{ Sampling}}$$





Sensor Energy Consumption



Radio Energy Consumption

With ASA, as we can see, the number of sample is very low and is very reduced respect of the fixed sampling.

About the energy consumption instead, as we can see in the figures, the energy consumption of ASA is very near to the optimal, for both radio and sensors. Instead with oversampling, we sample data every x-seconds, and as we can see, the radio/sensor energy consumption is very high.

So instead of ASA, could we use Fixed-sampling with optimal value?

No because we are not able to say in advance the optimal sampling rate: since we know the dataset, we can exploit it to calculate the optimal sampling rate. But this after the simulation experiments with ASA. Of course is not feasible in reality

Model-based active sensing

Sensor samples some values to derive the model of the phenomenal, then ***learn the spatio-temporal relationship among measurements*** and use this knowledge to make the sensing process energy efficient.

A model of the phenomenon to be monitored is built and then updated dynamically, based on measurements from sensor nodes.

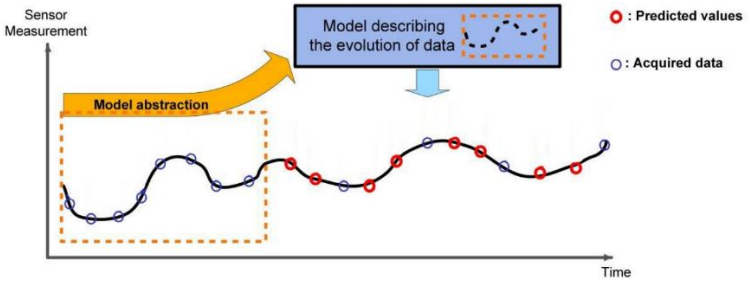
The sensor node decides whether:

- ***To acquire a new sample through a measurement***
- ***To estimate this new sample, with the desired accuracy, through the model***

The choice depends on the trade-off between acquiring a new sample, which requires energy and resources, and using the model, which may be more energy efficient.

So the model is used by the source node. **THE MODEL IS A DIGITAL TWIN OF THE SENSOR.**

After some times the model is no more valid, so the parameters must be changed and the model re-evaluated.



Limitations and other consideration on data-driven approaches

To increase the amount of energy saved, we can combine the different approaches seen so far.

But anyway, just reducing the amount of data does not necessarily implicates energy consumption reduction:

- Transmitting a message requires approximately the same energy, irrespective of the message size.
- Energy costs for maintaining the sensor network cannot be avoided
- Since data reductions eliminates data redundancy ***100% communication reliability is required*** (and it's not a valid assumption, since they are called low power and LOSSY networks).

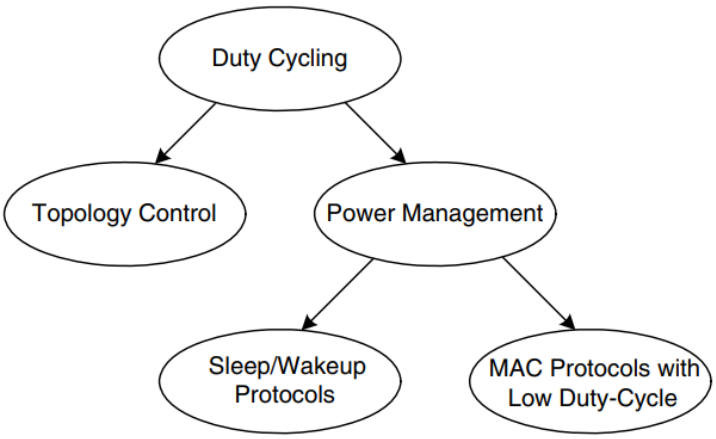
For example, techniques like idle listening for CMCA in Wi-Fi communication consume resources, moreover, if the packet is lost (due to a collision for example), it has to be retransmitted. Then also other packet are sent (routing tree management).
How much energy-consumption reduction in practice?

Duty-cycling

So duty cycling refers it's a technique to conserve power by periodically turning on and off the operation of devices or components. It involves cycling between active and sleep states, where the device is active for a certain duration and then goes into a low-power sleep mode for the remaining time.
A d**uty cycle** is the fraction of time an entity (in this case radio) spends in an active state as a proportion of the total time considered.

We have two ways of duty cycling (in both case, the radio is switching off):

- **Topology control:** ***network redundancy is exploit*** *(manly used when placement of sensor is random)*. Selects the minimum set of nodes that guarantees connectivity.
  All the other nodes are kept in sleep mode to save energy.
  How this is possible? We have redundant node, since we can't decide the strategic placement of the nodes (but we can redundancy for other reason too, for example because the network is unreliable) *so we can choose to not activate all of them and still have connectivity*: it increases the network lifetime by a factor depending on the degree of redundancy (typically in the order of 2-3).
- **Power management:** ***Exploits idle periods in the communication subsystem***: switches off the radio during inactive periods. For example, while monitoring temperature, we can switch-off radio for several minute, since the temperature doesn't change in seconds (or few minutes).
  Extends the network lifetime significantly: duty cycles of some percents are quite common in WSNs.



Duty-cycling - Topology Control

Goal: Find out the minimum subset of nodes to activate in order to ensure network connectivity. If we choose:

- ***Few nodes***, distance very large → high packet loss
  You can try to avoid packet loss by increasing the power of the antenna → more energy consumed
- ***Too many***:
  - At best, unnecessary energy consumption.
  - At worst, active nodes can interfere with each other → congestion on the channel → collision occurs → packet has to be retransmit

  "if you have more node than necessary, you are spending more energy than necessary!"

We have different "topology control protocols" approaches:

- ***Location driven***
  - needs to know the exact location of nodes : Geographic Adaptive Fidelity (GAF)
- ***Connectivity driven***
  - more flexibility : ASCENT, SPAN

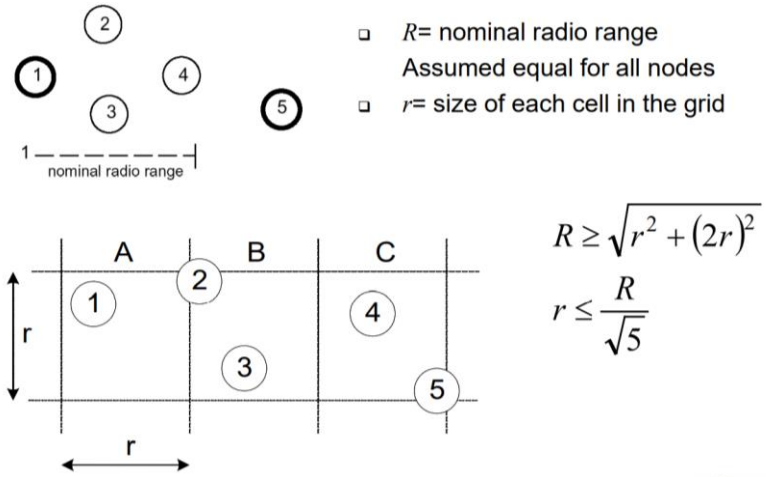
Geographic Adaptive Fidelity (GAF)

Each node knows its location (GPS): nodes are arranged in a grid with square-shaped cells with a side length of ***r***.
It's a cluster approach so we need to ***keep 1 node awake in each grid at each time (grid leader)***
Nodes alternate between:

- ***Active state*** *(grid leader)*
- ***Sleeping*** *(non grid leader)*
- ***Discovery*** *(leader election)*

After sometimes, ***energy of cluster-head exhausts: in order to avoid it, the role of cluster head is performed by each node following a rotation.***

Each cluster-head must be able to communicate with other ones placed into neighbor cell. Since R is the nominal radio range, we have to choose r such that the nominal radio range is greater the maximum communication range between cells (we have to remember that communication ranges has not a perfect circle shape).



The image is misleading: the formulas shown calculate r max or R max in order to enable communication between two adjacent cells (e.g. between node 2 and node 5).

All nodes inside a cluster, except the cluster-head, are sleeping: it's a sort of Hierarchical Routing.
As soon as the cluster-head detects an event, it wakes up all the other nodes in the cluster.
The cluster-head receives packets from cluster nodes, and forwards them to the sink node (no data aggregation).

Limitation

- We assumed that the nominal radio range is known, so the max distance at which is possible to transmit and receive. But in reality, it changes over time and sometimes is even different in different direction.
- Also we assumed that we know the exact position of each node in the sensing area. (GPS consumes energy)
- Once you have defines the cell size you can't change it!

Adaptive Self-Configuring sEnsor Networks Topologies (ASCENT)

It does not depend on the routing protocol and the decision about joining the network are based on **local measurements** (in an autonomous way decides if turn on a node or not).

- Each node measures the number of neighbors and packet loss locally.
- Each node then makes an informed decision to join the network topology or to sleep by turning its radio off.

Nodes can be in active or passive state:

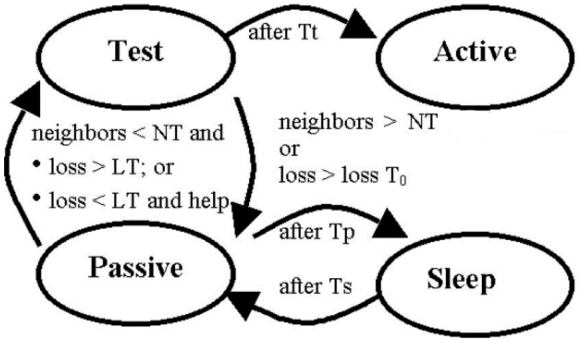
- Active nodes are part of the topology (or stay awake) and forward data packets
- Nodes in passive state can be sleeping or collecting network measurements.
  THEY DO NOT FORWARD ANY PACKETS

An active node may send **help messages** to solicit passive neighbors to become active if it is experiencing message loss.
A node that joins the network (test state) sends an **announcement message**.

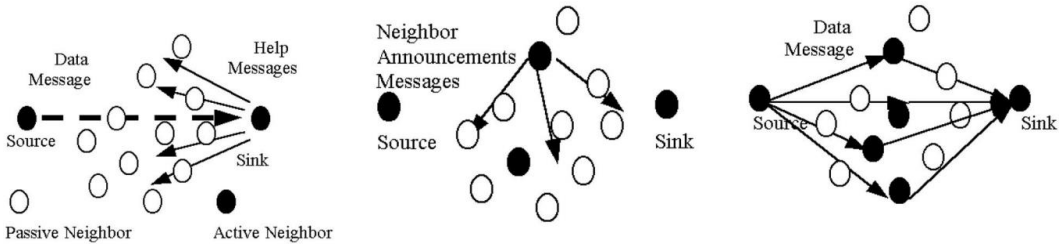
This process continues until the number of active nodes is such that the experienced message loss is below a pre-defined application-dependent threshold. The process will re-start when some future network event (e.g. a node failure) or a change in the environmental conditions causes an increase in the message loss.

There are some threshold domain dependent:

- **LT**: loss threshold
- **NT**: neighbors threshold
- **Tp** : if the node is in the passive node, after Tp time if nothing is sensed, the node return back to sleep node.
- **Ts**: period after the node goes from sleep to passive node.



This diagram represents the states that an intermediate node (neither the source node nor the sink node) goes through. Initially only the source node and the sink are active (the sink is always active), all the intermediate nodes are in the passive state.

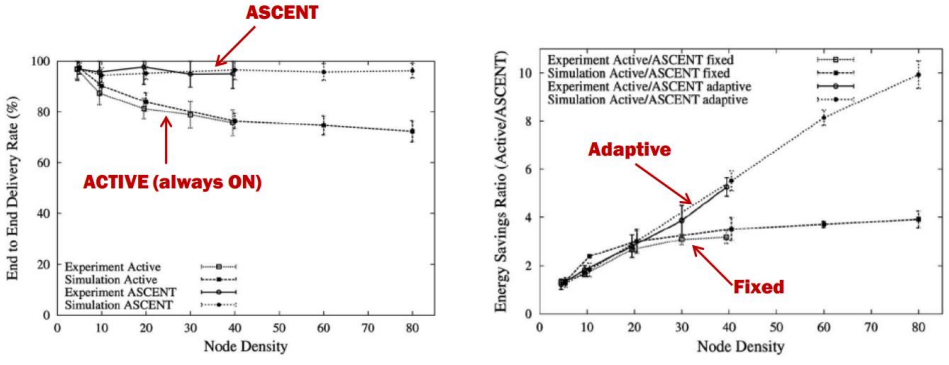


- After some time: the quality of communication between source and sink degrades, the sink nodes start to not receive some messages, or receive them out of order: ***a certain level of packet loss is reached,*** usually a small amount is tolerated.
  So the *sink start to transmit **HELP MESSAGES** to the intermediate nodes*, some of them will be able to receive packets (they are only listening not forwarding).
- Once they hear this message they will go in the ***test state*** (sending an announcement message)***.***
  - If the neighbors are too many or the quality is lower than before, the node come back to the passive state.
  - If not, the node go in the active state which is a ***final state***.

In this diagram, there is no way to change state once we have reached the active state. However, this is not the only possible solution. Other solutions may involve changing state if certain external conditions change.

Why do we check the number of nodes? Because if there are too many intermediate nodes which are active may face congestion towards the sink nodes (collision), since a lot of nodes will try to simultaneously transmit to the sink: from one side having a lot of intermediate nodes is good, on the other side you still have to control such number.

End-2-end Delivery Ratio **Energy Savings**

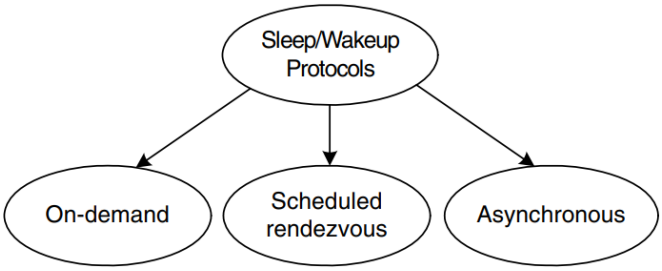


The second result is in line with our expectations, while the first result is completely unexpected. In terms of delivery ratio, when using topology control, which involves reducing the number of active nodes, can effectively mitigate collisions and improve the likelihood of successful packet delivery.

Duty cycling - Power Management

In general means that the radio (or a component or the whole sensor) alternate between sleep and active mode.
It can be implemented in 2 different ways:

- **General sleep/wakeup schemes** (higher layer in a form of protocol that alternates these 2 states)
  When should a node wake up for communicating with its neighbors?
  - When another node wants to communicate with it ***(on demand)***
  - At the same time as its neighbors ***(scheduled rendez-vous)***
  - Whenever it wants ***(asynchronous)***
- **Low duty-cycle MAC protocols** (implementation at MAC layer combined with MAC protocol)



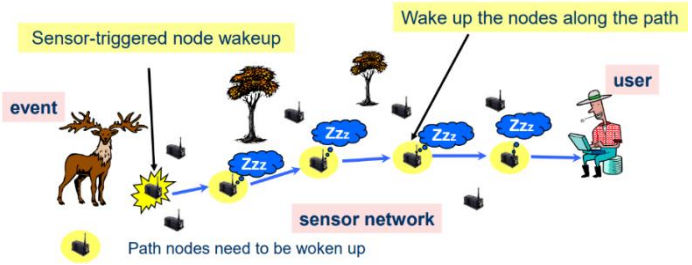
Sleep/Wakeup Protocols

sleep/wakeup schemes: ON DEMAND

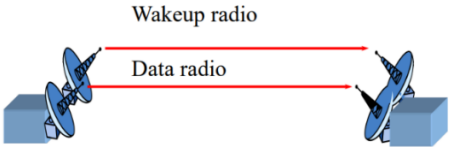
The sources detects an event, then we will send a wake up message to all the hop until we deliver it to the final destination.

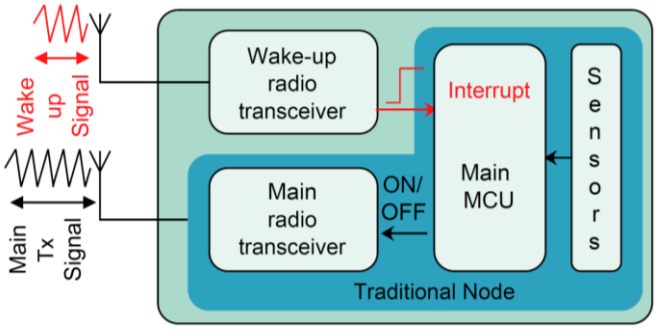
To implement this schema two different radios are needed:

- Data Radio (data transmissions)
- Wake-up Radio



We could also use one single radio with two different frequencies for data and wakeups.
If we use two different radios, the wake-up radio needs to be very energy-efficient, which results in a smaller communication range.





For the **Wake-Up Radio** (**WuR**) to operate effectively as part of the larger system in a multi-user environment, it should consider the following design points:

- ***Power consumption***. The most important feature of the WuR is its low power consumption in active mode. In fact, as its use requires the addition of new hardware on top of the main node, *the device itself must consume no more than tens of micro-watts*. Specifically WuR's active power should be below that of the main radio's sleep power to provide a positive balance between power saved and used. This is the main specification driving WuR design.

- ***Time to wake-up.*** The node attached to the WuR *must wake-up with minimum latency upon reception of WuS to avoid latency* incurred from multi-hops toward the sink and to increase the overall responsiveness of a purely asynchronous network. A range of protocols and applications can benefit from WuR based systems provided that the latency is low. For example, applications in health-care have strict latency requirements and cannot support

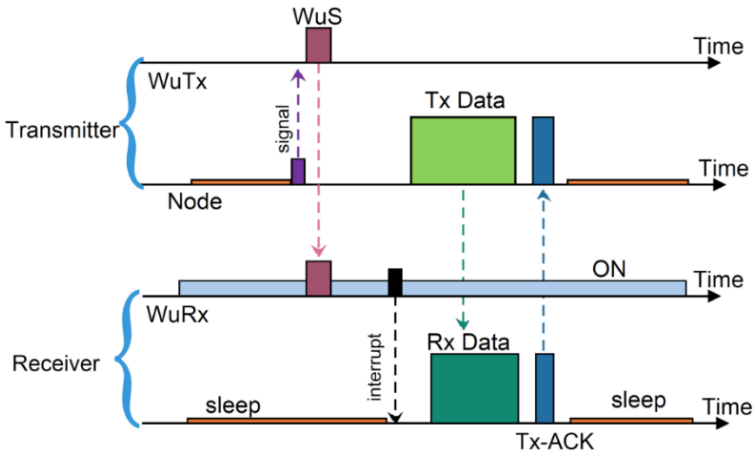
introducing long delays due to the wake up procedure.

- ***False wake-ups and interference***. If all nodes in a sensor network rely on the same wake-up strategy, when the WuTx tries to wake-up a node, it will trigger all the nodes in the neighborhood causing significant energy waste. This causes unnecessary activation of many nodes that should be avoided.
  There are two possible sources of false wake-ups:
  1. *nodes waking up when* **receiving a WuS intended for another node**.
     To ***avoid*** it: the WuR can employ a node ***addressing and decoding capability to trigger only the intended node***. *However it introduces complexity (check and interrupt) and often consumption at the WuRx*
  2. **interference from nearby devices operating at the same frequency**. How to avoid it:
     - Due to the low power budget, *only basic modulation techniques* can be used requiring a simple receiver structure. (Some interferences could be not detected)
     - *A preamble can be used to differentiate noise from a valid WuS*, thus avoiding false wake-ups. (Transmission of the preamble has a cost)
     - In addition, the WuS must not be missed by the targeted node, as retransmissions are costly in terms of power consumption and latency. To ensure this, a feedback loop such as WuS acknowledgment (WuS-ACK) can be employed by the WuRxs indicating the successful reception of the WuS.

- ***Sensitivity and range.*** In WuR design, receiver sensitivity is an important parameter as it provides the lowest power level at which the receiver can detect a WuS. Higher sensitivity requires more power hungry electronics at the receiver side. In contrast, low sensitivity for the same communication range will require high radiated power at the transmitter side. *Because of this, sensitivity requirements often leads to over-design to ensure reliable communication in adverse conditions.*
  The wake-up range that can be achieved with most current WuR designs is typically around 30m a value that can be improved by using techniques such as antenna diversity and directional antennas

- ***Data rate.*** A higher data rate can be seen as a way to improve energy efficiency and to achieve faster wake-up. While a high data rate reduces wake-up latency, a longer bit duration increases the communication range and the reliability of the WuS.
  A high data rate is not strictly required by the WuR, especially if it is only used as a triggering device as only a few bytes of data are required.

- ***Cost and size.*** To integrate the WuR into existing sensor nodes, it should be cost effective.

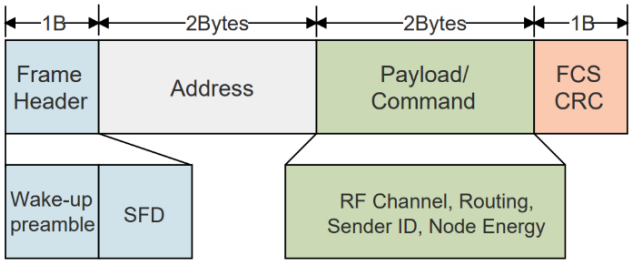


WuR	wake-up radio, the secondary low-power module
WuRx	wake-up receiver
WuTx	wake-up transmitter
WuS	wake up signal, the message sent by the WuTx

When a wakeup message comes, the wake-up radio sends an interrupt to the MCU. It's important that the wake up radio sends the interrupt with a minimum latency, in order to have synchronized receiver and transmitter. ***We also need node addressing in the payload of the wake up message, otherwise all the nodes in the sorrounding could be awaked erronously***.
In order to save energy and manage the energy consumption of the WuR, **Wireless Energy Harvesting techique** can be used.

Of course, even the WuTx and the WuS assume a very important role:

The **WuS** is a *6 byte* message:



- ***1 byte for Frame Header.*** The frame header consists of the *wake-up preamble* and *start frame delimiter (SFD)*, a **standard byte pattern agreed between the transmitter and the receiver.**
- ***2 byte for Address.*** It contains the node ID. It's length is variable depending on the WuRx capabilities. While a long address code is more robust against false wake-ups, it requires a long transmit time, hence more power is consumed. (Generally, using the address field, energy consumption is required to send and decode also the address, but it avoid fake wake-up).
  We could also use broadcast, it reduce the latency since it has to note decode the Wake up packet and triggered directly the main radio. However this choice can be costly.
- ***2 byte for payload/command***
- ***1 byte for error detection (CRC)***

Medium access control: we can characterized them in different ways:

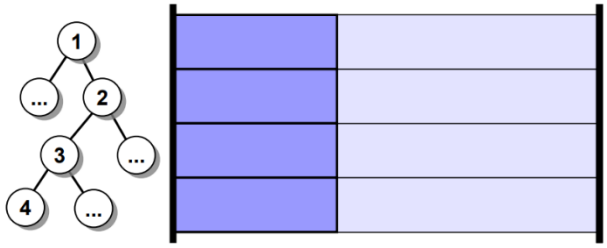
- **Communication initiator**
  - ***Transmitter***. In a Transmitter-initiated protocol, the node that has data to send initiates communication. It first sends a wake-up signal, whose receipt triggers the receiver to wake up its main transceiver. Data is exchanged using the main transceivers followed by Tx-ACK if transmission was successful. The nodes then go back into sleep mode.
  - ***Receiver.*** In Receiver-initiated systems, in this case, the receiver, often the sink, announcing its readiness to receive data. After this announcement, it switches to receive mode and monitors the wireless channel to receive any incoming packets. If we assume the WuRx on the sender side is always active and listening, when it receives the signal it activates its main transceiver to send the data packet. The session ends when the transmit acknowledgment (Tx-ACK) signal arrives at the sender from the destination node, after correctly receiving the data packet. All the nodes then go back to sleep mode. *This communication modality is most effective when transmissions are infrequent, and collisions at the receiver are unlikely.*
  - ***Initiator: Bi-directional***
- **Hardware:** *are communication nodes equally equiped?*
  - *Asymetric*
  - *Symetric*
- **Power Mangement:** even the WuR can be out under energy management. It can be always on or it can have Duty Cycle.

sleep/wakeup schemes: Scheduled rendez-vous

The basic idea behind scheduled rendez-vous schemes is that each node should wake up at the same time as its neighbors. Typically, nodes wake up according to a wakeup schedule, and remain active for a short time interval to communicate with their neighbors. Then, they go to sleep until the next rendez-vous time.

There are different approaches to implement scheduled rendez-vous schemes:

Fully Synchronized scheme



In this case all nodes in the network wake up at the same time according to a periodic pattern. More precisely, ***all nodes wake up periodically every*** T_{wakeup}, and remain ***active for*** a fixed time T_{active}. Then, they return to sleep until the next wakeup instant.

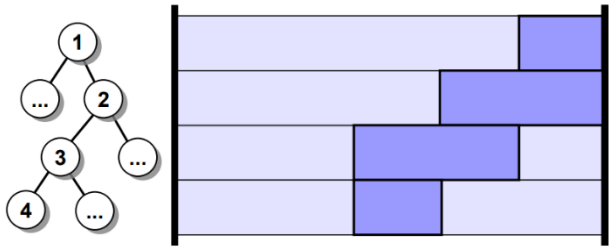
Problems:

During the active period all the nodes are active, so they can communicate: the approach is simple, but it is not energy-efficient because once the nodes wake up, they try to transmit simultaneously, resulting in a large number of collisions that will require retransmission.

If clocks are not synchronized, this implementation is not possible since nodes cannot communicate. Synchronization is not so simple but moreover is energy consuming.

The scheme ***is not very flexible since the size of wakeup and active periods is fixed and does not adapt to variations in the traffic pattern and/or network topology.***

Fixed staggered wakeup pattern



In this schema *nodes* located at *different levels* of the *data-gathering tree wake up at different times*. Obviously, the active parts of nodes belonging to adjacent levels must be partially overlapped to allow nodes to communicate with their children.

The wake-up time will be shorter compared to the full synchronized schema because now fewer nodes need to transmit within a communication window. With fewer collisions and less data to transmit using this scheme, it is possible to have a smaller communication window.

This approach is more energy efficient than the previous one, but nodes still need to be synchronized (need of periodic synchronization update).

Adaptive Staggered Scheme

This scheme is designed to overcome the limitations of other approaches. "Adaptive" indicates that the duration of the "active period" can be made adaptive, no longer fixed, depending on the operating conditions. Each node can have a different duration for its active period compared to other nodes.
For example, one can take into account the number of children, as child nodes can join or leave at any time. Therefore, if a node has many children, it is expected to handle a larger number of packets, requiring a longer active period. Another example is adapting it to the current network traffic.

This approach is excellent from an energy perspective, but it is complicated to administer because when a node changes its active period, we need to communicate this to its parent and child nodes to enable communication between them. This requires coordination among the nodes, which is achieved through exchanging packets. However, this coordination introduces challenges such as increased message exchange, which may or may not reach their destinations, and other issues.

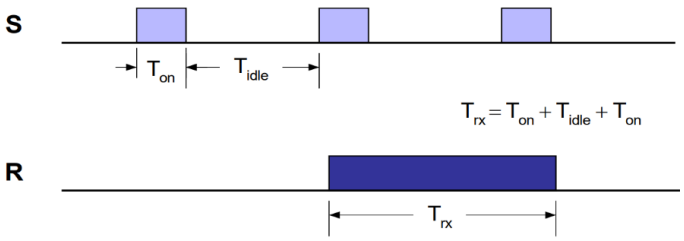
sleep/wakeup schemes: Asynchronous

Asynchronous schemes enable each node to wake up, transmit whenever they desire, and then return to a sleeping state. But how it works? The receiver? If it asleep, it cannot receive, there is not coordination between them.

Now we will explore two versions based on **Asynchronous Sender and Periodic Listening**.

Receiver based

- The ***sender*** transmit ***periodically*** the ***same message***, without any coordination with receiver.
- The ***receiver*** wake up periodically to listen if there are messages. It must be active *at least **Trx** in order to be sure to receive one entire copy of the message*.

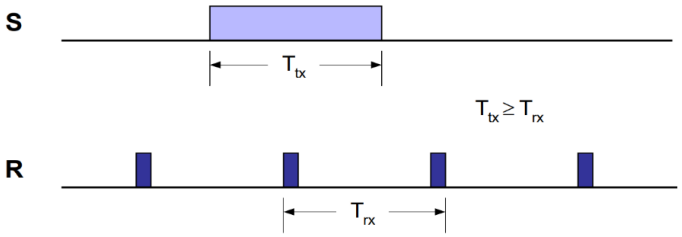


Why is T_{rx} so long? Wouldn't $T_{on}+T_{idle}$ be enough? No. While it's true that I would receive the entire message, it would be divided into two parts. However, the receiver has no way of combining them, so it needs time to listen to the complete message in its entirety.

The problem with this approach is that the sender is required to transmit the same message periodically. Additionally, since there are multiple potential receivers, they need to wake up periodically and remain active for a significant amount of time, resulting in high energy consumption across multiple receivers.

Sender based

- The ***sender*** transmit ***the message just once***
- The ***receiver wake up periodically for very short time and listen:*** if there's no message it go to sleep again, instead *if there's a transmission, the receiver remain active to receive the message (it's not shown in the figure).*

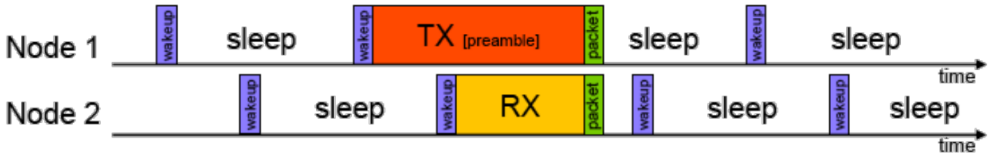


In this second approach, we shift the energy consumption from the receivers to the sender. Since there is only one sender while there are many receivers, and the sender transmits sporadically, we can conclude that this variant is more energy-efficient.

Problem: what happen if the message is not so long? You have to extend the message transmission in order to satisfy the condition.

Asynchronous - Low Power Listening (LPL)

Nodes periodically sleep and perform LPL. Nodes do not synchronize on listen time.
Sender uses a long preamble before each packet to wake up the receiver.



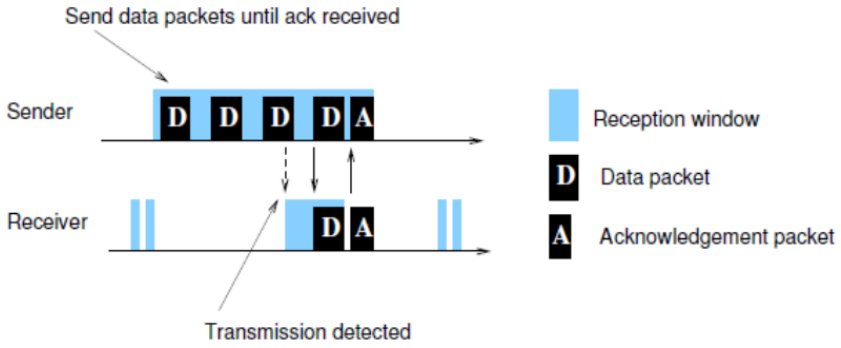
Constraint: check interval ≤ preamble duration

Problems: if the destination is only one, the *all other potential receiver remain active for all the transmission time only to discover, that they are not the final destination of the packet.* Consider also that every transmission wakes up all neighbors!
Furthermore, we notice that the receiving node must remain active for the entire RX interval. This can be considered an energy waste, but in the case of an asynchronous scheme, we can justify this "waste" because it is necessary for communication. By doing so, we allow the transmitting nodes to save energy.

If the receiver sleep time is increased, the duration of the preamble must be increased too, but this implicates more energy consumed for the sender.
If packet transmission is one sporadically, this approach is good.

Asynchronous - ContikiMAC

The idea is to optimize the previous protocol in order to reduce the total energy consumed by the two nodes. Various types of improvements have been proposed in the literature, but the one we will now see is the one used in the Contiki operating system.



In ContikiMAC, the receiver behaves in a similar manner as before. It periodically wakes up, and if no activity is detected, it goes back to sleep. However, if activity is detected, indicating the presence of a transmitted data packet, the receiver remains active and receives the complete data packet. After receiving the data, the receiver sends an acknowledgment (ACK) in the break between two consecutive packets. The sender waits for a certain interval of time between two transmissions to allow the receiver to reply with an ACK.

The improvement made consists of replacing the transmission of the long preamble with a repeated transmission of the message to be sent. Furthermore, the active time required for both the receiver and the sender is shorter compared to the baseline approach. Considering that staying active during communication consumes a significant amount of energy, this results in a substantial energy saving.

Let's now examine additional optimizations introduced in ContikiMAC to avoid erroneous situations and reduce energy consumption by the center-order receiver when such situations occur.

Fast Sleep Optimization

ContikiMAC fast sleep optimization:

1. if a silence period is not detected before ***tl*** (maximum packet transmission time), the receiver goes back to sleep.
2. if the silence period (which is the time between two consecutive packet transmission) observed by the receiver is longer than ***ti***, the receiver goes back to sleep.
3. if no packet is received after the silence period, even if radio activity is detected, the receiver goes back to sleep.

These three events occur when there is noise on the communication channel. Therefore, if any of these three cases occur, it means that what the receiver is observing is not a valid packet, but rather just noise.

Transmission Phase Lock

We previously mentioned that the sender continuously sends copies of the data packet until an ACK is received. The exact number of copies to be transmitted cannot be predicted in advance as it depends on the receiver's behavior. However, after the first successful exchange, the sender received the ACK from the receiver, indicating that the message has been received. At this point, the sender can calculate the number of copies it has sent. Using this information, the sender can estimate when the receiver will be listening during the next transmission. By doing so, the number of copies that need to be sent can be reduced, leading to energy savings.

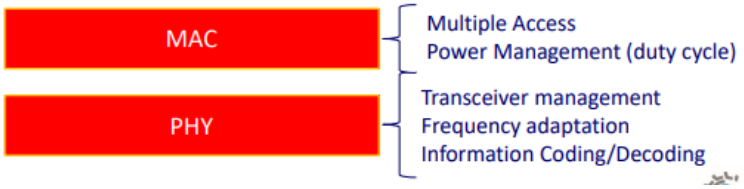
MAC Protocols with Low Duty-Cycle

We will discuss this type of protocols in the next set of slides.

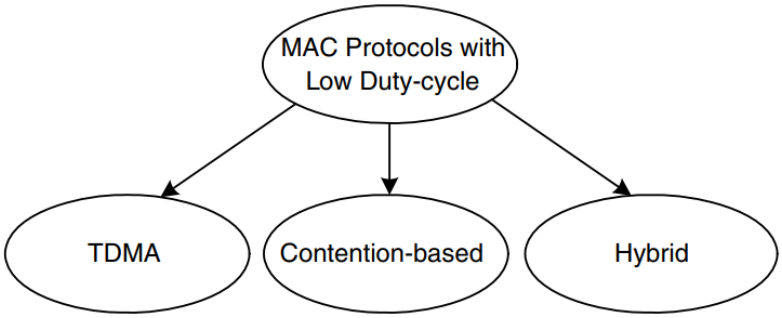
5- Communication technologies for LLNs

Goals

- Introducing low-layer protocols
  - Physical Layer
  - MAC
    - ⇨ Classification of MAC protocols
    - ⇨ Multiple Access + Power Management



MAC Protocols with Low Duty-Cycle



- Time-Slotted Access Protocols
  - ✓ effective reduction of power consumption
  - × need precise synchronization, lack flexibility
- Contention-based MAC
  - ✓ good robustness and scalability
  - × high energy expenditure (over-hearing, collisions)
- Hybrid schemes
  - Both TDMA + Contention Access Schemes
    - ⇨ 802.15.4 MAC
  - Switch between TDMA and CSMA, based on contention
    - ⇨ Z-MAC
- Polling-based Access Protocols
  - Master-slave approach
  - Slaves must be polled by the master node
    - ⇨ According to a certain schedule
  - ✓ effective reduction of power consumption
  - × need precise synchronization

Time-Slotted access protocols

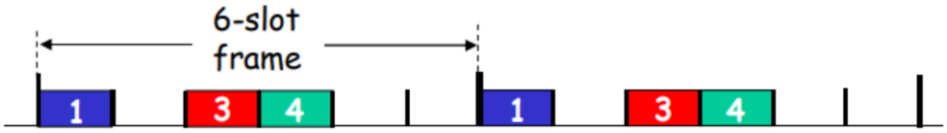
Time Division Multiple Access (TDMA) schemes naturally enable a duty cycle on sensor nodes as channel access is done on a slot-by-slot basis.

Pro:

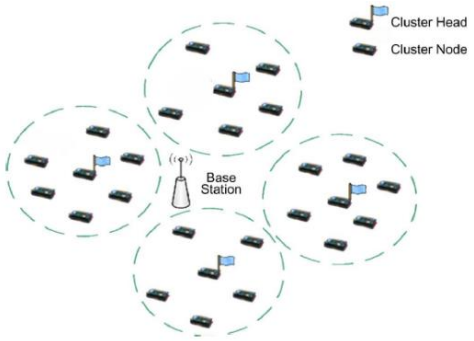
- ***High Energy efficiency***: each node is active only during its own slot(s), and can sleep during the other slots
- ***Guaranteed Bandwidth***: each node gets one or more fixed-length slots in each round. *You can also assign more or less time slots in a certain amount of time*
- ***Bounded and Predictable Latency:*** each node accesses the same slot(s) in every frame. You can calculate the max time the node has to wait before to send the packet. This time is also bounded, after a certain time, the node receive a time slot and it can communicate. It may happen error in communication, but this is another problem

Cons:

- ***Synchronization***: sensor nodes need to synchronize their clocks
- ***Lack of flexibility***: Join and leave of nodes may cause slot re-allocation and *unused slots go idle*
- ***Vulnerability to Selective Jamming attacks***: for example you can try to transmit simultaneously during a specific slot, so you can destroy the communication of a specific slot (selecting jamming attack) and it can be easily confused with communication error.



LEACH - Low Energy Adaptive Clustering Hierarchy



- Nodes self-organized to form clusters: after a certain period there will be some clusters in network
- Nodes elect one node as Cluster Head
- Nodes report data to their CH through TDMA
- Since the CH has the highest energy consumption, sensor nodes rotate in the role of CH

In general, time division-based protocols has an high energy efficiency, but have also limited flexibility (topology change may require a different slot allocation pattern), limited scalability and need synchronization.

Contention-based MAC protocols

What is the difference between Contention-based and CSMA-based protocols? CSMA protocols are part of the family of contention-based MAC protocols, but there are also other approaches to contention.

These type of protocols ensure ***large flexibility***: a *topology change do not require any re-configuration* or schedule update notification. Furthermore, as the number of nodes increases, the number of **collisions** also increases, which reduces reliability and leads to increased energy consumption due to overhearing → this approach is good when the number of node is not so high.
Anyway there's ***not synchronization required***.
In conclusion, these protocols have *low energy efficiency*: nodes may conflict, energy consumed for overhearing.

B-MAC

B-MAC is a protocol designed before the IEEE 802.15 MAC (we will analyze it later).
B-MAC design considerations: simplicity, configurable options, minimize idle listening (to save energy).

In B-MAC, nodes take turns transmitting and receiving messages on a specific channel. Before sending a message, a node listens for a short period of time to check if the channel is busy. If the channel is idle, the node sends its message.

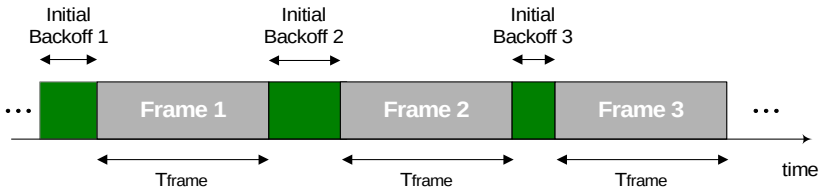
How it works?
When there is a packet ready for transmission, a ***random delay is generated before transmitting*** that is uniformly distributed in [15-68.3] ms
Then the channel assessment; after waiting the nodes sense the channel:

a. *If Channel free → transmit (and wait for ACK)*
b. *Else (Channel busy) → random backoff*
    a. *Uniformly distributed in [12.08 - 193.3] ms*

In B-MAC, several options are ***configurable***, like:

- ***CSMA with or without RTS/CTS***, if the packet size is not so large, comparable to RTS, is not convenient using RTS: since the payload is very small, and the RTS increase the energy consumption, is better avoid it.
- ***Low-power listening***: it can be not used (rare), and all the parameters like sleep interval can be turned
- ***Acknowledgements***: The use of ACKs can be enabled or disabled based on the requirements of the network. Disabling ACKs can reduce the communication overhead and improve the energy efficiency of the network, but may also result in lower reliability if messages are lost and not detected.

With only one node:



Hybrid schemes

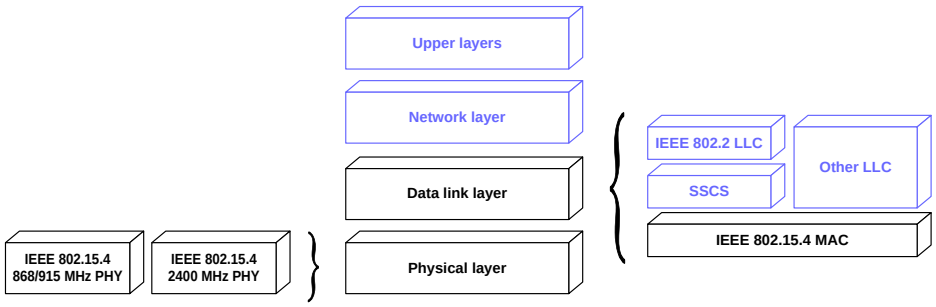
These schemes combine time-slotted access protocols with contention based MAC. This approach can be useful when there are different type of traffic in the same network (different type of QoS necessary, for example some data are time sensitive or sensitive in terms of reliability)

Polling-based access protocols

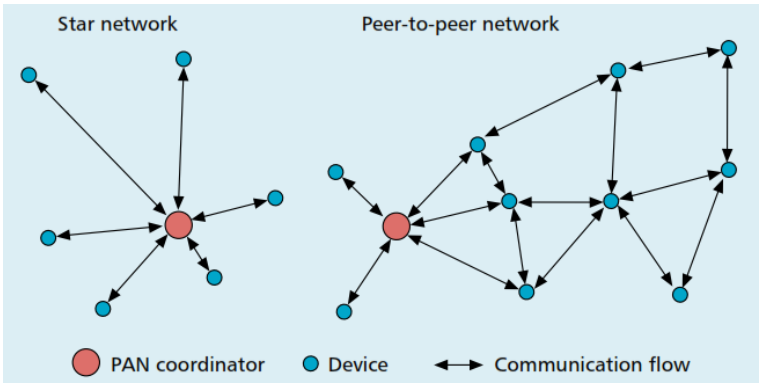
This protocols use a master-slave approach (e.g. Bluetooth).
Slaves must be polled by the master node according to a certain schedule.
There's an effective reduction of power consumption but need precise synchronization.

IEEE 802.15.4 standard

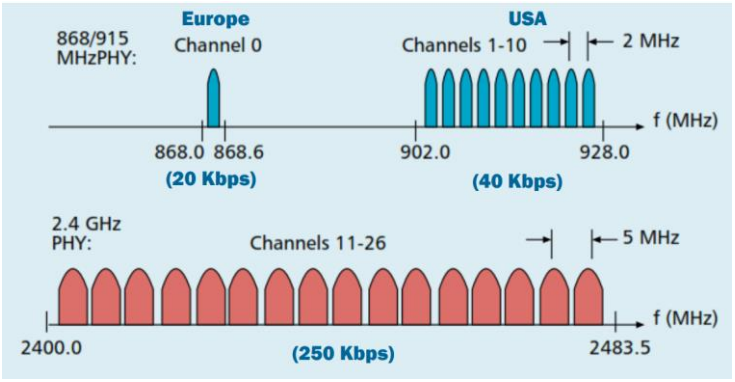
It's a standard for low-rate, low-power, and low-cost Personal Area Networks (PANs). This standard covers only physical and data link layer. IEEE 802.15.4 has been widely used as a basis for other network protocols such as Zigbee and 6LoWPAN, which implement additional functionalities and application-specific features.



The **network topologies** supported by the standard are:



Communication channels:

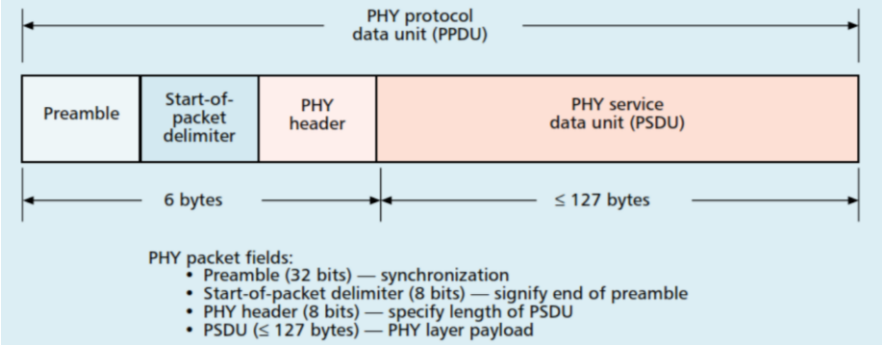


Channels 11-26 provide a higher bitrate of 250 Kbps compared to other channels, making them the most commonly used. Throughout this course, we will refer to these specific channels.

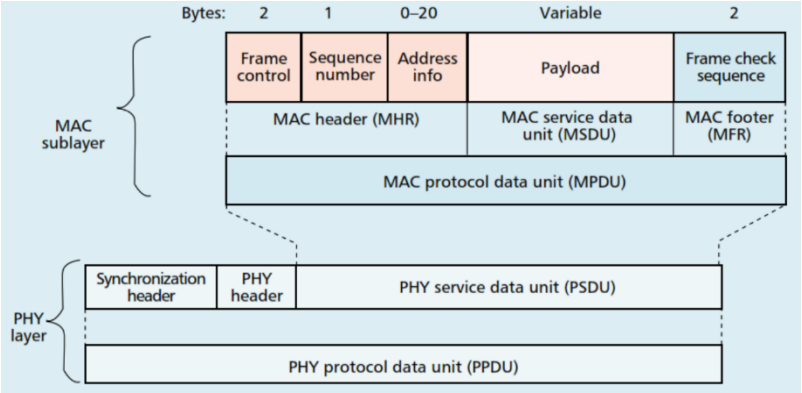
Addresses

- *PAN Address*: ***Identifies the PAN*** within a certain geographic area. Allows the communication between different PANs. *16 bits* (Broadcast PAN Address: 0x FFFF)
- *Device Address*: **Identifies the device within a PAN**.
  - Extended Address: *64 bits*
    - Most significant *24 bits* assigned by IEEE represent the OUI (Organizationally Unique Identifier) to uniquely identify manufacturers and other organizations that produce network devices.
    - Least significant *40 bits* assigned by manufacturer
  - Reduced Address: *16 bits*
    - Negotiated with the PAN coordinator
    - *Can replace the extended address in all communications*
    - Broadcast address: 0x FFFF

Frame format



- ***MAC header (3 to 23 bytes):***
  - Frame control
  - Sequence number
  - Address Info, from 0 to 20 byte: 10 bytes (2 PAN addr + 8 extended device addr) for each of the two destination and source addresses
- ***Payload***
- ***Frame check*** can be used for reliability check to check if there are errors

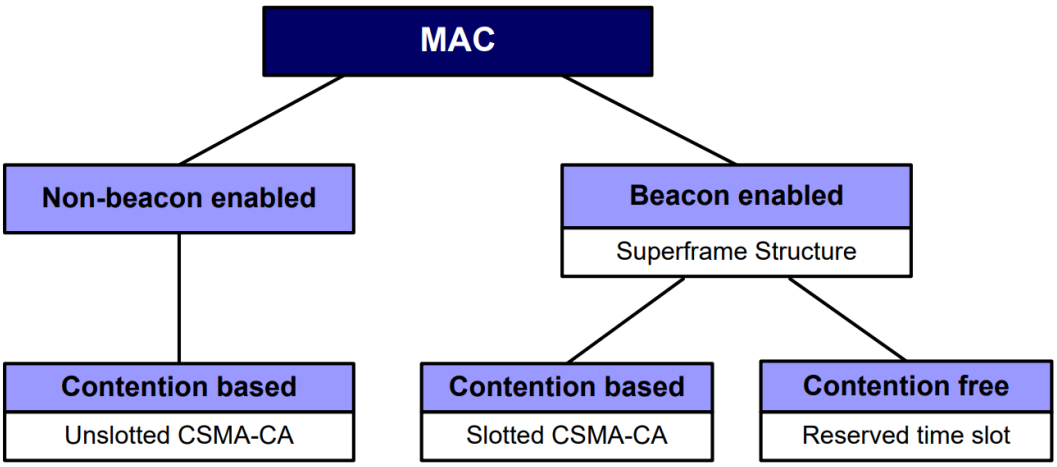


IEEE 802.15.4: MAC Protocol

The **MAC layer**, short for Media Access Control layer, is a sublayer of the data link layer.
It is responsible for controlling access to the physical medium and managing the communication between devices in a local network. The MAC layer defines protocols and procedures for transmitting data frames over the network and ensures that different devices can share the network resources efficiently and fairly.

There are two different channel access methods:

- **Beacon-Enabled** duty-cycled mode
- **Non-Beacon Enabled** mode (aka Beacon Disabled mode)



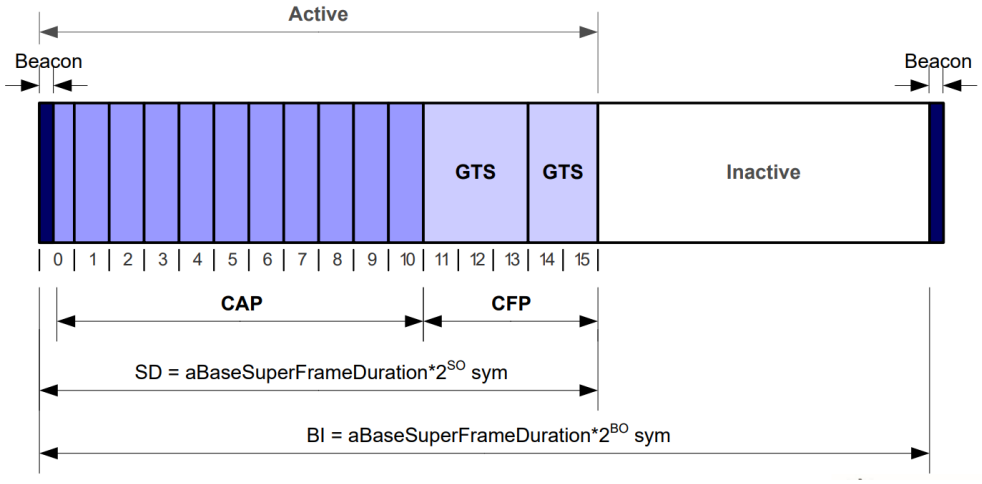
A **"beacon"** is a special frame that contains control information and is periodically transmitted by the coordinator node. It is used to synchronize and coordinate other devices within the network.

In Non-Beacon Enabled mode, there are no beacons, and nodes can communicate at any time they want.

IEE 802.15.4 MAC protocol: Beacon enabled mode

We refer to the star topology: the PAN coordinator is responsible to send periodic Beacon (the period is a parameter that can be changed).
The beacon enabled mode provides a power management mechanism based on a duty cycle. It uses a **superframe structure** which is bounded by beacons, i.e. special synchronization frames generated periodically by the coordinator node(s) that divide time into multiple Beacon Intervals (BI).

In reality, time is divided into Backoff slots (or periods), which represent the smallest unit of time. Beacon Intervals and Superframe have a duration expressed in multiples of Backoff slots.
To synchronize the network, all node activities, such as starting the transmission of a frame, can only begin at the beginning of a Backoff slot.



(This image is misleading. In reality, the active part is much shorter than the inactive part)

Inactive period: nodes enter a low-power state to save energy.

Active period: nodes communicate with the coordinator they associated with.
The active period is denoted as Superframe Duration (SD)

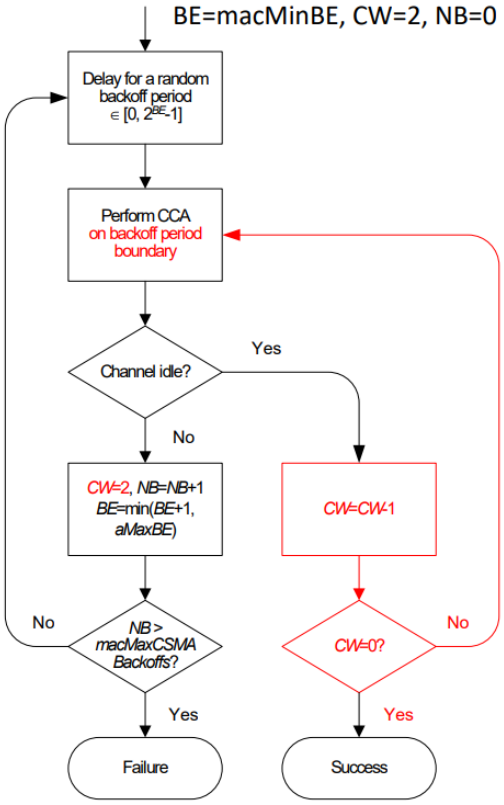
Active period is divided into:

- **Contention access period** (**CAP**), during the CAP a slotted CSMA/CA algorithm is used for channel access. During this interval, nodes can request to be assigned a GTS
- **Contention free period** (**CFP**), communication occurs in a TDMA style by using a number of Guaranteed Time Slots (GTS), pre-assigned to individual nodes (for the current superframe)

It is important to note that if a node successfully sends its request to transmit in a GTS, it can only transmit in the next Beacon Interval. This is because the response from the coordinator node will arrive within the next Beacon.

How communication occur during Contention Access period that follow immediately the reception of the beacon:

CSMA/CA algorithm



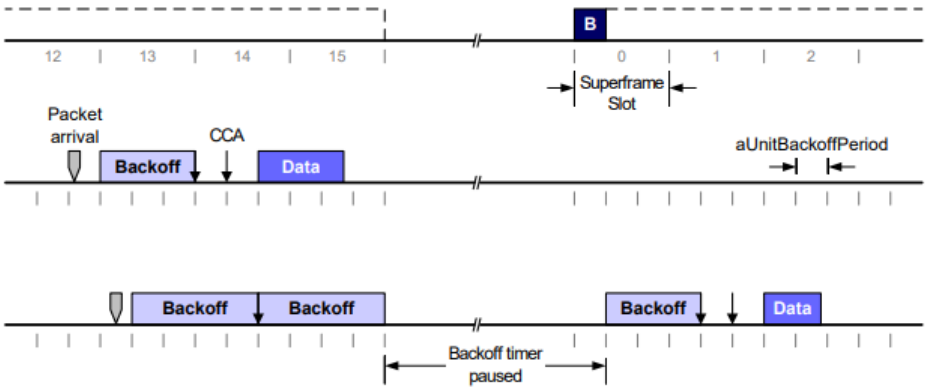
1. A set of state variables is initialized:
   a. ***CW = 2***, Contention Window, it is a counter that is decremented every time a channel assessment is performed, and it is initialized to two because the algorithm requires two consecutive channel assessments before allowing the transmission.
   b. ***NB = 0***, counts the number of "backoff stages" carried out for the on-going transmission
   c. ***BE = macMinBE***, stand for Backoff Exponent used for deriving the backoff interval
2. A *random backoff time*, uniformly distributed in the *range [0, 320·(2^{BE}-1) μs*], *is generated* and used to initialize a backoff timer. (*320 μs* is the time unit for the backoff delay in the 802.15.4). *The starting time of the backoff timer is aligned with the beginning of the next backoff slot.* ***In addition, if the backoff time is larger than the residual CAP duration, the backoff timer is stopped at the end of the CAP and resumed at the beginning of the next superframe.*** When the backoff timer expires, the algorithm proceeds to step 3.
3. A ***Clear Channel Assessment (CCA)*** is performed to check the state of the wireless medium.
   a. If the medium is busy, the state variables are updated as follows:
      i. NB = NB+1
      ii. BE = min(BE+1, macMaxBE)
      iii. CW = 2

      If the number of backoff stages has exceeded the maximum admissible value (i.e. NB>`macMaxCSMABackoffs`), **the frame is dropped**. Otherwise, the algorithm falls back to step 2.
   b. If the medium is free, then CW=CW-1.
      i. If CW=0 then the frame is transmitted
      ii. Otherwise the algorithm falls back to step 3 to perform a second CCA.

Be aware that the "success" state just means that you start transmitting your packet, but it does not guarantee that it will be received correctly.

Why two consecutive CCA?

If the clocks are not aligned, when two nodes sense the medium, both sense the medium free and they could try to communicate simultaneously. For this reason two channel assessment are performed.



What if the packet is not received correctly?

ACK is not mandatory, can be enabled or disable depending on application reliability requirements.

If enabled: Retransmission if ACK not (correctly) received within the timeout, but at each retransmission attempt the backoff window size is re-initialized. Only a maximum number of retransmissions allowed (`macMaxFrameRetries`). Remember, it increase the energy consumption.

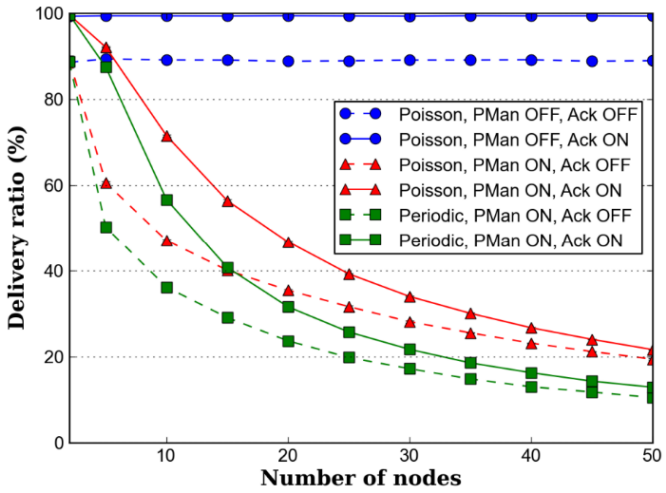
Security

The standard specifies three security mode:

- **Unsecured Mode**: No security service provided
- **ACL Mode**: Access Control based on Access Control List (ACL), any node has an ACL specifying nodes authorized to perform specific actions
- **Secured Mode**: in addition to the ACL Mode we have replay attack protection, message integrity and confidentiality

Performance

Performance are compered by comparing the MAC performance under **Poisson** (packet generated apparently random, following exponential distribution) and **Periodic traffic patterns**, when power management is enabled and disabled.



When nodes are **always active**, packets are transmitted immediately after their generation. Since generation times with Poisson traffic are distributed along the Beacon Interval there is almost no contention among sensor nodes.

Instead, ***when power management is enabled, the packets generated during the sleeping time are deferred to the beginning of the next Active Period, when all nodes wake up at the same time***. Therefore, ***channel access attempts tend to become synchronized***.
If the data generation process is Periodic (instead of Poisson), packets are generated just before the beginning of the Active Period so as to minimize the latency. Hence, *channel accesses are perfectly synchronized and this increases contention among nodes.*

Why the 802.15.4 MAC Reliability Problem? Possible answers are:

- The access method is CSMA: Contention increases with the # of active nodes
- The periodic Beacon synchronizes nodes' accesses: All sensor nodes contend for channel access upon receiving a beacon

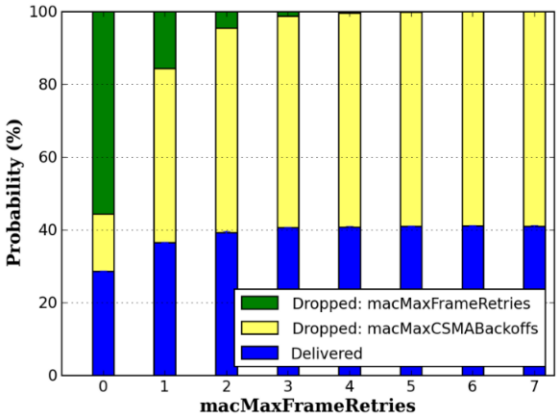
However, when comparing the performance of the CMSA/CA algorithm we just discussed with the CMSA/CA algorithm used in Wi-Fi technology (which is also based on CSMA but slightly different from the one we just saw), we understand that the reason for the poor reliability we encountered in our case is not solely due to the use of CSMA or synchronization via Beacons. These same principles are used in Wi-Fi technology, but in that case, we do not have these significant reliability issues.

Can the delivery ratio curve be improved? ***Which are the impact of CSMA/CA parameters?***

Parameter	Allowed Values		Description
	2003 Release [47]	2006 Release [9]	
macMaxFrameRetries	Constant: 3 (*aMaxFrameRetries*)	Range: 0-7 **Default: 3**	Maximum number of retransmissions
macMaxCSMABackoffs	Range: 0-5 Default: 4	Range: 0-5 **Default: 4**	Maximum number of backoff stages
macMaxBE	Constant: 5 (*aMaxBE*)	Range: 3-8 **Default: 5**	Maximum backoff window exponent
macMinBE	Range: 0-3 Default: 3	Range: 0-7 **Default: 3**	Minimum backoff window exponent

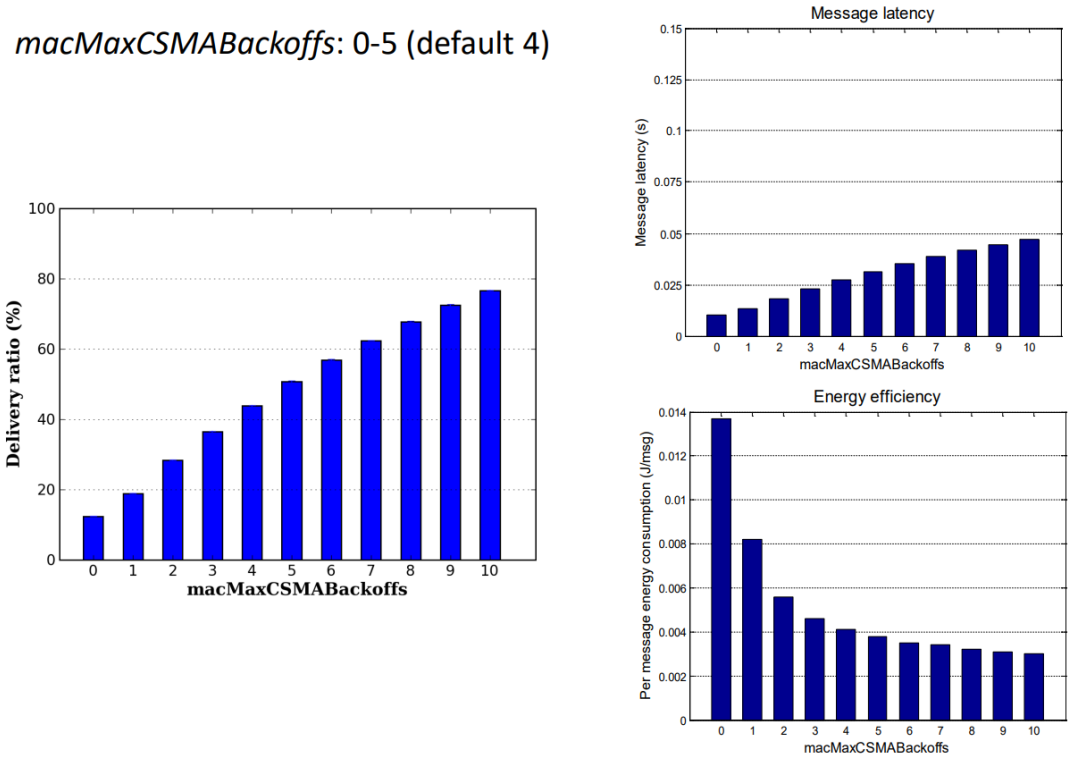
Analysis of each single CSMA/CA parameter:

- ***macMaxFrameRetries*** 0-7 (default 3)



As we can see, from 3 to 7 nodes, the reason for our low performance is not that packets are discarded after exceeding the maximum number of retransmissions. *The main problem* lies in the *maximum number of backoff and not the retries,* packets are not transmitted neither.

- ***macMaxCSMABackoff*** (0 – 5, default 4)



Increasing this values, packets have more chances to be delivered and the *delivery rates increased*, but it *increased also the avg latency* (as expected, since more packet are transmitted, notice also that when the number of backoff is 0 is not good, we have low latency but because we drop a lot of packet before they can be send).
The energy per message decreases. Here we are considering the average energy consumed for transmitting a packet, calculated by dividing the total energy used by the network of nodes by the number of packets received successfully. While the total energy may increase, the increasing number of successfully received packets causes this ratio to decrease.
Another conclusion is that even if we increase the number of backoff stages to very high values (like 10), we observe that the delivery ratio remains below 80%. So, if we are interested in achieving 100% reliability, we are unable to provide these performance levels.

- For the sake of simplicity, we will not include the plots for the other two parameters, but the trend is quite similar to that of the previous test. (delivery rates and avg latency increases, energy per message decreases)

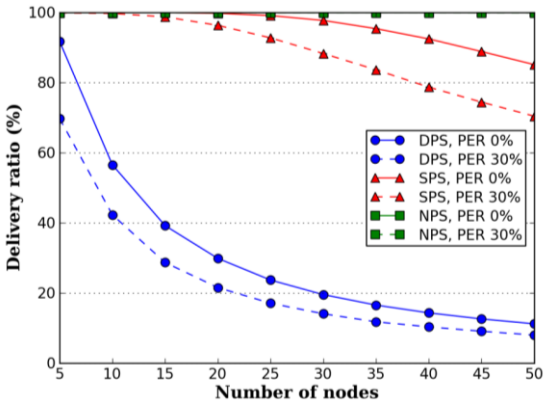
From these tests, we can conclude that **the MAC unreliability problem is mainly due to the CSMA/CA algorithm:** the periodic beacon synchronizes channel accesses thus maximizing contention. However, the problem is exacerbated by the

default MAC parameter values, which are not suitable for Wireless Sensor Networks (WSNs) operating in Beacon-Enabled (BE) mode.

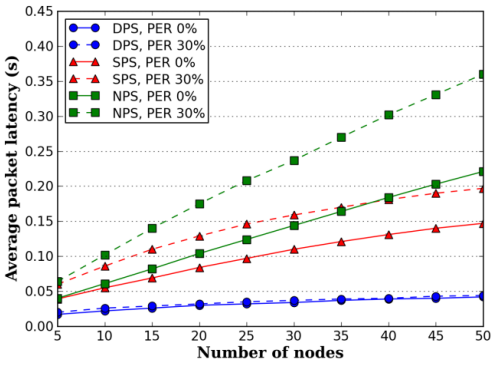
How to avoid the MAC Reliability Problem? (spoiler: Use the maximum value allowed by the standard)

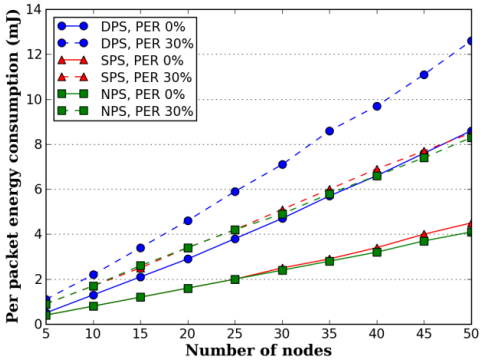
- **Default Parameters Set** (**DPS**). This set consists of the default values specified by the standard.
- **Standard Parameters Set** (**SPS**). All parameters are set to the corresponding maximum value allowed by the standard.
- **Non-standard Parameters Set** (**NPS**). This set of parameter values is not compliant with the 802.15.4 standard. In particular, all parameters are set to values beyond the maximum ones allowed by the standard.

	macMinBE	macMaxBE	macMaxCSMABackoff	macMaxFrameRetries
DPS	3	5	4	3
SPS	7	8	5	7
NPS	8	10	10	10



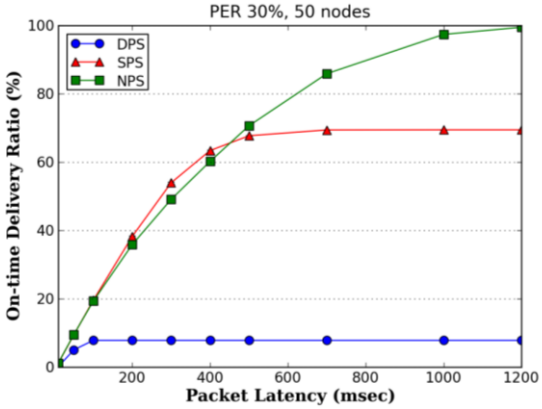
Avg. Latency Energy/msg





PER stands for Packet Error Rate, which refers to the rate at which sent packets are not received correctly.

The following figure shows that, while it is always convenient using parameter values larger than the default ones, using very large values might not be so convenient, depending on the type of industrial application. For example, when the deadline is less than or equal to 100 ms, the fraction of packets delivered on time is below 20%, irrespective of the parameter set.

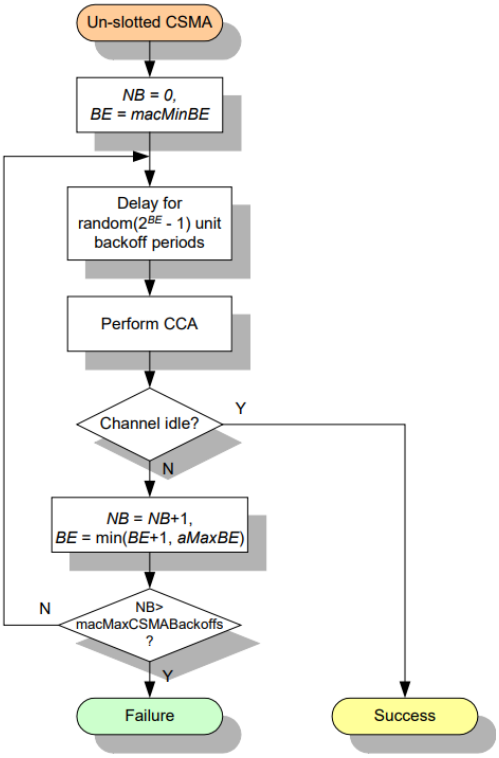


Conclusion

This MAC protocol is not suitable for applications with strict delay constraints. It is not possible to achieve both high reliability and low latency simultaneously because high reliability comes at the expense of increased latency.
So this protocol is good for non critical applications.

IEE 802.15.4 MAC protocol: Non-Beacon enabled mode

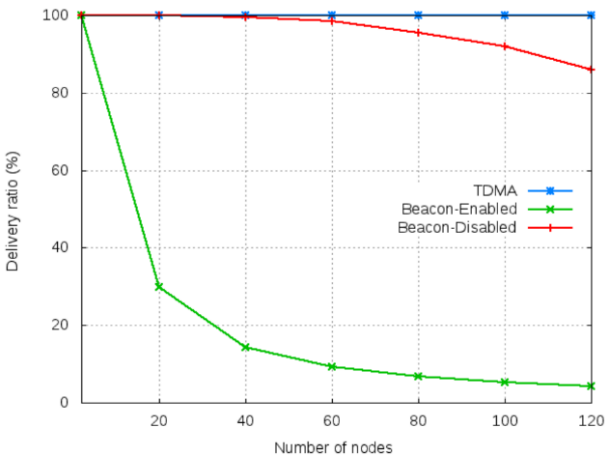
There are no Beacons, and therefore no Beacon Intervals. There are no active periods or inactive periods.
Nodes are allowed to transmit at any time, and the only protocol used for communication between nodes is the CMSA-CA protocol we discussed in a previous lesson, but in a slightly different version.



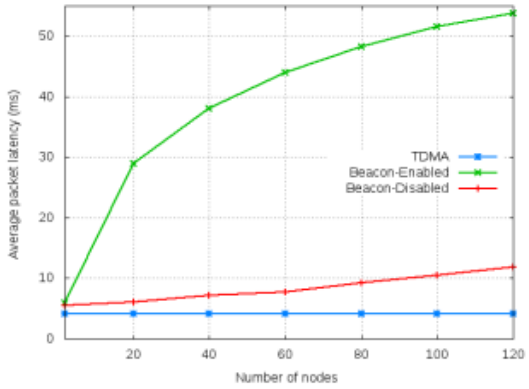
The main difference (beyond the time-slots) is that the CCA is performed only once.
Even with this approach, the **acknowledgment mechanism** is optional. The sender retransmits the message if it does not receive an ACK after a certain time interval. There is a parameter that specifies the maximum number of retransmission attempts.

Performance

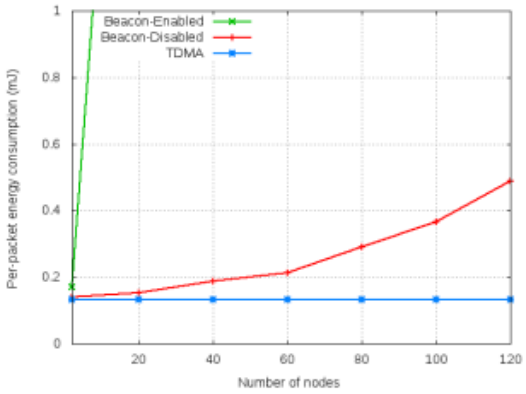
This approach has much higher delivery ratio than the beacon-enabled one. This because communication attempts are distributed over the time and they are not “synchronized” after the beacon is received.



Latency



Energy/packet



We conclude that using the non-beacon enabled approach is better than using the beacon enabled approach.

Beacon enabled mode have the advantage of power saving, but actually it can be performed also in non-beacon mode, since the PAN coordinator(s) must be always active, while regular nodes don't need it. Since nodes generally have only to transmit packets, they can go to sleep and then wake up when they have to transmit something.
In addition, the Beacon enabled mode requires synchronization among the nodes, which is a costly process.

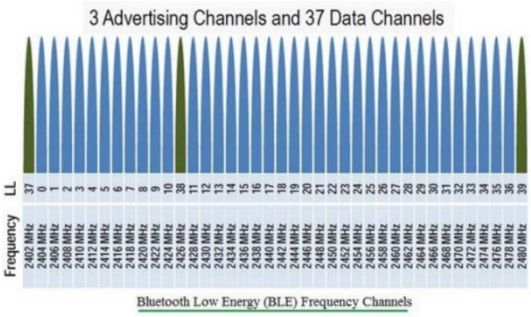
Other LLN Technologies

Bluetooth low energy (BLE)

It works very similar to the classic Bluetooth, but consuming less energy.

It allows two modes of communication:

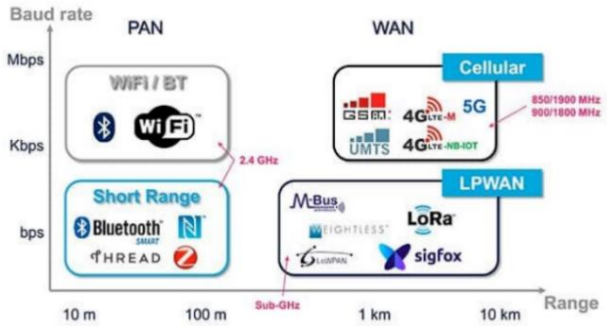
- **Communication Mode:** designed to enable communication between nodes, it uses a master-slave approach.
- **Advertise Mode:** involves broadcast communication where we have two types of nodes. **Advertisers** are Beacon devices that send messages, and then we have **observer** nodes that wait to receive the messages.



It can work with a "**hub and spoke**" topology where nodes utilize a hub to communicate with each other, or with a "**mesh topology**" where devices are directly connected to each other to create a local network.
In order to be connected to the Internet, from Bluetooth 4.2, devices can be connected to an application gateway (translation from Bluetooth stack to Internet stack) or directly to the router, using an IPv6 address.

Long-range technologies

The main limitations of BLE and IEE 802.15.4 is the short communication distance and the sensitivity to the interferences. In environments where we need higher communication distance and there are interferences, we can't use them. We can use more energy, but there are other protocols that try to save it and increase the distance or the robustness.



IEEE 802.15.4g

It is an evolution of the 802.15.4 standard, specifically targeted towards Wireless Smart Utility Networks. It allows longer communication range than IEEE 802.15.4 with similar transmission power due to the fact that it operates at frequencies below 1 GHz (instead of 2.4 GHz) and these frequencies are less crowded. In order to do this, new physical layers have been introduced (linked to the modulation of the signal).

Is 802.15.4g suitable to IoT applications requiring a long range?
The communication range strongly depends on the external environment: buildings, trees, and other objects that attenuate the signal strength.
This is because obstacles may hinder signal propagation. From experiments, it has been found that in any case, the communication range is lower than 1 km, but this range decreases as the number of obstacles increases.
We understand that this protocol may not be suitable for certain applications that require a long communication range. For such applications, other technologies like low-power WiFi or 5G are necessary.

5G

Goal: Connecting everything around us with 10x increase in peak bitrate, 10x decrease in latency, 100x increase in traffic capacity over 4G.
There are 3 standard:

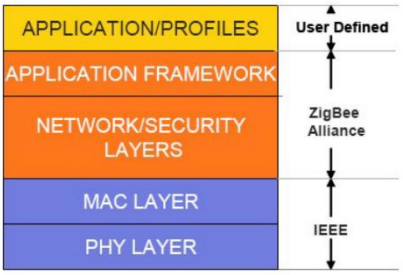
- **Enahnced Mobile Broadband (eMBB)**: Increased bandwidth and reduced latency, wrt 4G, used for *Mobile Augmented/Virtual Reality, 360°Video Streaming*
- **Massive Machine Type Communications (MTTC)*:*** Narrowband access for sensing, metering & monitoring apps, *reduced Power Consumption*
- **Ultra-Reliable Low Latency Communications (URLLC)*:*** Latency below 1 ms, used for *factory Automation, Autonomous Driving, Healthcare app (reliable application)*

6- Non-IP Technologies

The goal is to connect a Low-Power and Lossy Network (LLN), which uses one of the short-range protocols seen earlier to communicate within the network of devices, to the internet in order to reach the Internet of Things (IoT).
There are two types of solutions available: **non-IP solutions** and **IP-based solutions**. Let's begin by exploring the non-IP solutions.

ZigBee

ZigBee is a non-IP solution for multi-hop communication (another example is Bluetooth).
ZigBee is complementary to 802.15.4, meaning it builds upon it for the lower layers of communication. ZigBee specifically focuses on implementing the network and application layers, while relying on 802.15.4 for the lower-level communication aspects.

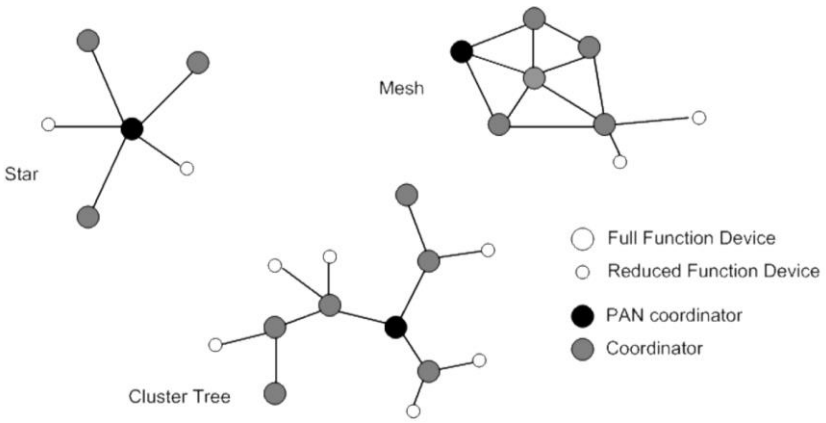


It can be used for: patient monitoring, fitness monitoring, TV remote, light control, irrigation, security, etc.

ZigBee device types:

- **ZigBee Coordinator** (ZC): One required for each ZB network. Initiates the network formation and manages all the activities inside the network.
- **ZigBee Router** (ZR): Participates in multi-hop routing of messages. Typically they are not sensor (do not produce data) but they are relay nodes. They receive data from other ZB nodes and just forward this data to the final destination.
- **ZigBee End Device** (ZED): Does not allow association or routing. They just produce data and send data to a ZB routers. Enables very low cost solutions.

ZigBee topologies



Routing in Zigbee

The routing algorithm depends on the topology used in the sensor network. Two topologies can co-exist and routing algorithms can be switched based on the destination.

Tree-based Routing (for cluster-tree topology)

In a tree topology routing can *only happen along the parent-child links* established as a result of join operations (this is called ‘‘tree-based routing’’). ***Routers maintain only their address and the address information associated with their children and parent.***
Forwarding:

- The packet destination is one of the node’s children: directly forwarded
- If no, it’s forwarded to the node’s parent

This kind of routing algorithm is not necessarily the most energy-efficient but is very simple to implement and allows routers to operate in a beacon-enabled network.

Mesh Routing (for mesh topology)

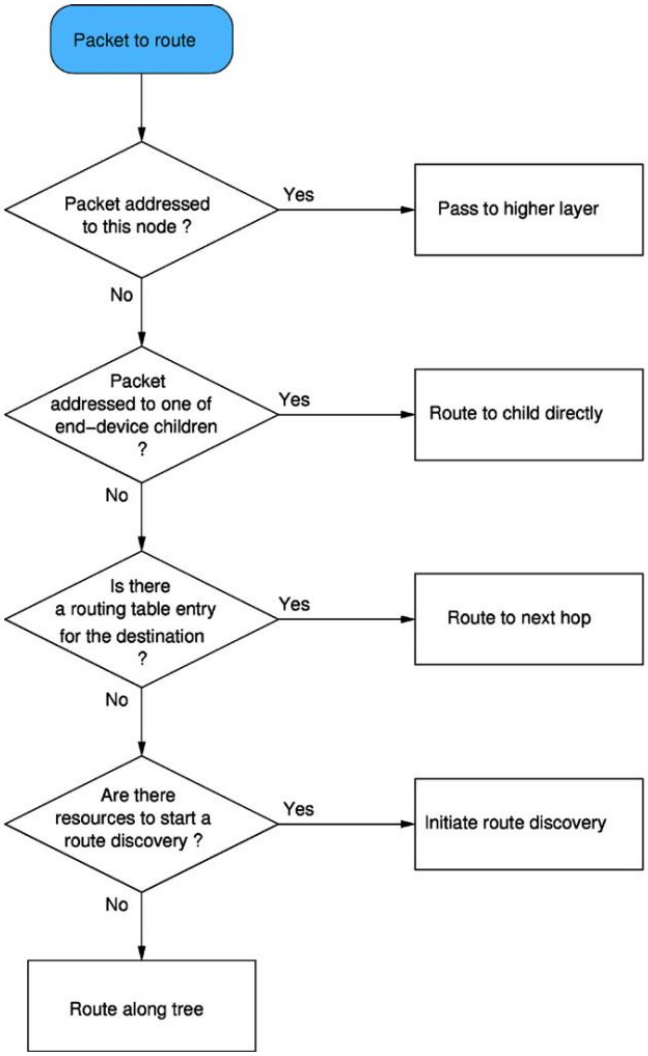
The mesh network topology is more complex to handle and beaconing is not allowed but is more robust and resilient to faults. Routers maintain a **routing table (RT)** and employ a route discovery algorithm to construct/update these data structures on the path nodes.

Destination Address	Next-Hop Address	Entry Status
16 bits	16 bits	Active/ Discovery/ Inactive

Routing table

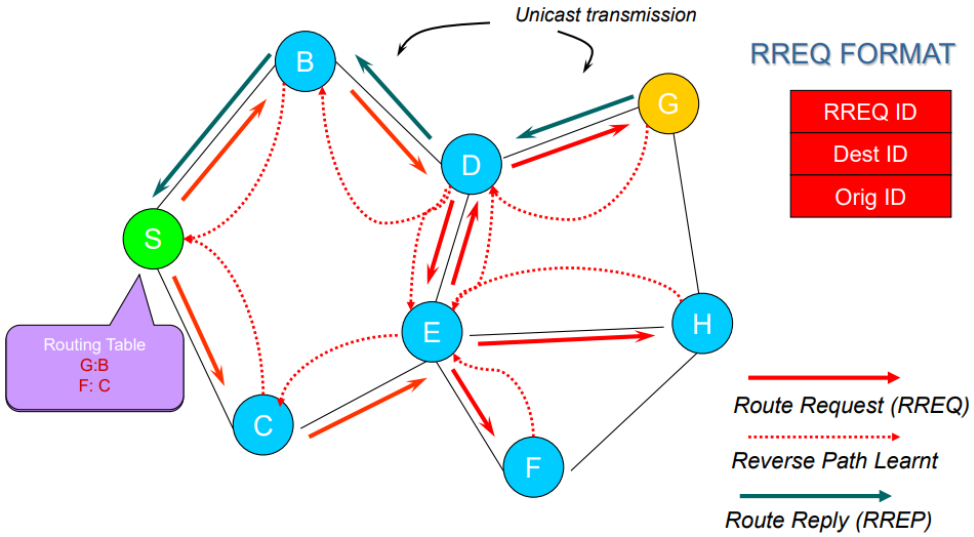
In this approach, the routing table acts as a cache where information is stored only after it has been obtained, rather than being acquired in advance. As a result, there is no proactive approach as seen in traditional routing protocols. Instead, the routing table is populated on-demand, allowing for dynamic and efficient routing based on real-time information.

The algorithm to forward a packet:



Route discovery is based on AODV: Ad-hoc On-demand Distance Vector

It's based only on Non-Beacon Enabled mode with routers are always active and other nodes can exploit duty cycling. In this algorithm information are not derived in advance, but only when necessary. This means that the path that a packet needs to traverse to reach the destination node is calculated dynamically at the last moment, specifically when the packet needs to be sent. There is no pre-computed path in advance. This approach is called **Reactive approach** (for the active approach adopted in the traditional routing there aren't enough resources).



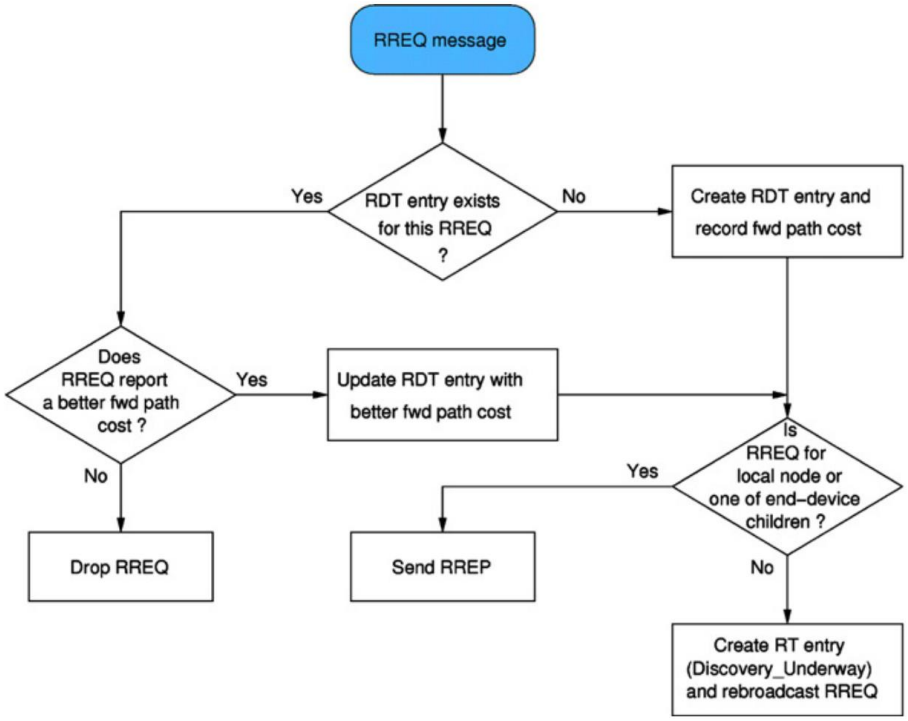
In addition to the Routing Table there is another table, the **Route Discovery Table** (**RDT**) which is maintained by routers and the coordinator to implement route discovery. For each entry in the table, the following fields can be found:

FIELD NAME	DESCRIPTION
RREQ-ID	Sequence number given to every RREQ message being broadcasted
SOURCE ADDRESS	Network address of the RREQ initiator
SENDER ADDRESS	Network address of the device that sent the most recent lowest-cost RREQ
FORWARD COST	The accumulated path cost from the RREQ originator to the current node
RESIDUAL COST	The accumulated path cost from the current node to the RREQ destination
EXPIRATION TIME	Number of milliseconds until this entry expires

Why RREQ-ID is used? To prevent loops and to select the most recent path.

RREQ Processing

When a node requires a route to a specific destination, it initiates the route discovery process by broadcasting a **Route Request** (**RREQ**) message. This RREQ message propagates through the network, being forwarded by intermediate nodes, until it reaches the destination node or a node that has a valid route to the destination.
Each RREQ message carries a RREQ ID which the originator increments every time it sends a new RREQ message. This way the RREQ ID and source address can be used as a unique reference for a route discovery process.
When a node receive RREQ:

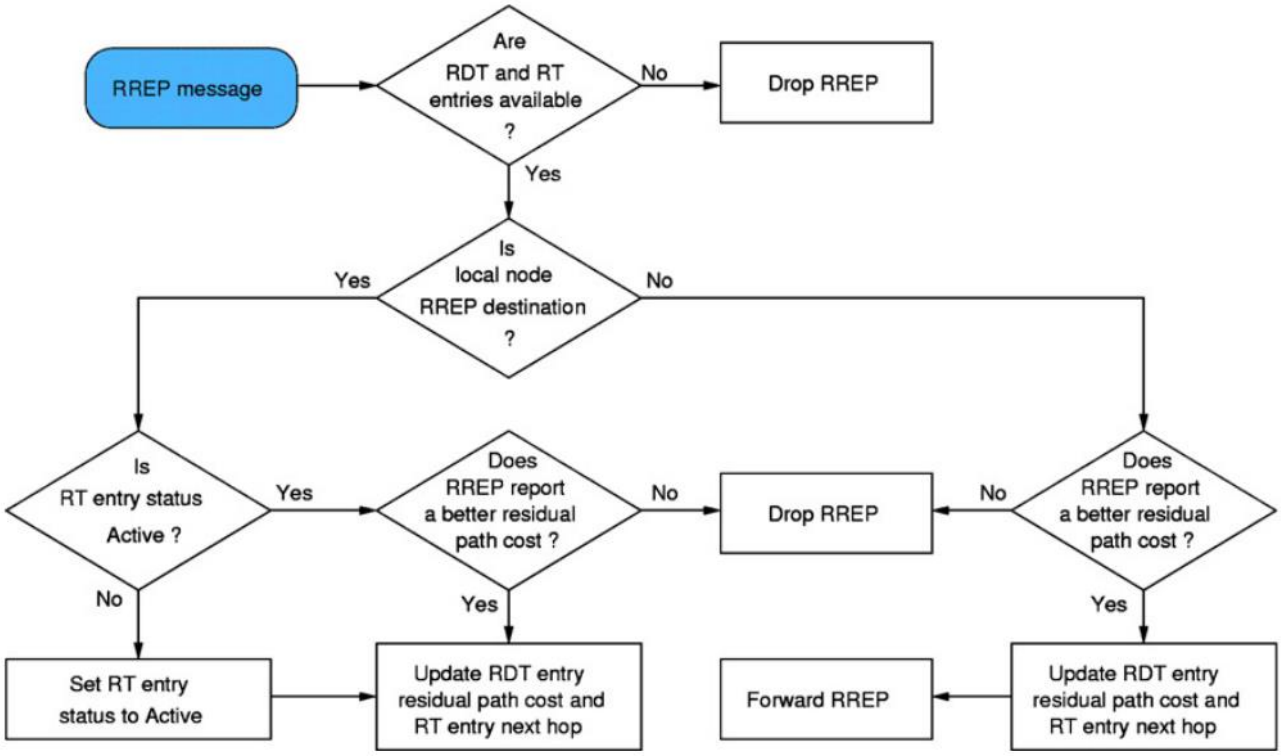


RREP message is addressed to the route discovery originator *(unicast)* and carries with it a *residual cost value field that each node increments as it forwards the message*.
Upon receipt of a route reply (RREP) message, a node retrieves the RDT and RT entries for the associated route discovery.

- If the node is the *RREQ originator and this is the first RREP it received*, it sets the *RT entry to Active and records the residual cost and next hop in the RDT entry.*
- In all *other cases* it *compares the residual cost from the RREP with the one from the RDT entry*. If the former is higher the node discards the RREP message; otherwise it updates the RDT entry (residual cost) and the RT entry (next hop). A node that is not the RREP originator must also forward the RREP towards the originator.

Note that intermediate nodes never change the RT entry status to Active as a result of receiving a RREP message. They will only change the entry status upon reception (and routing) of a data message for the given destination.

This algorithm is related to the reception of a Route Reply (RREP) message. Whenever a node receives an RREP message, it executes this algorithm to process the message and update its routing table accordingly.



Route Maintenance

Link failures are considered frequent events in this context, unlike in traditional Internet networks where they are typically considered rare events. When a node detects a link failure, it sends an Unsolicited Route Reply (URR) message to all its neighboring nodes to inform them that the link in question is no longer available. The neighboring nodes, upon receiving the URR message, will also propagate the information to their neighbors in a similar manner.

How to connect LLNs to the Internet?

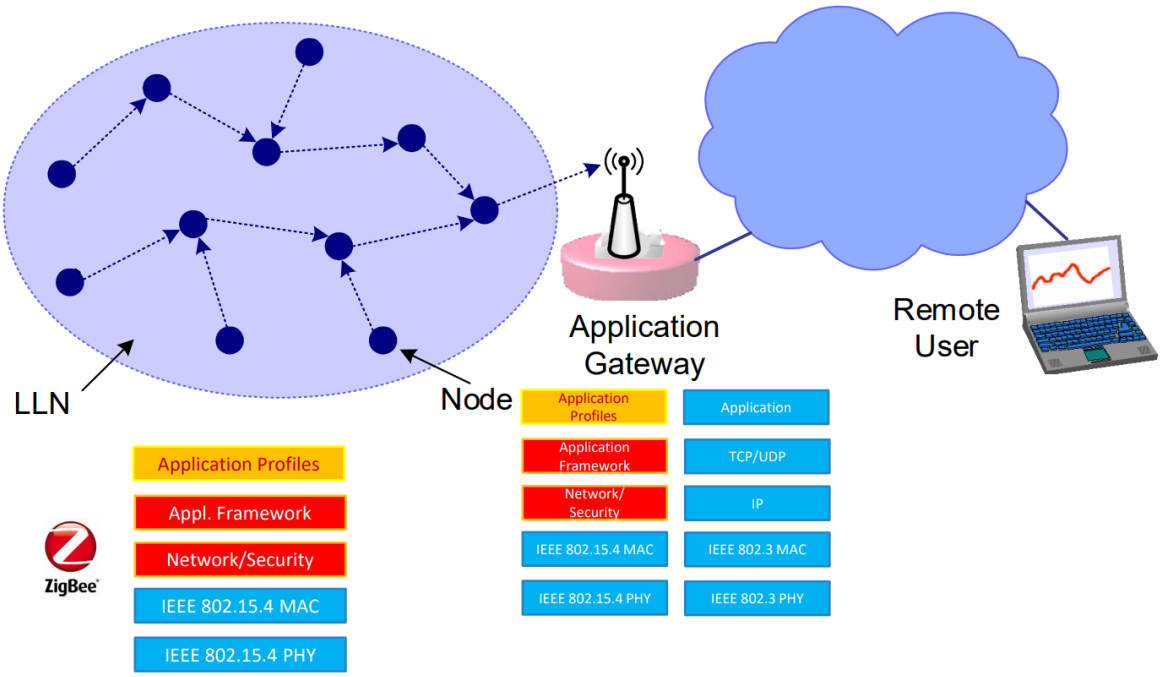
Assume having a LLN, how to connect it to Internet? Two possible approaches:

- Application Gateway

On one side we have a LLN (Low-power and Lossy Network) that implements a complete protocol stack, such as the ZigBee protocol stack, the Bluetooth protocol stack, or others. This implies that within the network of nodes, all the layers of the protocol stack are already implemented and operational. As a result, the nodes are capable of sending packets to a special node within the network called the Application Gateway. The Application Gateway serves as a bridge between the local network and external devices, allowing connectivity to the Internet.

Specifically, when a message arrives at the Application Gateway, it undergoes a translation process to prepare it for transmission over a different network that uses another protocol, such as the TCP/IP protocol. The translation is

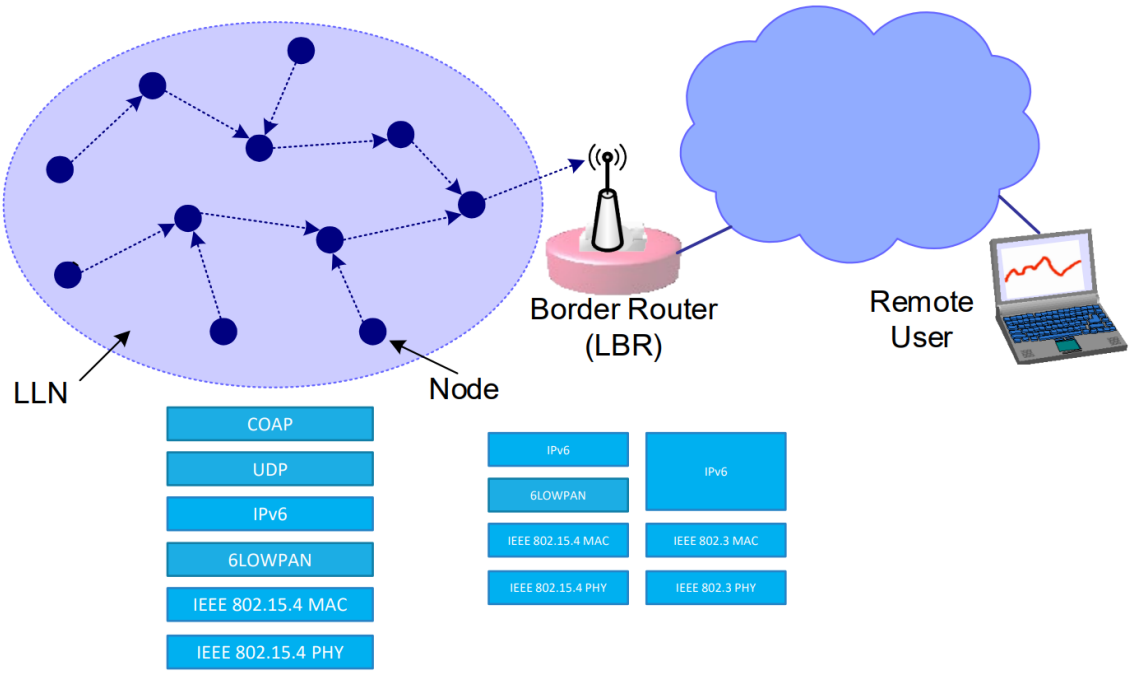
performed in the application layer.
So the Application Gateway functions as an interpreter, taking on the responsibility of translating and forwarding messages between the two networks.



- Fully-IP Solution

Nodes in the LLN implement the IoT Protocol stack, which is based on IPv6. Each node is assigned an IPv6 address and is capable of communicating in the language of the IoT.

To enable communication between the local network and the external network, a **border router** is used. This border router has two interfaces, allowing it to connect to both networks.
On one side, it implements the IoT protocol stack up to the networking layer, specifically up to the IPv6 protocol. Below IPv6, it incorporates the 6LOWPAN adaptation layer, which is necessary to facilitate communication between the 802.15.4 protocol and the IPv6 protocol. On the other side, the border router implements the traditional Internet Protocol Stack, where the 6LOWPAN adaptation layer is not required.



7- IPv6 for LLNs

IPv6 is Network layer protocol, Is the "evolution" of IPv4. Differences are:

- ***Larger Address Space***: to cope with the ever increasing number of connected objects (from 32 to 128 bit)
- ***Auto-configuration***: with very large networks management becomes challenging
- ***Header Changes***: Simpler structure to *speed-up processing* at routers and to *support QoS* (which was not supported in IPv4)
- ***Authentication and Privacy***: Authentication, data integrity and confidentiality
- ***Security:*** IPSec is mandatory for IPv6

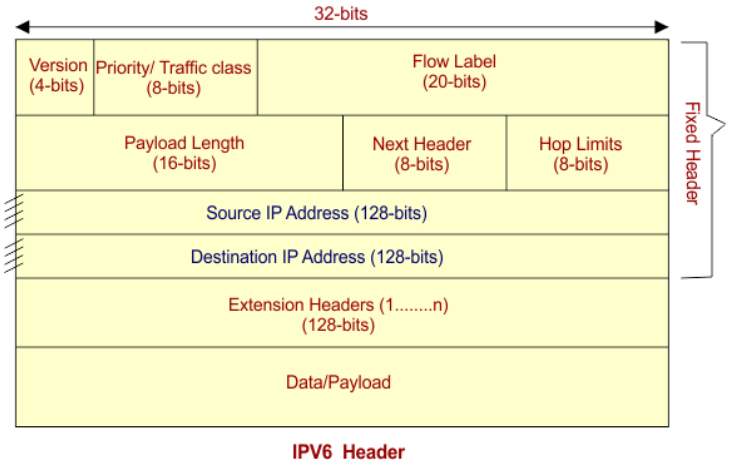
IPv6 header

IPv6 header has a ***fixed dimension (40 bytes):*** it speed-up the processing steps.
In order to do this, *some fields of IPv4 header were removed*. For example we have no more fragmentation support (it reduce forwarding time at routers) and checksum (it reduce processing time at routers).
However:

- Fragmentation is supported with the extended headers
- Checksum mandatory for TCP and UDP (it was not on UDP over IPv4)

New field have been introduced to support QoS (traffic class and flow label).

Fields:



IPV6 Header

- ***Version*** (4 bits) = 6
- ***Priority*** (8 bits)
  Class of service of the datagram: Determines its priority with respect to other datagrams
- ***Flow label*** *(20 bits)*
  Identifies datagrams in the same flow.
  Tells the router how to manage datagrams belonging to different flows. The concept of flow is not well defined
- **Payload Length (16 bits)**: Payload size in bytes
  It excludes the fixed header size, but includes the extended headers sizes
- **Next Header (8 bits):** Identifies the next extended header (if any).
- ***Hop Limit*** *(8 bits)*: Maximum number of hops allowed for a datagram, is decremented at each router before forwarding (similar al TTL)
- **Source address** (128 bit)
- **Destination address** (128 bit)

Extended headers mechanism

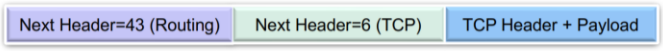
The option field, which existed in the IPv4 packet format, has been removed in order to increase processing speed. In IPv6, optional information can be included using the next header mechanism. Instead of having a separate field for options, optional information is now sent in the payload. However, in order to identify and process these optional information, specific mechanisms are used.

This is achieved using extended headers, specifically they are used by creating a chain of headers, in each of them, the next one is indicated.
Extended headers follow the fixed header (the standard header of IPv6) and precede the transport header. So the IPv6 header is the first header in the datagram and the transport protocol header is the last header in the datagram but between these two headers there are maybe other headers used to include additional information.

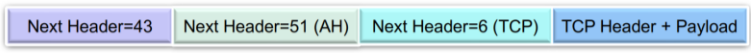
Remember that the presence or absence of optional headers is indicated by the "**next header**" field within the IPv6 datagram header.

1. No Extended Header

2. With a Routing Header



3. With a Routing Header and Authentication Header



There are different type of EH, the most common are:

- Number 0, **Hop-by-Hop Options**
  Options that must be processed by all the routers along the path (including source and destination)
- Number 60, **Destinations options header**
  Options that must be processed only by the destination node

Hop-by-Hop Option and *Destination Option* Header Format

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	*Next Header*								*Hdr Ext Len*								*Options and Padding*															
4	32	*Options and Padding*																															
8	64	Optional: more *Options and Padding* ...																															
12	96																																

- Number 43, **Routing Header**
  Specifies a set of nodes that must be traversed by the datagram to reach the destination. Loose source routing (not all nodes need to be listed)

Routing Header Format

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	*Next Header*								*Hdr Ext Len*								*Routing Type*								*Segments Left*							
4	32	*Type-specific Data*																															
8	64	Optional: more *Type-specific Data* ...																															
12	96																																

- Number 44, **Fragment header**
  As we said, the fragmentation isn't managed by the IPv6 and this protocol assumes that each link is able to transport a payload of *1280 bytes* MTU. But this is not the case of LLN that use IEEE 802.15.4 which has a maximum MAC frame size of *127 bytes*.
  For this reason we need a ***Path Maximum Transmission Unit Discovery (PMTU)*** which is used to discover the minimum MTU along the path, by sending periodic ICMP messages and then caching the value for each destination. Once the MTU of the path is discovered, if the size of the message to be sent exceeds this MTU value, we can utilize the fragment header to implement fragmentation.
  The options are very similar to IPv4 options (**Offset**, More fragment or not (**M**) and **Identification**)

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	*Next Header*								*Reserved*								*Fragment Offset*													*Res*		*M*
4	32	*Identification*																															

All zeros

Please note that for each Extension Header that does not have a standard length, the length is indicated.

IPv6 addresses

Ipv6 addresses have 128 bit. There are different type of IPv6 addresses:

- **Unicast Address**
  - Uniquely identifies a single network interface
  - A datagram sent to a unicast address is delivered to that specific interface
- **Anycast Address**
  - Identifies a group of interfaces, usually belonging to different nodes
  - A packet sent to an anycast address is delivered to just one of the member interfaces, typically the nearest host, according to the routing protocol's definition of distance
- **Multicast Address**
  - Identifies a group of hosts
  - A packet that is sent to a multicast address is delivered to all the interfaces that have joined the corresponding multicast group

BROADCAST ADDRESS DOESN'T EXISTS! But we can replicate the broadcast packets.

Classification of Unicast Addresses

Different from IPv4, in IPv6, we no longer have only one address for one interface. Instead, we have:

- **Link Local Address**, it's not routable. It's used to communicate with other host inside the same subnet for auto configuration, neighbor discovery, etc. They have the prefix ***fe80/10.***
- **Global Unicast Address**, similar to public IPv4. Uniquely identifies the host within the Internet.
- **Unique Local Address (ULA)** Intended for local communication. Must be used by devices that must not be reachable from outside the local link (comparable to IPv4 private addresses).

Address representation

The address is represent with hex ciphers, every 4 ciphers there are 2 point. We have 8 "hextets" (sixteen-bit aggregation). We can compress the address using the following rules:

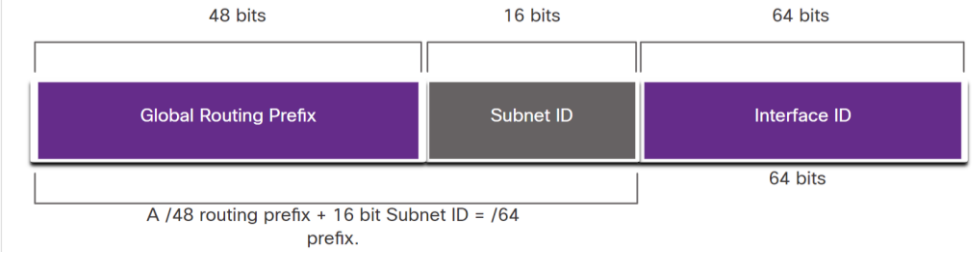
- 0 on the head of 16 bit group are omitted (es: 00A2 → A2)
- If there are groups of 0, the ":" are omitted too. But this notation can be used only once inside the address, the best practice is to use it with the longest string of 0s.

For example:

Preferred	2001 : 0db8 : 0000 : 1111 : 0000 : 0000 : 0000 : 0200
Compressed/spaces	2001 : db8 : 0 : 1111 : : 200
Compressed	2001:db8:0:1111::200

Unicast Addresses

As for IPv4 there is a part of the address that identifies the network and another one that identifies the host. These parts are called ***Prefix*** and ***Interface ID***. The prefix is generally formed by 64 bits: it's usually assigned by the ISP (the first 48 bits). Between them and the interface ID, the subnet ID is present.



Link-local address

Used for addressing on a single link for purposes such as automatic address configuration, neighbor discovery, or when no routers are present.


```
|   10     |
|  bits    |          54 bits          |          64 bits           |
+----------+---------------------------+----------------------------+
|1111111010|            0              |       interface ID         |
+----------+---------------------------+----------------------------+
```

These addresses always start with the prefix **`fe80`** (10 bits), followed by a section consisting of consecutive zeros. The Interface ID (*64 bit*) is assigned by the node itself. Without the presence of DHCP server the Interface ID can be generated:

- Originally derived from the *MAC address of the NIC (EUI-64),* but then we'll have undesirable address changes when NIC are replaced and security and privacy issues.
- Now MAC-based method is replaced by an *hash-based method*

Anycast addresses

A packet sent to an anycast address is delivered to just one of the member interfaces.
Syntactically indistinguishable from a unicast address.
Format: Subnet prefix (n bits) followed by 128-n bits all equal to zero

```
|                        n bits                 |   128-n bits    |
+-----------------------------------------------+-----------------+
|                    subnet prefix              | 00000000000000  |
+-----------------------------------------------+-----------------+
```

Multicast addresses

Identifies a group of interfaces:

```
|   8    |  4 |  4 |                   112 bits                    |
+------ -+----+----+-----------------------------------------------+
|11111111|flgs|scop|                   group ID                    |
+--------+----+----+-----------------------------------------------+
```

- ***Well-known multicast addresses:***
  - ***All-nodes multicast group:*** **`FF02::1`**, it's the equivalent of the broadcast address we had in IPv4. When a packet is sent to this address, it's received and processed by ***all the IPv6 enabled interfaces*** on that subnet.
  - **All-routers multicast group: `FF02::2`,** , when a packet is sent to this address, it's received and processed by ***all the IPv6 enabled router*** on that subnet

- **Solicited node multicast addresses: `FF02:0:0:0:0:1:FFXX:XXXX`**: formed by taking the low-order 24 bits of the node (unicast or anycast) and appending those bits to the prefix.

Neighbor Discovery Protocol (NDP)

The Neighbor Discovery Protocol (NDP) is a protocol used in IPv6 to discover and maintain information about neighboring entities within a network. It is responsible for managing various network functionalities, including address discovery, address resolution, node availability determination, and network parameter configuration.

*Provides a set of key **autoconfiguration features***

- *Presence of neighbors on a link*
- *Discovery of a router that provides network prefix*
- *Discovery of link-layer additional addresses*
- *Maintenance of reachability*

Services provided by NDP:

- **Router discovery:** Hosts can locate routers residing on attached links
- **Prefix discovery:** Hosts can discover address prefixes that are on-link for attached links
- **Parameter discovery**: like MTU

- **Address autoconfiguration**
- **Address resolution:** to map IP address to link layer address (like ARP in IPv4)
- **Next-hop determination**
- **Neighbor Unreachability Detection (NUD)**, to determine if a neighbor is still reachable or not
- **Duplicate Address Detection (DAD)**, to check if an autogenerated address is already used
- **Packet redirection:** process allowing a node to find a better next-hop route for a certain destination

In order to implement these services, 5 different **ICMPv6 packet types** have been introduced:

- **Router Solicitation**, **RS** : is used by host to locate link in an attached network
- **Router Advertisement**, **RA**: used by router to advertise its presence (it can be send periodically or in response to RS message). It can carry also parameters like prefix, DNS, default gateway address.
- **Neighbor Solicitation**, **NS**: use in NUD and DAD and for address resolution
- **Neighbor Advertisement, NA**
- **Redirect**

IPv6 autoconfiguration

IPv6 Auto-configuration is a feature of IPv6 that allows hosts to automatically assign IP addresses and configure other network parameters without manual intervention or the need for a DHCP server.
Autoconfiguration is extremely important in IoT environments since we have a very large number of nodes and it's not easy configure the IPv6 for all the nodes. IPv6 autoconfiguration allows the node to generate Link Local address and Global address.
Stateless and stateful autoconfiguration exist.

The autoconfiguration process for generating addresses consists of several steps:

1. Creation of a Link-Local address which is used only locally for autoconfiguration purposes
2. Determination of what should be configured
3. Determination of whether using a stateless or stateful procedure for address autoconfiguration

Generation of the Link-Local address

As we have already said, this allows for local communication between all nodes that are connected to the same link.

It starts with `FE80` (10 bits), followed by 54 bits equal to 0. The second part (64 bit), is generated starting from the MAC address and using a well known address hash algorithm.

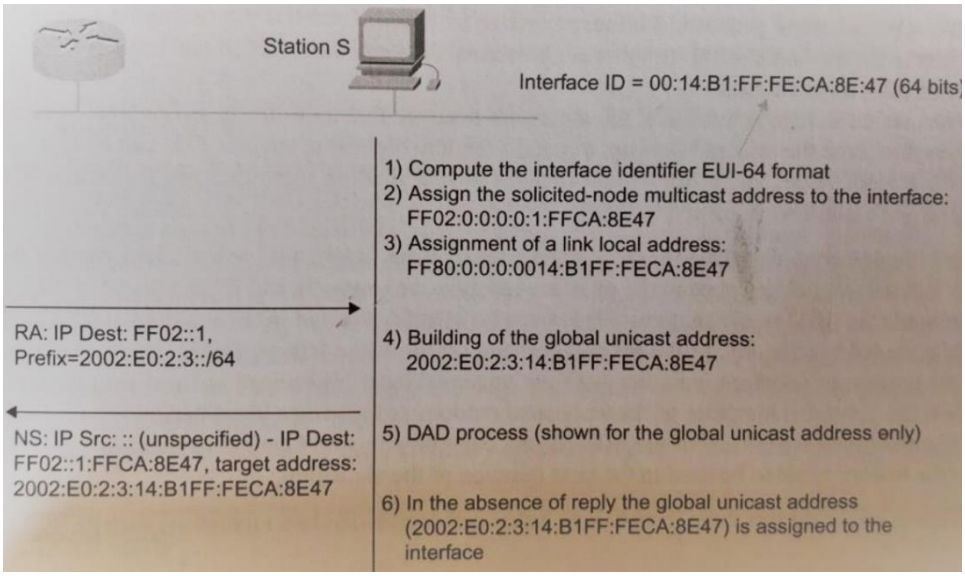
After generating its possible link-local address, the node needs to ensure that it is unique within the local network. The Duplicate Address Detection (DAD) process will be invoked (the same we will see for the Global Unicast address).

Generation of the Global Unicast address

The **network prefix needs to be obtained from the router**. The router periodically sends out Router Advertisement (RA) messages, which contain the necessary information. Alternatively, the node can send a Router Solicitation (RS) to request the information. Upon receiving the RA, it will contain the network prefix information.

From this point, autoconfiguration can occur in multiple ways:

- **SLAAC** (Stateless Address AutoConfiguration):
  RA: *"I have everything you need including the prefix, prefix length, and default gateway address."*
  - Prefix: is contained into the RA messages
  - Interface ID: EUI-64 process or generated using hash function



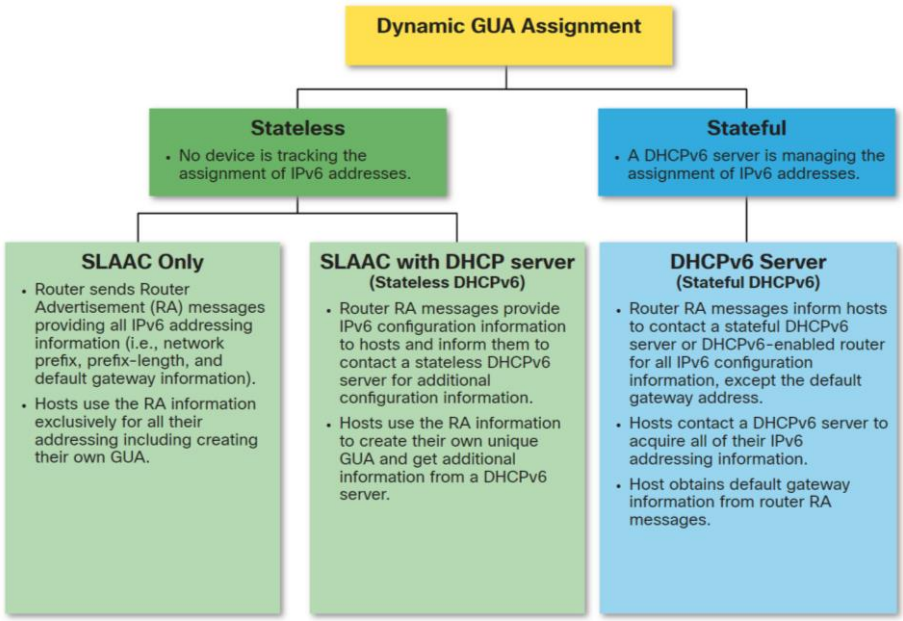
Note that the host can wait for RA after step 3 or send an RS in order to receive the RA.
Note also that in the DAD process, a **Solicited node multicast addresses** is used.
The IP address of the sender is set to "unspecified" because the node cannot use the global address it just generated. This is because it is not yet certain if the address is available or in use by another node.

- **SLAAC with stateless DHCPv6**

RA: "*Here is my information but you need to get other information such as DNS addresses from a stateless DHCPv6 server.*"
As in the previous case, the RA message provides the necessary information to configure the IP address.
The only difference with SLAAC only is that a DHCPv6 solicitation message is sent by the host to obtain information like DNS address, etc.
***Note**: stateless DHCPv6 server DOESN'T ALLOCATE addresses!*

- **Stateful DHCPv6**

RA: *"I can give you your default gateway address. You need to ask a stateful DHCPv6 server for all your other information."*
The host obtains from RA the default gateway and the IPv6 of the DHCPv6 server: it must contact it to obtain the GUA, DNS server address and other useful information.



In IPv6, where there is no broadcast address for sending DHCP requests, a requesting node can contact the DHCP server by sending a DHCPv6 request to a special multicast address represented by FF02::1:2.

8- 6LoWPAN Adaptation

The 6LoWPAN (IPv6 over Low-Power Wireless PAN) Adaptation Layer serves as an intermediary between the IPv6 layer and the MAC layer, enabling the adaptation of the IPv6 protocol to the characteristics of the low-layer used in LLNs.

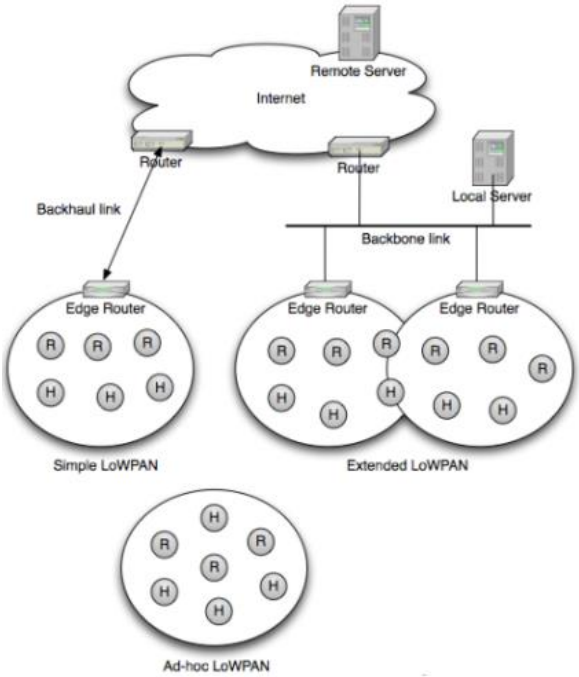
Standard IPv6 is not well suited for LLNs due to various challenges. One such challenge is the **payload size**, as LLN devices often have limited memory and processing capabilities. The 6LoWPAN Adaptation Layer addresses this issue by **compressing IPv6 headers** and reducing the overall packet size, making it more efficient for LLN communication. Another challenge is the **Neighbor Discovery process**, which is not optimized for LLNs. Neighbor Discovery is a mechanism used in IPv6 networks to discover and manage neighboring devices. In LLNs, where devices may have intermittent connectivity or limited power, a more efficient neighbor discovery process is needed. The 6LoWPAN Adaptation Layer helps optimize this process by introducing mechanisms such as address resolution and route optimization.

In conclusion, the 6LoWPAN Adaptation Layer offers a range of functionalities, including: packet fragmentation and reassembly, header compression and optimizations for Neighbor Discovery.

Context

We are in a low-power and lossy network consisting of IEEE 802.15.4 devices, it implicates that:

- We have small packet sizes (127 bytes)
- 16 bit for short and 64 bit for extended MAC address
- Low data rate
- Constrained devices
- Unreliable wireless link
- Etc.



Reference architecture

6LowPANs nodes:

- **Hosts**
- **Routers** (6LR): used to forward the information produced by hosts towards the edge routers
- **Edge Routers** (6LBR): is a special router that is connected to the rest of the Internet

Topologies:

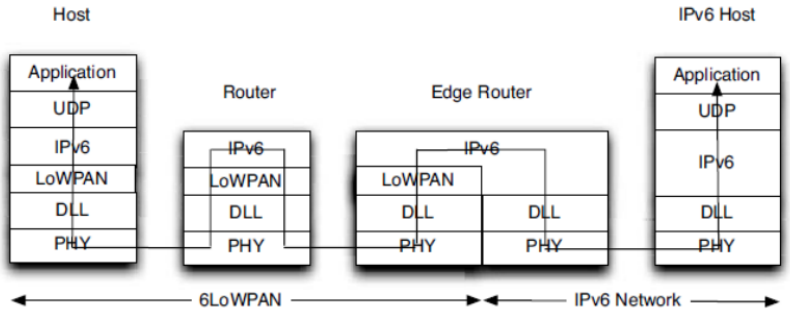
- ***Simple***: Single Edge Router
- ***Extended***: Multiple Edge Routers with common Backbone Link
- ***Ad hoc:*** No Edge Router

Network architecture

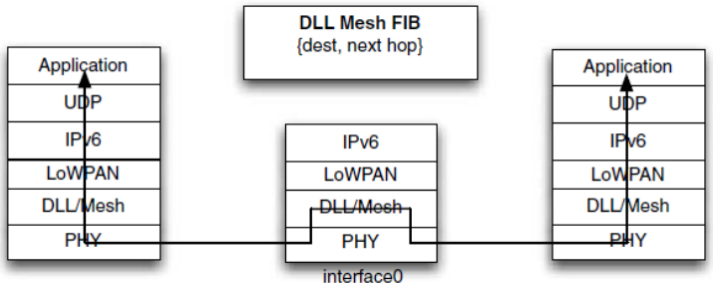
We analyze the network architecture focusing on one node.
Considering each node: ***all the node inside the communication range, forms the IP link of that specific node***, so there's not a single broadcast domain but for each node there is a broadcast domain, different from the other → **Broadcast domain = Transmission range**.

Forwarding phase:

- **Route over**: forwarding is performed at IP layer. It's like in the traditional internet.



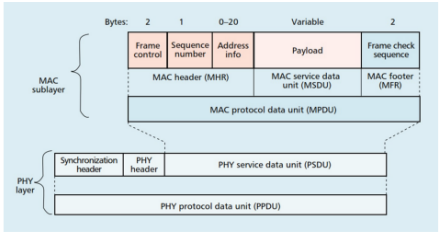
- **Mesh under**: forwarding is performed at data-link layer. The path traversed by the packet is transparent to the adaptation layer.



Fragmentation and Header compression

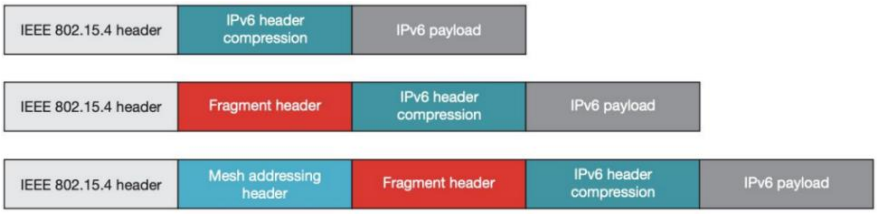
Why is it necessary?

- IPv6 mandates an MTU of 1280 bytes
- IEEE 802.15.4 links have an MTU of 127 bytes
  - Maximum frame overhead: 25 bytes
    - ⇨ Frame control (2 bytes)
    - ⇨ Sequence Number (1 byte)
    - ⇨ Addresses (up to 20 bytes)
    - ⇨ Frame Check Sequence (2 bytes)
  - MAC security header: 21 bytes
    - ⇨ AES-CCM-128 (21 bytes)
    - ⇨ AES-CCM-64 (13 bytes)
    - ⇨ AES-CCM-32 (9 bytes)
  - Payload (for IPv6 datagram): 81 bytes
    - ⇨ IPv6 header (40 bytes)
    - ⇨ UDP header (8 bytes)
  - Payload (for Application Data): 33 bytes



There are 3 different type of headers:

- IPv6 header compression
- Fragment header
- Mesh addressing header



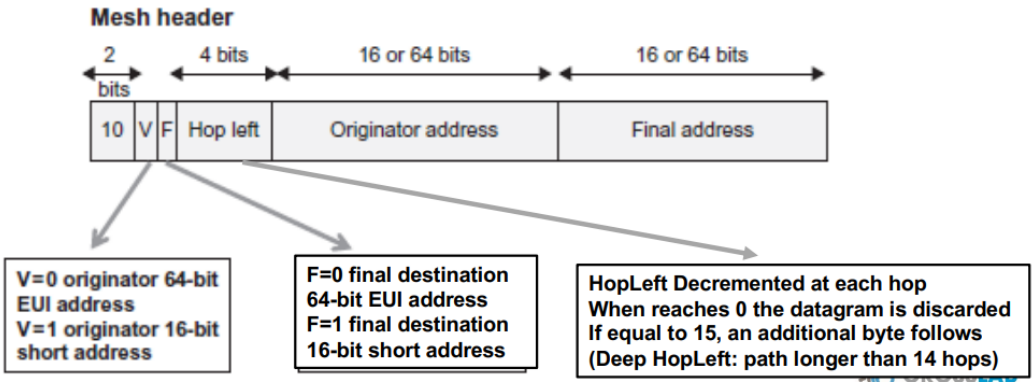
They must be used in the following order if there all present: *mesh – fragment – ipv6*

In each of them the first byte is the **dispatch byte**: in this byte, the first two bits indicates the type of packet

00	**Not a 6LoWPAN frame**
01	IPv6 addressing header
10	Mesh header
11	Fragmentation header (6 lower bits are 100xxx)

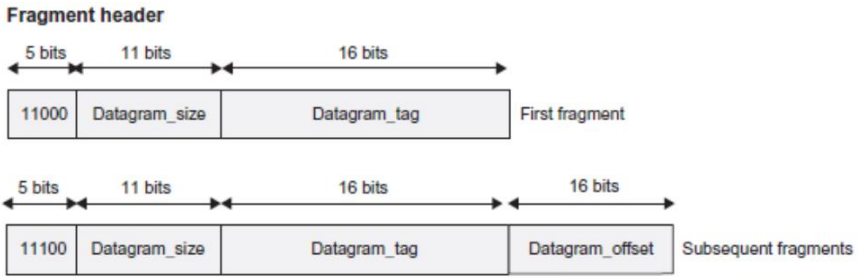
Headers: Mesh addressing header

Is used with a mesh-under routing approach, which implies that packets have to traverse the network utilizing multi-hop communication between nodes.



Headers: fragmentation header

Fragmentation is performed whenever the IPv6 exceeds the IEEE 802.15.4 MTU, so the datagram broken into several fragments.



Characteristics:

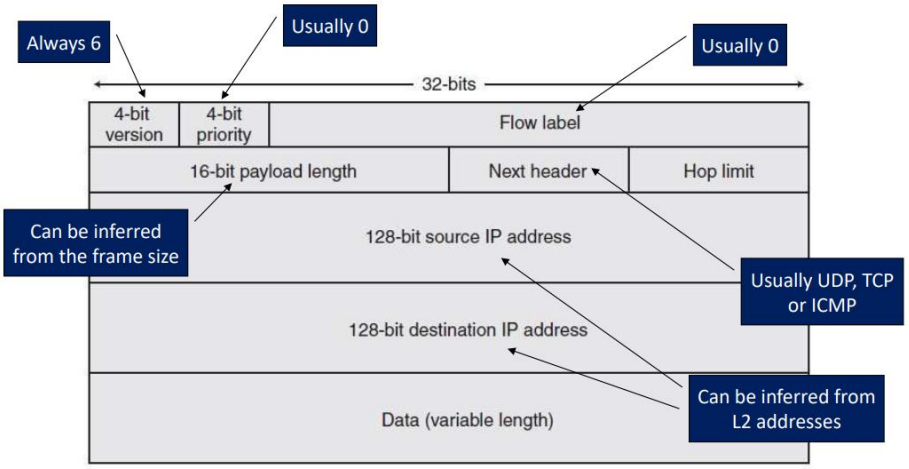
- The **datagam size** represents the size of the original IPv6 datagram, expressed in units of 8 bytes
- The **first fragment** is different from the others:
  - The first 5 bits are 11100 instead of 11000
  - It doesn't contain the Datagram_offset
- In order to recognize all the fragment originated by the same original datagram, the **datagram tag** must be identical for all link fragments
- The **datagram offset** indicates the offset from the beginning of the payload datagram (in 8-byte units)

A reassembly timer is started when receiving the first link fragment: upon the expiration of the reassembly timer, if not all link fragments have been received, all fragments must be discarded.

Headers: IPv6 header compression

Why compress the header? 6LoWPAN is most efficient when all IPv6 packets can be made to fit into a single IEEE 802.15.4 packet. Even when fragmentation is not needed, compression is still convenient as it reduces energy costs and bandwidth requirements.

Moreover, there are a lot of redundant information into the IPv6 header. For example, IPv6 address can be inferred from MAC address since they are generated starting from it.



Different types header compression can be used:

- **Stateful header compression**
  Useful for flow-based information, but this is not the case for 6LoWPAN, which is characterized by short-lived flows with a limited number of bytes.
  Stateful compression requires network nodes to maintain a compression state to keep track of the information needed to properly decompress packet headers. This compression state can be maintained in a table or database within the nodes. When a packet is received, the node leverages the state information to decompress the IPv6 header.

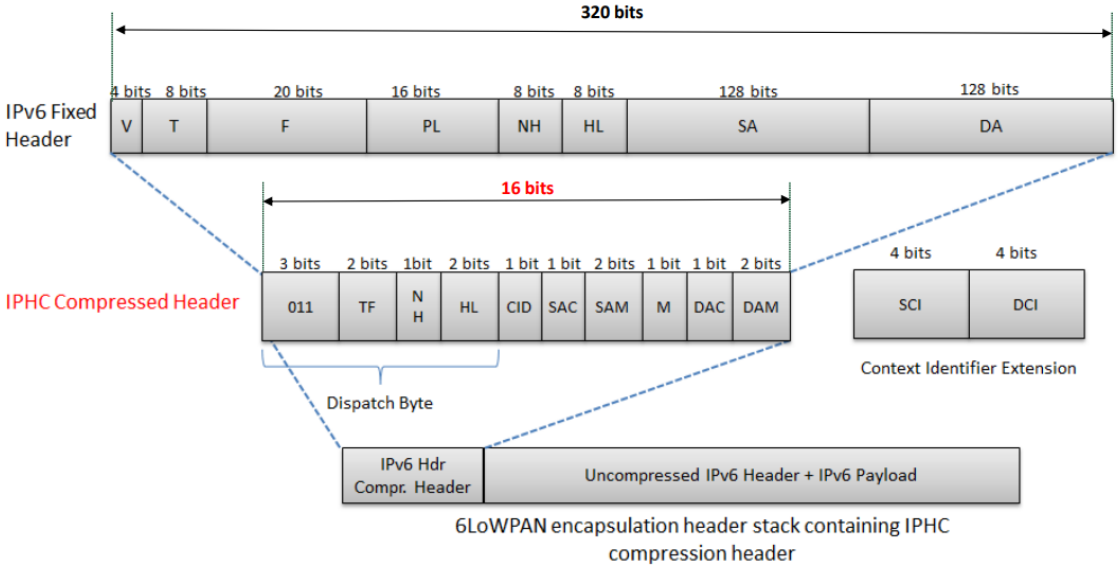
- ***Stateless header compression***
  Consists in avoiding information redundancy across layers
  - Efficient for link-local addresses (ND, DHCP) → if the traffic is local, the stateless header compression is very well suitable
  - Limited effect on packets destined to outside the LowPAN → if it's not local, not good

  Stateless compression does not require the compression state to be maintained in nodes. Instead, it utilizes predefined compression schemes based on known rules and patterns.

- **Context-based header compression**
  Exploits some shared context between source and destination node

stateful compression VS stateless compression

- **Stateful compression** can provide higher compression efficiency but requires additional processing to maintain and manage the compression state.
- **Stateless compression** is simpler to implement and requires less processing resources but may not achieve the same level of compression efficiency as stateful compression.



6LoWPAN encapsulation header stack containing IPHC compression header

The maximum result is to reduce the IPv6 header from 320 bits (40 bytes) to 16 bits (2 bytes), but there are a lot of intermediate situations.

Fields:

- Version Field (V) is always elided
- Payload Size (PL) inferred from the 802.15.4 frame (also the mac frame as a field containing of the mac frame, if you know the size of all the control fields, you can inferred the Datagram and its payload)
- Traffic Class (TC) + Flow Label (FL) → ***TF (2 bits)***
  - 00: Both TC and FL are carried in line (subsequent 32 bits)
  - 01: TC field compressed (2-bit ECN), FL in line
  - 10: TC uncompressed and in line, FL elided
  - 11: Both TC and FL compressed

- Next Header (NH) → ***NH (1 bit)***
  - 0: original NH field (8 bit) in line
  - 1: original NH field elided
- Hop Limit (HL) à ***HL (2 bit)***
  - 00: original HL field in line
  - 01: original HL field elided, hop limit = 1
  - 10: original HL field elided, hop limit = 64
  - 11: original HL field elided, hop limit = 255
- ***Context Identifier extension (CID, 1 bit)***
  - 0: No additional context information
  - 1: 1-byte context information immediately after the DAM field
- ***Source Address Compression (SAC, 1 bit)***
  - 0: Stateless address compression
  - 1: Stateful address compression, based on context
- ***Source Address Mode (SAM, 2 bits)***
  - SAC=0: Stateless address compression
    - 00: full 128-bit source address in line
    - 01: first 64 bits elided, remaining 64 bits carried in line
    - 10: first 112 bits elided, remaining 16 bits carried in line
    - 11: address fully elided; the first 64 bits are the local prefix, the remaining 64 bits are inferred from the IEEE 802.15.4 frame
  - SAC=1: Stateful address compression, based on context
    - 00: the address is the unspecified address (::)
    - 01: the 64-bit prefix is derived from context information, the remaining 64 bits are inline (64 bits)
    - 10: the 64-bit prefix is derived from context information, the remaining 16 bits are inline (16 bits)
    - 11: the address is derived from context information and, potentially, the link-layer frame (0 bits)
- ***Multicast compression (M, 1 bit)***
  - 0: the destination address is not a multicast address
  - 1: the destination address is a multicast address
- ***Destination Address Compression (DAC, 1 bit)***
  - 0: the compression of the destination address is stateless
  - 1: the compression of the destination address is stateful ð based on context
- Destination Address Mode (DAM, 2 bits)
  - M=0, DAC=0 (unicast address, stateless compression)
    - 00: the full 128-bit address is in line (128 bits)
    - 01: The first 64-bits of the address are elided. The value of those bits is the link-local prefix padded with zeros. The remaining 64 bits are carried in-line (64 bits)
    - 10: The first 112 bits of the address are elided. The value of the first 64 bits is the link-local prefix padded with zeros. The following 64 bits are 0000:00FF: FE00:XXXX, where XXXX are the 16 bits carried in-line (16 bits)
    - 11: The address is fully elided. The first 64 bits of the address are the link-local prefix padded with zeros. The remaining 64 bits are computed from the encapsulating 802.15.4 frame header (0 bits)
  - M=0, DAC=1 (unicast address, stateful compression)
    - 00: Reserved
    - 01: The address is derived using context information and the 64 bits carried in-line (64 bits) – Bits covered by context information are always used. Any IID bits not covered by context information are taken directly from the corresponding bits carried in-line. Any remaining bits are zero
    - 10: The address is derived using context information and the 16 bits carried in-line (16 bits) – Bits covered by context information are always used. Any IID bits not covered by context information are taken directly from their corresponding bits in the 16-bit to IID mapping given

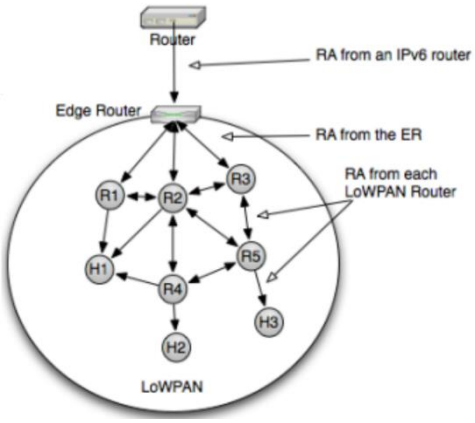
by 0000:00FF:FE00:XXXX, where XXXX are the 16 bits carried in- line. Any remaining bits are zero.
        - 11: The address is fully elided and is derived using context information and the encapsulating header (e.g. 802.15.4 or IPv6 destination address, 0 bits) – Bits covered by context information are always used. Any IID bits not covered by context information are computed from the encapsulating header as specified. Any remaining bits are zero.
    - M=1, DAC=0 (multicast address, stateless compr.)
        - 00: the full 128-bit address is in line (128 bits)
        - 01: The address takes the form – FFXX::00XX:XXXX:XXXX (48 bits)
        - 10: The address takes the form – FFXX::00XX:XXXX (32 bits)
        - 11: The address takes the form – FF02::00XX (8 bits)
    - M=1, DAC=1 (multicast address, stateful compr.)
        - 00: This format is designed to match Unicast-Prefix-based IPv6 Multicast Addresses (48 bits)
        - 01: Reserved
        - 10: Reserved
        - 11: Reserved

Neighbor Discovery & Registration

The standard IPv6 ND does not work well for LowPANs because of multiple overlapping broadcast domains and the cost of multicast communication within the LoWPAN.

6LoWPAN Neighbor Discovery introduces simple optimizations to:

- IPv6 Neighbor Discovery
- IPv6 addressing mechanisms
- Duplicate Address Detection (DAD) for LowPANs



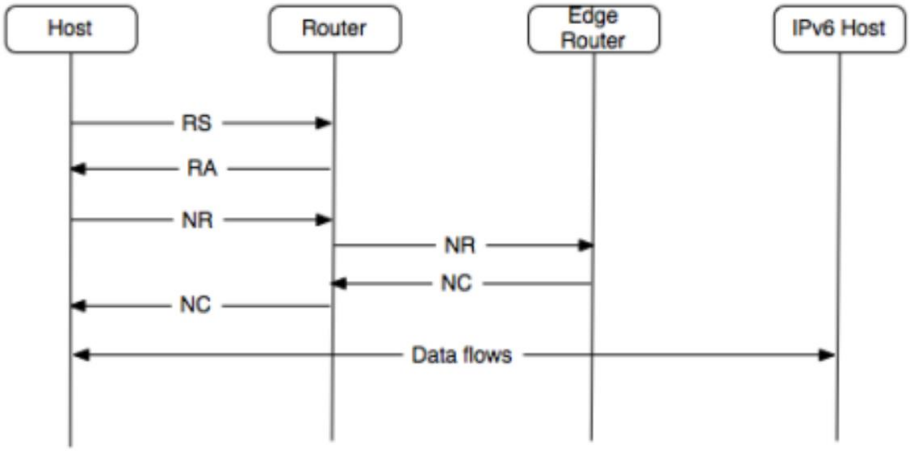
It introduces also the role of **edge routers**, which are responsible for maintaining centralized state and managing the registration process.

Edge routers are very important, especially in DAD, but it can happen that one host cannot communicate directly with the edge routers, in that case we need intermediate routers and use a multi-hop communication approach.

DAD in LowPANs

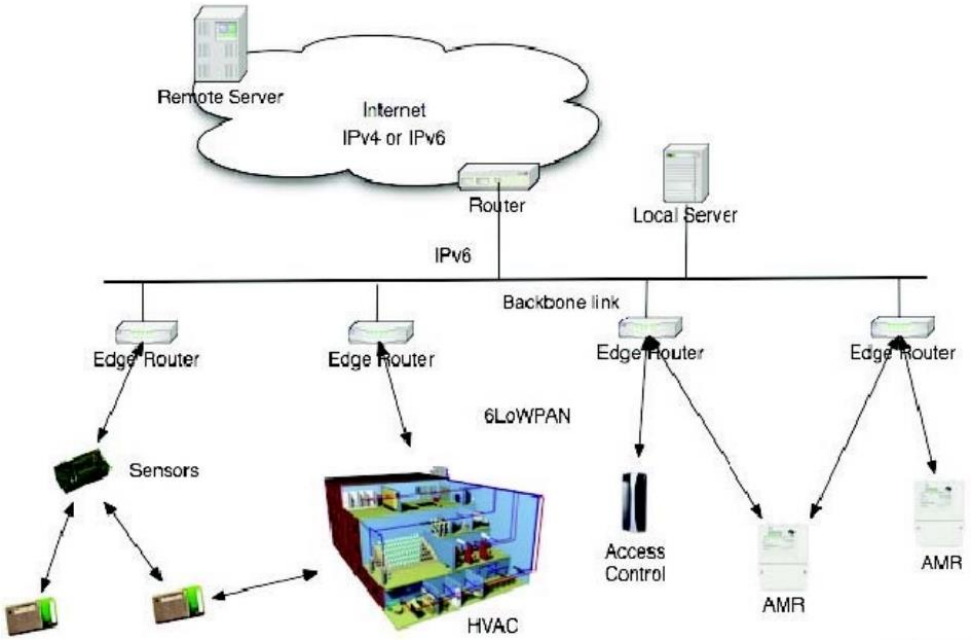
Duplicate Address Detection (DAD) is a process used in Neighbor Discovery to ensure that an IPv6 address is unique within a network.

RAs are also used to automatically disseminate router information across multiple hops, but it's difficult to implement like in the standard IPv6, so a centralized state at the Edge Router(s) is used.
ERs keep track of nodes in their network, and perform DAD, address resolution and short address generation on behalf of nodes.



As in IPv6, the host send an **RS** message and a router (note it can be a router or an edge router) replies with an **RA** containing some information useful to the host.
Then the node send a **Node Registration** (**NR**) message: if it can't communicate directly with the edge router, the NR is sensed by other routers and forwarded until it's received by the edge router.
Edge router registers the host (the address) inside its table and replies with **Node Confirmation** (**NC**) message.
After that, data can flows.

A 6LowPAN example:



- **Edge Router**

  Since in this example we have different LowPAN, we have an edge router for each LowPAN.
  Has both a 6LoWPAN interface and a regular IPv6 interface and it is responsible for:
  - *Diffusing RAs to the 6LoWPAN network*
  - *Address Resolution* of a node
  - *Maintaining a whiteboard of IPv6 addresses*
  - *Duplicate Address Detection* for the addresses it defends
  - Listening for RS and Node Registration messages
  - *Can also generate IPv6 addresses using IEEE 802.15.4 short addresses on behalf of the registering nodes*

  It forward packets:
  - From one 6lowPAN to
    - another 6LowPAN
    - backbone network: 6LoWPAN adaptation layer is stripped off, the header is uncompressed
  - From the backbone network to one 6LowPAN: adds 6LoWPAN specific adaptation layer and possibly 6LoWPAN IPv6 header compression mechanism and then forwards them to the 6LoWPAN

- **LowPAN router**

  Forwards data packets across links and relays control packets between the Edge router and the 6LoWPAN nodes and responds to the RS messages from hosts on the same link with Ras.

- **LowPAN Host (Node)**

  After bootstrap, forms an optimistic IPv6 link local address from the EUI64 bit MAC address and then registers with the edge router

9- Routing protocols

How to perform multi-hop data delivery? To implement multi-hop communication we need a Routing Protocol, the one we will see is called RPL.
While IPv6 (+ 6LoWPAN) handles auto-configuration and forwarding of IPv6 datagrams, the routing protocol plays a crucial role in determining the optimal path for datagrams to traverse towards their intended destination.

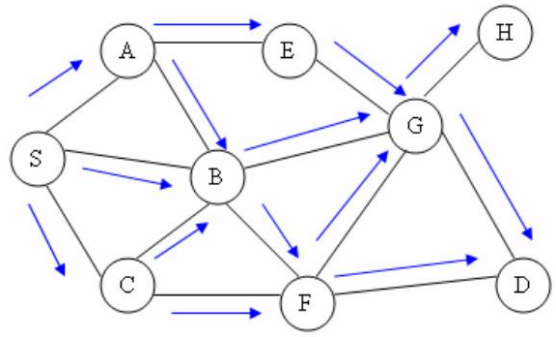
Taxonomy of routing schemes

- **Flat routing**
  No hierarchy, no special role and the routing protocols doesn't exploit any organization.
  Examples are flooding, gossiping, AODV.
- **Hierarchical routing**
  there's always an organization of the node in the network, which is very suitable for data aggregation and routing protocols can take advantage of this hierarchical organization.
  The organization can be different like:
  - Cluster-based, as in LEACH
  - Tree-based organization as MinTRoute
- **Location-based routing**
  Node location is exploited: the location information can be relative (refer to the position respected to anchor nodes) or absolute.

How to select the parent node?

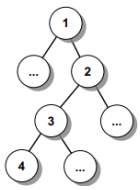
Flat routing: flooding

"Flooding" is an example of a float routing technique. In this approach, packets are transmitted to all nodes within the communication range without following a predetermined path. When a node receives a message, it retransmits a copy of it to either all or a subset of its neighboring nodes.
It implies that the final destination receive different copy of the same message from different paths.



Very expensive approach because there is a very huge number of exchanged message.
Since each node retransmit the packet after receiving it, interferences may occurs: it implies retransmission, it means energy and bandwidth consumption.
This approach is unsuitable for LLNs (bandwidth and energy consumption is a problem).

Hierarchical routing: tree-based



Sensor nodes *self-organize in a tree*, each node is associated with one parent node.

Forwarding of messages occurs in two step:

1. Sensor nodes send data to his parent node
2. Parent nodes forward data towards the sink node

There are different approaches to forming the topology, such as "Broadcast and Reverse Path" and "Shortest Path".

Broadcast and reverse path

- The *sink periodically broadcasts beacons*, only to *announce* the *presence* of the node.
- The other nodes re-broadcast beacon messages (possibly after a random delay to avoid collisions)
- **Nodes use the source address of the first received beacon to select the parent node**

Note: in the initial phase flooding is used, because there is no topology! But once the tree topology has been formed, then the forwarding of messages will be based on the tree topology.

Limits? 1) The ***selected path may not be the shortest path*** in terms of hops; 2) ***Link quality is not considered***; 3) Lead to trees with many long and unreliable paths

Shortest path

It's a conventional Distance Vector (DV) protocol.
A node is a neighbor if a packet is received from it. Each node selects its parent node based on the minimum hop count.
During the propagation of messages to construct the tree, these messages contain a hop count counter that increases each time a node retransmits the message.

Limits? ***Link quality is not considered***: hop count is not an adequate metric in lossy links, since the expected # of transmissions would be more appropriate

Possible Variant: ***Shortest Path with link-quality Threshold*** (SP(t)): a node is a neighbor if the link quality exceed a pre-defined threshold t (e.g., 70% or 40%)

RPL protocol

Routing Requirements depends on the specific application domain and for each domain, on the specific application: Industrial Settings, Smart Cities, Home Automation, Building Automation…
Before RPL, no existing IETF routing protocol was suitable for LLNs and so they decided to design a new protocol from scratch.

RPL stands for **Routing Protocol for LLNs**, is a Distance Vector and Source Routing protocol used in the following *communication paradigm*:

- Multipoint-to-point (collection or sensor networks, sensor nodes to sink)
- Point-to-multipoint (when sink node transmitted message to all nodes)
- Point-to-point

RPL was specifically designed for Low-Power and Lossy Networks (LLNs), but not specifically for IEEE 802.15.4, as it is just one of the possible low-level communication technologies.
RPL is a Layer-3 routing protocol!

RPL Design principles

Based on **Destination-Oriented Directed Acyclic Graph (DODAG):** is a tree-like structure, but nodes can associate to more than one parent (in tree topology a node could only have one parent).
Once the DODAG has been constructed, the goal of the RPL protocol is to compute the best paths to reach various nodes within the network.

Type of nodes

- *Low Power and Lossy* ***Border Router*** *(LBR):*
  Root of a DODAG. The LBR acts as a gateway between LLN and Internet.
- ***Router***: A device that can generate and forward traffic. A router does not have the ability to create a new DAG, but associates to an existing one.
- ***Host****:* End device, can generate data traffic, but cannot forward traffic.

RPL topologies

- ***Hierarchical Topology****:* Nodes are ***forced to form DODAGs*** (based on parent-child relationship)
- ***Mesh Topology***: RPL *allows the forwarding through siblings*, when needed

RPL is a hierarchical routing protocol that also incorporates a mesh topology, as it allows for forwarding to both parent and sibling nodes, rather than solely relying on parent nodes for routing.

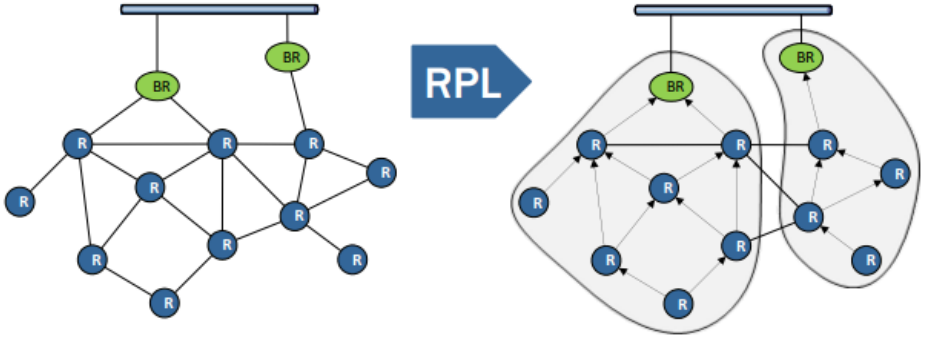
Main features

- **Auto-configuration**
  A LLN is a dynamic network where changes can occur rapidly. The network topology may change, nodes may fail

or become inactive due to power management. In such a dynamic environment, RPL is designed to handle these challenges by reconfiguring the network topology. RPL leverages IPv6 functionalities, such as neighbor discovery and dynamic configuration, to adapt to the changing conditions of the network.

- **Self-healing**
  Adaptation to network topology changes and node failures: this is very useful since links and nodes in LLNs are not stable and may vary frequently, so there is a mechanisms that choose more than one parent per node in the DAG to eliminate/decrease the risks of failure.
- **Loop avoidance and detection**
  RPL triggers recovery mechanisms when the loops occur
- **Independence and transparency**
  RPL *can operate over multiple link layers*: it's independent from data-link layer technology
- **Multiple edge routers**
  There may be multiple edge routers and so it is possible to construct *multiple DAGs, each with a root*. *A node may belong to multiple instances, and may act different roles in each instance*. This led to High availability and Load balancing.

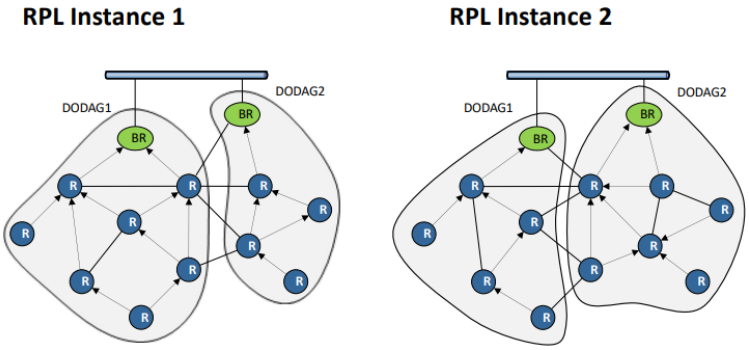
For example, in case of multiple sinks, one DODAG per sink is created:



RPL Instances

RPL Instances refer to separate instances or copies of the RPL routing protocol running in a network. Each RPL Instance operates independently and has its own routing table and configuration settings.

RPL Instance are identified by ***RPLInstanceID***, so we can have one or more DODAGs, identified by the same RPLInstanceID. All DODAGs within the same Instance share the same metrics/constraints.



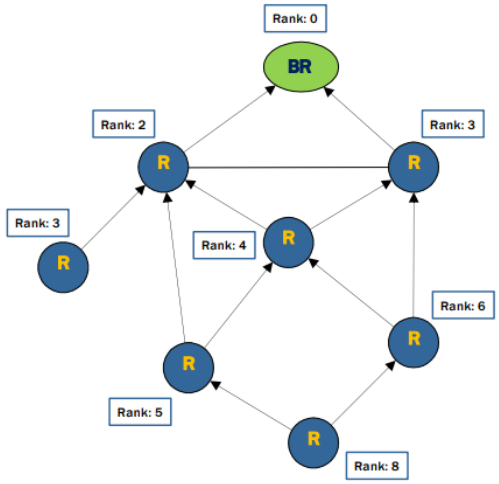
RPL nodes may belong to different instances, but to one and only one DODAG per RPL instance.

How to construct a DODAG?

We have to introduce some concepts:

Rank

Each *node* in the DODAG is assigned a ***rank***: an integer that *represents the location of a node within the DODAG*. The rank value strictly increases in the downstream direction: ***lower value → higher rank***.
(It is not a measure of the path cost towards the root).



The rank represents the desirability of a node to become a parent node in the routing topology. Nodes with lower ranks are preferred as parent nodes, as they are considered more suitable for relaying data packets.

The rank value is also used to avoid and detect routing loops.
Nodes in RPL select the next hop for forwarding packets by considering both **parent** nodes (with lower rank) and **sibling** nodes (with equal rank).
Nodes maintain a list of candidate parents and siblings used if the current parent loses its routing ability.

How ranks are generated by nodes? Objective Function (OF).

Objective function

Defines how nodes translate one or more metrics/constraints into ranks. Taking into consideration, for example, delay, bandwidth, reliability, node energy, node attributes, and hop count.
Once the objective function is defined, it is used to select and optimize routes based on path cost.

Objective Function defines:

- How to compute the rank
- How to *compute the path cost*
- How to *select parents* (when, who, how many)
- How to *advertise the path cost*

Node Metrics and Constraints could be:

- Node State and Attributes: CPU overload, 1 bit to signal congestion
- Node energy:
  - how the node is empowered, e.g. batteries or electrical grid (2 bits)
  - Estimated residual lifetime, calculated based on the remaining battery capacity
  - Other metrics (TBD)
- Link Throughput
- Link Latency
- Link reliability:
  - Link Quality Level (LQL: 0-7), 0 represents the minimum level and 7 the maximum level
  - Expected Transmission Count (ETX), often considered in order to measure the quality of a link

These metrics can be used to attract/avoid specific links for specific traffic types.
Several Objective Functions can be used, even concurrently: deployments greatly vary with different objectives, a *single network may support traffic with very different requirements, in terms of path quality*.

Example: network with two main applications

- Alarms: time sensitive paths, short delay, highly reliable paths
- Telemetry: non-time sensitive paths, minimum number of hops, not traversing battery-operated nodes to preserve their energy

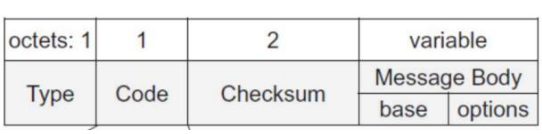
Which are the objective Functions used in practice?

- **OF0 - Objective Function Zero**: basic OF that does *not require any metric to be measured*.
  Only relies on default configurations. ***NONE of the previous constraints and metrics are used.***
  This is the simplest objective function you can have: minimize the hop count without considering link quality or energy consumption.
- **MRHOF - Minimum Rank with Hysteresis Objective Function**: computes a node's rank based on the additive metrics (often, ETX is used as routing metric).
  The MRHOF objective function assigns ranks to each node in the network based on various metrics such as link quality, node energy, and path cost.

RPL Control Messages

Based on new-type ICMPv6 messages:

- Message Header:
  - Type: set to *155* in case of RPL
  - *Code*: identifies the RPL control message:
    - Ox 00: DODAG Information Solicitation (DIS)
    - Ox 01: DODAG Information Object (DIO)
    - Ox 02: Destination Advertisement Object (DAO)
    - Ox 03: DAO Ack
  - Checksum
- Message Body:
  - Message Base
  - Options



RPL Control Messages

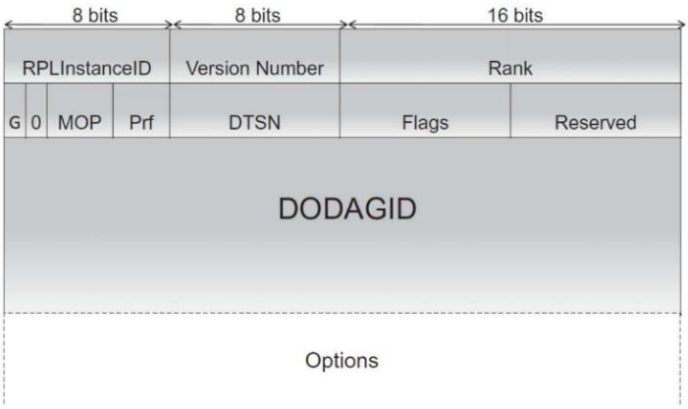
In the RPL protocol, there are several control messages used for network management and routing operations. These messages are exchanged between RPL nodes to establish and maintain the routing topology.
The control messages DIO, DIS, DAO, and DAO-ACK are commonly used control messages in the RPL protocol.

DIS - DODAG Information Solicitation

This message is used by a node to solicit a DIO message from an RPL node**.**
The DIS may also be used to check neighbor nodes in adjacent DODAGs, in order to understand if there are neighbor nodes in the communication range.

DIO - DODAG Information Object

Used to construct a new DODAG. DIO messages are periodically broadcasted to all nodes by the root (i.e., the border router) to provide network information.



Allows a node to:

- discover a *RPL instance*
- *learn* its configuration parameters
- *select* a DODAG *parent set*
- *maintain* the DODAG

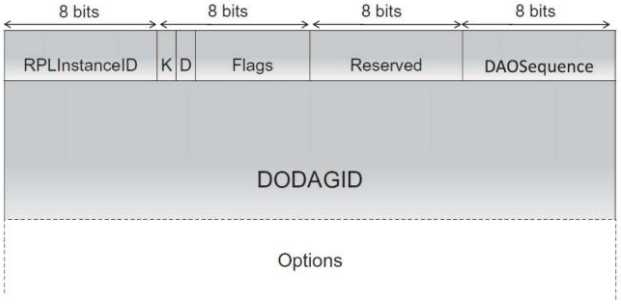
Fields:

- ***RPLInstanceID***: ID of the RPL instance the DODAG is part of
- ***Version Number***: Version number of the DODAG. It is incremented every time the network is updated.
- ***Rank:*** The rank of the node sending the DIO message
- ***Destination Advertisement Trigger Seq Number (DTSN***): Used to maintain downwards routes
- ***Grounded (G):*** a flag indicating whether the DODAG satisfies the application-defined objective
- ***Mode of Operation (MoP)***: set up by the root, identifies the mode of operation.
  A node must be able to cope with the MOP to join as a router
  It's important to decide which node can be router in the DODAG, because if the node cannot cope, it cannot be a router in the DODAG. (Talking about story-mode, see later)
- ***DODAG Preference (Prf):*** preference between DODAG routes with respect to other DODAG routes.
- ***DODAG ID:*** Uniquely identifies the DODAG. Set by the DODAG root.

DAO - Destination Advertisement Object

It's used to propagate reverse route information by recording the nodes visited along the path.
Sent by all the nodes, other than the DODAG root to populate their routing tables with addresses of children and to advertise their address to their parents.
When the root receives a DAO message from a node, it becomes aware of the reverse path for reaching that specific originator node.

DAO messages are sent by a node to advertise its reachable destinations to its parent nodes. DAO messages carry information about the node's own address and the destinations it can reach.



Fields:

- ***RPLInstanceID***: which was learned from the DIO
- ***K flag:*** when a message is received with this flag set, the node must respond with an ACK.
- ***DAOSequence***: Sequence Number, incremented at each DAO message sent
- ***DODAG ID:*** Set by the DODAG root. Uniquely identifies the DODAG

DAO-ACK - DAO Acknoledgement

Sent by a recipient in response to a DAO message.
Fields: RPLInstanceID, DAOSequence, Status.
Status code greater than 128 indicates a reject, and as a result, the node should select an alternate parent.

RPL DODAG Construction

It's base on two phases:

- **Broadcast transmission of DIO messages**:
  - *started by the root* and *repeated by all the other nodes*.
  - To build **upward** routes (from nodes to the root)
- **Unicast transmission of DAO messages:**
  - *Sent by nodes to the DODAG root*
  - To build **downward** routes (from the root to nodes)

Phase 1 - build upward routes

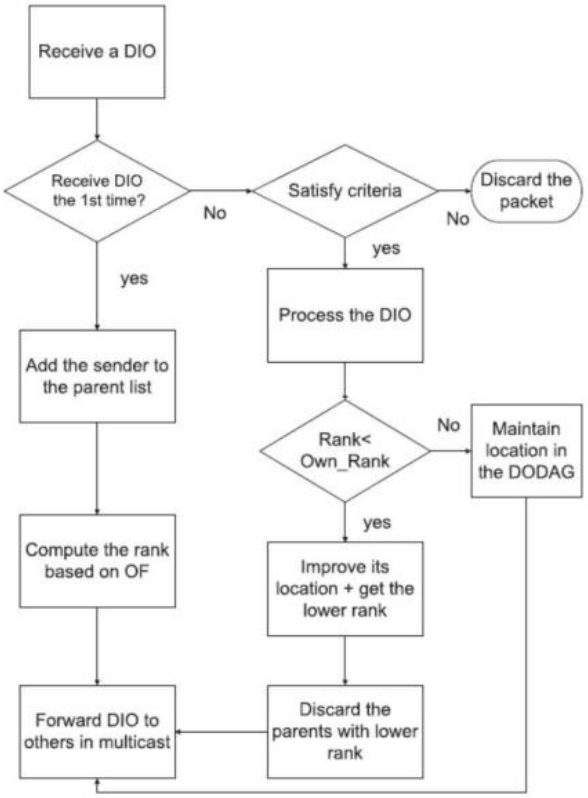
The root broadcasts a DIO message to announce:

- DODAG ID
- Rank, initially equal to 0, since the root has the highest position.
- Each node, before re-broadcast the message, increase the value of the rank. It allows each node to determine its position along the DODAG.
- Objective Function (specified in the DIO Configuration Option Field)

The DIO message can be received by both a node willing to join the DODAG or a node that has already joined the DODAG. By receiving DIO messages, a node learns the set of nodes in the one-hop neighborhood and determines its neighbor set.

Using the Objective Function of the RPL instance, a node:

- Determines its own rank, based on ranks received from neighbor nodes
- Selects one or more parents from the neighbor set
- Selects a preferred parent from the parent set: ***the preferred parent is the default next hop in the route towards the root***



Routing Update

DIO messages are periodically re-broadcast to update routing information, but how to select the update period?
A ***small period***: reduces the probability of having stale information or persistence of loops, but of course *uses more bandwidth and energy.*

Phase 2 - build downward routes

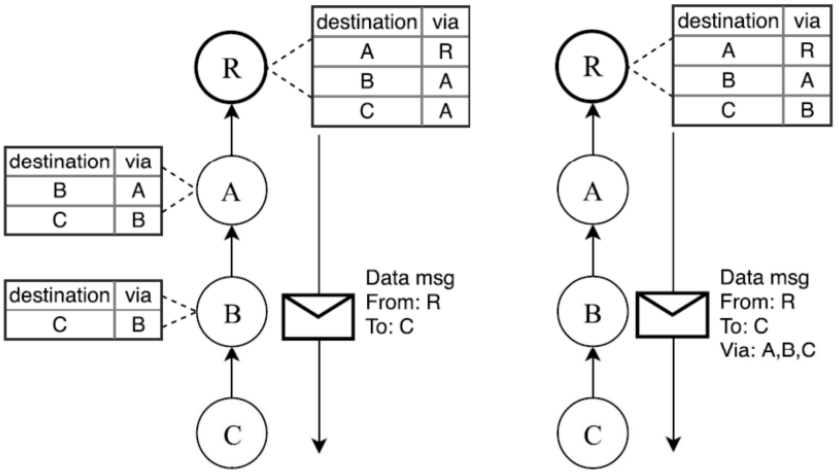
Downward routes from root to nodes are supported: they are built and maintained through DAO messages.
Setup procedure for downward routes:

- each node sends a unicast DAO message to the root
- Upon receiving a DAO message, a *visited node*
  - Inserts its address in the DAO message
  - Retransmits the updated DAO message upward
- *Upon receiving a DAO message from a node the root learns the downward route towards that node*

Mode of operations for downward routes:

- **Storing mode:** in the storing mode, a DAO message is sent in unicast by the child to the selected parent, which is able to store DAO messages received by its children before sending the new DAO message with aggregate reachability information to its parent.
- **Non-storing mode:** in the non-storing mode, the DAO message is sent in unicast to the root, thus, intermediate parents do not store DAO messages, but only insert their own addresses to the reverse route stack in the received DAO message, then forwards it to its parent.

Storing mode is more efficient because intermediate know how to reach the destination node, however, this approach has the **problem of memory used** → is often used non-storing mode.



(a) Storing mode (b) Non-storing mode

RPL Network management

The RPL Network management consists of three functionalities: DODAG repair operation, loop detection and avoidance, and security.

DODAG repair operation

DODAG Repair Operation: ***triggered whenever an inconsistency is detected in the DODAG.*** For example detection of *link failure* (for example node out of battery or interference). Strategies to deal with these problem are:

- **Local repair**: Triggered upon detection of a network inconsistency (like link failure or local loop)
  The current path is replaced by an alternate path, so a new preferred parent is chosen from the list of possible parents: may not be an optimal path. It cost less in term of message to be sent and overhead.
- **Global repair:** Triggered by the DODAG root when several inconsistencies are detected. In this case, the entire DODAG is reconstructed.

Loop Detection and Avoidance

During the DODAG construction phase, *loop can be created if for some reason one node receive a DIO from its neighbor before receiving it before receive it from one of its parent*. Moreover, a loop occurs whenever a node loses all its parents and attaches to a node in its sub-DODAG.
Loops are undesirable problems in routing protocols, however, *over-reaction to loop detection is not recommended, as traffic in LLNs is typically low!*

- **Loop Avoidance:** Based on rank, a node cannot select a parent with a rank higher than the node minimum rank + `DAGMAXRankIncrease`.
  Does not prevent loops from occurring but avoid count-to-infinity when a loop occurs.

- **Loop detection:** *check if packet is moving in forward direction toward the root node and it's not experimenting any loop* exploiting control information inside data packets. Control data are flags making part of packet headers.
  There's also a limitation in forwarding packet: you can forward packet to one sibling only once (added a flag in the header).

Security

The are some security modes that are implemented by the RPL protocol, specifically it implements three different security modes, so nodes in the network can operate according to three security modes, that are as follows:

- **Unsecured**: RPL control messages sent without any security mechanism
- **Pre-installed**: Nodes have a pre-installed key for securing RPL control messages (all nodes share a key).
  Nodes exploits this pre-installed key to guarantee *Message confidentiality and Message integrity*.
  Not save you from internal node bad compromised by an attacker.
- **Authenticated**: Nodes have a pre-installed key. *Used only in the joining phase*, but not for data exchange.

RPL performance Evaluation

RPL Performance, what is important to consider?

- **DODAG Convergence Time**: *time at which the DODAG is completely constructed and all the nodes have joined the network.* **This time should be short as possible**, since there are DIO broadcast: **during this phase, nodes use a flooding approach! Nodes must be active**, so we need that this time is as short as possible.
- **Power Consumption**: average energy consumed by each node during the DODAG construction process.
- **Path Length**: the number of hops to traverse from a node to the root (LBR).
  Path Stretch: average/maximum path length. ***Is not very significant, since we can have a short path but not very reliable path.*** Is it better have long reliable path or short not so reliable path?
- **Latency**: (round-trip) time between two nodes placed at a certain number of hops

All this metrics depends on the number of nodes and on the characteristic of links.

RPL instability:

The preferred parent can be changed: if high packet loss is experienced with the preferred parent, RPL selects a new preferred parent (local repair).
Makes RPL adaptive to network conditions, but introduces a change in the path to the final destination.
This change in the path can have negative impact, there may be an issues flow-based (real-time) traffics: Resources shared with the old parent must be deallocated, resources with the new parent must be allocated (especially if the MAC protocol is TSCH).
This could case some instabilities and some problems in the data flow.

10- Industrial Internet of Things

So far, we have focused on applications that do not have any specific requirements, such as latency or reliability. However, now we will shift our focus to critical applications that have special Quality of Service (QoS) requirements that need to be fulfilled by the underlying network. Industrial applications fall into this category of applications with special requirements.

Is the IETF IoT architecture suitable for critical/industrial applications? Moreover, what are critical applications?

Characteristics

- Industrial applications are applications with ***special requirements about latency/reliability.***
- Typically random topology is not used since you're able to deploy sensor in strategic way (***pre-arranged topologies is used***, for example star topology, with the border router in the center)
- Often ***complex sensors***: the energy consumption for sensing cannot be neglected with respect to communication.
- ***Large Data Volumes***: Cameras, complex sensors

Requirements

- ***Reliability*** in data collection is often required (Both *sink-to-sensors* and *sensors-to-sink* reliability required, so both to collect data and to send command)
- ***Real-time constraints in data collection***
- Energy Efficiency, often not the most important requirements to guarantee, may can be most important guarantee real time communication
- ***Scalability:*** A hierarchical architecture is typically used

We can classify industrial applications:

- **Safety-critical applications**
  Reliability must be maximum, all emergence message generated must be received and they must arrive with a very low delay. Strange constraints on reliability and latency
  - Class 0: Emergency actions
- **Control applications**
  Control applications, but with different requirements in terms of reliability and latency.
  - Class 1: Closed-loop regulatory control
  - Class 2: Closed-loop supervisory control
  - Class 3: Open-loop control
- **Monitoring applications**
  - Class 4: *Alerting*
  - Class 5: *Logging and Monitoring*

Wired networks used for critical functions (Classes 0, 1, and 2): wired links provides high reliability, fault tolerance, Deterministic latency and better Security with respect than wireless networks.

Wireless networks used for non-critical functions (Class 5, 4, 3, 2): of course reliability, Bounded latency, Scalability, Energy efficiency, Security are important, but the *advantages are the flexibility, the mobility, the easy of deployment and the limited cost.*

In WSN (LLNs) we have to pay attention to reliability threats:

- Node failures (remember we are in an industrial environment, so we can have liquid or other agent that can make the node fail)
- Transmission errors may occurs, for example due to interferences produced by other devices/machinery or due to the presence of obstacle
- Collisions/Congestions

Countermeasures:

- **Packages in order to protect the device**
  If the device is a sensor, they may introduce problem for sensing
- **Node redundancy**
  Increase the cost of the deployment but also the lifetime of the system.
- **Frequency hopping**
  Not use always the same frequency. Consider that if you use always the same frequency and in that frequency there are many interferences your communication is very bad (or you cannot communicate at all). *In this way interferences impact less on communication, of course sometimes you transmit in a "wrong" frequency, but not always.*
- **Redundancy**
  For example **packet retransmission**: if a packet is not received correctly, it can be retransmitted to ensure its successful delivery. Another technique is **forward error correction**, where redundant coding is used to encode the information and if some bits are not received correctly, the destination can still decode the errored packet by utilizing the redundant information.
- **Redundant paths (routing)**
  This approach is particularly useful when there are physical obstacles that could hinder communication between two nodes. In cases where an alternate path is available to reach the destination node, this problem can be circumvented. By utilizing a different route, the communication can bypass the obstacles and ensure reliable delivery.

Communication Technologies for Industrial Applications

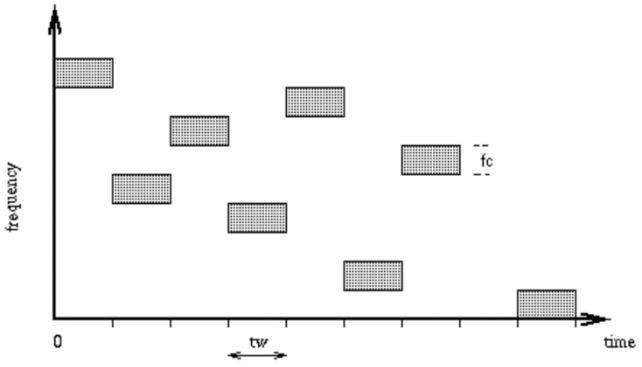
In this course, we will explore two communication technologies for industrial applications: **IEEE 802.15.4 TSCH** and **WirelessHART**. Both of these technologies are built upon the **IEEE 802.15.4** protocol, enhancing it with additional features.

As we know the IEEE 802.15.4 standard has some limits:

- ***No guarantee on end-to-end delay***, not guarantee the maximum delay that a packet may experience.
- ***Limited communication reliability***: in many case a packet can also be dropped because it can be over the max backoff or the maximum retransmission number.
  The MAC unreliability problem is mainly due to the random nature of the CSMA/CA algorithm.
  The periodic beacon synchronizes channel accesses thus maximizing contention. The problem is exacerbated by the default MAC parameter values: which are not appropriate for WSNs with a large number of nodes, especially when power management is enabled.
- No built-in frequency hopping technique

Channel Hopping

To avoid these limits, a **channel hopping** technique is used: subsequent packet are sent using difference frequencies (even if no interference is detected), forming a pseudo-random hopping pattern. Of course transmitter and receiver must be synchronized to the same frequency.
If a transmission fails, the packet is retransmitted in a different frequency. It's used a **time slotted** approach:



In this way there's a greater chance of successful transmission, reducing the negative effect of the interferences. It also increases the security: against *Selective Jamming attack*.

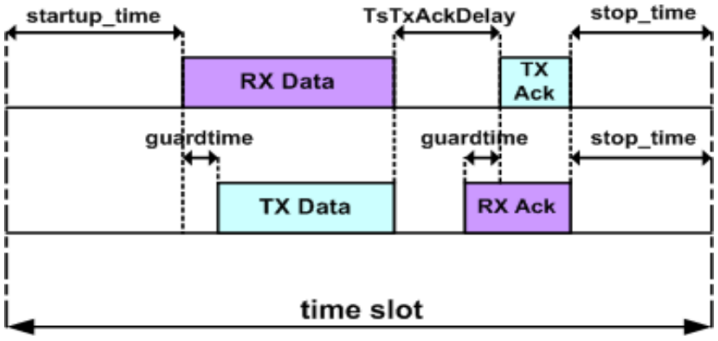
IEEE 802.15.4 TSCH

TSCH is the standard for critical application operating on Low-Rate Wireless Networks, since the original IEEE 802.15.4 protocol doesn't guarantee the requirements for industrial application.
Time slotted access, multi-channel communication and channel hopping are the main characteristic of the standard.

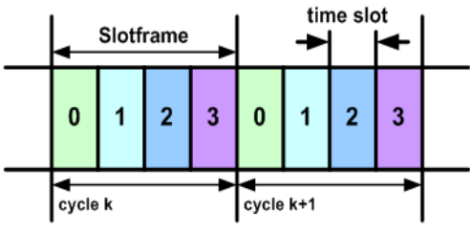
- **Time-slotted access**:
  Time is divided into slots, and each node is assigned specific time slots during which it can transmit. The duration of each time slot is designed to accommodate the transmission of a fixed-size packet and the subsequent reception of the corresponding acknowledgment.
  Since packet are transmitted inside the slots, we have
  - Predictable and bounded latency, since each node know when it can transmit
  - Guaranteed bandwidth
- **Multi-channel communication**: ***more nodes can communicate at the same time*** (i.e., same slot) ***using different channels*** (different operating frequencies). It increased network capacity.
- **Channel hopping**: *mitigates the effects of interferences* and improves reliability

Inside a single time slot:
Since nodes are not exactly synchronized, it's necessary to introduce a delay before transmission, in order to avoid overlapping communication.
Let us suppose to have two nodes, A (the initial transmitter) and B (the initial receiver), that want to communicate:

- Nominal start of the time slot: the communication doesn't start in this moment
- startup_time: after a certain period the *node B* become ready to receive a packet
- After an additional guardtime the *node A* node can transmit it's message (to ensure that the receiver is truly prepared to receive the packet)
- After the transmission of the packet, there's a certain period were both A and B switch their radio from Tx mode to Rx mode and vice-versa.
- Once the node A is ready to receive the ack, but again there's a guardtime that the node B must wait before transmit the ACK.
- After the ACK there's a stop time before the next time slot.



Slots are grouped to form slotframes. Time slots inside each slotframe are typically assigned to a couple of nodes. This behavior will repeat in successive slotframes.

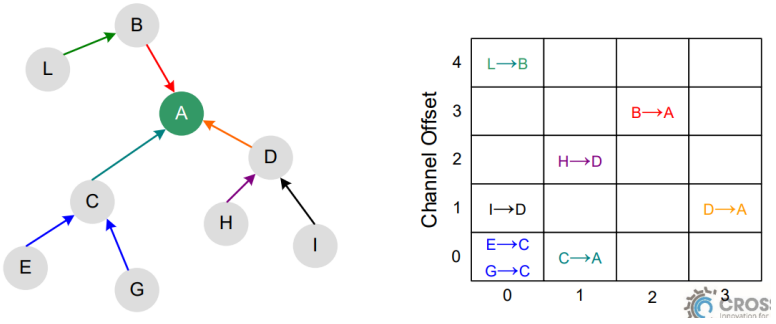


TSCH link

If we consider multi-hop communication, we can visualize it as a two-dimensional grid, where one dimension represents time divided into time slots, and the other dimension represents different frequencies. This concept can be likened to having multiple channels available simultaneously within each time slot, enabling communication between different pairs of nodes. Each of these channels is uniquely identified by a **channel offset**.

A single cell in the resulting table is called a "**Link**," which is defined as a pairwise assignment of directed communication between devices in a specific time slot, utilizing a particular channel offset.

To avoid confusion with the term "link", it is common to replace it with the term "cell".



We can imagine the communication schedule like a matrix with dimensions #channelOffset X #timeslots.
From the matrix mentioned above, we can observe that we can have both dedicated links and shared links:

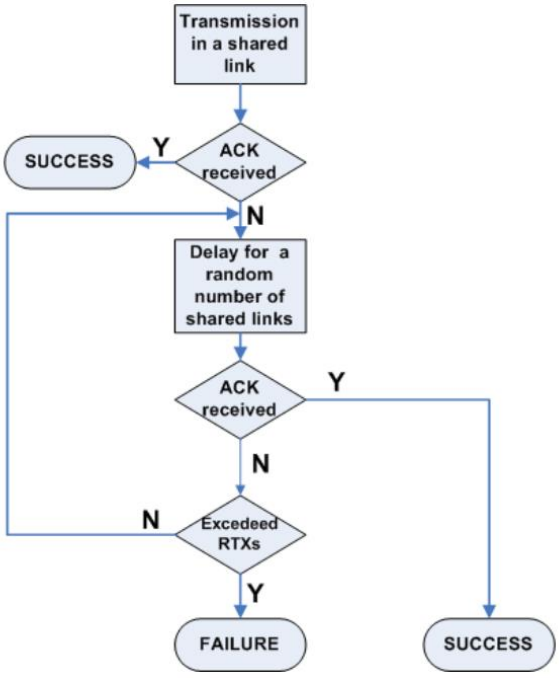
- **Dedicated links:** in dedicate links we have
  - deterministic traffic
  - typically used for Periodic transmissions (subsequent slotframes)
  - One transmitter – One receiver, direct access (absence of collision)
- **Shared links:**
  - Sporadic and unpredictable traffic
  - typically used for discovery and routing messages (which are just control message and does not contain application data)
  - Multiple transmitters/receivers and CSMA-CA based access.

Why should I choose to use shared links if they bring about the same problems that I wanted to avoid with this new protocol, such as collisions? Shared links have been introduced for specific purposes. For example, they can be used for transmitting control messages, like beacon messages in the MAC protocol, which are typically transmitted over shared links.

Shared links are used also to request the allocation of dedicated links.

TSCH CSMA-CA for Shared Links

The communication in shared links is performed in this way:



Note: There's a sort of ***backoff delay = the number of shared link that must be skipped by the specific node to retransmit the packet.*** Then the node try to transmit again and wait for ACK

We observe that this CSMA-CA algorithm is similar to the CSMA-CA algorithm used in the original IEEE 802.15.4 standard. However, there are some differences between them.
Differences with 802.15.4 CSMA-CA:

- The **backoff mechanism** is triggered only after a node encounters a collision, in order to prevent further collisions (unlike the previous algorithm where: generating a backoff time was the first action taken).
  In this case, the backoff unit duration is expressed in *shared slot* (in 802.15.4: duration of the backoff was expressed in multiples of backoff slots).
- **Clear Channel Assessment** (CCA) are not used to prevent collisions among nodes, but to avoid transmitting a packet if a strong external interference is detected.
- Then a packet is **dropped** only if it reaches the maximum number of retransmissions. (in 802.15.4: there were 2 reasons to drop a packet: exceeded number of retransmissions or exceeded number of backoff stages)

Frequency Translation- Channel Hopping

The frequency f:

$$f = F\{(\mathrm{ASN} + chOf) \mod n_{ch}\}$$

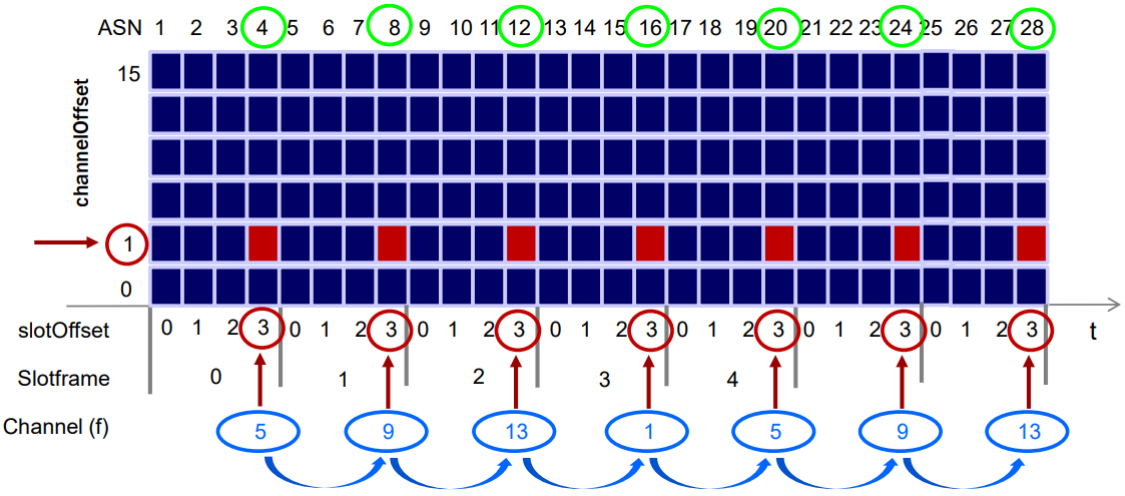
ASN: total # of slots elapsed since the network was deployed

n_{ch}: number of used physical channels

F is implemented as a look-up-table containing the set of available channels

F is often the Identity function.

Example: let us focus on the red cell



Remember that the operating frequencies are enumerated from 11 to 26, so the frequency 5 is actually the operating frequency 16.

As we can see from the example, ***the number of hopping (change of frequency) is limited***, in the example the number of different operating frequencies used by a cell is only 4 over 16 available. In this way not all the frequencies are used, reducing the performances.
In order to use all the frequencies the number of channels and the size of the slotframe should be relatively prime (the standard suggests some values for slotframe sizes).

Network formation

So far, we have assumed the existence of an operational network that we can use for sending and receiving packets, and we can evaluate its performance. However, there is a preliminary phase in which the network is not operational because nodes are still joining the network.
The network formation is based on ***Enhanced Beacons (EBs)*** regularly emitted by the PAN Coordinator and other nodes. EBs are special frames *containing:*

- **Synchronization information**: allows new devices to synchronize their clocks with the network. Extremely important since the network is based on time slots.
- **Channel hopping information**: *allows new devices to learn the channel hopping sequence*
- **Timeslot information**: describes when to expect a frame transmission and when to send an acknowledgment
- **Initial link and slotframe information** allows new devices to know:
  - when to listen for transmissions from the advertising device
  - when to transmit to the advertising device

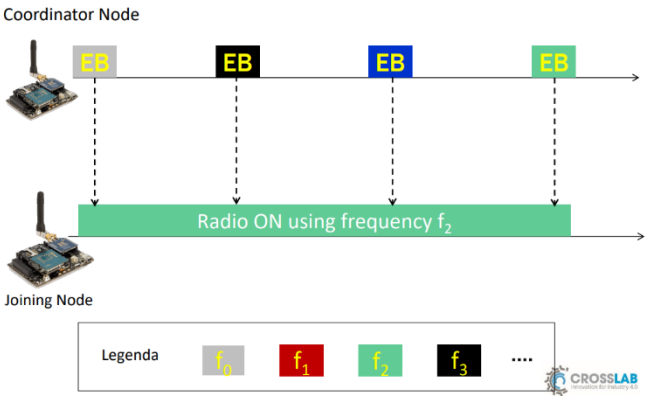
The steps are:

1. EB is transmitted by the border router, who initiate the network formation
2. EBs are received by all the other nodes who wants to join the network (there are some frequency consideration to perform)
3. Once a node join the network will retransmit the beacons to announce is presence to other node who wants to joins.

EBs are transmitted by nodes part of the network, but **the EB advertising policy is not part of the TSCH protocol**.

Frequency and time consideration:

- A joining device starts listening for possible EBs on a certain frequency
- Upon receiving an EB, the MAC layer notifies the higher layer
- The higher layer initializes the slotframe and links using information in the received EB message and switches the device into TSCH mode
- At this point the device is connected to the network. Then, the device allocates communication resources and starts advertising, on its turn



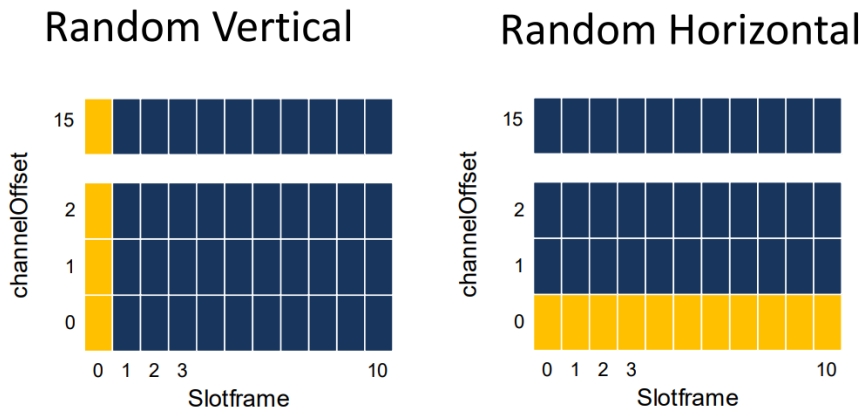
Coordinator node is a node that already joined the network

In this example after some time, the joining node receive the EB and then it can join the network: ***Devices must keep the radio ON during the joining phase,*** for this reason ***the joining phase should be very short*** in order to reduce the amount of energy consumed by a node that want to join the network.
This time depends on the EBs frequency → EBs should be sent frequently. However, as is often the case, transmitting EBs more frequently leads to a reduction in communication resources (as bandwidth resources are consumed) and an increase in energy consumption at network nodes.

When should EBs be transmitted?
Use only one cell in each slot frame to send EBs: Increase the joining time!
We have two strategy to reduce the joining time:

- **Random vertical**: use *more than one cell* (may be up to 16), *but all cells devoted to EB correspond to the first time slot in each slotframe with different chOffset*. Since different nodes use the same time slot but different chOf (different operational frequencies) *reduce the probability of collision, even if neighbor nodes send EbS in the same timeslot.*
  So a large number of cell devoted to the EB.
- **Random horizontal:** *more than one cell devoted to the EB* (up to the number of time slots in a slotframe), *however slot devoted to EB are allocated in different time slot in the slotframe, but all at the same channelOffsett*.

The random horizontal approach is generally preferred because the frequency at which the EBs are sent is constant.



The larger the number of slots, the better the performance: reduced the number of collision.

What is the optimal value?

Optimization Problem

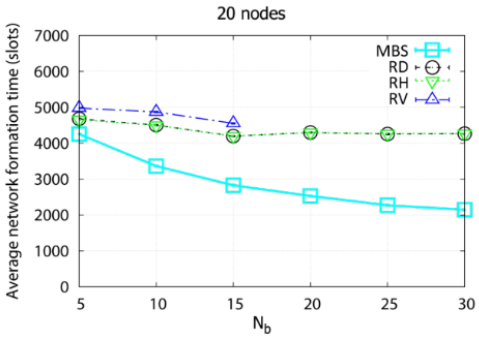
$$\begin{cases} \min_{d_j} \tau_{join} \\ \sum_{j=0}^{N_b} d_j = T \\ d_j \in N \ \forall j \end{cases}$$

- $d_0 \dots d_{N_b-1}$ are the unknowns
  - distances, in terms of timeslots, between successive EBs
- T: slotframe duration

Model-based Sheduling Algorithm: <u>given the total number of EBs to allocate provides the EB allocation strategy that minimizes the joining time</u>.

In this approach the importance is the distance between two consecutive cell and not the channelOffsett or the slotframe.

With this method, changing the number of EBs per superframe and the number of nodes, it has better performance than other techniques like Random, Random Vertical, Random Horizontal where the performance index is the average network formation time.



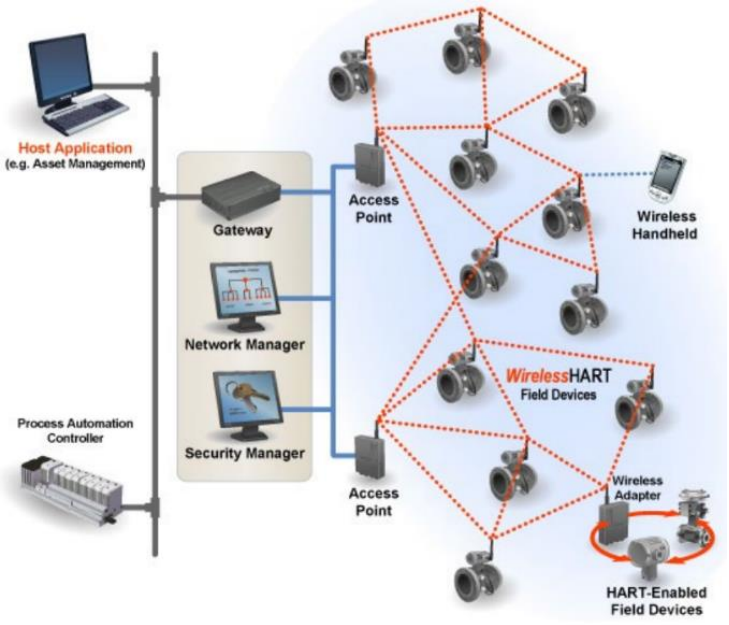
WirelessHART

Many ideas in IEEE 802.15.4e TSCH are derived from WirelessHART, a wireless communication standard specifically designed for the industrial sector.
WirelessHART is based on the physical layer of the IEEE 802.15.4 standard. Its MAC protocol is a time slot protocol, similar to the one we have described thus far, with some minor variations.

It uses *IEEE 802.15.4 in the 2.4 GHz ISM Band as **PHY***, TDMA-based as MAC protocol.

WirelessHART architecture:

- **Field Devices** = sensing/actuating nodes
  They communicate in a wireless way. Devices connected to process or plant equipment
- **Access point** as the *same functionalities of the border router* (considering the ITF standard)
- **Gateways**: Enable *communication between these devices and host applications connected to a high-speed backbone* or other existing plant communications network.
- **Network Manager** is *responsible for configuring the network, scheduling communications between devices, managing message routes, and monitoring network health*



Gateway, network and security are usually grouped inside the same device.

In order to guarantee bounded latency, the path from a node to the access point must be fixed: in this environment is important that the topology remains static (or quasi static) in order to guarantee a bounded latency. Why? Because if the path change every time we are not able to guarantee a bounded latency.

Routing: *is both graph routing and source routing*

- **Graph Routing**: *routes are calculate in advance by network manager based on the topology and traffic info received by all the devices*; once the routes are calculated, the routes are sent to the devices. Each device knows its position in the network and know the next hop.
  A graph is a collection of paths that connect network nodes. Paths in each graph are explicitly created by the network manager and downloaded to each individual network device.
  - To send a packet, the source device writes a specific path ID in the network header.
  - All network devices on the way to the destination must be pre-configured with graph information that specifies the neighbors to which the packets may be forwarded.
- **Source routing**: in addition to (supplement of the) graph routing *aiming at network diagnostics*. The source device includes in the header an ordered list of devices through which the packet must travel. Each device utilizes the next network device address in the list to determine the next hop

In summary, WirelessHART utilizes a centralized approach where the network manager handles all the management tasks. This approach is implemented to ensure that the application receives guarantees in terms of time-bound delay and reliability. In cases where strict requirements are necessary, such as a static network, the centralized approach is considered the most suitable.

TSCH-IoT integration

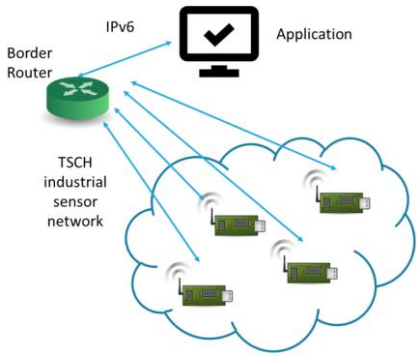
How to integrate TSCH network and the Internet, in order to obtain the so called Industrial IoT?
The IoT paradigm assume that:

- the underlying access mechanism is asynchronous;
- smart objects are always active and can be added and removed dynamically. Assume that the underlying protocol is IEE 802.15.4.

These assumptions are not applicable to the TSCH protocol due to the following reasons: Firstly, the underlying mechanism is synchronous as the TSCH protocol operates on a time slot basis. Secondly, smart objects are not continuously active but rather operate only during their assigned time slots, remaining in a sleeping mode for the remainder of the time.
So the challenge now is how to combining a dynamic and flexible routing mechanism (e.g. RPL) with **802.15.4e TSCH**.

In order to integrate TSCH networks into the traditional Internet, which is based on IPv6, the IETF established a working group called "IPv6 over the TSCH mode of IEEE 802.15.4" (commonly known as **6TiSCH WG**). The primary goal of this working group is to enable IP-based communication over low-power and lossy networks, specifically for soft real-time industrial applications (because for hard real-time requirements is better to use wired technologies).

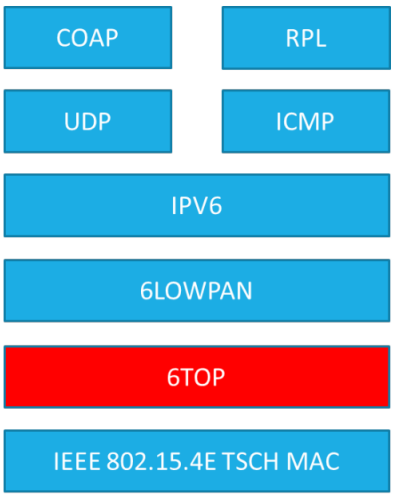
The overall architecture of a 6TiSCH network includes one or more border routers to connect the TSCH industrial network to external applications through IPv6:



The problem is that TSCH does not define:

- *policies to build and maintain the communication schedule*
- mechanisms to *match the schedule to the multi-hop paths maintained by RPL*
- mechanisms to *adapt the resources allocated between neighbor nodes to the data traffic flows*
- *techniques to allow differentiated treatment of packets*:
  - data packets generated at the application layer
  - signaling messages needed by 6LoWPAN and RPL to discover neighbors and react to topology change

TSCH **provides** necessary mechanisms for communication, but it **does not include** resource allocation and management.
Specifically, it does not define a policy for constructing the communication schedule.
So, a new sublayer has been introduced within the **6TiSCH Protocol Stack**:



The new layer defines *allocation algorithms and protocols for negotiation of communication resources.*

Let's clarify why the 6top layer was introduced

Most other IoT protocols were designed with the assumption that the underlying MAC protocol is the original 802.15.4 MAC protocol, which is based on the asynchronous CSMA-CA algorithm. This algorithm assumes that nodes can transmit at any time. However, in the case of TSCH, the underlying MAC protocol is not asynchronous, it is time slotted, meaning nodes are only active during specific time slots and are in sleep mode at other times. This leads to several differences compared to the traditional standard.

If you directly use the TSCH MAC protocol below the 6LowPAN adaptation layer, it won't work as expected. Specifically, certain functionalities are not implemented because TSCH provides necessary mechanisms that allow communication between a pair of nodes within a cell, but it does not define any policy for slot allocation and management.
The TSCH MAC protocol intentionally left these functionalities to be implemented in the upper layer.
However, the 6LowPAN adaptation layer was designed for the CSMA-CA protocol and did not incorporate these slot allocation functionalities. As a result, a new layer called the 6Top layer was introduced to bridge this gap and provide the necessary functionality for slot allocation and management in TSCH-based networks.

6TiSCH scheduling

6TiSCH Scheduling: how to allocate cells in the TSCH matrix.
Several approaches are possible:

- **Static scheduling**: used *during network bootstrap* or when a better schedule is not available.
  Cells are allocated statically: this is not a very efficient scheduling, but since nothing is allocated you have no better alternative. During the operation phase other techniques are used
- **Centralized scheduling**: like in wirelessHART network, based on a central entity
- **Distributed scheduling**: neighbor-to-neighbor cell negotiation
- **Autonomous**: No neighbor-to-neighbor cell negotiation. Cells are allocated autonomously by nodes
- **Hop-by-hop**: Cell reservation along the path using an end-to-end signaling. Not yet defined
- **Hybrid solutions**: Combines different scheduling approaches

Static scheduling

It is **pre-configured**, meaning that the number of slots to be used for communication is defined in advance and learned by the node upon joining the network.
Minimal config: is introduced to allow broadcast of RPL control message, and it involves configuring a predetermined number of timeslots specifically for the transmission of RPL messages from the beginning.

Once the network has been formed (node joined the Tisch network) you have to calculate the paths for each of them to the border router, and this is performed through exchange of RPL messages. In order to transmit RPL control messages, a specific cell is used, which is the initial slot (0,0).
Used only initially to setup the network. Once the network has been set up it is better to use another communication schedule.

Centralized scheduling

Path Computation Element (PCE) that manages all the details of the centralized schedule:

- Collects:
  - network state information, related neighbors of each node and the quality of links between neighbors
  - traffic requirements from all nodes, useful to build the topology of the network
- Builds the communication schedule that depends on the topology of the network
- Installs the schedule on the network. So each node learn which cells are allocated to them (their communication schedule)

Centralized Scheduling Algorithms:

- **TASA** (Traffic Aware Scheduling Algorithm)
  Assumes tree topology, all the nodes send packet to the root of the tree (typically the border router). Good for sensor networks. The coordinator has a single radio interface.

This algorithm is designed for a converge-cast communication model, where all nodes in the network send their data to a common destination.

- **MODESA** (Multi-channel Optimized Delay Slot Assignment)
  Tree network topology. The coordinator has multiple radio interfaces so it is able to perform simultaneous communications. Also this algorithm is designed for a converge-cast communication model.

The main difference is that the coordinator has multiple radio interface.

TASA

There is a centralized manager called the Path Computation Element (PCE), which we can refer to as the Manager for simplicity.

- ***Every*** *node regularly updates the Manager with the list of other nodes it can hear and the amount of data it generates.*
- ***the Manager*** can *calculate the topology and the schedule of the network.* Since he received this information by all the node, based on the list of node it *can derive the connectivity graph: once it's derived, the manager can assign a slot to the couples of nodes that want to communicate*.
- The manager informs each node about links it's involved in
- ***If the connectivity graph changes,*** *the manager updates its schedule and informs affected nodes.* This is possible thanks to the regularly update sent to the manager by the nodes.

MODESA

- like TASA it takes an iterative approach: iterates over the set of nodes with data to transmit and having an available interface with their parent in that slot
- Sorted according to their priorities
- Picks the node with the highest priority and schedules its first transmission on that timeslot ♣ on the first channel offset
- Then, selects another node: If the node is in conflict with a previously scheduled node its transmission is allocated on a different channel offset, otherwise, the same channel offset is used
- Continues until transmissions of all nodes have been scheduled

Summary of centralized scheduling approach

Of course if the network is static, and the topology does not change (or change very slowly): the centralized approach is very good, since the manager calculate the graph, then the optimal communication schedule, then diffuse this communication schedule and in principle the situation does not change.

Not appealing for: dynamic networks (mobile nodes, power management) and large-scale networks. If you have a very large network, the centralized approach may not be the most suitable approach since the time to collect the information of the network may be very large.

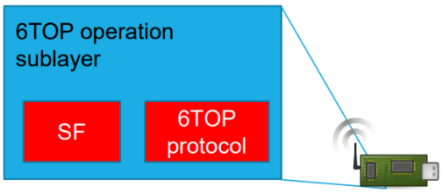
Distributed scheduling

Typically in a wireless environment the network is dynamic since the quality of the links depend on external condition: for this reason, centralized scheduling is not suitable but we may have to consider a distributed schedule approach. In this approach, there is no central entity, and each node collaborates with its neighboring nodes to compute its own schedule. This is achieved by exchanging local, partial information with neighboring nodes.

There's not a global knowledge of the network so optimal schedule cannot be calculated. On the other hand, the overhead is limited, making it suitable for energy-constrained nodes. No need to send information to a centralized node. Nodes agree on a common schedule with neighbor-to-neighbor negotiation.

During the negotiation phase, nodes agree on a common schedule by determining which cells to allocate and how many, in order to satisfy the QoS requirements along the path. This negotiation takes into account the number of packets that a node needs to send, ensuring an appropriate allocation of cells.

How to negotiate? 6Top protocol



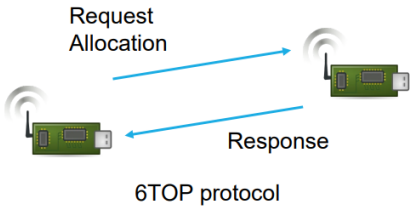
The negotiation protocol alone is not sufficient because before negotiating the number of cells needed to transmit packets, it is necessary to know how many cells are required for the message transmission. The **scheduling function** is responsible for determining the number of cells to be allocated for communication with a specific neighbor node. It defines the policy adopted by nodes to allocate and deallocate cells and also determines the algorithm for adding or deleting cells. The scheduling function dynamically adapts the number of reserved cells between neighbor nodes based on current bandwidth requirements and network conditions.

Indeed, it is important to note that even if an application generates a constant number of packets that does not change over time, external operating conditions such as the quality of communication due to the instability of the wireless link can lead to packet errors and the need for retransmissions. Therefore, the number of packets may vary over time, even when the application's packet generation remains constant. As a result, the number of cells to be negotiated may also change over time. The scheduling function takes into account these operating conditions and decides when to increase or decrease the number of allocated cells accordingly. It ensures that the allocated resources adapt dynamically to the changing conditions of the network.

Once a node has determined the number of cells to allocate or deallocate using the scheduling function, it utilizes the 6P protocol to negotiate the allocation or deallocation of these cells with its neighboring nodes.

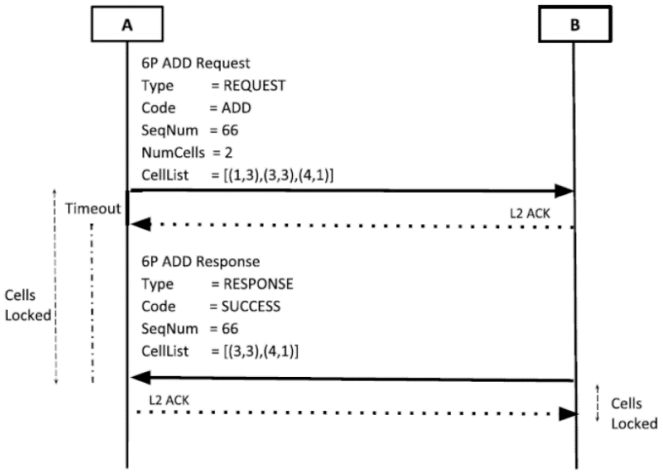
6Top Protocol (6P)

It's a ***request-response protocol***



We can have a **2 way handshake**:

1. The SF at Node A has decide that 2 cells must be allocated based on the traffic. So the Node A has to perform an ADD request (**Code**: ADD)
   SeqNum: to differentiate number of message
   CellList: list of possible cell usable for allocation (A suggest to node B which cells consider for allocation)
2. Node B send and ACK, and select 2 of the 3 cells (of course, node B will only choose from the cells that are not already in use by any of its neighboring nodes). It send the ADD response
3. Node A responds with an ACK

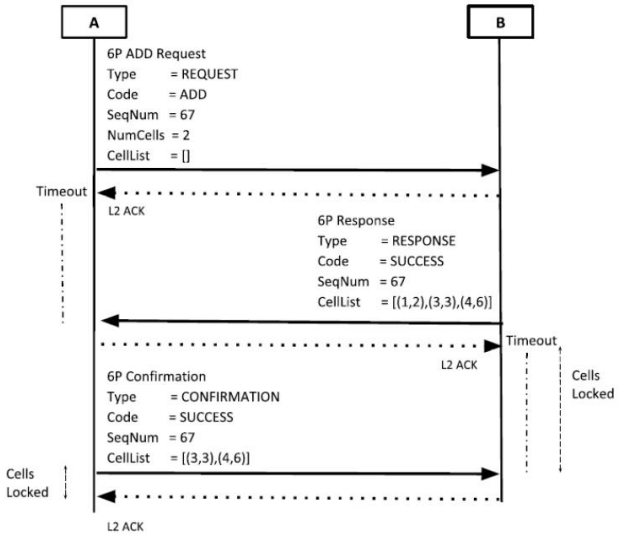


We notice that the **SeqNum** is the same for both messages, which indicates that both messages belong to the same request.

It is possible that all the cells suggested by node A are already used by neighboring nodes of node B. In this case, B will respond with a **failure message**.

(the same thing could be performed with a delete)

Or it can be performed with 3 way handshake



In this case that A doesn't specifies a list of cells, node B suggested a list of cell (6P Response) and A decide which use (6P confirmation).

Typically 2 handshake Is preferred, since one less message (and one ack) is sent, so less energy consumption.

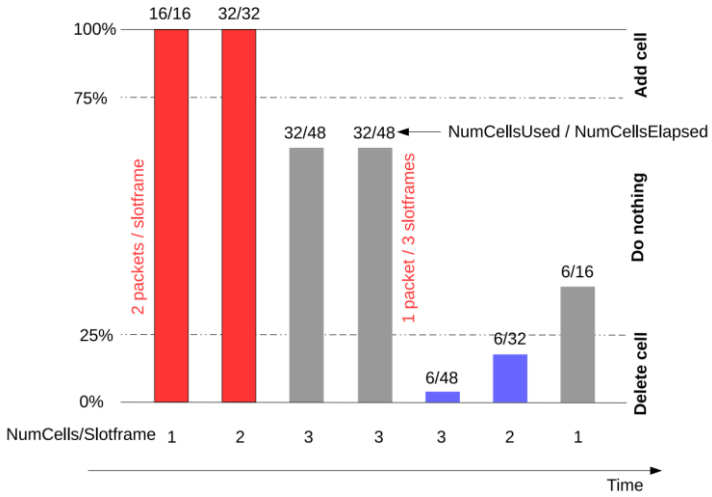
Distributed Scheduling Functions

It's the policy that decides how to maintain cells and trigger 6P transactions.

Minimal Scheduling Function (MSF)

It's the current default SF defined by 6TiSCH WG. It's an "hybrid" scheduling function since there are both autonomous and negotiated cells.

- ***Autonomous cells***: allocated without any negotiation and minimal amount of bandwidth (they are not negotiated with neighbors, bandwidth used only for control message).
  These cells are used to transmit control messages, such as 6P messages for the negotiation of cells.
- ***Negotiated cells***, additional cells :
  - *Negotiated through 6P protocol*
  - *Can be dynamically added/removed depending on traffic and network conditions:*
    - It increments ***numCellElapsed*** at each elapsed cell, and *increments numCellUsed each time that cell was used to send or receive a frame from that neighbor*.
    - *Every 16 slotframes, MSF computes the cell usage as the ratio between numCellUsed and numCellElapsed.*
      - *If the cell usage is greater than 75%, MSF issues an 6P ADD request to add one (Negotiated) dedicated cell to that neighbor.*
      - *If the cell usage is smaller than 25%, MSF issues a 6P DELETE request to delete one (Negotiated) dedicated cell to that neighbor*.
        By default, at least one negotiated cell is maintained in the schedule to communicate with the parent node.



At the beginning the node needs to send 2 packets per slotframe to its preferred parent, but has only a single cell to that neighbor in its schedule. After 16 slotframes, the cell usage is 16/16=100%. MSF issues a 6P ADD request to add one TX cell to its preferred parent. After the cell is installed, and after another 16 slotframes, the cell usage is still 32/32=100%. Then MSF calls 6P again to add another TX cell. With the three cells scheduled to that neighbor, the cell usage drops to 32/48=67%, a stable region for MSF. Let's assume that, at that point, the node's traffic requirement drops to 1 frame every 3 slotframes. The cell usage hence drops to 6/48=12.5%. MSF issues a 6P DELETE request to delete one (Negotiated) TX cell to its preferred parent. After this action, cell usage increases to 6/32=18.75%, still below the 25% limit, triggering 6P to delete another cell. This causes the cell usage to increase to 6/16=37.5%, a stable region for MSF.

*Note how increase and decrease **just one cell!*** This algorithm takes some time before adapting to the new situation.

On-the-Fly Scheduling Function (OTF)

It's previous SF considered by 6TiSCH WG with only Negotiated Cells.
The node analyzes the traffic information from uppers layers and estimates the number of packets that will be generated by the specific node. The node can also consider the number of packet received by neighbor nodes.
The algorithm outputs the number of cells that need to be managed by the node. This number is then passed to an allocation algorithm, which determines the number of cells to be negotiated. These cells are subsequently negotiated using the 6P protocol.

The main difference: the MSF measure the utilization and based on it decide if allocate or deallocate one cells.
OTF estimate the number of cells that are required by the node, based on the traffic that the node must manage and based on this analysis estimate the required cells.

ALGORITHM 2: OTF Allocation Algorithm (S_c, R_c, T)

Input:
S_c = Number of scheduled cells
R_c = Number of required cells
T = Hysteresis Quantum
Output:
ΔS = Number of cells to add/delete

1 **if** $R_c > S_c$ **then** $\Delta S = R_c - S_c + \lceil T/2 \rceil$ $\Rightarrow$ ADD CELLS
2 **else if** $R_c < S_c - T$ **then** $\Delta S = S_c - R_c - \lfloor T/2 \rfloor$ $\Rightarrow$ DELETE CELLS
3 **else** $\Delta S = 0$ $\Rightarrow$ DO NOTHING

6P Evaluation

When evaluating the 6top protocol, several parameters are taken into consideration. These include the **success rate**, which is the percentage of successful transactions out of the total number of transactions issued. Another important parameter is the **transaction delay**, which measures the time interval between the transmission of a request and the reception of the corresponding successful response.
In addition to these performance metrics, it is also important to evaluate the causes of failure.

Some consideration about 6P:

- MSF have an higher failure rate than OTF, why? Lack of resources (especially at high rates), due to overprovisioning and Buffer Overflow at MAC layer
- Anyway even OTF fails, especially at high traffic rates. Indeed, we observed that when transitioning from T=2 to T=6, many negotiations fail due to overprovisioning.

From experimental results we can say:

- 6P Transaction Delay in the order of seconds: very far from being negligible (regardless of the selected SF)
- Significant Failure Rate: even more than 50%! This strongly depends on Scheduling Function and Parameter Setting
- → Non-negligible negotiation overhead

So the general conclusion is that 6P introduces a non-negligible negotiation overhead, primarily due to the high transaction delay and failure rate. This factor must be taken into consideration when evaluating the performance of a scheduling function.
So reduce the number of 6P transactions is beneficial since it reduce negotiation overhead.

A solution could be **overprovisioning**: instead of allocating the minimum number of cells, you allocate more cells. This can help avoid frequent cell allocation and deallocation, thereby reducing the number of 6P transactions. However, this approach consumes more bandwidth due to the increased number of allocated cells, and more importantly, it can lead to failures due to a lack of resources.

SF Evaluation

When evaluating a scheduling function, several parameters need to be taken into consideration:

- **End-to-end Reliability**: This parameter measures the reliability of the communication link, specifically the ratio between the number of packets received by the sink (destination) and the total number of packets sent by all the nodes in the network. A higher reliability indicates a more effective scheduling function.
- **End-to-end Latency**: This parameter measures the time interval between the generation of a packet and its correct reception at the sink. Lower latency is desirable as it indicates faster and more efficient communication.
- **Cell Utilization**: This parameter evaluates how efficiently a node utilizes the allocated cells. Higher cell utilization implies better resource utilization and improved network performance.
- **Duty Cycle**: Duty cycle refers to the ratio between the number of cells where a node is active (transmitting or receiving) and the slotframe size. A lower duty cycle indicates more efficient energy usage by the node.

Some consideration about SF:

- **RPL** has an important impact since its dynamism may result in frequent parent changes. When a change occurs, the node is unable to transmit anything for a certain period of time.
  The link-quality assessment mechanism used by RPL can lead to select a parent node characterized by a poor link quality.
  Both these factors can dramatically increase the queue size, resulting in large delays and/or packet dropping.
- **6P** has also an impact: 6P transaction failures cause inconsistencies in the schedule, so it must be reset and re-negotiated. This increases the queue size.

So ***Congestion periods are unavoidable***, they are caused by other IoT protocols (RPL, 6P, TSCH):

- OTF is not capable of reacting to congestion at all
- MSF reacts better, but the time required to detect and recover from congestion may be in the order of minutes, or tens of minutes

Enhanced OTF (E-OTF)

Based on these evident problems, it was decided to make modifications to the OTF algorithm. The modified version now takes into account the link quality, specifically the Expected Transmission Count (ETX), when calculating the necessary number of cells. It includes a mechanism to timely react when the queue size increases over a certain threshold (the Congestion Bonus, line 2).

ALGORITHM 3: Enhanced-OTF Allocation Algorithm $(S_c, R_c, T, B, Q, U, ETX, \beta, \alpha)$

Input:

S_c = Number of scheduled cells
R_c = Number of required cells
T = Hysteresis Quantum
B = *Congestion Bonus (CB)*
Q = *Average Queue Occupancy*
U = *Average Cell Utilization*
ETX = *Estimated Link Transmissions*

Output:

ΔS = Number of cells to add/delete

1 $R'_c = R_c * ETX$
2 **if** $Q > \beta$ **then** $\Delta S = B$ $\Rightarrow$ *ADD CB*
3 **if** $R'_c > S_c$ **then** $\Delta S+ = R'_c - S_c + \lceil T/2 \rceil$ $\Rightarrow$ ADD CELLS
4 **else if** $R'_c < S_c - T$ **then** $\Rightarrow$ DELETE CELLS
5 **if** $U > \alpha$ **then** $\Delta S = B$ $\Rightarrow$ *REMOVE CB*
6 **else** $\Delta S = S_c - R'_c - \lfloor T/2 \rfloor$
7 **else** $\Delta S = 0$ $\Rightarrow$ DO NOTHING

Line 5: if utilization is high.

Comparing the performance of this modified allocation algorithm with that of the MSF and the original OTF algorithms, we observe that it delivers excellent results in terms of congestion management. In fact, it is able to quickly recover from potential congestion, and this positive effect is attributed to the introduction of the congestion bonus.
The congestion bonus is applied when congestion is detected, and it involves allocating a larger number of slots to the congested node. This allows the node to recover quickly by transmitting its queued packets more frequently, thus reducing the queue size rapidly.

Limits of Distributed Scheduling approach:
Relies on 6P protocol, so the performance of the scheduling function is negatively affected by the network overhead introduced by the 6P protocol:

- 6P transaction experience an avg. delay in the order of seconds
- A significant % of 6P transaction fail
- 6P negotiations increases the network overhead

Autonomous scheduling

One idea to overcome the limitations of the 6P protocol is to adopt a different approach called **autonomous scheduling**. In autonomous scheduling, the communication schedule is computed locally and cooperatively by the nodes. This approach is also decentralized, similar to distributed scheduling, but it differs in that there is no neighbor-to-neighbor cell negotiation and the 6P protocol is no longer used for cell allocation.

In autonomous scheduling, nodes independently calculate the number of cells to allocate based on their own criteria. They also autonomously allocate these cells using a **pre-defined hash function** that is computed based on the node's address.

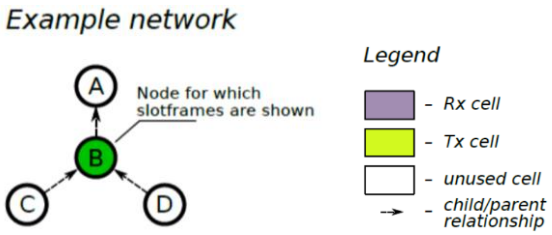
So, two neighboring nodes, with knowledge of only their neighboring node addresses, are able to allocate a certain number of cells in a coordinated manner, allocating the same cells without communication, simply by leveraging the hash function.

What are the pros? *No additional overhead due to negotiation* that implies *no additional delay introduced by 6P transactions* and *no 6P security attacks*.

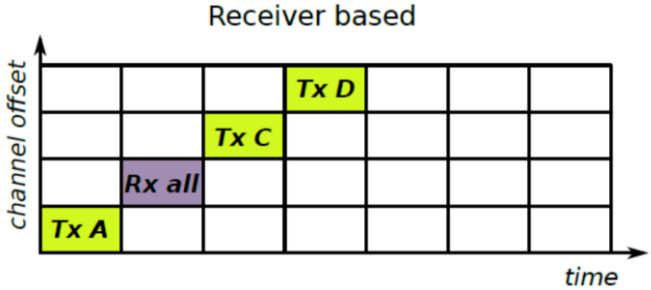
How to allocate cells autonomously?

- Node based approach: Receiver-based approach, Sender-based approach
- Link-based approach

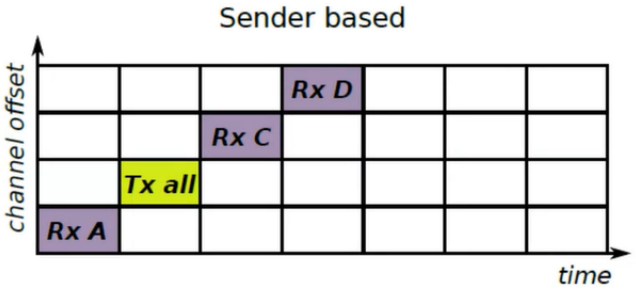
Node based approach



- **Receiver based approach**: with respect to this example, the node *will allocate one single cell to receive packet from all other nodes*.
  In addition to this, allocate also one cell to transmit for each of all its neighbor.



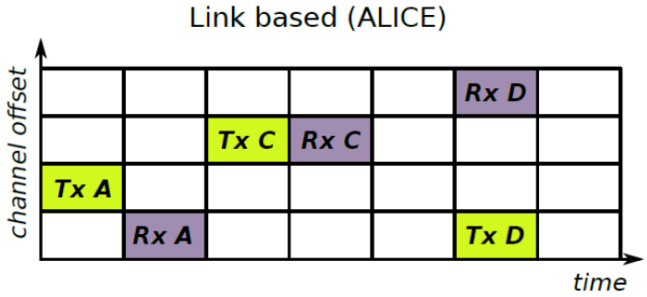
- **Sender based approach**: is the opposite. One cell is allocated for sending to any of the neighboring nodes, and one cell is allocated for receiving from each individual neighbor.



In the ***receiver-based*** approach ***both node, C and D, should use the same cell to transmit to B → collisions.*** This approach is prone to collision, since there is one single cell where the node B is able to receive but there could be several node that want to transmit.
In the ***sender based*** approach ***this doesn't happens!*** Even though node D and C want to transmit, collision will not occur.

Link-based approach

In a link-based approach, the number of allocated cells is determined by the number of links (number of neighbor nodes).



For each link allocate a couple of cell (one for receive, the other one to transmit). A higher number of cells is allocated compared to previous approaches.

When the slot frame size increases, the performance of ALICE deteriorates because the allocation in ALICE is static, meaning it does not vary even if the number of packets that a node wishes to send increases. As a result, as the slot frame size grows, the bandwidth dedicated to a specific node within a given time period decreases.

Performance

Receiver-based generally performs worse: having a single Rx slot for all neighbors implies that all nodes contend to transmit on the same Rx slot → many collisions.

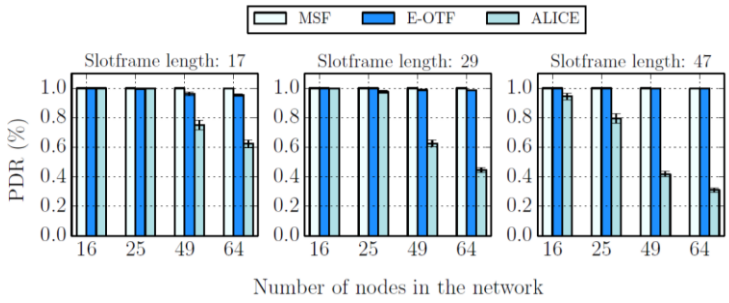
ALICE performs better than sender-based: due to its link-based allocation, instead of node-based allocation. More slots per slot frame are allocated to the same node.

In ALICE, in order to reduce the collision due to the collision of the hash function, a **cell rescheduling** is performed at each slot frame. However, it has been observed that this rescheduling has no significant average-case effect. It may only have a significant impact in worst-case situations, and therefore rescheduling is typically not used in practice.

Comparison of SF

E-OFT and MSF are adaptive: The number of allocated cells is varied, depending on Traffic rate, Link quality, Queue level (E-OTF), Cell utilization (MSF)
ALICE is not adaptive: Fixed number of cells per slotframe, but ALICE with Frame Pending bit is a simple mechanism to make ALICE adaptive. It's a mechanism provided by the underling TSCH access protocol (basically is a Flag in the Frame Control Field of the TSCH frame). When set, the FP bit indicates that the recipient should remain active in the next timeslot and on the same channel unless the next cell is already allocated for other purposes.

Periodic traffic



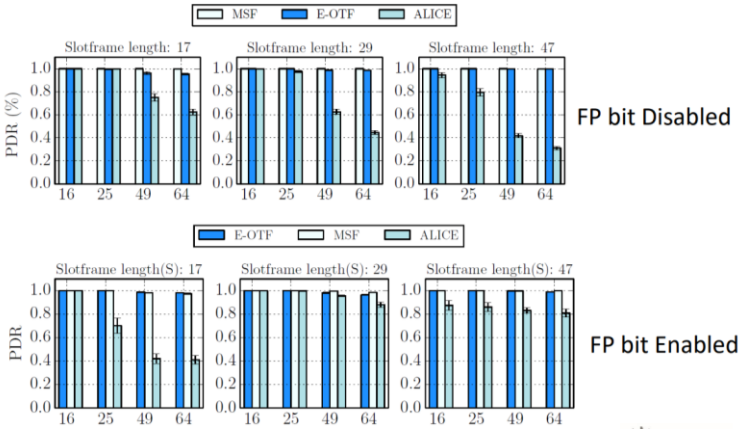
In network not very big, all SF works well, but when the size increases, ALICE performs worse than the other ones, because if the slotframe size increase, it allocates a certain number of cells: if you increase the size of the slotfram, you decrease the bandwidth dedicated for each specific node.
In addition, another consequence of the static allocation in ALICE is that its duty cycle is lower compared to other approaches, which is positive. However, this leads to worse end-to-end reliability and latency.

Bursty traffic

Nodes alternate between Active and Inactive Periods. Traffic is generated only when events occur (event-driven apps): even in this case network not very big, all works well. But when the size increases, ALICE performs worse than the other ones. *E-OTF works a little bit better since it react to network traffic faster than MSF.*

ALICE with FP bit enabled

With the FP bit enable in ALICE, it works better when the slotframe is not so big and not so small, but it's difficult to find the optimal slotframe size in advance.



Lessons Learned

Upward Traffic (from nodes to the root of the DODAG):

- Periodic Pattern - Moderate Offered Load: ALICE-FP is the optimal choice.
  No negotiation overhead, Lower duty cycle, longer lifetime
- Periodic Pattern - High offered load: E-OTF and MSF are mandatory.
  Traffic peaks are promptly handled, due to dynamic cell allocation at the cost of an additional cost in terms of duty cycle.
- Bursty traffic (very dynamic networks): E-OTF outperforms both MSF and ALICE-FP.
  It adapts more rapidly to changes in traffic conditions

Downward Traffic:

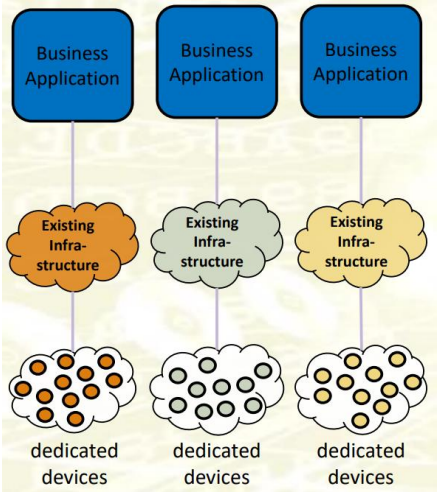
- ALICE-FP is always the best option.
  Slots allocated for upward and downward traffic. MSF and E-OTF optimized for upward traffic. Downward traffic is not so frequent in IoT environments.

Professor Vallati's lecture notes

IoT Cloud Integration

When an IoT system is developed, there are 2 approaches:

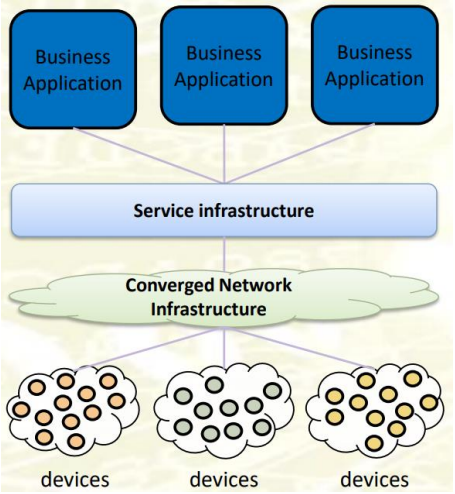
- **Vertical approach**
  - Designed to serve one single purpose.
  - Inefficient: each device is dedicated to a single application.
  - Operating in isolation: no (or very limited) cooperation.



Since we have many products we have many solutions, what are the characteristics of those systems?
Those system have been designed without having interoperability and exchange of data in mind, this means that each one of those systems are usually designed and implemented to be not interoperable at all. The main result of this is that every system is designed to serve one single purpose.

These set of vertical systems that do not exchange data and are not interoperable each other is highly inefficient. Sensors take data but only send it to their system and this is a problem in those situations where cooperation between various sensors can be really useful.

- **Horizontal approach**
  - Converged infrastructure.
  - Unified sensing and actuating infrastructure.
  - Support for multiple applications.



Application are instantiated over the same service infrastructure and can operate with devices. Like smartphone: a set of sensors/functionalities and as service infrastructure the Marketplace, and over it the applications. And then the application can change its behavior based on which devices are available.

This is the vision, since many of the actual IoT system were developed in a vertical way or, however, they are moving towards this direction.

How do we create this converged IoT infrastructure? First thing is to standardize some communication protocols in order to ensure exchange data between devices manufactured by different companies. The idea is to start by creating a converged IoT network based on IPv6 which is at the core of the common stack for IoT communication.

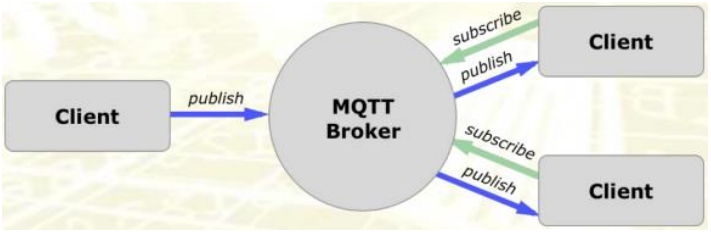
On top this converged infrastructure we need to have an **application protocol** that guarantees that the communication between devices and applications is meaningful, example of these protocol are ***MQTT*** and ***COAP***. In addition to define application protocols that enable applications interact with devices we have another standardization effort that are part of ***IP for Smart Objects Alliance*** (IPSO) that specifies a set of data encoding mechanism and semantic option in order to make sure that the data produced by sensor has the same meaning for all the application or other sensor involved.

Direct Cloud Integration (MQTT)

With the current solutions exploited today, which are mainly Vertical, we can say however that the ***Horizontal IoT is already available***, why? The vertical solutions in the market almost all use the same technology and architecture. The reason is that all the IoT solution providers relies on Cloud providers, in order to speeding up, reduce costs and have a scalable system**.** On the cloud an application is deployed in order to receive and process the data coming from the sensors.
Cloud providers decide to include in their portfolio solutions specifically designed to IoT service. So they choose to sell IoT “packages”: complete IoT systems.

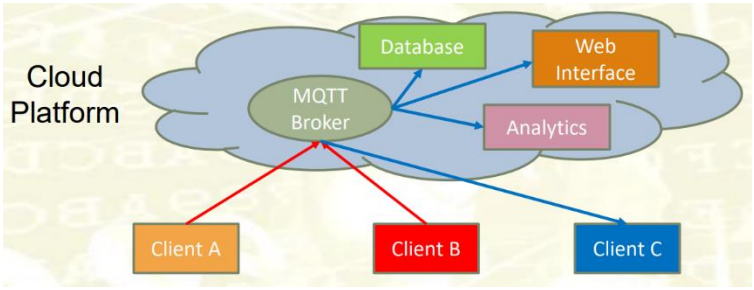
Almost all the cloud providers exploit the same **technology (but those packages are not interoperable with other cloud providers)**.

They are all based on **MQTT**:
MQTT is a ***publish/subscribe messaging protocol*** designed for lightweight M2M communications.
MQTT has a ***client/server mode***l, where every sensor is a client and connects to a server, known as a broker, ***over TCP.***



The **MQTT broker** is responsible for receiving and forward data from/to other devices.
Every message is published to (an address, known as) a **topic**, which is a string that represents the type of data produced by the data producer. The topic is linked with the information produced.
Clients have to **subscribe to topics** of interest to receive information, if needed they can subscribe to multiple topics.
Every client subscribed to a topic receives every message published to the topic.
A client can publish a message on a given topic at any time**.**

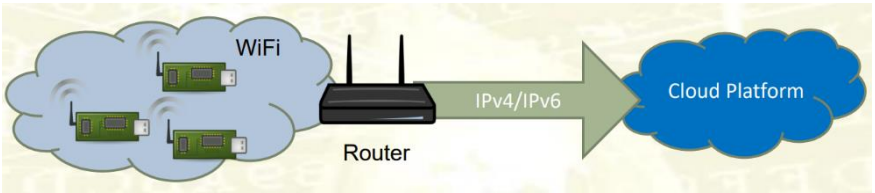
A cloud-based platforms:



- An MQTT broker is instantiated in the cloud (so it have high availability, is scalable, etc)
- Sensors (MQTT Clients) publish on pre-defined topics
- Other sensors (so other MQTT clients) or other modules of the cloud platform subscribe to the topics

The other components of the IoT system is the application. The application would like to receive data from sensors and/or send command to the actuators.
In the cloud application you can have/interact with modules like in the figure above.

Peculiarity of the solution provided by cloud providers is that they provide some kind of packets ready to be used, in order to deploy the IoT solution with minimal changes.

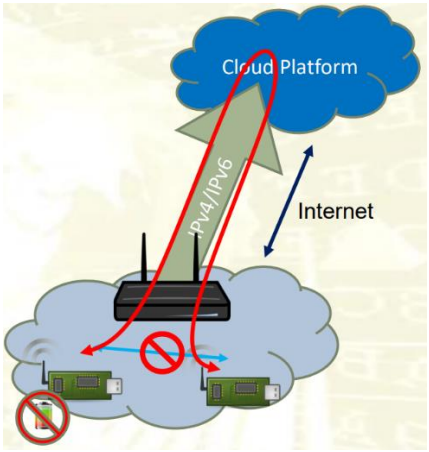


The IoT marketplace have a lot of rapid IoT prototyping platforms, those IoT prototyping platforms are a set of embedded systems produced specifically to support the creation of IoT devices to ease and reduce the time for the development of those IoT solutions. Many of the IoT rapid prototyping platform available today connects via a local Wi-Fi network, which is usually pre-existing *(no new network equipment is required)*. So the cloud platform provide also a skeleton of the code to program a set of popular boards.

PROs:
- Rapid and simple deployment
- It does not require the installation of a new infrastructure, Wi-fi is already there
- It scales with Cloud infrastructure, as the number of devices IoT grows, the number of brokers grows as well

CONs**:**

1. Cloud is always involved:
    - Low latency applications are not supported (main drawback)
    - Persistent connectivity
    - Machine-to-Machine interactions not possible
2. Wi-Fi was not designed with IoT in mind: Energy consuming (no battery powered devices) and coverage equal to the radius of Access Point/Router (wi-fi does not include multi hop communication).



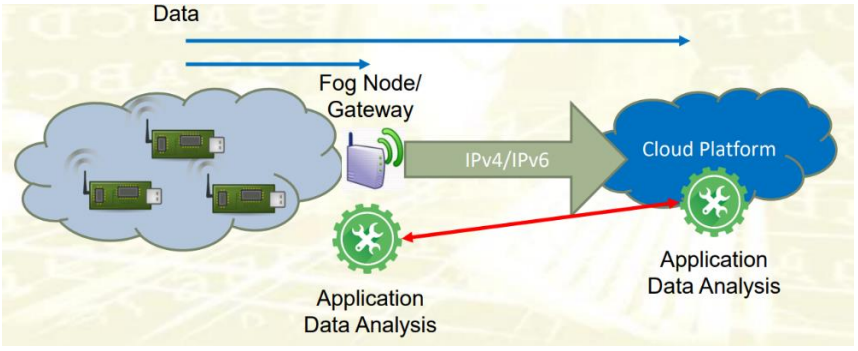
Local vs Fog computing

If we use this approach, we know that for some use case, like manufacturing inspection, autonomous robots and augmented reality it is an issue, since all of them require low latency and M2M communication!
What we would like to have is to go back to the traditional approach: data collected and analyzed by server located on the same location of the user.

Why? Low Latency, beneficial for applications for closed loop control. The communication is direct; M2M interactions are allowed without involving the Cloud and requiring persistent connection; Loss of connectivity still ensure system functionalities, is all local, so no problem.
But we cannot simply go back: Cloud as too many benefits! So ***Fog (Edge) Computing***!

Fog computing is a trend that expand the centralized cloud computing architecture in order to introduce intermediate computing layer between the iot devices and the cloud platforms. The idea is to introduce a middleware layer called Fog Layer to execute locally a part of the application and data analysis process that were executed only on the cloud platform. Fog Layer is composed of one or more Fog Node that are in direct communication with iot devices thank to the fact that they are usually installed on the same LAN.

So the idea is to create a multi-layer computing architecture in which you have at least 2 layer: the cloud layer and the fog layer.

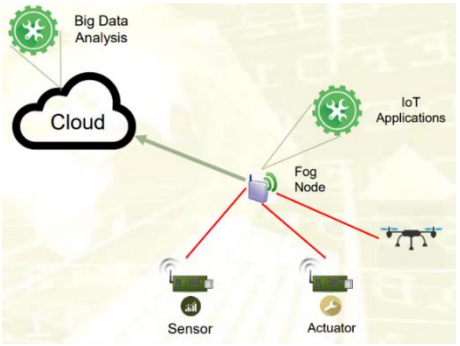


Advantages

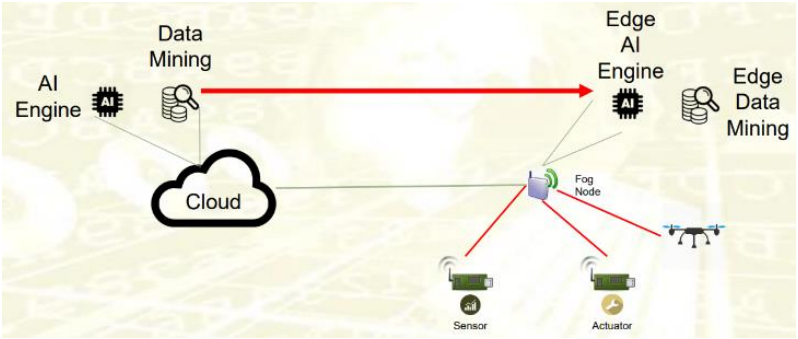
Executing applications and analyzing data analysis in proximity of the cyber-physical systems ensures:

- Low-latency
- Resiliency to network outages: the system still works even when network interruptions occur
- Privacy: data is not offloaded to external systems

Meanwhile advantages of cloud computing are preserved: data (or a compressed/reduced set) can still be uploaded to the cloud for big data analysis and historical data collection.



Fog systems are characterized by limited computing and storage capabilities, however, they support the implementation of simpler Data Mining e AI techniques to analyze data in proximity.



The fog layer is not a separated layer, but it's integrated in the cloud system!
CC is not only scalability, CC has many advantages and many of them come from virtualization techniques, in order to preserve those advantages also fog computing system adopts the same virtualization technologies, in particular they adopt lightweight virtualization technologies (like containers, docker, …) in order to meet with the reduced capabilities that those embedded systems provide. The result of this virtualization is that application developers for fog computing system can abstract from the hardware and system components to ensure compatibility and rapid deployment.

An application can be moved from the cloud to the fog or vice-versa since the environment is the same thanks to the isolation provided by lightweighted virtualization.

Where are those fog nodes placed? Fog nodes can be deployed at various points in the network chain leading to the Cloud provider, but the most common solution is to place them directly on the gateway. By replacing the traditional internet router/gateway, fog nodes offer additional advantages such as the ability to support multiple network protocols, not just Wi-Fi. These new gateways can implement protocol translation functionalities, allowing them to provide connections not only through Wi-Fi but also utilizing protocols like IEEE 802.15.4, for example.

The result is that this multi-layer architecture of Fog computing facilitates the interaction of various wireless technologies, including those discussed with Professor Anastasi. The new generation of gateways is responsible for providing connectivity to local devices and enabling the integration of new wireless technologies. Additionally, these gateways can incorporate security functionalities to regulate access to IoT devices from the external Internet.

Web of Things

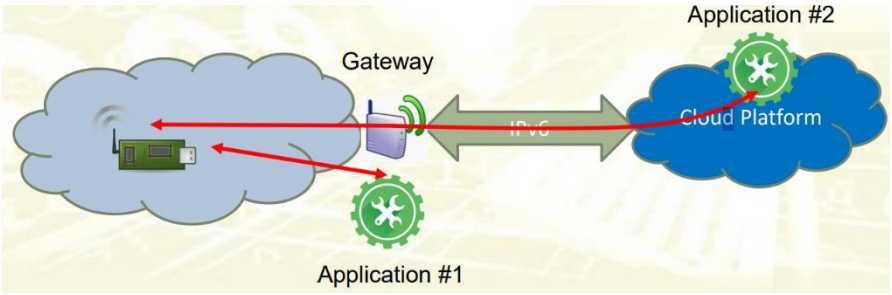
We have discussed in the previous lesson the direction in which the IoT world is heading. Now let's return to the currently available solutions, specifically focusing on MQTT, which is widely adopted today. We need to assess if MQTT is a good fit for this multi-layer architecture. Initially, MQTT was considered the best option for centralized cloud-based solutions, where everything was integrated in the cloud, including the MQTT broker. However, this centralized MQTT architecture may not be well suited for a multi-layer architecture, where the application logic can be distributed across the cloud and fog layers.

MQTT does not fit well in this new architecture. The main problem is where we place the broker?

- Deploy on the cloud? No, because we will still have long delay
- Deploy on the gateway? The application run on the gateway: we resolve the latency problem, but applications are running on the cloud for several reasons, for example scalability… we have scalability issues with this solution! Why? We can have a lot of applications that want to communicate with that specific broker.
- Use two broker: one on the cloud one on the fog node. Extremely inefficient, especially when we talk about data flow: to which broker must send its data?

A Web of Things approach instead, fits well in this architecture: it **shift the focus on the IoT devices** and putting it at the center of the whole architecture.
Every IoT device provides an interface to expose their services to applications: the interface exposed by the devices is invoked directly by applications when needed. The operations are performed in the same way, from the gateway or from the cloud. So, in the Web of Things paradigm, applications interface directly with IoT devices instead of relying on an MQTT broker.

IPv6 enables communication between devices and applications in the cloud. With its large addressing space, IPv6 allows devices to have unique, globally reachable addresses.



Advantages:

- Simple: a *broker is not needed*
- *Energy Efficient*: data is transferred only when needed (if no application consumes the data it is not transferred), meanwhile, In the MQTT architecture, data was transferred from the sensor to the broker even though there was no subscriber.
- *Common interface*: data retrieval or function invocation are performed in the same way on different hosts
- Better integration with the Web since the web of things adopts the same approach

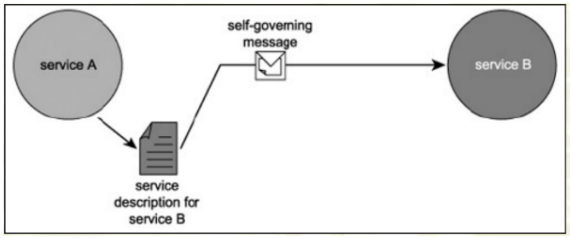
Service Oriented Architecture

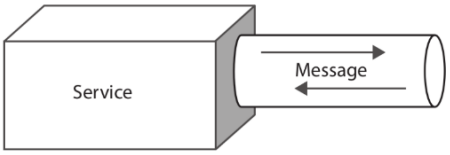
The Web of Things is characterized by a **service oriented architecture**: without the services we have one, complex approach, limit to expansion of those applications.
Why? Tightly coupled approach within components, so you cannot create and maintain in a easy and scalable manner due to the fact that small changes on the system will have a cascade effect on all the other parts of the distributed system. In order to overcome this limitation a different approach was considered, based on the **loose** coupled distributed systems where the components of the distributed system were only paired on a loosing manner.

The new idea was to conceive a different programming paradigm were the components of the system was only slightly correlated each other in order to ensure the possibility to change the implementation of a module without triggering a cascade effect. A different programming paradigm was created: service oriented programming paradigm.

Service := basic computing module (of a certain program) that encapsulates a self containing application logic.

The idea is to keep *isolated as much as possible different components.* Service are aware of each other through the use of service descriptions and exchange information through messages.



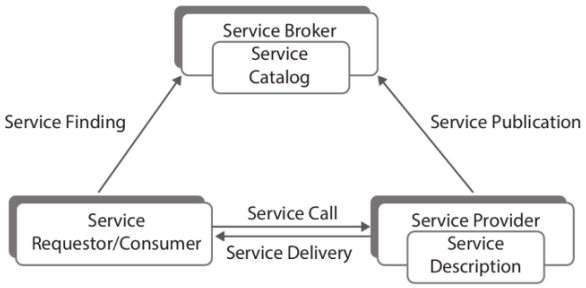


In the Service Oriented Architecture applications ***are built as a collection of services (or microservices) from existing software components.***
Services within the application communicate via ***messages*** *passing*: each message has a ***specific schema that defines the format and the exchange of information.***
In order to ensure interoperability, the structure of the message, must be defined in a very precise manner.
Every message is received and then handled by the service in a manner completely hidden from the exterior.

In a SOA interaction we have two roles: a ***consumer*** service and a ***provider***.

- A consumer service sends a ***service request*** to a provider service via message passing
- The provider return a ***response to the consumer containing the expected results***

A service can always be a consumer, a provider, or it can be both, thus implementing one role or the other depending on the specific situation.



In general service providers and consumers do not interact directly with each other initially, as the consumer should know the address and the descriptor of the provider: An additional software module is usually employed in the middle, a ***broker.***
The broker is responsible for creating a service catalog (or registry) in which all the available services are enlisted

A client service can discover a provider service only if the interface is published to the broker: after the consumer queried the broker for a specific service and obtained its identity, it can interact directly with the provider. The address and the descriptor of the broker are well known.

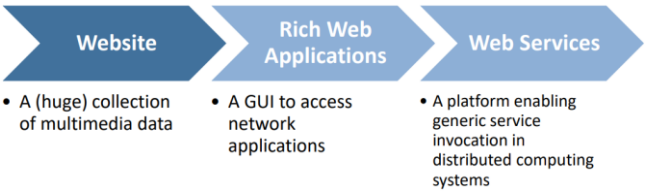
Advantages

The ***implementation of services remains hidden***, thus making possible to fix the implementation without affecting other services as long as the service declaration/interface remains intact.
This separation enables a ***loose coupling*** feature that ***allows to plug and unplug specific services from an application effortlessly.***
This feature makes the system **flexible to changes and expandable**, i.e. new features can be added effortlessly in short amount of time.
SOA services are perfect to adapt to the scalability of the cloud infrastructure.

World Wide Web

Everything started with the World Wide Web technologies, before moving to web services the web itself originally included several mechanism, in particular three of them:

- identify a resource over the internet in a unique manner
- to exchange data between entities
- a language that defines how information should be encoded to achieve interoperability



Web services is the heart of a web application, is what stays below the graphical user interface.
Now let's see now how the three mechanisms that we talked about before are implemented.

URI- Uniform Resource Identifier (to identify resources)

This is a mechanism to identify object or resources on the internet.
"is a compact string of characters for identifying an abstract or physical resource"

```
<uri> ::= <scheme>":"<scheme-specific-part>
```

URI are also the URN (**Uniform Resource Name**), i.e.: urn:isbn:0-395-36341-1, and also URL.

HTTP (to exchange data)

The second mechanism is a protocol for ensuring interoperable communication across different software components, the HTTP is an example of this type mechanism created for the web which allows communication between web server end client.

HTTP := "… is an application-level protocol for distributed, collaborative, hypermedia information systems".

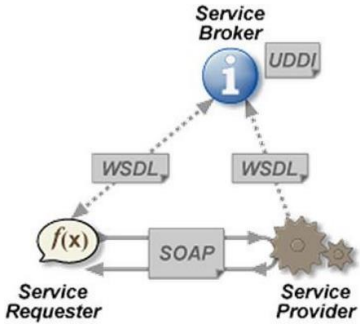
XML (to encode data)

In the web the data encoding part was standardized as **HTML** which is a markup language specifically designed in order to encode data about web pages (layout, information, images, video, …). This standard was too much specific for the web, so it is not a very good encoding language for web services which requires something more general, for this reason, the eXtensible Markup Language (XML) has been introduced.

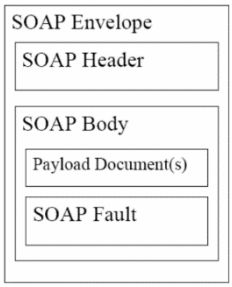
Defines a set of rules for an encoding markup language that is both human and machine readable.
XML specifies the **syntax**. XML is **extensible** because the tag set is not pre-defined.
XML is in a **hierarchical** structure. An XML **element** is a document component inside a begin tag and end tag.

Web Services (SOAP and WSDL)

How do we exploit all these components for creating web services according to the service-oriented architecture?
The first component we require is a component to specify the interaction between consumer, producer and broker. In order to standardize interaction with a broker the Web Services Description Language (**WSDL**) has been chosen.



Each Web Service must have an interface described in WSDL. Via WSDL each service provider register itself to the broker, while the service consumer uses WSDL to look up for a specific service.

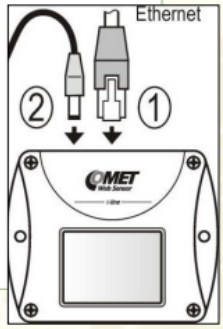


After this initial phase where the consumer look up for a functionality, we have a direct interaction between the producer and the consumer and in web services that happens with another language named Simple Object Access Protocol (**SOAP**), that provides a standard messaging format.

SOAP provides a standard packaging structure for transmission of XML documents over various internet protocols (like HTTP). The structure depends on the type of the application.

A **SOAP message structure** is called **envelope** and is made of a header and a body. The **header** contains security and specific information, while the **body** includes the message payload (request or response) and a faults and errors section.

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <InsertTx5xxSample xmlns="http://cometsystem.cz/schemas/soapTx5xx_v2.xsd">
      <passKey>13960932</passKey>
      <device>4145</device>
      <temp>1.4</temp>
      <relHum>91.9</relHum>
      <compQuant>0.3</compQuant>
      <pressure>-9999</pressure>
      <alarms>hi,no,no,no</alarms>
      <compType>Dew point</compType>
      <tempU>C</tempU>
      <pressureU>n/a</pressureU>
      <timer>60</timer>
    </InsertTx5xxSample>
  </soap:Body>
</soap:Envelope>
```



Web Services for IoT

This web services was the playground were initially the iot was designed, so when the iot paradigm was born the web services realm was the realm that was predominant in the world of information technologies. The first design of iot system was to actually adopt this kind of solutions for web services on the realm of the iot.
So the idea was actually to adopt this package of solutions URI, HTTP, SOAP and WSDL also on the iot world. The iot devices had to expose their functionalities and their data as web services, so we had those iot devices implementing the SOAP protocol to interact with other services and the WSDL to publish in a broker their services description. So other services could interact with the broker and retrieve the services description and then eventually interact with the device, in the same exact manner cloud services would interact with other services running in the cloud.

The type of interactions that these iot devices (that are supposed to be simple and to run on constrained hardware) required to implement too many functionalities in order to be integrated into a full web services architecture.
Fortunately a new concept for programming model for service oriented architectures was created and his name is REST.

REST

The REpresentation State Transfer is a revisitation of the service oriented architectures in order to ensure an architecture that is more lightweight, more simple and with a reduced set of functionalities.
The idea behind REST is still the same of web services since it is a resource oriented architecture for distributed systems where the URI is still adopted and where the HTTP protocol is still adopted as communication protocol across different services.

The main concept that changed in REST compared with traditional web services was how services and applications invoke other services provided by other systems, the idea of REST was to simplify significantly how one service invoke a functionality or retrieve some data from another service. The idea is to have **services not invoke other services but invoke resources** and a set of functionalities associated with those resources.

REST replaced both SOAP and WSDL protocols

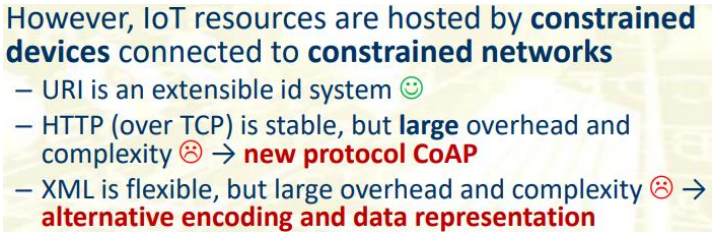
Four standard **action** (CRUD) were defined to be invoked on a specific resource exposed by a specific service. Those four CRUD actions were mapped into the four HTTP **methods**.

Interaction is realized by **four standard actions (CRUD)**
- **Create** a new resource from the request data
- **Read** a resource state
- **Update** a resource state from the request data
- **Delete** a resource

HTTP methods associated to actions
- **POST**: Create a new resource from the request data
- **GET**: Read a resource
- **PUT**: Update a resource from the request data
- **DELETE**: Delete a resource

The content of data is still transmitted using XML as part of the HTTP content, but the additional markup required by SOAP is removed.

REST allows many standard formats like XML, JavaScript Object Notation (JSON) or plain text as well as any other agreed upon formats for data exchange.
XML is still useful to guarantee interoperability (check for example correctness of the document), but for SOAP it's too much, so JSON is more suitable for it.



Constrained Application protocol (CoAP)

CoAP is a web protocol for the special requirements of constrained environment (LLNs).
It could be seen as an evolution of HTTP, in fact, its purpose is precisely to replace the HTTP in web of thing architecture. The basic idea is the same, but reverse, since the sensor nodes are the "server". HTTP is not usable since is too heavy for constrained devices, and moreover doesn't consider the M2M communication.

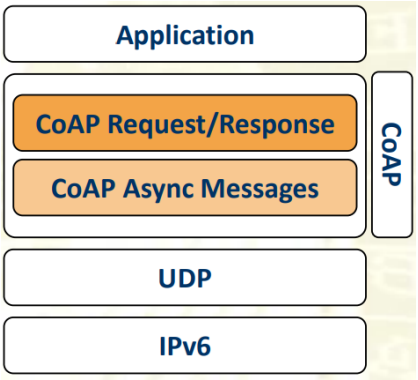
The goal was to realize an application protocol for web services (common with HTTP) but optimized for M2M applications, CoAP is an evolution of HTTP (not just a compressed version), in fact is based on HTTP.
Each node has an IPv6 assigned and they are reachable from outside the network (so the application, in fog or in cloud, can use the interface exposed by the sensors).

Main features of CoAP:

- It use ***UDP*** as transport protocol, TCP is too complex (reliability, duplicated detection and congestion controls make TCP not suitable, too many variable and operation to perform)
  Some of those functionalities will be included in CoAP
- ***Request/response*** communication model with support for asynchronous message exchanges (this is new)
- ***Low header overhead and parsing complexity.*** Thanks to binary version format, CoAP messages are easy to parse. The length is fixed so it can use macros and mask.
- Simple proxy and caching capabilities.
- Stateless HTTP mapping
- Support for multicast communication
- Support for discovery of resources

Coap interaction model:
Still preserve the request/response of the HTTP, this means that in the client-server communication, the client sends a request and the server reply with a response.
Those request and response are exchanged in an ***asynchronous*** manner with ***optional reliability***.
The reliability of this message exchange in not provided by TCP (as it was done in HTTP) therefore an optional reliability is introduced at the application layer through retransmission and acknowledgment. There's a set of application in which reliability is crucial, this is done by defining 4 different type of messages:

– Confirmable (CON)
– Non-confirmable (NON)
– Acknowledgement (ACK)
– Reset (RST)



In case optional reliability is enabled only confirmable message are transmitted, meaning that request and responses are sent as **confirmable** messages. A confirmable message must be acknowledged by the receiver, the sender implements a time-out mechanism to retransmission.
If reliability is disabled, request and responses are sent as **non-confirmable** messages which do not require acknowledgment.

The CoAP layer is usually divided into two sub-layers:

- **CoAP Request/Response**: handle the delivery and parse of the messages
- **CoAP Async Messages**: takes care of handling retransmission

Request/Resp protocol:

Same model as HTTP (client/server).

- every **Request** is characterized (as in HTTP) by:
  - Method
  - Request URI, to identify the resource on which the method is invoked
  - Payload and optional meta-data (CoAP options)
- the **Response** includes:
  - Response Code, specifies the outcome of the processing of the request
  - Payload and optional meta-data, which are the data created by the process

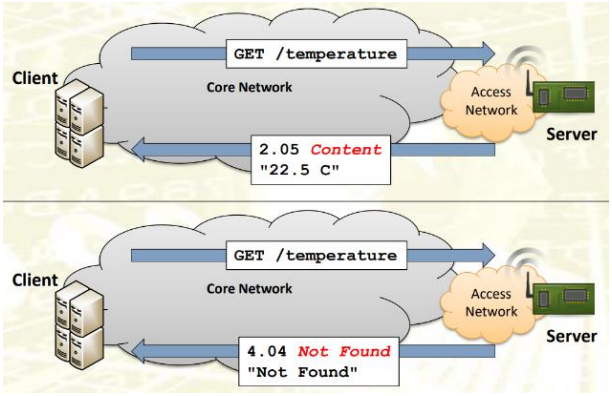
It use also the same **methods** oh HTTP (get, post, put or delete):

- **GET**: retrieves a representation of the state of the resource (safe and idempotent).
- **POST**: requests that the representation enclosed be processed (neither safe nor idempotent). It usually results in a new resource being created or the target resource being updated. May be used for configuration purposes.
- **DELETE**: requests the resource to be deleted (not safe but idempotent)
- **PUT**: requests the update of the status of an existing resource with the enclosed representation (not safe but idempotent). Is generally associated with an actuator (get associated with sensor).

Safe := it doesn't change the internal state of the system
Idempotent := doesn't change the response, so if I call a function of this type multiple times I will always get the same result

Response codes:

- **Success (2.xx)** – the client request was successfully received, understood, and accepted
- **Client error (4.xx)** – the client seems to have erred, request erred, or request for something not there (like 404)
- **Server error (5.xx)** – the server is aware that it has erred or is incapable of performing the request



Request/Response transport

CoAP is bound to non-reliable transport (i.e., UDP). A lightweight reliability mechanism is needed:

- Stop&Wait retransmission with exponential back-off (in order to implement a very simple a congestion control, the exponential back-off is used). Adopts a retransmission and acknowledgement managed with a time-out.
- Duplicate detection: detection and management of duplicate packets during data transmission is performed by observing the Message ID of the various messages.

Requests and responses are carried by CoAP asynchronous messages, which implement optional reliability.

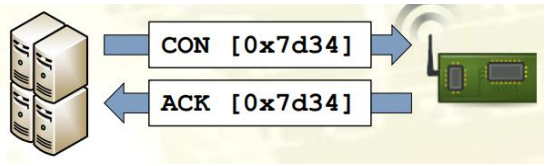
Reliable transmission

Reliable transmission in CoAP is implemented trough CON messages, so if an application wants to have a reliable communication between client-server, the application must be configured in order to send out their request and responses as CON messages.

CON messages are received by the server and by the client and explicitly acknowledged by sending back an acknowledgement message. Otherwise if the server doesn't want to receive that message it must reject it by sending back a **RST message**, because just ignore the message is not a good idea since the sender may send back some retransmissions of the same message.

If a confirmable message is **not acked** in a certain timeout, the sender will send the first message again. It will resend a messages at exponentially increasing intervals (2s default start ack timeout) until it receives and ACK (or RST) or it runs out of the attempts.

CON and ACK messages are matched by the same *16 bits* **Message ID**.



Unreliable transmission

Unreliable transmission is initiated by sending a **NON message** – e.g. regularly repeated readings from a sensor. The recipient MUST not acknowledge the message.
At the CoAP level, there is no way for the sender to detect if a NON message was received or not!

To detect duplications, the **Message ID** is still included in NON messages.

Interaction Model

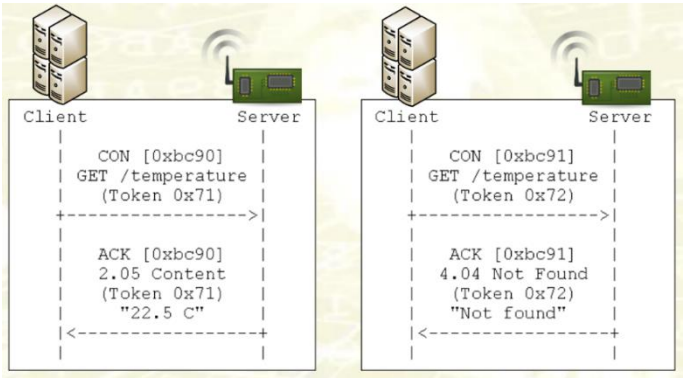
Requests are carried in CON or NON messages: a ***token matches responses to requests***.
There's no relationship between Tokens and Message IDs.

If in a CON message:

- Responses may be **carried in** the resulting ACK messages (piggy-backed mode), *synchronous one.*

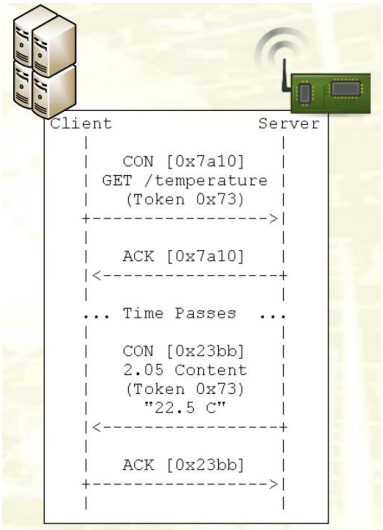
  In this case, the request and response share both the same token and the same message ID.

  [Potrebbe mancare un ACK di ritorno]



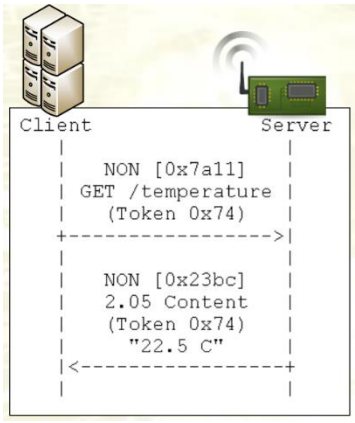
- Responses could be carried in a **new** CON message (separate mode), *asynchronous one*. Should be adopted if the server use a lot of time to generate the response. Why? Because otherwise, due to the "congestion control", the 2s expire and after a backoff, messages are sent again and it generated useless traffic

  The message ID is new because it is a new exchange of messages however the token of the response is the same of the request.



If in a NON message:

Responses are sent in a new NON message.
This new NON message will have a different message ID, however the token remains the same because it links together request and response.



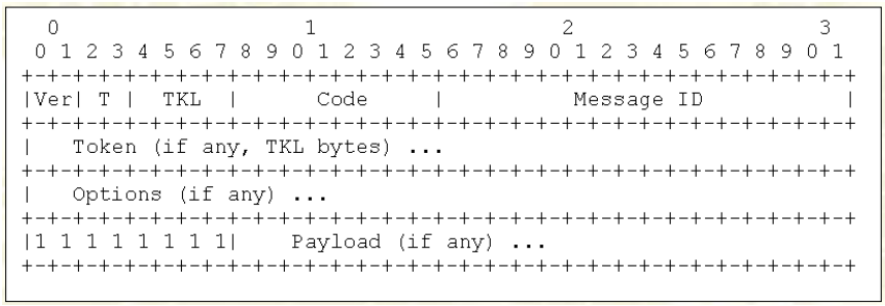
Empty messages

An empty message contains only the header. They are usually used to send ACK or RST, meanwhile.
Otherwise, CON empty message are allowed in order to implement a function named **CoAP ping** that is used only to check if a specific server is up and running. So you can send an empty CON message to a server and if an ACK is send in response this means that the server is running.

Message format

Another significant difference between CoAP and HTTP is the format of the messages. CoAP adopt a binary format to minimize the size of the messages, HTTP instead adopt a text format.

The header of the CoAP message is of a fixed size *4 bytes*.



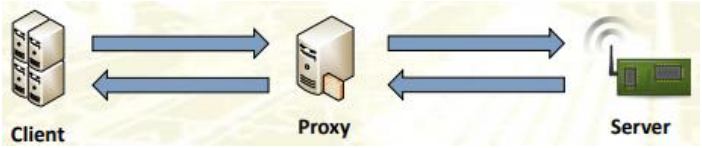
- Version: currently 1
- Type (T): message type (0 CON, 1 NON, 2 ACK, 3 RST)
- Token Length: 8 bytes at most
- Code: split into 3-bit class and 5-bit detail (c.dd)
  - Class: request (0), success response (2), client error response (4), server error response (5)
  - Detail: specifies the method (e.g., 0.01 is a GET) or the response code (e.g., 2.04 is 'Created')
  - Special code 0.00 indicates an Empty message
- Token
- Options (if any)
- Payload (if any)

Proxing

CoAP allows (like HTTP) for the introduction of intermediaries.

Intermediary: endpoint that can be introduced in the middle between the CoAP client and the origin server, in order to implement some additional functionalities that cannot be implemented on the server or the client. It behaves as client towards the origin server and as server towards the client.
Endpoint: any device that implements even partly the CoAP functionalities
Origin server: is the actual server that produces the request

One type of intermediary defined in CoAP is the **proxy CoAP**. A proxy is an intermediary concerned with receiving the request from the client, forwarding the request to the server, receiving the response from the server and forwarding it to the client.

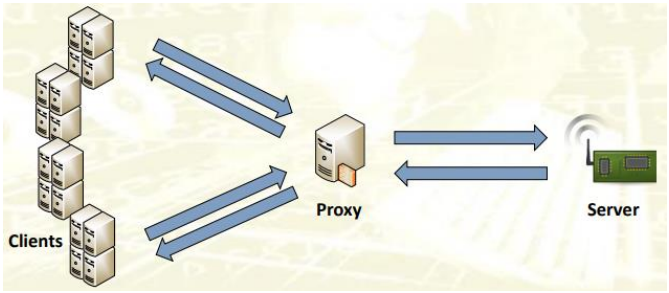


This proxy is placed here to perform additional functionalities like caching, protocol translation, protocol translation or security functionalities rejecting unwanted requests by not forwarding them to the server.
Caching is the most important functionality, this allows to save bandwidth and energy.

Two different options for the proxy are introduced in CoAP specifications:

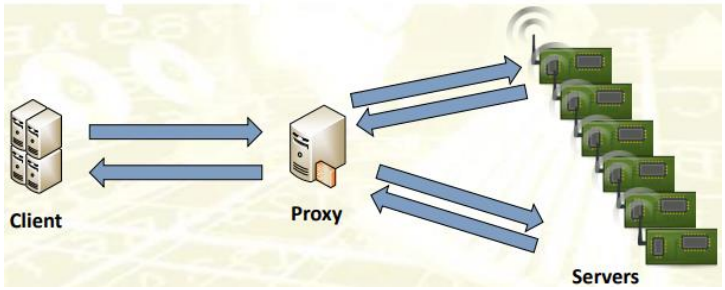
1) **Forward Proxy**

Performs requests on behalf of the client. Clients issues their request directed to the origin server, this request however is sent by the client to the proxy, this forward proxy receives the request and issues the request to the origin server. The origin server send back a response which is forwarded to the client.
The endpoint (the server) is selected by the client since he have to specify to full URI. In this process, the client is fully aware that a proxy is present on the middle, in fact, the client has the flexibility to choose whether and which proxy to use for communication.
This configuration is usually exploited when there are some functionalities on proxy, like protocol translation.



2) **Reverse Proxy**

Satisfies requests on behalf of one or more other server(s). Is installed to provide coverage for a network of sensors or servers, and its purpose is to handle requests on behalf of one or more servers. In this case, the proxy is completely transparent to the client.
The proxy exposes the resources of the servers as they were its own. Client interact only with the proxy (but they are have no knowledge of this) and the server only exposes the functionalities through the proxy. Usually adopted when the architect wants to hide completely the architecture of IoT devices.

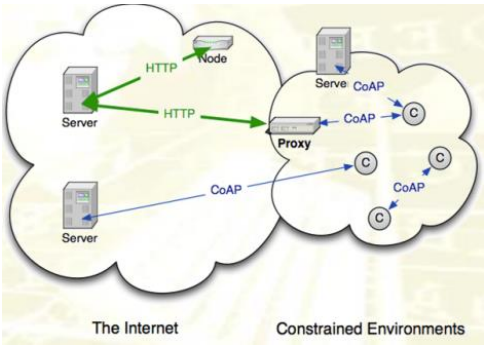


Proxy translation

One of the functionalities that could be implemented by a proxy is a protocol translation functionality.
The proxy can implement the **Cross-protocol proxy** (**Cross-Proxy**): a proxy that translates between different protocols.

The CoAP specification specifies two different cases for protocol translation proxy and it specifies how that should be implemented in practice.

- **CoAP-to-HTTP proxy**
  Is a proxy that is installed on a CoAP network to have the CoAP clients (sensors) interacting with HTTP servers.
- **HTTP-to-CoAP proxy**
  Is a proxy that translates HTTP requests into CoAP request, so it can be used to have regular HTTP clients to interact with CoAP servers.



Usually proxy are both of them at the same time, to allow devices that send a request to receive a response.

CoAP – Discovery

HTTP includes a mechanism to discovery resources hosted on web servers. We need a similar mechanism in CoAP, to discover all the servers available in a specific network.
This functionality is implemented by defining a mechanism based on **multicast**. A multicast CoAP request is sent to the network and this request is directed to a specific IPv6 multicast address named "All CoAP Nodes". If this message is received by a server, it has to process this message and reply to the client in order to acknowledge his presence in the network. In addition to allocating a multicast address, a specific port has been allocated for this type of messages: 5683.

Once the servers are discovered, the client wants to know the resources hosted by every server, for this reason has been defined a new discovery mechanism named **resource discovery**. It allows to get the list of the URI of the all the resources hosted on servers and a very short description describing the type of the resources, the functions and other details…, described in a standardized language named **CoRE Link Format**.

The client can retrieve the list of resources and the description by invoking the GET method on a specific URI: `"/.well-known/core"`.

CoRE Link Format
Link format is very similar to HTTP header values.

example: `</sensors/temp>;if="sensor"`

Collection of links (separated by commas). Each link is described by: a relative URI and a number of attributes.

Attributes:

- **Resource Type ('rt')** – Opaque string used to assign an application specific semantic type to a resource
- **Interface Description ('if')** – opaque string used to provide specific interface definition used to interact with the target resource, e.g.: the URI of a Web Application Description Language
- **Maximum Size Estimate ('sz')** – an indication of the maximum size of the resource representation
- **Content Format ('ct')** – provides a hint about the Content-Formats this resource returns
- **Observable ('obs')** – a hint indicating that the destination of a link is useful for observation

The resource "`/.well-known/core`" can be used not only to retrieve the list of the resources but as an entry point to perform a query on the list of the resources available on a specific server.

example: `GET /.well-known/core?name=value`

CoAP – Resource observing

The idea behind resource observing is to enable to observe the status of a resource for an indefinite amount of time. This functionality was not present in HTTP protocol because you didn't have the need to inspect a resource for a long period, but in CoAP instead it can be very useful. In order to support that, a new method was added to the traditional 4 and this is the ***observing method***.

A first way to do it is the **Continuous Polling**, the client periodically send out a GET request to the server.
This solution is not efficient due to the overhead, furthermore, the period of the request is decided by the client which is not fully aware of all the implications like the status of the physical resource, so the timing wont be accurate.

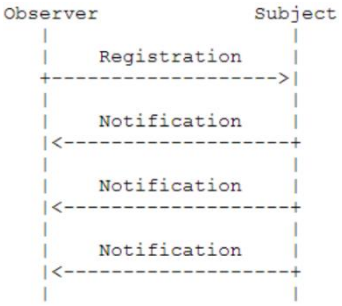
The **Observer Pattern** approach solves all the problems present in the previous option. This update pattern is inspired by the PUBlish/SUBscribe interaction that was predominant in other IoT protocol (like MTTQ protocol).
The idea is to have a mechanism trough which the client signals an interest on the updates of a specific resource to the

server, after that the server is aware that an unsolicited notification must be sent back to the client every time the physical measurement linked to that specific resource changes.

Observer Pattern – protocol overview

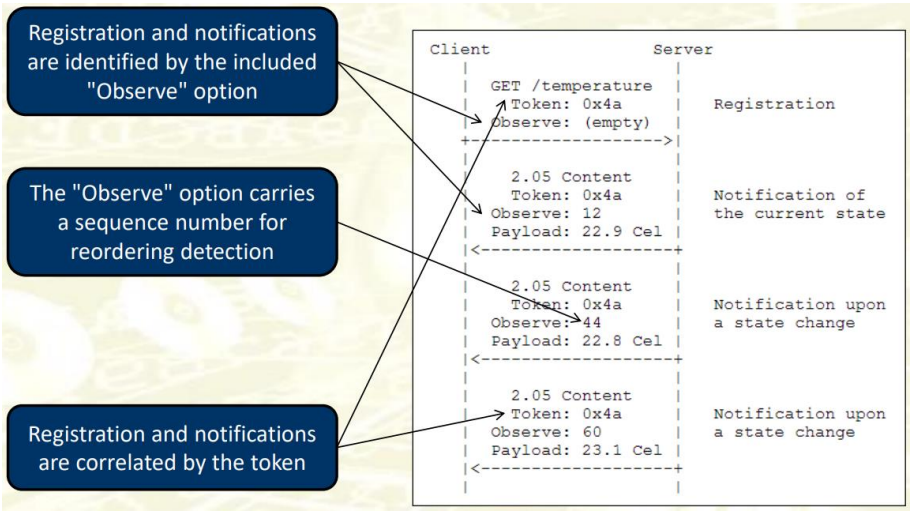
The Observer method introduced in CoAP has been inspired by the observer design pattern that allows a provider of information to store a list of observer for a specific resource and that list is used to send unsolicited notifications.

We have an **observer** (a CoAP client) and a **subject** (a resource on a CoAP server). The observer send a "registration message" to manifest his interest on a specific topic, that in case of CoAP is a resource, so the observer is added to the list of observers and every time the resource changes a notification message is sent.



The "registration message" is in the form of a **extended GET request**, that include the observing option on the header.
The notification instead is a simple CON or NON message, that is sent from the server to the client and that include the new status of the resource.

example:



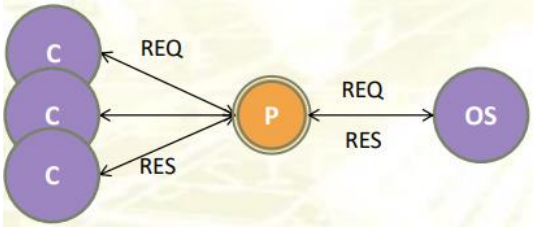
Not all the resources exposed by a server could be observable, so is up to the programming of a server to define if a resource is observable or not. If a resource is observable the server is also responsible to define, whenever the state of the resource changes, when a notification should be send out in a way that is useful in the application context.
For **example**, in the case of a temperature sensor, we can send a signal every single time the temperature changes or we can send it only when it drops below a certain threshold that can be set by the client through a specific query.

Otherwise, the server can choose to send a notification periodically according to a certain time interval, in this case it is up to the server to choose the interval. It is important to underline that unlike Continuous Polling, here the interval is chosen by the server itself which is the only one who can know the internal dynamics of the measurement.

The observing relationship might lose a little bit consistency so the client and the server view might be out of sync due to latency, lost and/or reordered notifications, ecc… In these cases a threshold "**max age**" is used.

Observer Pattern – with proxy

Observing can also be used with **proxy**. In this case, the proxy will issue an observer relationship with the origin server. That observer relationship is not simply forwarded as it is to the orgin server, but instead is processed by the proxy and this latter issues by itself another relationship with the orgin server.



Multiple clients can issue it's observing relationships with a proxy, but the proxy eventually issues only one observing relationship with the origin server, to achieve scalability.

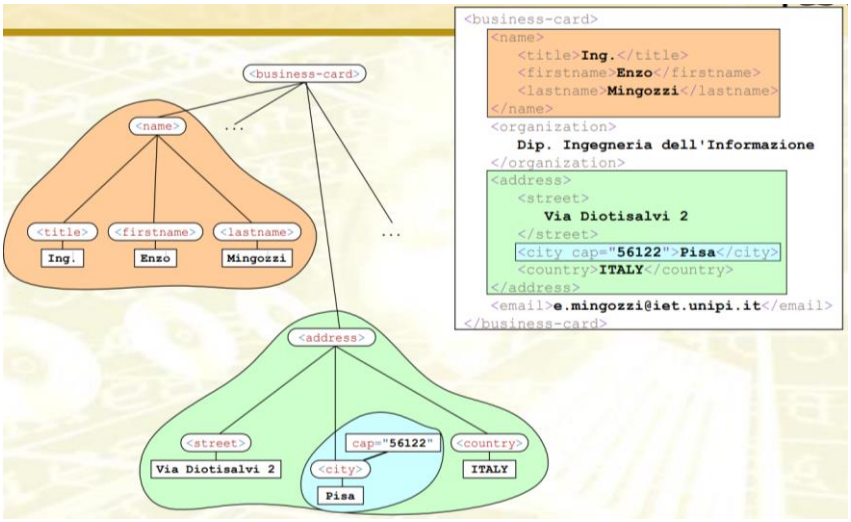
So in case of proxy the observe relationship is broken in two pieces, one between the clients and the proxy and another between the proxy and the origin server. Every time a notification is sent from the server to the proxy this notification is send out to all the client that are subscribed. Consequently, if there are many clients that want to observe the same resource, placing a proxy in the middle is a good idea.

Data encoding

The picture that we have until now is that clients interact with IoT devices through the Web of Things paradigm, so the iot devices behave as web server, on the other side we have the applications that interact with them in the same paradigm adopted for web services.
At the core of this interaction we have one protocol: the HTTP or his constrained version CoAP, in which request and responses are allowed to specify a payload. The payload on the request might contain parameters, value to set or updates to be sent, on the response the payload might contain the current representation of the resource of the result of a computation. In order to encode that payload you need an encoding language and that's why different encoding languages have been introduced.
The main solution adopted by web services is XML, although another solution JSON was introduced recently, but it is not properly related to IoT devices. New set of encoding languages is EXI, specifically designed for IoT (a reengineering of XML and JSON in order to minimize complexity and overhead produced).

XML

An XML document, which stands for Extensible Markup Language document, is a structured text file that stores data in a hierarchical format and is organized in entities (highlighted by tag), which should represent something real. Every entity can be organized in order to have entities inside.



Element

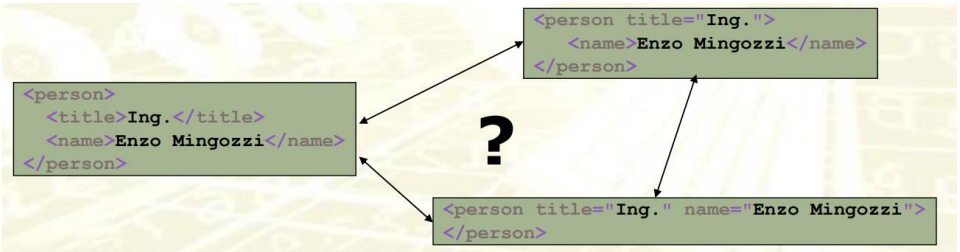
The basic building blog is an **element**. An element consists of:

- A start tag: <nome>
- A content
- An end tag: </nome>

"`nome`" is the element type: it can be of any length, must start with a letter or an underscore and each start tag must have its corresponding end tag.

Attribut

Each element can have one or more ***attributes:*** name/value pair associated to an element.
`att-name = "att-value"`. The use of attribute is unregulated:



Entity references

An entity is a well known XML document fragment with an assigned name.
e.g.: "3,14159265" with name PiGreco → `&PiGreco;`. Entity reference syntax: e.g: & for char &.

CDATA sections

CDATA (Character Data) sections are used to escape blocks of text that may contain special characters or reserved symbols that would otherwise be interpreted as markup.
Into CDATA any kind of data can be stored. To skip XML syntax rule application on a piece of text

```
<![CDATA[
    if ( this->getX() < 5 && v[0] != 3 )
    cerr << this->displayError();
]]>
```

Well-formed documents

An XML document is well formed if:

- One and only one root element (the document element)
- Elements must be correctly nested
- End tags names corresponding to start tags
- Unique attribute names

Since XML documents usually combine different languages, conflicts may arise in element name: a **name space** uniquely identifies a collection of elements (e.g. the XHTML name space). It's identified by an associated URI.

To declare a new namespace in an XML document, you can use the `<xmlns>` **tag** at the beginning of the document. By associating a (alias) **prefix** with the namespace, you can qualify element or attribute names to indicate their membership in that namespace. In the following example, the prefix `edi` is used.
Additionally, you need to specify a **URI** that provides a list of all the elements associated with that namespace.

```
<x xmlns:edi='http://ecommerce.org/schema'>
   <!- the prefix edi" is associated to the
        URI http://ecommerce.org/schema for the
        element "x" and all the sub-tree
   -->
   <edi:name>My Name</edi:name>
</x>
```

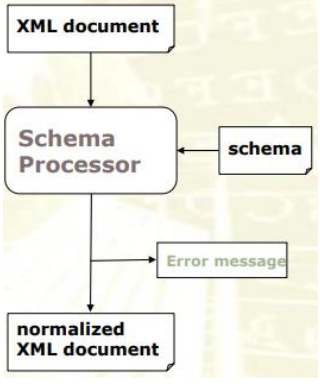
If you don't define a namespace, all the elements are part of the **default namespace**.

Well-formedness is not sufficient!

```
<business-card>
   ...
   <address>
      ...
      <city cap="56122">Pisa</city>
      <country>ITALY</country>
   </address>
   ...
</business-card>
```

```
<business-card>
   ...
   <address>
      ...
      <city cap="56122">Pisa</city>
      <country>ITALY</country>
      <country>SPAIN</country>
   </address>
   ...
</business-card>
```

Both documents are well-formed, but the second should not be 'valid'! Does it represent correctly a specific set of data?
Grammar rules are needed to characterize valid documents: need of an **XML document schema**.
A schema is a set of grammar rules to check if a specific xml document has meaning in a specific context**,** it's specified by the designer of the XML documents.



Schema processing: performed by a **schema processor** (a specific software) on a document based on a schema. If a XML document is well-formed, so if it respects the basic rule of the xml language, but it doesn't follow the grammar rules specified by the XML schema, it has no meaning for the application (or however for the one who read the document)
Check well-formedness then check validity (compliance with the schema) → If valid, normalize the document, e.g., insert attributes with default values.

A **schema language** is a formal language to specify schemas. Example are DTD (Document Type Definition) or the most popular **XML Schema**.

XML Schema

The XML schema is a description language of a XML file (it's the only one with a official validation from W3C). As all descriptive XML content languages, it's goal is to define what elements are allowed on the XML file, what type of data are associated with them and what is the hierarchy between elements.

Main constructs:

- Definitions:
  - **Simple type**: defines a family of Unicode text strings
  - **Complex type**: defines a collection of requirements for attributes, sub-elements, and character data in the elements that are assigned that type
- Declarations:
  - **Element**: associates an element name with either a simple type or a complex type
  - **Attribute**: associates an attribute name with a simple type

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:b="http://businesscard.org"
        targetNamespace="http://businesscard.org">

  <element name="card" type="b:card_type"/>
  <element name="name" type="string"/>
  <element name="title" type="string"/>
  <element name="email" type="string"/>
  <element name="phone" type="string"/>
  <element name="logo" type="b:logo_type"/>

  <complexType name="card_type">
    <sequence>
      <element ref="b:name"/>
      <element ref="b:title"/>
      <element ref="b:email"/>
      <element ref="b:phone" minOccurs="0"/>
      <element ref="b:logo" minOccurs="0"/>
    </sequence>
  </complexType>

  <complexType name="logo_type">
    <attribute name="url" type="anyURI"/>
  </complexType>

</schema>
```

In a XML document the schema for a document is specified by the attribute `schemaLocation` whose value will be a URI indicating where to find the schema. By inserting this attribute, the author asserts that the instance document is intended to be valid with respect to the schema.

XML schema types

A ***simple type***, also called a datatype, is a set of Unicode strings together with a semantic interpretation of the strings.

```
<element name="productid" type="nonNegativeInteger"/>
```

This example allows: all strings that represent non negative integers

A number of derived simple types are also predefined, for example: string, Boolean, negative Integer, etc…
Sets of simple types are used to create complex type.

Complex type

Are elements that include at least some simple elements but may contain other complex type in a hierarchical way.

```
<complexType name="card_type">
…
</complexType>
```

Other characteristics like min/maxoccurs, or sequence/choice tag are not reported here.

XML for IoT

XML was not defined for IoT devices: it has a lot of redundant and addition information (start and end tag, additional text at the beginning for schema and type of document) and it introduce significant overhead for transmission of the document. Moreover, well-formed and schema must be checked each time an XML document is received.
Good sides: strict set of rules, opportunity to define the schema, to check grammar and validity of correctness of the document. **Avoid unless you're forced.**
When you have large IoT project in which interoperability is mandatory and you need to check documents in a rigorous manner. You need a mechanism to check interoperability.

JSON

Origin: Web applications and asynchronous data interchange (AJAX), for this reason is based on a subset of JavaScript. It's text format, but it is a lightweight data encoding, based on only two structures:

- Collection of name/value pairs (Objects)
- Ordered list of values (Arrays)

```
{
  "Image": {
    "Width":  800,
    "Height": 600,
    "Title":  "View from 15th Floor",
    "Thumbnail": {
      "Url":    "http://www.example.com/image/481989943",
      "Height": 125,
      "Width":  "100"
      },
    "IDs": [116, 943, 234, 38793]
  }
}
```

JSON may be a limitation for complex system with interoperability mandatory, but it's lighter.

So JSON has 2 advantages: 1) is simpler than XML, there are no formal method to verify well-formed or validity of the document, it's flexible and expansible. 2) Overhead is reduced, thanks to non-redundancy of "useless" chars.
However, like XML has another limitations, since is not designed for IoT, is still text based! Text is bad every time you have LLNs because is very complex to be parse.

Binary encoding - EXI

The idea is to create new data encoding language designed directly for IoT environment, but it had no sense to start from scratch.
Binary representation of XML documents for payload compression. Several standards available: • Fast Infoset (ITU) • Binary XML Encoding Specification (OGC) for geo-related data • **Extensible XML Interchange** – **EXI** (W3C)

The result is a set of rules on how can translate the XML document in a set of 1 and 0.
EXI helps in significant reduce of overhead (reduce the size of the document) and reduced complexity in parsing in the sensors side.

Due to its highly successful data compression capabilities, the EXI data encoding representation paved the way for the introduction of other binary encoding languages, such as **CBOR** (Concise Binary Object Representation). CBOR, based on JSON, is a binary representation specifically designed for IoT.

We won't delve into the intricacies of these two languages as they are considerably complex.

Sensor Data Representation

The same 'type' of devices (resources) may be represented in a different manner by different vendors, **without a common understanding of the meaning of the data:** *it's not a data encoding problem* but something like "On" / "Off" or 1 / 0 … or maybe 0 / 1 or TRUE / FALSE.
Data encoding tell us how the data is structured, but we need something that tell us what the data means.

We have two problem:

- **Semantic problem**: we want that an IoT device of a company has a common understand on what the data is with an IoT device of another company. → SenML
- Another problem is **automated discovery**: applications would like to discover automatically the sensors and understand the type of sensors, the set of information/functionalities they can provide. → IPSO Smart Objects

How do we achieve this goal? The solution is to define ***new languages***, new meta-language that can be ***used in order to describe a domain.***

Sensors Markup Language (SenML)

SenML stands for Sensor Measurement Lists. It is a lightweight and compact **format** for representing sensor measurements in the context of IoT. SenML provides a standardized way to structure and exchange sensor data, making it easier to interpret and process sensor measurements across different platforms and applications.
SenML can be used on top of an encoding language like: XML, JSON, CBOR and others...

A SenML object represents a set of measurements and related metadata. It contains:

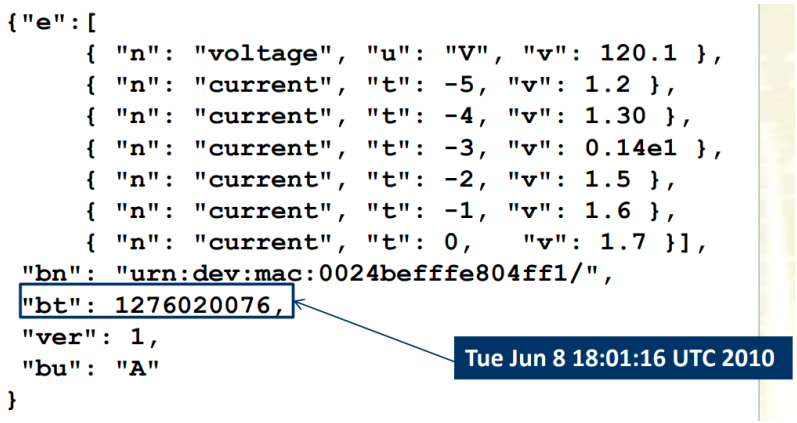
1. A set of optional **attributes**
2. A mandatory **array**, that must contains at least one measurement.

Base Name (optional)	A string that is prepended to the names found in the entries
Base Time (optional)	A base time that is added to the time found in an entry
Base Units (optional)	A base unit that is assumed for all entries, unless otherwise indicated
Version (default = 1)	Version number of media type format
Measurement or Parameter Entries Array (mandatory)	Values for sensor measurement or other generic parameters (such as configuration parameters)

For each measurement we have:

Name (optional if Base Name is present)	Name of the sensor or parameter. The Name attribute concatenated to the Base Name attribute must result in a globally unique identifier for the resource (an URI is recommended).
Units (Optional)	Units for a measurement value
Value (Optional if a Sum value is present)	Value of the entry. Only three basic data types: • Floating point numbers ("v" field for "Value") • Booleans ("bv" for "Boolean Value") and • Strings ("sv" for "String Value")
Sum (optional if a Value is present)	Integrated sum of the values over time
Time (optional)	Base Time + Time = Time when value was recorded (if zero, the time is roughly "now")
Update Time (optional)	A time in seconds that represents the maximum time before this sensor will provide an updated reading for a measurement

E.g with JSON encoding:



IPSO Smart Objects

IPSO Smart Objects provides a common design pattern, an object model, to provide high level interoperability between Smart Object devices and connected software applications.
They define a common URI structure that is used for automated discovery. Is a common methodologies to encode what is the type of IoT device and what are the functionalities exposed by IoT device.

Standardize a large database where a common set of IoT devices are presented:

- ***Category***: For instance: *smart lighting category, smart vehicle* etc.
  - ***Inside*** that categories there are set of ***functionalities*** optional for that categories (for example: turning on/off the light inside the smart lighting category).

Every category and every functionalities has an ID associated.
Every devices exposed a functionalities where the id of the category and of the functionalities are reported.

Object URIs: **Object ID/Instance ID/Resource ID**

- Object ID → overall category of IoT devices where the device places it
- Instance ID → valid only locally, usually 0, more than 0 if the IoT device has more than one sensor that belong to the same category
- Resource ID → reference to a specific functionality in that category

Designed for CoAP and for the WoT in general.

Why semantic is so important?

- Problem 1: ***Scalability*** – The IoT involves a large number of devices generating vast amounts of data. Managing this immense volume of information requires an autonomous and automatic process to handle devices and data in a scalable manner.
  How the semantic approach can help? Machine-interpretable markups: The use of semantic markup allows structuring the data generated by devices in a format that machines can easily interpret. This facilitates automated data processing and enables devices to communicate with each other.

- Problem 2: ***Interoperability*** – The IoT ecosystem comprises devices with different technologies, domains, and computational capabilities. This heterogeneity poses challenges in terms of interoperability, which refers to the ability of devices to communicate and work together seamlessly.
  How the semantic approach can help?
  - Shared understanding over a domain: The semantic approach allows for the definition of ontologies and knowledge models, establishing a shared understanding among devices and systems within a specific domain. This shared understanding promotes interoperability by enabling devices to use a common vocabulary and have a coherent view of the domain.
  - Precise and unambiguous vocabularies: to ensure the accurate and unambiguous definition of concepts and relationships within the IoT domain.

IoT platforms and oneM2M

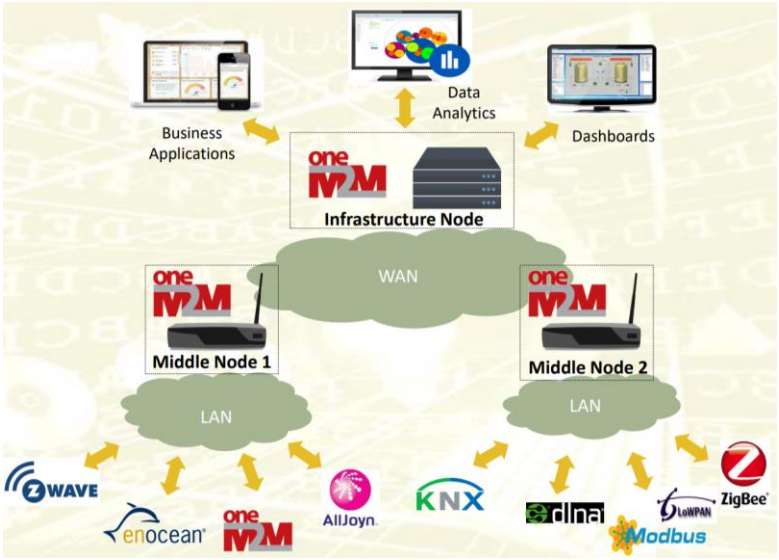
We would like to have an horizontal approach, and to do that we have already IPv6, Data Encoding Language, data Semantic, etc. But developers life is not easy, since they need to reinvent the wheel every time. We'd like to write software once and then run their software in many infrastructure as possible.
There's a missing software middleware layer between the service infrastructure and the application layer.

We want a common platform that will run on all the devices of different networks. Since this is missing, there are SEVERAL IoT solutions as IoT platforms. They are not interoperable between them, it's like different OS (Windows, Linux, etc.. they have different interfaces).

oneM2M is a global partnership project founded in 2012. The goal of the organization is to create a global technical standard for interoperability concerning the architecture, API specifications, security and enrolment solutions for Machine-to-Machine and IoT technologies based on requirements contributed by its members.
OneM2M is a standard that describe architecture of an IoT platform, functionalities provided by an IoT application and interface exposed by it.

Infrastructure:

- The standard must be as much general as possible
- On one side we have IoT devices, oneM2M allow the presence of heterogeneous IoT devices. Why? We are in a transition phase, we don't want to dismiss who don't respect the oneM2m specification now! We want a smooth transition.



The overall architecture can be divided into three main components:

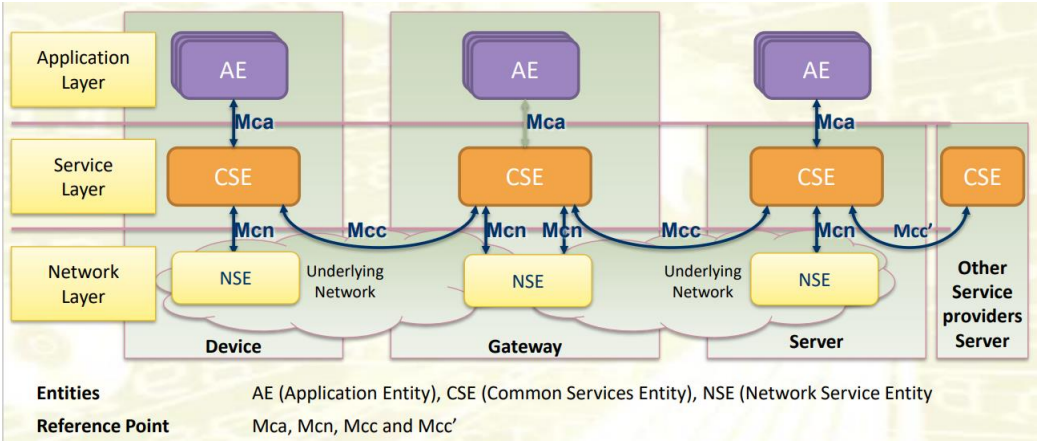
1. The upper part, referred to as the "**infrastructure level**", consists of an Infrastructure node, which is typically a logical entity that implements certain onem2m functionalities and is usually hosted in the cloud. This infrastructure node supports the execution of applications such as analysis applications or dashboards.
   So this first layer is equivalent to the cloud computing application that we have at the moment running in a datacenter somewhere.
2. The central part is known as the "**field level**", which corresponds to the fog/edge layer where application logic or security measures can be deployed.
3. The bottom part represents the **IoT device layer**, where multiple devices are present. Each device may implement different communication or integration standards. Within this layer, there are existing devices integrated with onem2m or new devices designed and produced according to onem2m specifications.

One of the main challenges we face in the future is the fact that many IoT devices currently belong to vertical solutions, which lack standardization. Consequently, future IoT platforms should focus on providing a solution for integrating these devices seamlessly into the platform.

Provide also functionalities that a node should implement, grouped in layers:

- Application layer
- Service layer
- Network layer

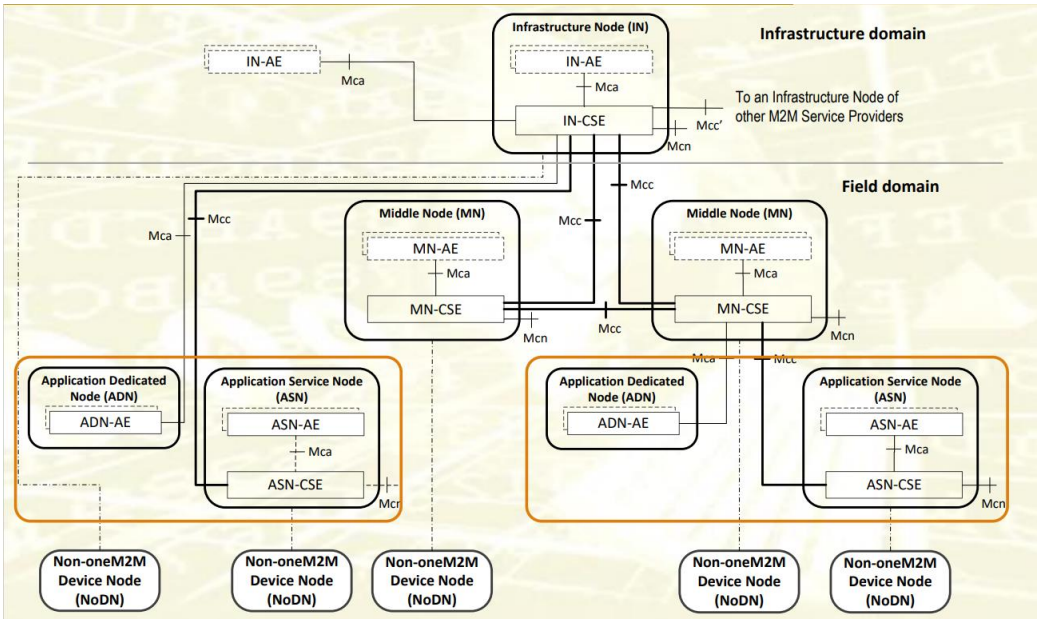
Every functionalities, exposed an interface through upper/lower layer, but also through other “node”.



- **AE**: *Application Entity*, containing the application logic of the M2M solution like home management functions, fleet management, blood sugar monitoring. *This layer could be deployed on different devices*, device (usually host a simple application and not the full one), gateway, server.
- **CSE**: Common Service Entity containing a set of common service functions (CSF) that are common to a broad range of M2M environment (verticals). **This is the main part of the oneM2M specification**. *This is where the common functionalities to all IoT application are implemented* (so it’s good to have these functionalities in the IoT platform, so developers doesn’t have to reimplement them every time), ***like security, registration, data storage***, etc.
- **NSE**: Network Service Entity, provides network services to the CSE, like device triggering, device management support, location services. These services are related to the underlying network capabilities. ***(It’ is the only functionality mandatory for all devices, it allow the communication between device, gateway, server)***

Interfaces are well specified by oneM2M, interface between layer on the same node but also between nodes.
This set of interfaces guarantees interoperability, allowing onem2m platforms to be composed and deployed as a combination of various modules provided and programmed by different companies. If all these modules adhere to the same set of interface specifications, it ensures seamless interoperability.

Those set of modules: applications entity, common service entity and network service entity can be deployed on different nodes that are part of the m2m architecture, however onem2m specifies also a set of configurations that are allowed in the system or not.



- *Infrastructure domain* (where we have the ***infrastructure node***, ***only one per Service Provider***): it must have a NSE and of course a CSE. Usually is not a problem since is usually developed on a cloud infrastructure. It may host AE, the complex application logic is usually implemented here.
- Important: it may happen that a devices (especially in field domain) it may not have enough resources for the functionalities oneM2M requires, then the IoT device is named ***application dedicated node***: is an IoT device, with not enough resource to run other software rather than application logic. It communicates with the CSE placed in a middle node or in a infrastructure node, using the appropriate interfaces.
- ***ASN (Application service node):*** it has AE, but also CSE. In that case, the workflow is different since all the functionalities are placed in the same node.
- Depending on the application domain, there can be in the between of the architecture, a middle node. It has a CSE and it can have also AE.

They are not depicted here but as we already said every node has at least one network service entity NSE that is mandatory for device, gateway or infrastructure node to communicate with other devices.

Common Service Functions

The core part of oneM2M is the CSE. The functionalities provided by it are:

- **Registration**: is the process of *delivering AE or CSE information to another CSE in order to use M2M Services.*
  These functionalities is exploited during the network bootstrap.
  The Registration (REG) CSF processes a request from an AE or another CSE to register with a Registrar CSE in order to allow the registered entities to use the services offered by the Registrar CSE.
- **Discovery**: The Discovery (DIS) CSF searches information about applications and services as contained in attributes and resources.
  The result of a discovery request from an Originator depends upon the filter criteria and is subject to access control policy allowed by M2M Service Subscription.
  An Originator could be an AE or another CSE. The scope of the search could be within one CSE, or in more than one CSE. The discovery results are returned back to the Originator.
- **Security**: basically *encryption* of the data and *basic functionality of identity management*.
- **The Subscription and Notification** (SUB) CSF provides notifications pertaining to a subscription that tracks event changes on a resource. ***Implements the pub-sub design pattern***.
- *Collecting BIG data is out of scope of the standard*, but the provide several functionalities to collect small amount of data. IoT device “publish” data on intermediary node and then “cloud” application retrieve them from the middle node. Available only with small amounts of data.