Università degli studi di Modena e Reggio Emilia

Dipartimento di Ingegneria Enzo Ferrari

Automotive Connectivity

Anno Accademico 2024/25

Indice

Capitolo 1

Introduction

1.1 Structure and Content

- **Module 1**:
  1. ***intra-vehicles communications***: nodes, sensors, ECU
  2. ***signal busses***: CAN, LIN, FlexRay, MOST, Ethernet [T1/T1S]
  3. ***car domain and OS***
- **Module 2**:
  1. ***inter-vehicles communications***: *V2V* and *V2X* (car is a node)
  2. ***wireless technologies***: Bluetooth, LoRa, C-V2X, IEE 802.11p (bd)
  3. application, messages, broadcast, GPS

Different **domain** or **application** needs different *communications protocols*, is important to understand how each nodes in domain communicate each other (inside the car).

1.2 Intra-Vehicles

From the 80's, where the car's control unit are isolated an there was a dedicated wires connect sensors and actuators with less electronic than now, until the reach the greatest goal of evolution in the automotive sector: autonomous drive. The complexity of the number of connection from each ECU's to the other, also the number of ECU's for each car, is growing. While the number of signal increase in a liner way, the connection between ECU's is growing with a quadratic complexity $O(n^2)$.

If we examine the evolutions of the ECUs number inside an "Audi A6" we can observe that in 1997 it has 5 ECUs and in the 2007 it has 50 ECUs, instead the "Tesla M3" in the 2017 has 70 ECUs. The quadratic increase of ECUs number, however has reach a cap for two main reason: the cost and the space inside the car. Traditionally one ECUs is responsible of one task, but nowadays it could be two type of trends:

1. *distributed of function across ECUs*
2. *integration of multiple function in one ECU*

1.3 Architectures

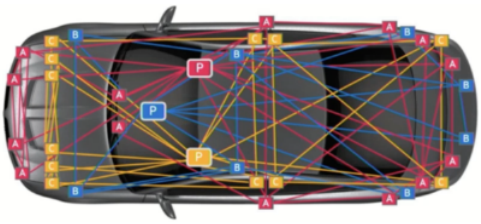


Figura 1.1: *Domain Architecture*

1. central domain controller (**P**) or high performance computer
2. ability to handle more complex functions
3. cost optimization
4. cable harness is rigid and expensive

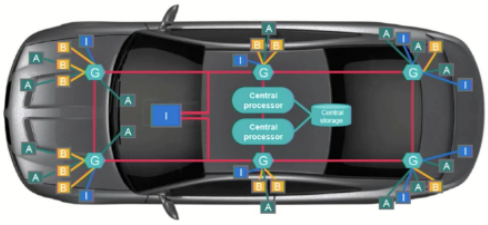


Figura 1.2: *Zonal Architecture*

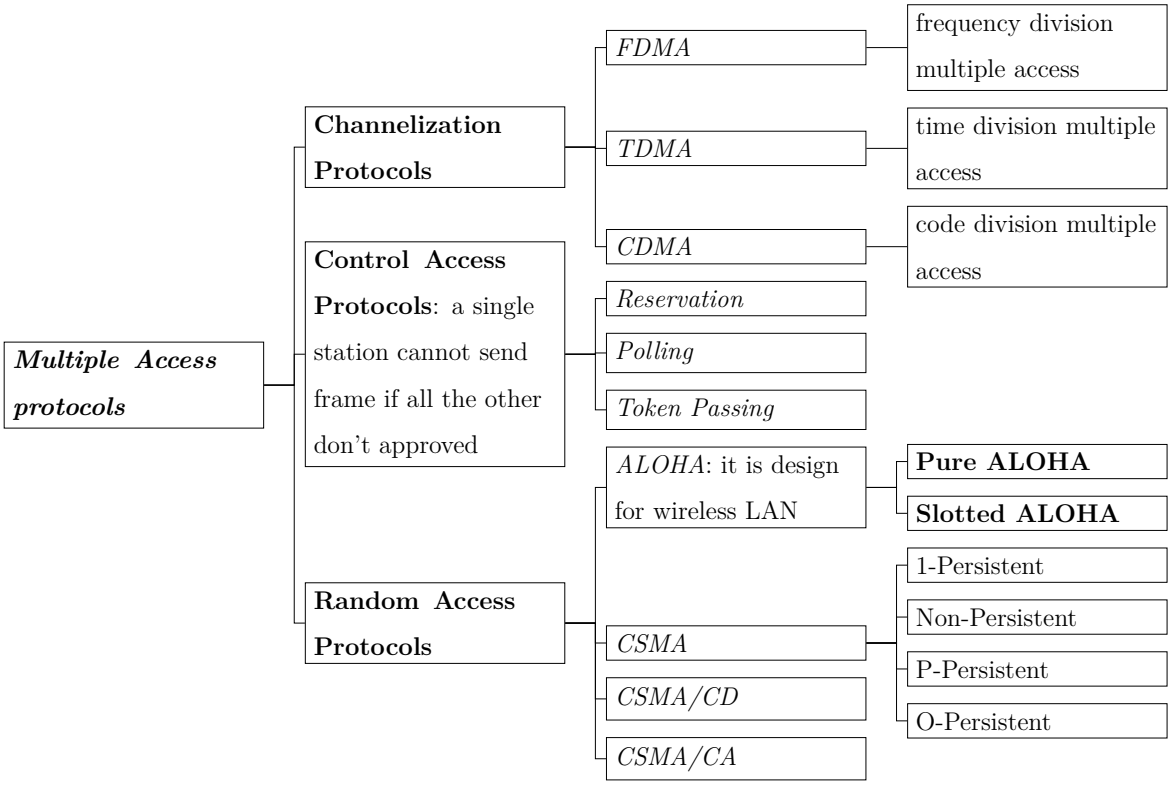
1. local ethernet per zone (**G**)
2. ultra high-speed secured backbone between zone
3. centralized software
4. central computer storage

1.4 Basic Knowledge

1.4.1 Multiple Access Protocols

In the ISO/OSI stack the first layer is the *data link layer* and it is used, in a computer network, to transmit the data between two or more devices or nodes. The data link layer it is normally split in two different sub-layer:

1. **data link control**: is a reliable channel for transmitting data over a dedicated link using various techniques such as framing, error control and flow control of data packets in the computer network.
2. **multiple access protocol**: if the link doesn't connect only two nodes, but multiple nodes can access to the physical link is possible that two or more nodes start to communicate in the same time, and it could be possible to have collision and cross talk between two or more devices. In this case the *multiple access protocol* is required to reduce the collision and avoid cross talk between the channel.



In this course it could be useful to see in dept three type of *Multiple Access Protocols*: the first one is ***Carrier Sense Multiple Access - Collision Detection***, next is the ***Carrier Sense Multiple Access - Collision Avoidance*** and the last one is

the ***Time Division Multiple Access***. In the autonomotive domain indeed there is needs to have a bus topology network and it is important to avoid collision.

CSMA/CA - Carrier Sense Multiple Access - Collision Avoidance: the idea is that before transmitting, a node first listens the shared medium to determine if the channel is not used (**idle**), if not it could start to transmit, but the problem start when two nodes begins to write on the nodes together. The **Collision Avoidance** part get in the game when two or more device try to write in the channel simultaneously in this case if another nodes is sense the transmitting node wait for a period of time (usually random) before re-start the writing procedure.

CSMA/CD - Carrier Sense Multiple Access - Collision Detection: is use in early Ethernet technology for LAN. It use carrier-sense to detect if the media is **idle** and it is combined with collision-detection in which a transmission station sense collision by detecting transmissions from other stations while it is transmitting a frame.

1. is the frame ready for the transmission? if not, wait for the frame.
2. is medium idle? if not, wait until it becomes ready.
3. start transmission and monitor for collision during transmission.
4. did a collision occur? if yes, go to collision detecting procedure.
   (a) continue the transmission (with **jam signal**) until minimum packet time is reached to ensure that all receiver detect the collision.
   (b) increment re-transmission counter.
   (c) was the maximum number of transmission (time out) attempts reached? if yes, abort transmission.
   (d) restart from 1.
5. reset the transmission counter and complete frame transmission.

TDMA - Time Division Multiple Access: is a channel access method for share-medium networks. It allow several users to share the same *frequency channel* by dividing the signal into different time slot. The users transmit in rapid succession, one after the other, each using its own time slot. This type of access to the physical medium has higher syncronization overhead tha *CSMA*.

1.4.2 Bit Coding

The first thing is to introduce the ***Electromagnetic Inferference - EMI*** that is a disturbance generate by an external source that affects an electrical circuit by *electromagnetic induction*, *electromagnetic couplig* or from conduction. For reduce EMI there are three possible way: add shield to wires, used twisted pair wiring or use coding with few rising/falling signal edges. At this point we can introduce the two main coding techniques: ***NRZ - Non Return to Zero*** or ***Manchester Coding*** (original variant).

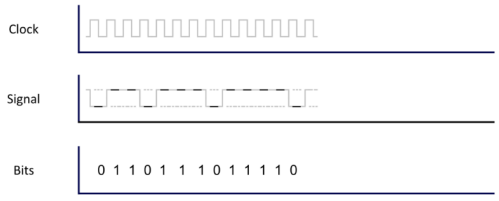


Figura 1.3: *Non Return to Zeros*

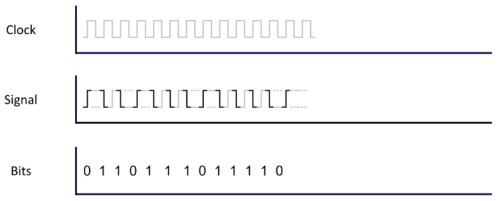


Figura 1.4: *Manchester Coding*

In the ***Non Return to Zero*** the digital ones is, usually, the positive voltage, while digital zeros are represented by other significat condition, like negative voltage.

In the ***Manchester Coding*** (original variant) the digital ones is the rising edge of the signal, instead the digital zeros are represented by the falling edge of the signal.

In both case it must be identify the digital zeros or one on the rising edge of the clock, so the syncronization problem between the clock of the transmitting node and the receiving nodes it is foundamentals.

Capitolo 2

Intra-Vehicles

2.1 ISO/OSI Layers

In telecommunication the idea is to divide each steps into layers starting from the application layer to the fisical ones, every layers have different function and it needs different protocols. Each layer can interact with the one that is above or below it and the communication of two layers follow rigid and specifics rules. Nowadays the standard *de iure* is the **ISO/OSI**, instead the the *de facto* standard is the **TCP/IP** that relax the rigid guidelines. The *ISO/OSI* has seven layers (bottom to top):

1. **physical layer**: specifies the mechanical and electrical properties to transmit bit (in the "real" world) and to control time synchronization.
2. **data link layer**: checked the transmission of the frame, error checking, frame synchronization and flow control.
3. **network layer**: it is used for the transmission of the packets, it is also know as *IP Layer*, in is normally use in ethernet.
4. **transport layer**: reliable end to end transport segment, you can manage how the data have to flow. In 99.99 % of the car domain it doesn't need.
5. **session layer**: establish and tear down sessions.
6. **presentation layer**: define the syntax and the semantics of information.
7. **application layer**: uses data transmitted via physical medium.

In the first module we need only two layers: **physical layer** and **data link layer**. We have to study the behaviour of the communication protocols like CANBus, LIN, FlexRay, MOST and Ethernet in this two layers. Starting from the **transmission medium**, normally the hardware pieces that we use to interact with is:

- **transceiver**: is used to "convert" analog signal to bits (brain less).
- **controller**: control the communication (brain full).

Initially the idea is to focus a little more on **CANBus**, the ***Physical Layer***: is compose by three component: **Physical Signaling - PLS**, **Physical Medium Attachment - PMA** and **Media Dependant Interface - MDI**.

1. **physical signaling**: the main purpose is to understand the bit encoding/decoding (if it is *NRZ* or *Manchester*) and to mantein the synchronization all over the network, every transceiver it must have a the same clock source. The synchronization is the most important things both for the bit encoding/decoding and for don't introduce delay in the communication.
2. **physical medium attachment**: driver/receiver characteristics based on the communication protocol.
3. **media dependant interface**: the connector for access to the physical medium.

Data Link Layer is compose by two component: **Logical Link Control - LLC** and **Medium Access Control - MAC**.

1. **logical link control**: from now on, we start to call *frame* the data that are send/receiver from the physical channel. It is used for *acceptance filtering* that permit to decide if a frame is important for the application above the *controller* and if not discard it. This component include also the *overload notification* and *recovery management* in the case there is an error on the communication they could ask to a re-transmit the data.
2. **medium access control**: is purpose is **error detection** it could check the data encapsulation/decapsulation, frame coding and error detection/signaling/handling.

2.2 Network Topology - The Bus System

Figura 2.1: *Line Topology*

Figura 2.2: *Star Topology*

Figura 2.3: *Ring Topology*

In the **Line** topology also know like ***Bus*** topology each node is connected by interface connectors to a single center cable. It is cheaper than the others and it has lower complexity but it is not very robust.

In the **Star** topology every peripheral nodes is connected to a central node called *hub* or *switch*. It has an higher cost and complexity than the *bus* topology, but it is much more robust (if the *hub* goes down it is a *single point of failure*).

The **Ring** topology is a *daisy chain* in a closed loop. When a node sends data to another, the data passes through each intermediate node on the ring until reach its destination (it use only one direction). It is not too munch expensive, but has higher complexity (if you want add a new node it could be troublesome).

In the autonomotive domain it is chosen the ***Bus Topology***, why? The first thing is that in the automotive industry it is mandatory to maintain lower the cost. The *busses* are very cheap for the materials, the weight and the volume. In the *bus* topology it is possible to have higher modularity, you can *plug & play* a node "when you want", in that way it is possible to have fully customizability inside the vehicles. The last things is that there is shorter development cycles. In the autonomotive field there is three main component:

1. ***transceiver***: it is the *physical layer definition* and implement the first layer of the *ISO/OSI* stack.

2. ***communication controller***: it is the communication protocol and implement the first and the second layers of the *ISO/OSI* stack.
3. ***ECU***: also know like **electronic controller unit** and implement the last layer of the *ISO/OSI* stack, the **application** layer.

The idea is to made possible to abstract the application layer in order to, if you want, change the first two layers, for example from CANBus to FlexRay, but nothing change at the application layer.

2.3 Controller Area Network

The **Controller Area Network** also know as **CAN** is a vehicle bus standard to enable efficient communication. It is originally developed to reduce complexity and cost of electrical wiring. ***CANBus*** use an **electrical** medium over wires and a **broadcast** data transmission. CANBus use the ***CSMA/CR*** like *multiple access protocol*, it means *carrier sense multiple access collision resolution* protocol, that permit to CANBus to have ***arbitration*** on the channel access. In this way there is random access to the physical channel, but it is impossible that there is some collision on the communications.

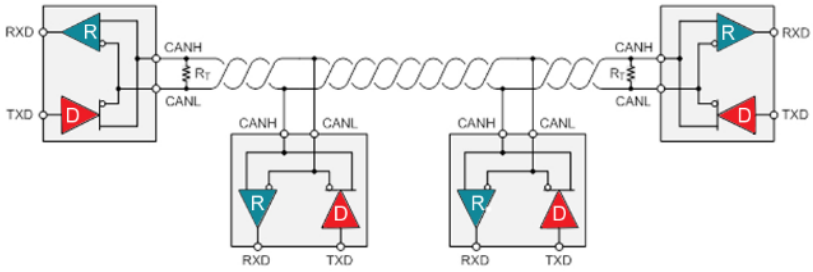


Figura 2.4: CANBus Network Topology

The **CANBus** network is compose by two wires: **CAN High** and **CAN Low**. The data is transmit over the wire using the *potential difference* on each transceiver. Two twisted wires are use because it gives to the protocol **noise resistance** and **increase resiliency**, if one brakes, CAN Low *survives*. At the end of the wire in the bus topology there are place two impedance R_T of 120Ω. Each CANBus node has three element:

- **CAN Transceiver**: is directly connected to the medium access by two pin (one on CANH and the other on CANL). It has the goal to translate the voltage level into bits (during the reception) and send it to the *CAN Controller* and translate bit into voltage level (during the transmission).
- **CAN Controller**: is connect to the *CAN Transceiver* by two pin (CANTX and CANRX) and is scope is to: message completion, control bus access, transmission and reception of the message, bit timing.
- **Microcontroller**: application software communicating with other ECUs via messages over the bus.

CAN Message							
1 bit	29 bit	1 bit	6 bit	0-64 bit	16 bit	2 bit	7 bit
SOF	CAN-ID	RTR	Control	Data	CRC	ACK	EOF

- **SOF**: is the **start of frame** is always set to *dominant **0*** to tell the other ECUs that a message is coming.
- **CAN-ID**: contains the message identifier - lower value have higher priority.
- **RTR**: is the **remote transmission request** allow to ECUs to "request" message from other ECUs.
- **Control**: informs the lenght of the *Data* in bytes (0 to 8 bytes), two bits are *reserved* for future implementation.
- **Data**: contains the actual data values, which need to be "scaled" or converted to be readable an ready for analysis.
- **CRC**: is the **cyclic rendundancy check** is used to ensure data integrity.
- **ACK**: is the **acknoledgement** this slot indicates if the CRC is OK all the bits must be **recessive** (*logical 1*).
- **EOF**: is the **end of frame** marks the end of CAN message all the bits must be **recessive** (*logical 1*).

The CANBus use a *message passing* technologies, it means, when a message is sent through the wire by an ECUs all the CAN Transceiver reciver the message, but if a application layer of one of another ECUs doesn't need that message it could ignore or if it need it, it could accept that message, using the *CAN-ID* as identifier. In other word the CANBus use the **receiver-selective** form of addressing. In the CANBus

the bit logic is pretty simple, each ECUs reads the wire (through a buffer) and each ECUs **can** write on the line (through a transistor), in this way the **basic state** is **up** ($+5V$ or logical ones) when one or more ECUs want to set signal low turn on transistor conductive (diode), this connect the bus to signal ground in this case the bus level is **low** ($0V$, or logical zeros) indipendently from other ECUs. The **0** is named **dominant level**. It could be see the CANBus wires as **logical AND** (if an ECUs write zeros the state is *zeros*).

The CANBus is an **event-driven** bus system, it means that there is no need to wait a scheduled time slot for sending data and there is the possibility of collision over the communication channel. If an ECU X registers an event e it is authorized to access the busses immediately and send data, but if another ECU Y is alredy transmitting data, then X waits. We want to calculate how long it takes a message to be sent, the first thing to do is to calculate the maximum bits number that is allow in a CAN Message: 130 bits. The CANBus can have lots of different bus speed $B \in \{5k \cdot \frac{bit}{s}, 125k \cdot \frac{bit}{s}, 250k \cdot \frac{bit}{s}, 500k \cdot \frac{bit}{s}, 800k \cdot \frac{bit}{s}, 1M \cdot \frac{bit}{s}\}$, let's consider the average $B = 500k \cdot \frac{bit}{s}$, the resulting time for sending a message is equal to $T_x(time) = \frac{M}{B} = \frac{130bit}{500k \cdot \frac{bit}{s}} = 0.25ms$, but what is happen if two ECUs start the communication on the same time? Let's consider the case where there are three ECUs X, Y, Z, X and Y are waiting Z because it is using the medium access, but probably they start to transmit in the same time when the busses is free, in this case we have a **collision**, the solution is how CANBus implement the ***CSMA-CR***, **carrier sense multiple access - collision resolution**, the two ingredients are how we can see the CAN busses (like a logical AND) and the **CAN-ID** to the logic prioritizing.

1. ECU X want to send: it must check if the bus is free (*carrier sense* - **CR**).
2. if it is busy the ECU have to wait.
3. when the bus is free, it could happen that one or more ECUs are ready to transmit, and start the communication together (*multiple access* - **MA**).
4. the last incredient is how to avoid the impending damage born from the collsion? (*collision resolution* - **CR**) $\rightarrow$ ***bitwise arbitration***.

All the ***bitwise arbitration*** is base on the first two field of the CANBus Message:

SOF (it is for everyone a **dominant bit**: ***0***) and **CAN-ID** (it could be 11 bits, in the standard CANBus and 29 bits for the extended ones). We know that in CANBus the ones with the lower *ID* has the greatest priority. Another basic know is that the CANBus network work like a *wired-AND* so if a nodes wrote on the bus a ***0*** the entire network has logically low value, also if someone else try to wrote a logically high value.

	ID 10	**ID 9**	**ID 8**	**ID 7**	**ID 6**	**ID 5**	**ID 4**	**ID 3**	**ID 2**	**ID 1**	**ID 0**
A	1	1	0	**0**	1	0	0	1	1	0	0
bus	1	1	0	***0***	1	0	0	1	1	0	0
B	1	1	0	**1**							

node B loses *arbitration*

→ stop sending and re-start sensing

wired-and bus logic		
sender *a*	**sender** *b*	**bus level**
1	1	1
1	0	0
0	1	0
0	0	0

We have three knowledge: the default value of the CANBus network is logically high, the bus work as *wired-AND* and the logic ***0*** si the **dominant** value, so if the *sender a* or *sender b* send over the bus the ***0*** value, it win the *arbitration* with the other *sender*.

arbitration logic		
sender	**bus**	**interpretation**
0	0	**next**
0	1	***fault***
1	0	**stop**
1	1	**next**

We alredy know that CANBus is *carrier sense* if the sender sent over the network a logical ***1*** but read logical ***0*** knows that it losts the *arbitration* with another *sender* and have to stops the transmission.

Priorities instead of Collision: the bus logic and arbitration logic not only prevent collision, it ensure a priority-controlled bus access: smaller ECUs ID, higher priority.

CANBus Message Integrity: the idea is to use the Data field to generate a CRC to permit the check on the integrity of the message, but wee need some basic knowledge before start: *polynomial division* and *XOR*.

Polynomial Reminder Theorem: given two polynomials $M(x)$ (the dividend) and $G(x)$ (the divisor), asserts the existence (and the uniqueness) of a quotient $Q(x)$ and a remainder $R(x)$ such that:

$$M(x) = Q(x) \cdot G(x) + R(x)$$

N.B. the degree of $R(x)$ is stricly lower than the degree of $G(x)$.

In the calculation of *CRC* depends on the arithmetic of modulo 2 polynomial. A modulo 2 polynomial is like:

$$a_n \cdot x^n + a_{n-1} \cdot x^{n-1} + ... + a_2 \cdot x^2 + a_1 \cdot x + a_0$$
$$a = \{0, 1\} \quad \forall a \in \{a_0, a_1, ..., a_n\}$$

An example of the representation of a binary polynomial is like: $x^3 + x + 1 = 1011$. If exist an x with a certain exponent e like: x^e in the binary representation the position e is fill with a 1.

$\oplus$	**0**	**1**
0	0	1
1	1	0

The ***XOR*** is a digital logic gate that gives a true (logical 1) when the input number is odd, otherwise is false (logical 0).

CRC Encoding:

1. we need to transmit a *n bits* **message** $M(x)$: $deg(M(x)) = n - 1$.
2. we have a *m + 1 bits* **generator** $G(x)$: $deg(G(x)) = m$.
   - the **remainder** $R(x)$ of the division $\frac{M(x)}{G(x)}$ will have strictly lower degree respect to $G(x)$ and, in the worst case, the maximum value will be $deg(R(x)) = m - 1$.

- $R(x)$ can always expressed with m bits.

3. add m zeros at the end of $M(x)$: this means to do the following $M(x) \cdot x^m$.
4. divide the **new message** $M(x) \cdot x^m$ with the **generator** $G(x)$ to obtain the ***reminder*** of m bits called ***CRC***.
5. the final message $B(x)$ is equal to $M(x) \cdot x^m + CRC$: this means to add the CRC bits at the end of the message replacing the m zeros padded before.

Example:

$$M(x) = 1101011011 \qquad G(x) = 10011 \quad (m = 4)$$
$$M(x) \cdot x^m = 11010110110000$$

```
          1100001010
10011)11010110110000
      10011
       10011
       10011
        000010110
            10011
             010100
              10011
                1110
```

The final message $B(x)$ is equal to: $B(x) = \underbrace{1101011011}_{\text{M(x)}}\underbrace{10011}_{\text{R(x)}}$

CRC Decoding

1. the receiver **acquire** $B(x) = M(x) \cdot x^m + CRC$.
2. the receiver **knows** $G(x)$.
3. the receiver **divides** the whole message by the generator: $B(x) = \frac{M_x \cdot x^m + CRC}{G(x)}$
4. if the receiver obtain **no reminder** the transmission was successfully (no errrors detected).

CRC Error Resistance: consider an error $E(x)$ occurs on the transmission channel and the receiver $B(x) + E(x)$ instead of simply $B(x)$, when the *CRC logic* can failed? The problem occure when $E(x)$ is multiple of $G(x)$ in this way $\frac{B(x)+E(x)}{G(x)}$ gives no reminder, so the receiver mark $B(x) + E(x)$ as a correct message. To avoid this problem we need to choose in appropriate way the generator $G(x)$, this is the reason why the $G(x)$ it is standard in the *CRC Encoding* (by the protocol).

CRC Design Priciples: $G(x)$ is extremely important in a way that $E(x)$ cannot easily be multiple of $G(x)$. For **detecting single bit of error**:

- $E(x) = x^i$ for error in i-th bit.
- if $G(x)$ has more than 1 term it cannot divide x^i.

Mathematical theory help us to desing powerful $G(x)$ with fancy characteristics, in CANBus the generator is: $G(x) = x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1$. **sender** and **receiver** must to agree on the **generator**.

CANBus bit coding: we know that there are two main bit coding algorithm: **Non return to Zero** (is less noisy) and **Manchester coding** (carries the clock with him on every single bit). In CANBus is important the clock for the synchronization between nodes, so it could be thinks that *Manchester coding* is the best one to be used. The *Manchester coding* has a big problem: the ***clock drift problem***. The *clock drift problem* is **caused** by natural variations of ***quartz*** (environment), for the correct working of CANBus the receiver must sample signal at the right time instant. *Clock drift* leads to ***de-synchronization*** of the clock that comport a bad interpretation of bit sequence. In order to avoid this type of problem, it is necessary to reduce the rising/falling edge of the signal, so it is advise the usage of **NRZ**.

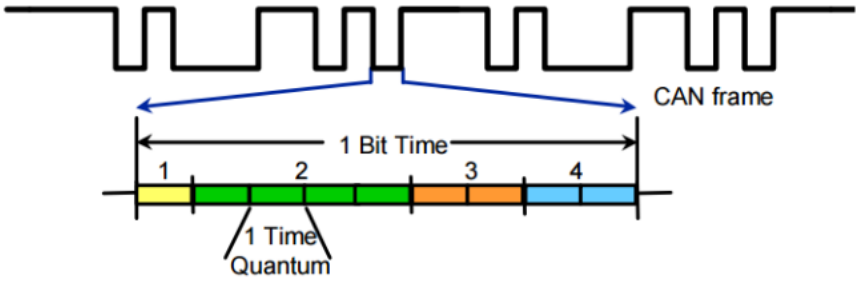
Problem

When using *NRZ* coding, sending many identical bits leaves no signal edges that could be used to compensate for the clock drift.

Solution

Insertion of extra bits after n consecutive identical bits $\rightarrow$ ***Bit Stuffing***. In CANBus $n = 5$.

Time Quanta (TQ): is the smallest time slice it could be count.



It is normally divided into four kind of field: *synchronization segment*, *propagation segment*, *phase buffer segment 1* and *phase buffer segment 2*. A *bit* it is compose from 8 to 25 **time quanta** and it is the smallest discrete timing resolution used by CANBus node. Each **TQ** is generated by programmable divide of the oscillator. Each segment is composed by an integer number of TQs and segments are non-overlapping. The bitrate is selected by programming the width of the TQ and the number of TQ in the various segments.

1. ***synchronization segment***: it is used to synchronization the various node, only the receiver nodes have to adjust their own clock during the receiver of the payload. The lenght of the segment is always **1**.
2. ***propagation segment***: if one node transmits to another faraway ones (geographically speaking) how we can synchronize the first *TQ* of the *synchronization segment*? The **propagation segment** allow the signal propagation across the network and through the nodes. This segment it could be compose from *1 TQ* to *8 TQs* and it is necessary to compensate for signal propagation delays on the bus line and through the electornic interface circuit of the bus nodes.
3. ***buffer segment one & buffer segment two***: this two segment it could have a programmable lenght between *1 TQ* and *8 TQs*. Between this two segment there is the ***sample point***. This point is used from the node to sample the information through the bus channel. This two segment are used to the ***re-synchronization***, in some circumstances we need to compensate the oscillator tolerances within the different CAN nodes.

Jump Width

The *jump width* is the amount of *TQs* that we can add (in the *phase buffer segment one*) or remove (in the *phase buffer segment two*) that permit to adjust the lenght during the *re-synch*.

Nowadays in many CANBus Modules the *propagation time segment* and *phase buffer segment one* are combined in a new segment named ***timing segment 1*** (the *phase buffer segment two* is renamed in ***timing segment 2***).

Dynamic Sample Position: programming the sample point position allow **flexibility**:

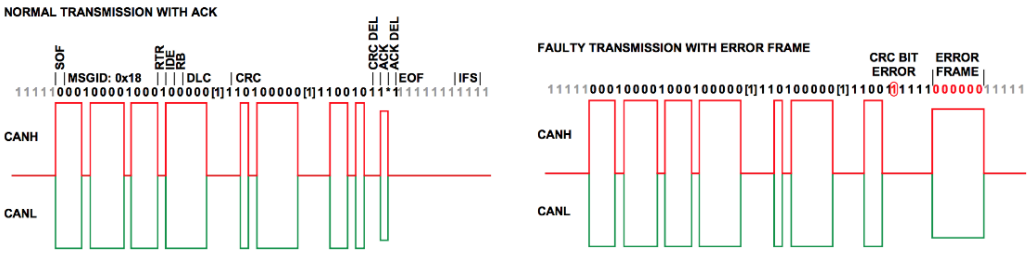
1. ***early sample***: decrease the sensitivity to oscillator tolerances and permit to use lower cost oscillators.
2. ***late sampling***: allow maximum signal propagation time (**reachability**), maximum bus lenght and poor bus topologies can be handled (more *time quanta* in the *propagation segment*).

CANBus Error: there are six possible different error:

1. ***Bit-Error***: write *logical 0* over the bus and sense a *logical 1* (or viceversa). In general if a transmitting ECU detects an **opposite bit** level on the CANBus we have a **bit-error**.
   - ECU writes *logical 0* and reads *logical 1* $\rightarrow$ very bad error.
   - ECU writes *logical 1* and reads *logical 0* $\rightarrow$ it is "possible" when there is the **bitwise arbitration** or it is expected that the bus state will change to dominant as other nodes acknoledge the message
2. ***Stuff Error***: reminder on the *bit stuffing*: it needs one opposite bit stuffed each 5 consecutive bits, it is used only from the beginning of the frame to the CRC delimiter. From the ACK field to the end is used the **fixed-form bit fields**. Each node receiving a message that breaks the bit stuffing rules will transmit an **error frame**.
3. ***Format Error***: if one of the *CRC delimiter field*, *ACK field* or *End Of Frame* have an divergent form, the receiving nodes perform a check to ensure these are

correct, if not send a **error frame**.

4. ***CRC Error***: *CRC delimiter field* is the only weapon to ensure the integrity of the message, it depends on the polynomials division, if the *CRC checks* (the reminder of message plus CRC divided by the Generator) is not 0 it generates a **CRC Error**.
5. ***General Error***: the seven **recessive** bits in the *EOF* are used to inform the CANBus nodes about a general error occured during the transmission. If a receiver node found out an error, it writes six consecutive "**zeros**" forcing an error in the current frame that can be captured from everyone.



6. ***ACK Error***: it happen when no one of the receiver nodes write on the busses an **dominant** bit in the *ACK field* of the transmitting frame.

> ***CANBus ACK***
>
> The transmitting nodes, after the DATA and the CRC, write in the bus a *logical 1* (**recessive**) and it hopes, in the mean time, that **at least** one receiver write a *logical 1* (**dominant**) in the ACK bit, if not the transmitting node (reads on the bus *logical 1*) and will resend the message.
> There is two bits for the *ACK field* to absorbe possible delay. We need to allocate space for "not perfect synchronized receiver" to push a **dominant** bit on the bus.

The *ACK* is triggered by another node so the voltage value could be slightly different. These technologies have some implication on the CANBus protocol, like:

- also the receiver node/s can (have to) transmit during specific frame slot (the *ACK field* or *EOF*).

- all the receiver must check the *CRC* very quickly in order to know if the message have pass the integrity checks.
- a CANBus network ***must have at least two nodes to work***, because with only one node no one can acknoledge a message.

For the calculous of the time in the circuit (in the CANBus controller) it is normally used ***time crystal***, the smallest *ICs* possible is the $8MHz$ *time crystal*. If we consider each clock cycle for the smallest unit in CANBus (*time quanta*) for each bit we have at least *8 TQs* (up to *25 TQs*).
If we minimize the size of the of a single bit we have to consider *8 TQs*. $\frac{8MHz}{8TQs} = 1MHz$ we can obtain the maximum bitrate for the CANBus.

CANBus Recap:

1. ***low cost***: the price is **always** a costraint, with it's two wires has a good price-performance tradeoff. This enables the use of CANBus outside the autonomotive domain.
2. ***reliability***: CANBus has sophisticated error detection and handling mechanisms. If failed the integrity checks of the frame it could repeat the sending of the same data and every nodes are informed about the error. CANBus has high immunity to EMI.
3. ***latency***: CANBus means real-time (soft) because there is low latency between transmission and request and actual start of transmission. CANBus has inherent arbitration on message priority due to the bitwise arbitration logic.
4. ***flexibility & speed***: CANBus nodes are "plug & play" and there are not limited number of nodes into a network.
5. ***multi master operation***: (ECU peers) each nodes is able to access to the bus, if there is a fulty nodes the bus communication is not disturbed and they switch-off from the communication.
6. ***broadcast capabilities***: message can be sento to single/multiple nodes and every node simultaneously receive common data.
7. ***Standardize***: *ISO-DIS 11898* (high speed), *ISO-DIS 115192-2* low speed.

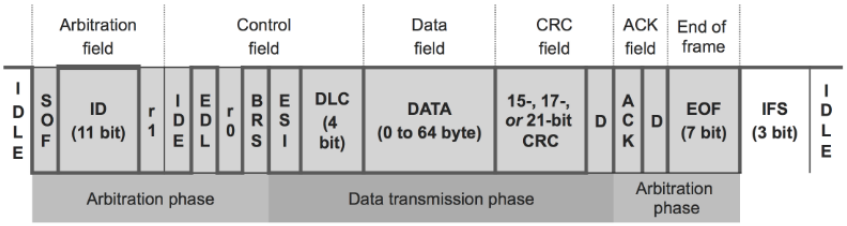
2.4 Controller Area Network Flexible Data-Rate

The ***CAN-FD*** is the evolution of the *CANBus*. The mainly disadvantages of CANBus are: $1MHz$ in some circumstances are not enough and only *8 bytes* of payload are often restrictive. To be compliant to standard CANBus the *arbitration phase* (before the data) and *ACK phase* (after the data) must be mantein to the same frequency.
CAN-FD data frames can be transmitted with two different bit-rates, in the *arbitration phase* and in the *ACK phase* the bitrate depends on the network topology and it is limited to $1MHz$, instead in the *data phase* the bitrate is limited by the *transceiver characteristics*:

- support a bitrate higher than $1MHz$.
- support a payload larger than *8 bytes*.

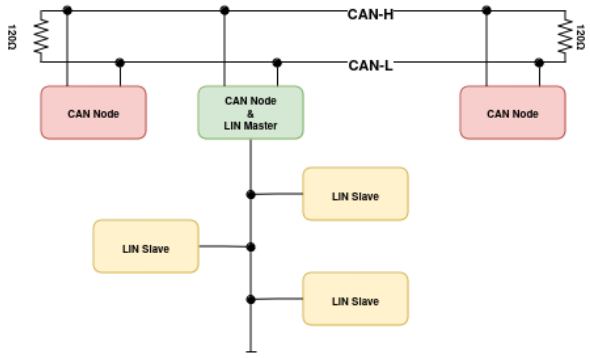
The increase of the frame speed is possible by shortening the bit time. We define the ***Bit Rate Shift - BRS*** is the bit in the *control field* used to inform **ALL** the nodes that sender will transmit faster in the *data transmission phase* and the ***Extended Data Lenght - EDL***. The implication of this change are:

- **larger payload**: it needs more *CRC bits* to maintain the robustness of CANBus, to have more *CRC bits* it needs a **larger generator**.
- **shorter bit-time**: new bit-time logic in the state machine, a *factor* is introduce between the bit time during arbitration phase and the bit-time during the transmission. The tipical factor is $8 \rightarrow$ considerering the fastest rate of CANBus *arbitration phase* and the longer header and CRC, the final result is more or less $6MHz$.



To summarize the **CAN-FD** could reach in the *data transmission phase* the speed transmission of $6MHz$ and the possibility of sending a payload large up to **64 bytes**

2.5 Local Interface Network



The ***LIN*** is a message oriented communication protocol that is design and developed to create something cheaper than low speed CANBus, this purpose it was reach partaway. Like the CANBus, LIN, works on the first two layers of the ISO/OSI stack (physical and data link layers), but it uses a **master-slave** concept. Using this architecture, the LIN busses, can have only a quartz on the master that manage the synchronization on all the LIN network, to achieve without waste space on the vehicle, the master of the LIN mesh is part of the CANBus network. This is the reason why the LIN is also know like a ***sub bus*** of CANBus (Fig. 2.5). As result on the communication we can say that LIN network is self-synchronize, but for this reason it needs to have lax timing constraints because only one node (the **master**) can schedule the order of the transmission. In addition to this, LIN busses, has other difference between CANBus:

- LIN is a **bidirectional one-wire line** and it can reach the frequency up to $20kHz$ (CANBus can reach $1MHz$).
- The **voltage** for the analog transmission over the channel is 40V, instead the CANBus is only up to 5V.
- **Bit Transmission** is *UART* like:

bit transmission		
1 bit	8 bits	1 bit
start bit	**data bits**	**stop bit**

In LIN protocol there is a rudimental error detection on the frame, it is a sum of all the payload bytes modulo 256 (in this way it can be stored into a single byte), but also on the channel, if a sender while monitoring the bus, read an unexpected state abort the

communication without correction. The **schedule** of the network is hardcoded on the **master's firmware** (static), the scheduler determines which node have to transmit in that specific slice of time. This consent to have a channel that is **mostly deterministic**, permit to the slave to not know how it is schedule the transmission and allow to the master to *change the order of transmission runtime*.

LIN Message: is divide into two component: *Message Header* and *Message Response*. The first one is sent over the channel by the **maaster node** and is like a request for a specific slave, instead it is possible to see the second one like the slave response.

Message Header			**Message Response**	
Break	*Sync*	*Identifier*	*Data*	*Checksum*
14 bits	8 bits	8 bits	0 to 64 bits	8 bits

Description for each field:

1. ***Break***: is composed by two kinds of fields: **13 low bits** (*dominant*) and **1 high bit** (*recessive*) that is used to delimiter of the field.
2. ***Sync***: is used to *synchronize* the bit timing of the slave, it is always **0x55** (01010101) in this way follow the profile of the clock.
3. ***ID***: is used to individuate the right slave, different from the CANBus, in this identifier there are parity bit, to have a check on the integrity of the ID, because in this case is very important for the correct master-slave communication (**protected field**). ID is divide in two segment:
   - from **LIN 2.0** the first **2 MSB bits** define the lenght of the payload that could be 2, 4 or 8 bytes, previous version of LIN used static 8 bytes data lenght.
   - **4[6] bits** for the "real" ID.
   - **2 parity bits**:

$$p_0 = id_0 \oplus id_1 \oplus id_2 \oplus id_4$$
$$p_1 = id_1 \oplus id_3 \oplus id_4 \oplus id_5$$

4. ***Data***: contain the payload, and it was send by the slave selected by the master with the lenght settled in the identifier.

5. ***Checksum***: is this case, like said before, the checksum is the sum of all the payload bytes modulo 256, in that way it can be stored into a single byte.

	d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0
carry		*1*	*1*	*1*		*1*		
first byte	0	0	0	1	0	1	1	0
second byte	0	0	0	1	0	1	1	1
third byte	0	0	0	1	0	1	1	0
forth byte	0	0	0	1	0	0	0	0
fifth byte	0	0	0	1	0	0	0	0
checksum	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**1**

LIN nodes are typically bundled in clusters each with a master that interfaces with the backbone CANBus. We have introduce the general message/frame format, but in LIN protocol there are six kinds of different messages (encoded in the ID field):

1. ***Unconditional Frames***: is defined by the ID ***0x00 - 0x3B*** and is the default type of frame, where the master send a header over the channel and the request slave reply.
2. ***Event Trigger Frames***: is defined by the ID ***0x00 - 0x3B***, the master polls multiple slaves, the slave who have updated data response, if there are collision, the communication end and the master switch to *unconditional frame*.
3. ***Sporadic Frames*** is defined by the ID ***0x00 - 0x3B*** in this type of message the master acts like a slave and reply to his own requests.
4. ***Diagnostic Frames*** is defined by the ID ***0x3C - 0x3D*** with this frame the communication becomes request-response, the ***0x3C*** is the ID where the master make the request, instead the ***0x3D*** is the ID where the slave reply.
5. ***User Defined Frames*** is define by the ID ***0x3E*** is a user-defined frame and it can contain any types of information.
6. ***Reserved Frames***: ID ***0x3F***

There is a reason why the **data lenght** of CANBus and LIN are equal. Since LIN is also called CANBus sub system, for compatibility reason the payload is equal. Message of CANBus can be sent over LIN too and viceversa.

2.6 FlexRay

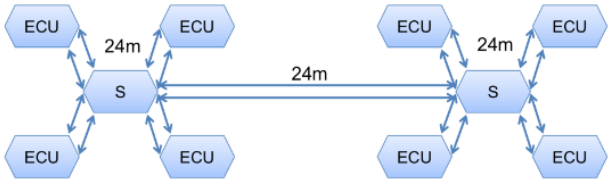
This communication protocol it is design with the purpose of be more reliable in therm of determinism than the CANBus. To achieve this goal is necessary to increase the price, the **FlexRay** is more expensive than CANBus. The CANBus is prone to failure, the reason behind this problem are the topology of the network: if the channel is broken a frame could not be delivery to every node on the bus and there are not rendundant link to avoid that.

The CANBus frequency is up to $1MHz$ ($6MHz$ is we consider CAN-FD), but it slowly regard the requirement of the modern vehicles (***X-by-Wire***) moreover in CANBus there isn't the assurance that each node have its own time slice where it can write over the busses, this is because in CANBus there is **not** a firmware that implement the **event scheduler**

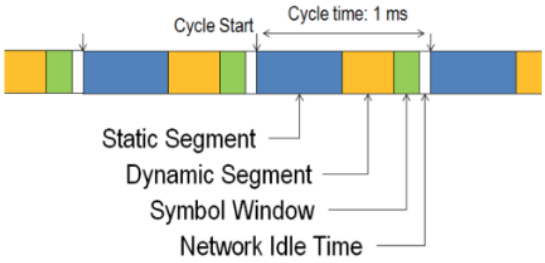
> ***X-by-Wire***
>
> Drive/Brake/Steer can be bypassed as an **input device**, the reason behind that is that it must be possible to control the vehicle's actuation not only in manual but also in autonomous drive. This involves two main requirements: **error tolerances** and **time-determinism requirements**

Requirements:

- **more resilient topology**: the topology of FlexRay is a ***star-type*** with **bus termination** where the maximum distance between each line is up to 24 m. Each connection is compose by two lines for rendundancy purpose (**boost error tolerance**). The second line it could be used or for backup or it could use to increase the frequency speed of transmission. Each line can reach the speed of $10Mbps$ if we split the message between the two line we can obtain the increase of the velocity by a $2x$ factor. Like in CANBus the default value is set to *logical 1* but the wire are **unshielded twisted pair**.



- **Determinism**: the busses operates using a **scheduler** that is replicated for each **time cycle** during the communication time. Each cycle is divided into four different kinds of *segment*:



  - **static segment**: is *preallocated* into slices that permit to be more deterministic and allow time constraints addressing, for each nodes are allocated a fixed period (at least one) into this segment.
  - **dynamic segment**: the idea is like CANBus, nodes can take control over the channel if the bus is available (not busy) can simulate an behaviour *event triggered*, normally used for event based data that does not require determinism.
  - **symbol windows**: typically used for network maintenance and signaling for starting the network.
  - **network idle time**: a know "quite" time used to maintain synchronization between node clocks.
- **different message class**: allow latency-constraints.
- **other characteristics**:
  - differential signaling on each pair of wires reduces the effects of external noise on the network without expensive shielding.
  - flexray busses require termination at the ends.
  - need synchronization clock in sender and receiver (and timestamp synchronization).

FlexRay - Access to the bus

The FlexRay uses the ***TDMA*** as methods for access to the physical medium, this allow to all nodes to be synchronize (using the same clock). In *TDMA* each nodes on the bus has its own turn (time slice of the cycle) where it can write on the channel. The *TDMA* permit **time consistency** that allow to the communication protocol **determinism**

(one of the constraints).

While for the CANBus each node has to know only the commnuication baudrate, in this case nodes on flexray must know the schedule of the transmission and all pieces of the network to communicate. For the autonomotive domain, there the majority of nodes are *embedded system* where there are a closed configuration and it is difficult to change firmware after the installation, this can be a problem for the scalarity of the system. After the installation it's difficult to add a nodes in the flexray network, but thanks to this there isn't the necessary for the nodes to use a discovery algorithm to understand how the network is composed.

For a **TDMA** network (such flexray) to work correcly, all node must be configured in a correct way:

- in the *embedded system* it is mandatory to have static configuration network.
- there is an increment of **relaibility**, but **not flexibility** this leads to configuration tradeoff:
  - data rate
  - deterministic volume
  - dynamic data volume
  - topology

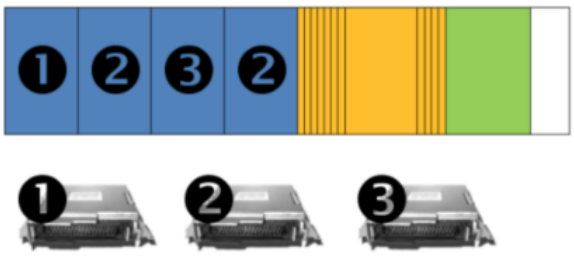


Figura 2.5: FlexRay Cycle

The **static segment** must be decided **a priori**, in this way it is possible to divide the segment in at least n slot, n as the number of the nodes on the network. In each slot a defined node can transmit data over the bus, while the other nodes have to wait their slot. In this way we know exactly when the nodes are going to transmit, this is the definition of **determinism**. The disadvantage of the *TDMA* is that, if the ECU number two doesn't have nothing to transmit we loose resource (time). If we are all

synchronize there is no way to collide.

The **dynamic segment** use a different protocol to access the medium, in this segmente there is the same as the CANBus, **CSMA-CR**. The segment is divide in mini-slot, each slot is assign to a specific ECU, for each slot the defined ECU sense from the bus if no one is broadcasting data, it can start to transmit, when it ends the time slice, the transmission interrupt, but the next node scheduled sensing the bus finds a busy state, so it doesn't start to sense. It could be possible for some ECU to lose each type of arbitration on the *dynamic segment*.

FlexRay: Frame

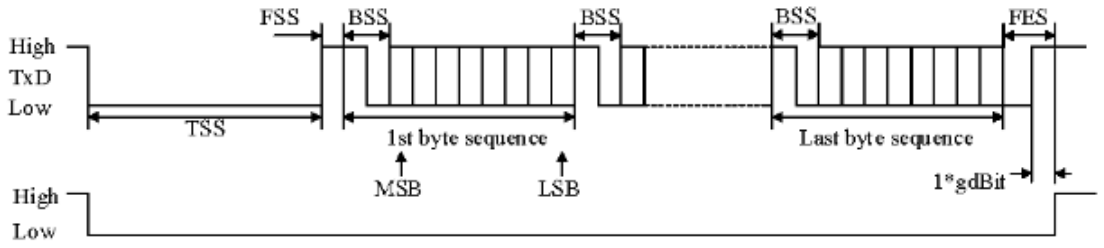


Figura 2.6: FlexRay Message Frame

FlexRay communicate using frames, each frame contains data as **bytes** $b_0 ... b_{m-1}$, and it is structure like:

1. **Transmission Start Signal** (**TSS**): logical value **0**.
2. **Frame Start Signal** (**FSS**): logical value **1**.
3. m bytes, composed each one of:
   - **first bit** is part of the start signal (**BSS0**).
   - **second bit** is part of the start signal (**BSS1**).
   - there are n bits for the data.
4. **Frame End Signal** (**FES**): logical value **0**.
5. **Transmission End Signal** (**TES**): logical value **1**.

The framing of data is usefull for the **determinism**. Bit stuffing is not used in this standard because given the number of bytes of a message you exactly know the amount of frame bits needed for the transmission.

FlexRay: Message Format

5 bits	11 bits	7 bits	11 bits	6 bits	n bits	24 bits
control bits	**frame ID**	**lenght**	**header CRC**	**cycle counter**	**payload**	**CRC**

- ***Control Bits***:
  1. b_0 is reserved: always **zero**.
  2. b_1 is used to distinguish static/dynamic slot message.
  3. b_2 is *null frame* indicator, it is used to sign a frame without payload (also in static segment).
  4. b_3 is *synch frame* indicator, it is used to sign a frame used for synchronizing clock, to be send to a few "reliable" ECUs.
  5. b_4 is *startup frame* indicator, used for synchronization during bootstrap and it is send by cold start node.
- ***Frame ID***: identify the message.
- ***Lenght***: lenght of payload.
- ***Header CRC***
- ***Cycle Counter***: global counter of passed bus cycles (it is important to always know the time).
- ***Payload***: it could be from 0 to 254 bytes.
- ***CRC***: it is used to check the integrity of payload.

Time Synchronization

In flexray is important to have synchronize all the clock of each node inside the network, it is important to have synchronize bit clock and slot counter. In flexray we do not want to have a dedicated node for that goal, but something distributed. Normally there are three nodes named *cold start node*. This node has the goal of the ***cold start procedure*** that consist:

1. check if bus is *idle*, if not abort the trasmission.
2. **transmit wakeup** [**WUP**] pattern, if collision occure abort the transmission, if not this is the **leading cold start node**.
3. the *leading cold start node* send over the bus a ***Collision Avoidance Symbol***

(**CAS**). It need to start regular operation (cycle counter start at 0) and to set the BSS0 and BSS1.

4. after the **CAS** signal the other cold stard node wait **4 frames** before start the cycle counter (for the other cold node start from 4).
5. they start the regular operation (BSS0 and BSS1).
6. other **regular** ECUs wait other **2 frames** before starting regular operation.

leading	**wup**	**wup**	**cas**	0	1	2	3	4	5	6	7	8	...
cold	**wup**	*abort*						4	5	6	7	8	...
cold	*abort*							4	5	6	7	8	...
regular										6	7	8	...
regular										6	7	8	...

Comparison

bus	LIN	CANBus	FlexRay
speed	40 kbit/s	1 Mbit/s	10 Mbit/s
cost	$	$$	$$$
wires	1	2	2 o 4

If we consider the typical application:

- **LIN**: body electornics → mirrors, power seats, accesories.
- **CANBus**: powertrain → engine, transmission, ABS.
- **FlexRay**: High performance Powertrain Safety (*X-by-Wire*) → active suspension, adaptive cruise control, keep lane assist.

2.7 Ethernet

2.7.1 Access

Ethernet use the ***CSMA/CD***:

- **CS**: *carrier sense*: if the bus is busy, do not transmit.
- **MA**: *multiple access*: once the transmission start, the collision could be happen.
- **CD**: *collision detection*: listen while communicate the information over the bus, if collision detected, jamming sequence.

The problem of this type of access protocol has a problem when two distant node (geographically speaking) try to transmit information in the same time, if the propagation of the frame over the channel it's not fast enough, the two different node does not see the collision and generate some noise. This case cannot happen into the autonomotive domain, it is necessary to fulfill a codition: the frame sent from a node have to reach **all** node before the end of the transmission.
The idea is to work on the **frame lenght** and the **bitrate**, ad example given the minimum frame lenght is possible to reach the maximum bitrate.

Hp

F is the minimum frame size, b is the maximum bitrate

c speed of light (considering a costant)

$t = \frac{F}{B}$ is the transmission time

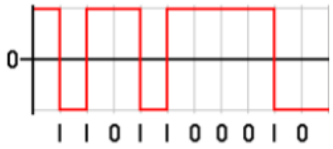
$l = c \cdot t$ l is the distance covered by the transmission

$d_{max} = \frac{l}{2}$ d_{max} is the maximum lenght between the two terminal node.

Ethernet **topology** was born like *bus*, but nowadays it have changed into *star* topology; and it is configured using a switch (layer 2 TCP/IP) or router (layer 3 TCP/IP) with all node connected with this component.
Ethernet use a two different type of **bit coding**:

- ***Non Return to Zero Inverted - NRZI*** (used until the ethernet version of $10Mbps$): the logical value is determine by the transition of the analog signal. If

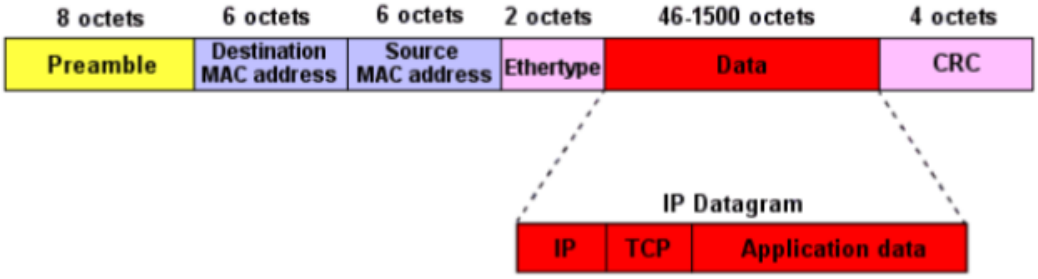
nothing change in the signal the logical value is **0**, but during the falling edge it will be a logical **1**.



- ***4B5B*** (used in the ethernet version of $100Mbps$): it maps 4 bits words in 5 bits group on the medium, using the **NRZ** (it prevents three equal consecutive bit).

Data (4B)	Codeword (5B)	Data (4B)	Codeword (5B)
0000	11110	1000	10010
0001	01001	1001	10011
0010	10100	1010	10110
0011	10101	1011	10111
0100	01010	1100	11010
0101	01011	1101	11011
0110	01110	1110	11100
0111	01111	1111	11101

2.7.2 Ethernet Frame



- ***Preamble***: it is used for the synchronization of the receiver (it is 8 bytes that alternating a logical 1 and a logical 0).
- ***Destination MAC Address***: is the destination address of the ethernet interface (**MAC - Medium Access Control**).
- ***Source MAC Address***: same as before, but for the source node.
- ***Ethernet***: is used for indicating the type of data carried by the frame or the data lenght.

- ***Data***: the data to be transmitted, for the incapsulating principles, inside it can be found the **iP Datagram**.
- ***CRC***: is compose by 4 bytes of **checksum** that it is calculate on the entire frame (without the *preamble*). It is always calculated like the remainder of the polynomial division by a known generator (*Standardize CRC-32*):

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

2.7.3 Ethernet in the Autonomotive Domain

Previous Limitation

Ethernet is a 45 years old protocol, but why only now is started the interest by the autonomotive company in the ethernet protocol, and why it coul be used right now?

1. ethernet produce to much EMI/RFI, for the automotive domain.
2. ethernet could not guarantee latency down to the low microsecond range (not guarantee time constraints).
3. ethernet did not have a way to control bandwidth allocation to different stream so it could not be used to transmit shared data from multiple types of source.
4. ethernet did not have a way of synchronizing time between device and having multiple device sample data at the same time.

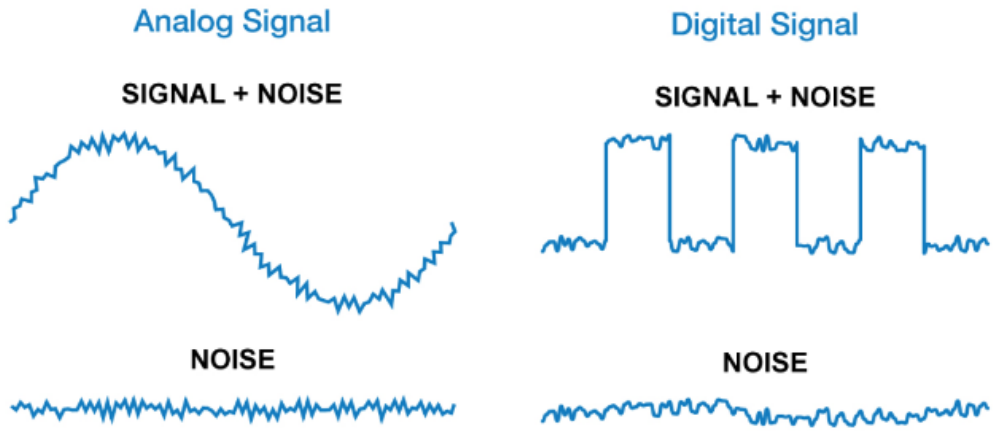
The big need for find something like ethernet for the autonomotive domain is the current research for the fully ***autonomous driving*** system for the autonomotive. For resolve these problem, it was born the ***One-Pair EtherNet (OPEN) alliance***, that is a non-profit, special interest group (SIG) of mainly automotive industry and technology providers collaborating to encurage wide scale adoption of Ethernet-based communication.

Solution

1. change the waveform of the signal we can put into effect the reduction of the EMI/RFI. The ***100Base-tx*** is the standard ethernet that has **fast rise time** and **three clear level**, instead; in the ***100Base-t1*** the autonomotive ethernet it is posssible to notice that there is **slower rise time** and the **three level are not very clear**. This reduce the pollution of the EMI/RFI.
2. to solve the problem to not guarantee time constraints, the *autonomotive ethernet* define a new type of frame: ***Express Packet***. For these type of frame is possible to interrupt the transmission of existing packet, that can be reload when the *express packet* have finished its work. The **express packet** can "guarantee" latency in the single microsecond range.
3. to control bandwidth allocation to different streams it is create two different approchs:
   - ***stream reservation***: simple reservation protocol to notify the various network elements in a path to reserve the resources necessary to support a particular stream.
   - ***queuing and forwarding for AV bridges***: defines rules to ensure that an AV stream will pass through the network within the delay specified in the reservation.
4. for the synchronization purpose it is created a new standard ***IEEE 802.1AS*** (Timing and Synchronization for Time-Sensitive Application in Bridge Local Area Network), this standard introduce simpler/faster methods for choosing master clocks.
5. (**bonus**) to reduce the wiring needed in a car, it is choose to use ***Power on Ethernet - PoE***

2.8 EMI

ElectroMagnetic Interface is a disturbance generated by an external source that affects an electrical circuit by electromagnetic induction, electrostatic coupling or conduction.



An **analog** signal absorbs the noise that becomes part of the information itsef. The only possibility is *filtering* (Fourier Theory).

In **digital** signal if you are still able to distinguish well between **1** and **0** you are **errorless** and **noise immune**. If the noise "change" a bit there is an **error**.

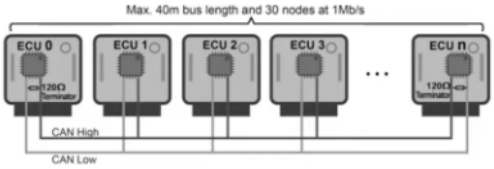
All intra- and inter-vehicle communications are digital, but there is an easy experiment to realize how much a car is "noise"

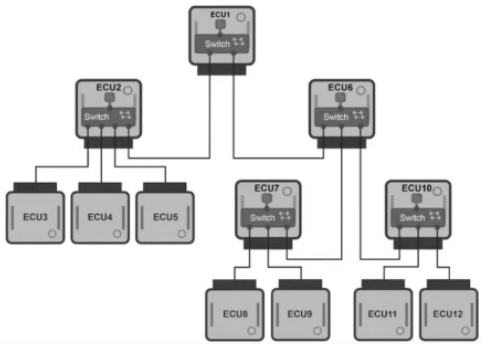
2.9 10BASE-T1S

First of all, it makes sense have a recap of why it is important ethernet for the intra-vehicle communication:

- provide more bandwidth than any other commnuication protocol (intra-vechicle). Up to $10Gbps$.
- iP based. Allow easily integration of many tech.
- Alredy used into Infotainment Application.
- switch based flexibility.

The main problem of the introduction of ethernet in the autonomotive domain is the different topology used between ethernet and the alredy used technologies of the intra-vechiles communication.





- CANBus/FlexRay have bus topology.
- no intermediate device.
- low bandwidth.
- cheaper than 10Base-t1.

- physical layer is point to point (p2p).
- the network is iP based.
- large bandwidth.
- significantly more expensive.

Motivation & Goal of 10BASE-T1S

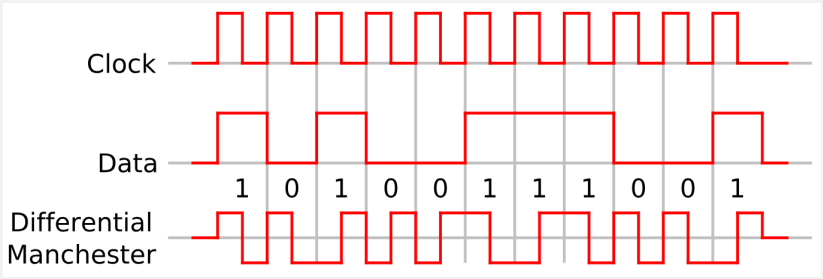
10BASE-T1S can provide an iP-based (ethernet) solution over the current topologies of automotive domain. It can allow transparent iP network with a simple design and deploy, without introduce the complexity of the gateway. Reduce the number of the wire of "normal" ethernet using the **PoE** technology (**Power over Ethernet**), but could introduce a little more complexity.

The ***10BASE-T1S***, also named **Ethernet over twisted-pair** is a technology that allow to have the **physical layer** similar to CANBus and FlexRay, because it uses a ***Single Twisted Pair (STP)* copper cabling** and introduce a new algorithm for **network access**: ***Physical Layer Collision Avoidance (PLCA)* Method**. This new algorithm allow the bus access to be deterministic (unlike the CANBus family), but reduce the number of node into a single channel and the lenght: maximum **8 nodes**, for **25m** of coverage. Increase the bitrate (not too much) up to $12.5 Mbps$ and support **PoE**.

2.9.1 Bit Coding

Differential Manchester Encoding (DME) is a line code in digital frequency modulation in which data and clock signal are combined to form a single two-level self-synchronizing data stream. Each data bit is encoded by a presence or absense of signal level transition in the middle of the bit period, followed by the mandatory level transition at the beginning. Recap:

- there are at least one transition of each clock period, this permit the synchronization of the clock between nodes.
- values are encoded based on transition: if into the clock period there is another transition it will be encode **logical 0** else if there is not other transition is **logical 1**. This behaviour allow to have better EMI immunity.



There is another version of **Differential Manchester Encoding** that invert the digital encoding: if there is a transiction between two clock cycle it will encoded into **logical 1** otherwise, if there is not transiction, into **logical 0**.

The 10BASE-T1S use the ***4B5B*** that maps groups of 4 bits of data onto groups of 5 bits for transmission. These 5-bit words are predetermined in a dictionary and they are chosen to ensure that there will be sufficient transictions in the line state to produce a **self-clocking singal** (this introduce a 20% of overhead on the data).

It can be noticed that not all the permutations of five bits are used. For the remaining codes, the **bit coding** is mapped into a **special function**. For this reason, the **4b5b** mapping is a non-surjective function:

1. $11111_{(2)}$: **silence**
2. $11000_{(2)}$: **sync/commit**
3. $10001_{(2)}$: **esderr**
4. $01101_{(2)}$: **esd/hb**
5. $00111_{(2)}$: **esdok/esbrs**
6. $00100_{(2)}$: **ssd**
7. $01000_{(2)}$: **beacon**
8. $11001_{(2)}$: **esjab**

***4B5Bs* encoding**

name	**4b**	**5b**
0	0000	11110
1	0001	01001
2	0010	10100
3	0011	10101
4	0100	01010
5	0101	01011
6	0110	01110
7	0111	01111
8	1000	10010
9	1001	10011
A	1010	10110
B	1011	10111
C	1100	11010
D	1101	11011
E	1110	11100
F	1111	11101

2.9.2 Medium Access

PHY-Level Collision Avoidance: in this type of scheme all nodes are assigned unique sequential numbers (IDs) in the range $1, 2, ..., N$. The 0th ID corresponds to a special **master** node that during the idle interval transmits the synchronization *beacon* (a special **heartbeat** frame). After the beacon each nodes gets its *transmission opportunity* (TO). Each opportunity interval is very short (typically 20 bits), in that way the overhead for the silent nodes is low. If the PLCA circuit discovers that the

node's TO cannot be used (the other node with a lower ID have started its transmission and the channel is busy at the beginning of the TO for this node), it asserts the "local collision" input for the MAC thus delaying the transmission. The condition is cleared once the node gets its TO. A standard MAC reacts to the local collision with a **backoff**, however, since this is the first and only backoff for this frame, the backoff interval is equal to the smallest possible frame - and the backoff timer will definitely expire by the time the TO is granted, so there is no additional loss of performance.

Recap

- each node on the BUS has an **ID**.
- the **ID 0** is the **coordinator** (master).
- at the begin of each TX cycle, the *coordinator* node sends a **beacon** which means "we are starting a new tx cycle".
- slave node (called **drop nodes**) are given a **transmit opportunity** (TO) in order of their assign IDs.
- like a round-robin scheduler without handshake, TO are counted by each node (it's a mix between flexray static and dynamic segment).

Example

Try to consider four nodes, in Fig 2.7 is possible to see the behaviour of a network where only the first and the third node have something to transmit, instead the other two keep quite.

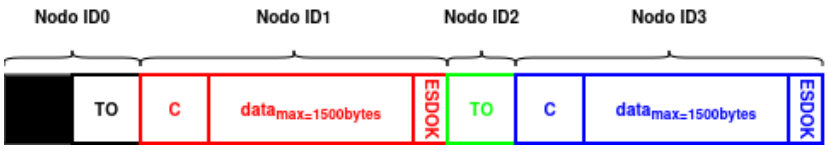


Figura 2.7: Communication between four nodes

When a node want to transmit send over the channel an **commit** message (*special function*) for indicates the start of data transmission and a **ESDOK** for notify other node that the transmission is finished, instead if the node does not have nothing to say, send over the bus an **transmit opportunity** that is long exactly 20bits.

Is possible for each node to activate the ***burst mode*** that allow to node to have more bandwidth and reduce latency (for audio/stream data).

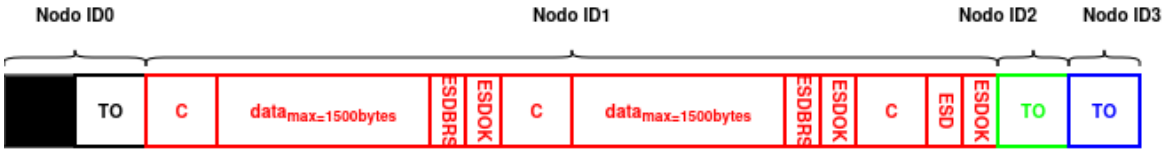


Figura 2.8: Communication with **burst mode**

In Fig. 2.8 is possible to see at the end of the transmission of node one the send of two package: **esd** and **esdok** that together they are named: **max_bc**. ***max_bc*** is a configurable parameter that allow nodes to transmit more than one packet per transmit opportunity. Value are from 0 to 255.

2.9.3 Worst Case Latency

1. **flexray**: worst case latency is "wait the next static slot" $1.5ms$ (an entire cycle).
2. **10base-t1s**: worst case latency means to have a 8 nodes (full net), that in one cycle each of them have something to transmit the maximum size of the data segment (1500 bytes) at $10Mbps$ without **burst mode**. $9ms$

2.10 CAN-XL

Goal:

- match the **bandwidth** of 10BASE-T1S.
- priority based on the IDs.
- **larger payload** (up to 2048 bytes) and **ethernet compatibility**.

Capitolo 3

Inter-Vehicles

3.1 Global Positioning System

The ***GPS*** in origin named **NAVigation Satellite Timing and Ranging Global Positioning System** (**NAVSTAR GPS**) is one of the space-based *global navigation satellite system* (**GNSS**) that provides **geolocation** and **time information** anywhere on the Earth, using the signal of at least four satellite. The geolocation information gives an **XYZ** coordinates.

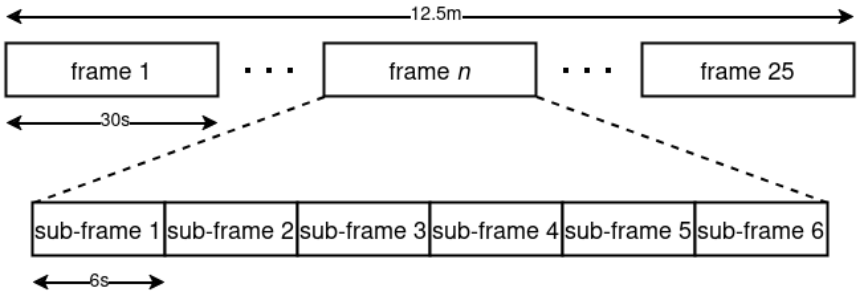
3.1.1 Actors

GPS is based on three elements: **space segment** (the satellite), **control segment** (ground station) and **user segment** (end-user equipments):

1. **space segment**: have the main goal to broadcast navigation message constantly.
2. **control segment**: ground antennas that *track*, *collect* and *correct* all the **sat orbits** (normally they are very precise and known a priory). They both recive/-transmit data from/to satellites.
3. **user segment**: cheap devices used to collect GPS signal and know the position (it has the maximum error approximately of 1 meter).

3.1.2 NAV Msg

The information **broadcasted** by the satellite is called ***Navigation Message*** (**NAV**).
The nav msg is composed by:

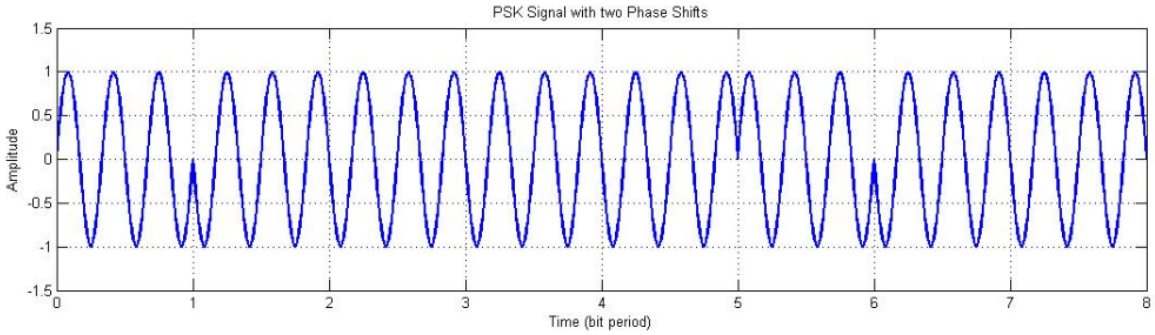


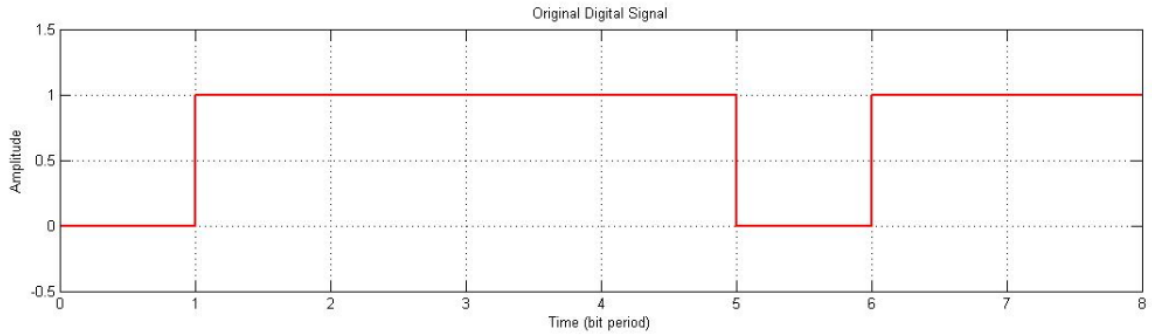
- **25 frames** (or **pages**) that takes 12.5 minutes to be transmitted.
- each frame takes 30 seconds to be transmitted and it is formed by **6 sub-frame**.
  1. the first sub-frame contains the **satellite clock information**.
  2. the second and the third give the information about the **satellite ephemeris** (**orbit**).
  3. the forth and thr fifth are different and are complete only receiving all the 25 frames of the NAV msg, they have the **Almanac & constellation status**.
- each sub-frame needs 6 seconds to be transmitted, and it is composed by **10 words**.
- each word consist of **30 bits** and it takes 0.6 second to be transmitted.

3.1.3 Bit Coding

GPS use the **Bi-Phase Shift Key** (**BPSK**) modulation technique. In BPSK the carrier signal is modified by altering its phase by 180 degree, for each symbol. A phase shift of 180 degrees denotes a binary 0 while no phase shift represents a binary 1. The advantages to use this type of mudaltion technique is:

1. rendundancy
2. jamming resistance
3. measure & remove the ionospheric delay
4. requires a dual frequency receiver (with a single one it is possible to survive but less accuracy) one at $1575.42MHz$ and the other one at $1227.60MHz$.





3.1.4 Working Principle

Let's distinguish the study of the working principle in two hypotheses: **theoretical**: receiver clock is perfectly synchronize with the satellite clock (*absolute clock*); and **reality**: receiver clock is cheap, non-atomic and not perfectly synchronize with the satellite clock.

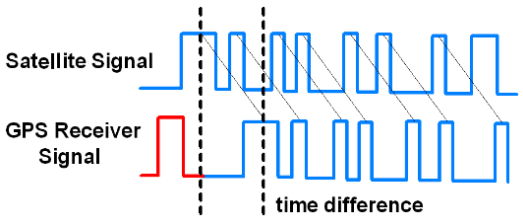
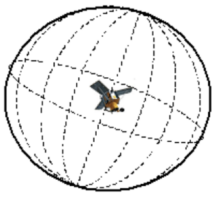


Figura 3.1: Satellite-Receiver clock synch

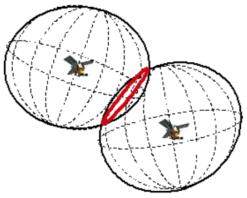
In the first hypothesis, we know: **satellite position** (written in the NAV), **satellite clock** (written in the NAV) and the **speed of light** c. So it is possible to obtain:

1. **signal travel time**: $\Delta t = clock_{recv} - clock_{NAV}$
2. **distance satellite-receiver**: $d = c \cdot \Delta t$

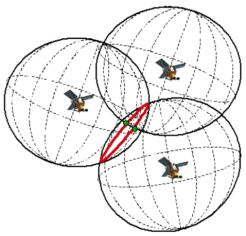
The problem is, if there is at least $1ms$ of de-sync between the satellite and the receiver, then will have at least an error of 200 miles. Solution: ***Trilateration***.



if it is knows **one** sat_i and the distance d_i we can be in any point of the spherical surface of radius d_i centered in sat_i.



if the sat known are **two** sat_i and sat_j both the position and the distance d_i, d_j we can be in any point of the border given by the intersection of the two sphere's surface.



adding the **third** sat, we have **three** sphere, that ends the intersection with two available points. One is on the Earth surface (us) the other is in the open space (discard).

In the **reality** the satellite and the receiver do not have the clock synchronize, so it is important to introduce a factor named *clock error* Δe:

$$d_i = \sqrt{(x_i - x_u)^2 + (y_i - y_u)^2 + (z_i - z_u)^2} + c \cdot \Delta e$$

where:

- x_i, y_i and z_i are the sat position, **know** (NAV).
- c is the speed of light, **know**.
- x_u, y_u and z_u are the position of the receiver, **unknown**.
- Δe is the clock error, **unknown**.

If we consider four satellites sat_i, sat_j, sat_k and sat_p we end with four equations and four unknown items: x_u, y_u, z_u and Δe.

$$\begin{cases} d_i = \sqrt{(x_i - x_u)^2 + (y_i - y_u)^2 + (z_i - z_u)^2} + c \cdot \Delta e \\ d_j = \sqrt{(x_j - x_u)^2 + (y_j - y_u)^2 + (z_j - z_u)^2} + c \cdot \Delta e \\ d_k = \sqrt{(x_k - x_u)^2 + (y_k - y_u)^2 + (z_k - z_u)^2} + c \cdot \Delta e \\ d_p = \sqrt{(x_p - x_u)^2 + (y_p - y_u)^2 + (z_p - z_u)^2} + c \cdot \Delta e \end{cases}$$

Δe is equal for all satellite, because all of them are perfectly synchronize, so the *clock error* is the same for each tuples (sat, recv). If recv and sat are perfectly synchronize (**time** given), with just 3 sat is possible to calculate your position. If you know where you are (**space** given), with just one sat is possible to have the clock sync, but if you do **not know** both **space** and **time** you need at least four sat to solve the equation.

3.1.5 GPS limitation

- it require a lot of power to work properly.
- GPS signal do not pass solid structure.
- affected by large buildings, unreliable in dense urban area.
- GPS accuracy is function of the signal reception, larger the antenna, better the signal. $miniaturization \quad \frac{1}{\alpha} \quad accuracy$

3.2 Bluetooth

Bluetooth is short-range wireless technology and it was introduce for the first time in 1994 to replace serial *RS-232* wired cables. Typically used for **point-to-point** technologies. It has a coverage of 10m and creates a network named **Personal Area Network** (**PAN**), the frequency range is between $2.4GHz$ and $2.485GHz$ with few *Mbps* of bandwidth. It is standardize in the *IEEE 802.15.1* like packet-base protocol.

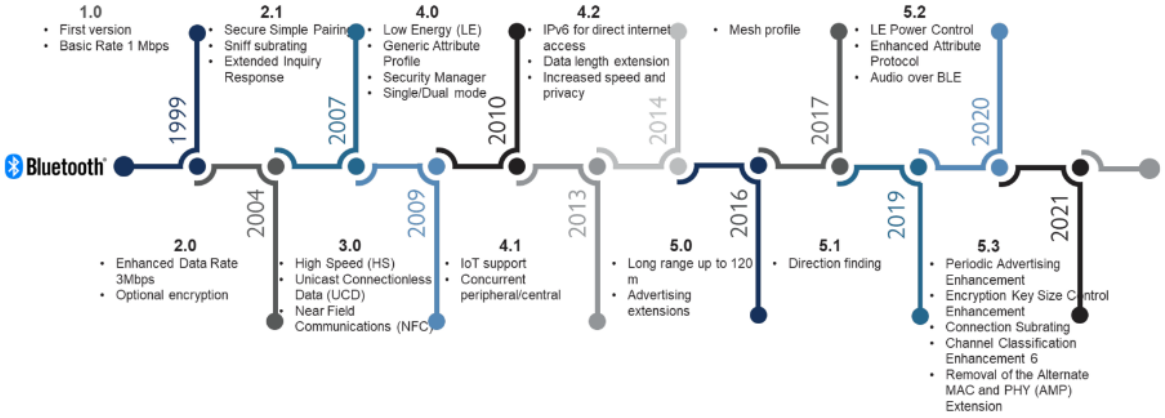


Figura 3.2: Bluetooth different version in the time

The bluetooth works at $2.4GHz$, like WiFi and ZigBee. A bluetooth network is named **piconet** and use a *master/slave* communication type.
In a bluetooth network there is always a **master** and up to seven **slave**, each slave can be only connected to one master. The master coordinates communication throughput the *piconet* it can send data to every one slave connect and can request data to each slave. The slaves can only talk with the master not between them.

3.2.1 Address & Names

The identifier for each node into a piconet (both for slave and master) its a **unique 48 bits address**, commonly abbreviated ***BD_ADDR***, usually is show as 12digit hexadecimal value, similar to the MAC address.

> **Name** is pretty different, it is also possible to give to each slave an user-frendly name. It can be up to 248 bytes long and two device can share the same name.

3.2.2 Connection Process

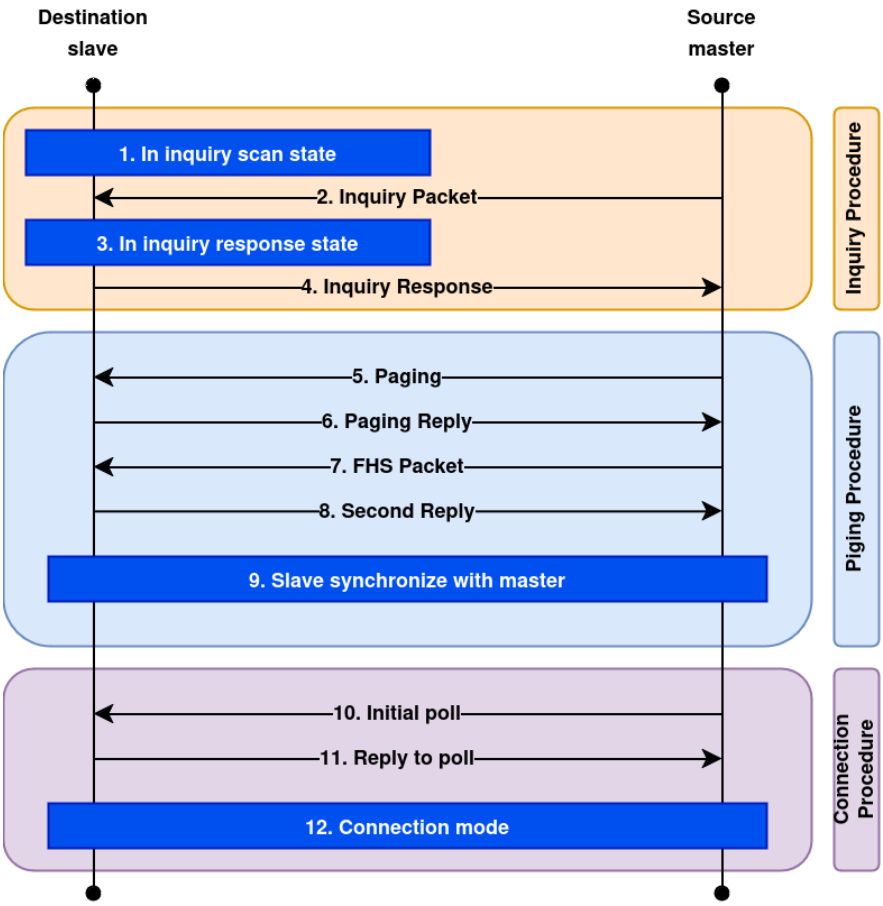


Figura 3.3: Connection Procedure in Bluetooth

Like show in Fig. 3.3:

- ***Inquiry Procedure***: the goal of this part is to retrive the information of each nearby nodes. A node run an inquriy scan and any listening device for this type of request, replies with its address, name and other info.
- ***Paging Procedure***: the purpose of this process is to create a connection between two bluetooth device. Each device **must knows** the address of the other. ***FHS***: **Frequency Hopping Sequence**.
- ***Connection Procedure***: after the *paging* process it is necessary to define the **connection state**.

3.2.3 Connection

A bluetooth connection can have four kinds of mode:

1. ***active mode***: this is regular connected mode, where the device is actively transmitting or receiving data.
2. ***sniff mode***: the device is active periodically for a certain amount of time (*power-saving mode*).
3. ***hold mode***: is a temporary, power-saving mode where a device sleeps for a defined period (not necessarily periodic) and then returns back to active mode when the interval passed. The master can command a slave device to hold.
4. ***park mode***: when the master command a slave to "park", that slave become inactive until the master tells it to wake up.

3.2.4 Pairing

Paired devices automatically establish a connection whenever thery are close enough. No UI interaction are required.

When device pair up, they share their address, name and profiles. Usually all this information are stored in memory. They also share common **secret key**, which allow them to bond whenever they want.
Pairing usually requires an **authentication process** where a user must validate the connection between devices, it could be different depending on the domain, some device ask to **press a button** (headset) or other can ask to **insert a code** (PC or smartphone).

3.2.5 Power Class

The transmit power and therefore range, of a bluetooth module is defined by its **power class**. There are three defined class power:

Class number	**Max Output Power**$_{dBm}$	**Max Output Power**$_{mW}$	**Max Range**
class 1	20	100	100m
class 2	4	2.5	10m
class 3	0	1	10cm

3.2.6 Profiles

While bluetooth **specification** define how the technology *works*, **profiles** define how it is *used*. Two bluetooth device are **compatible** if they **support the same profiles**. Some explanation of the most common bluetooth profiles:

- ***Serial Port Profile*** (***SPP***): permit to substitute a serial communication protocol (like RS-232 or UART). *SPP* is great for sending data between two device
- ***Human Interface Device*** (***HID***): is the profile to enable the receiving data from the user-input device (like mice, keyboard or joystick). **Bluetooth HID** substitute **HID** alredy defined for human input USB device, the innovation is the substitution of the USB cable for the communication.
- ***Hands-Free Profile*** (***HFP***) and ***Headset Profile*** (***HSP***): *HFP* implements a few features on top of *HSP* to allow common phone interaction (accepting/rejecting calls, etc.) that occur while the phone remains in your pocket.
- ***Advanced Audio Distribution Profile*** (***A2DP***): *A2DP* define how audio can be transmitted from one bluetooth device to another. *HFP* and *HSP* have a bidiretion communication, instead *A2DP* is one-way street, that permit higher quality potential.
- ***A/V Remote Control Profile*** (***AVRCP***): the audio/video remote control profile allows for remote controlling of a bluetooth device. It is usually implemented alongside *A2DP* to allow the remote speaker to tell the audio-sending device fast-forward, rewind, etc.

3.2.7 Bluetooth Stack

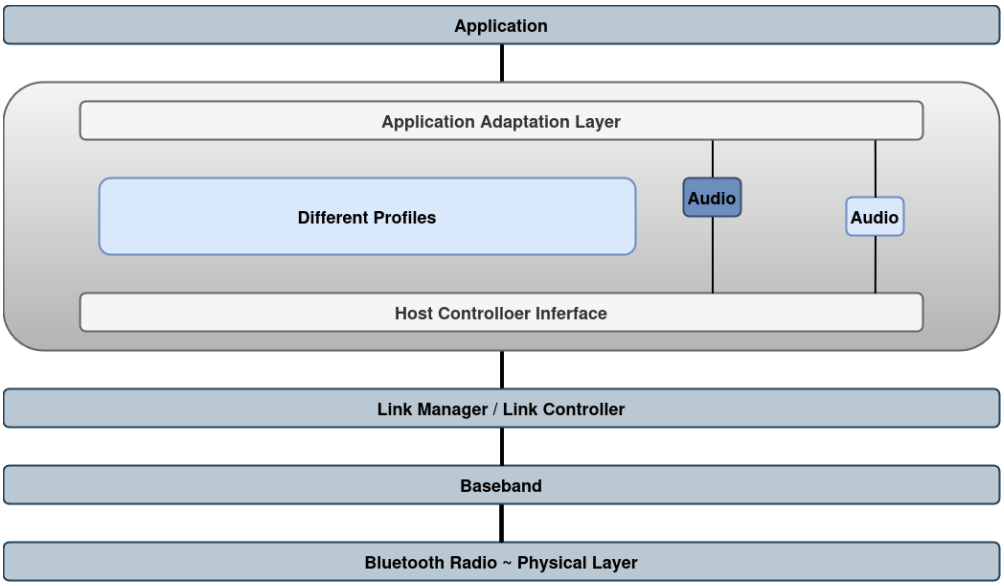


Figura 3.4: Bluetooth Stack

The bluetooth protocol stack is split in two parts: a **controller stack** (in Fig. 3.4 the blu/gray section) containing the timinig critical radio interface and **host stack** (in Fig. 3.4 the biggest container) dealing with high level data.

Radio Layer

The bluetooth radio layer is used for transmit bit from master to slave (or vice-versa) and it is similar to *physical layer* in ISO/OSI model. It is a **low-power system** with 10 meters range operating in the same frequency of WiFi and ZigBee: $2.4GHz$. It defines the specification for the transceiver device:

1. ***Frequency Bands and Channel Arrangement*** (***FHSS***)
2. **Transmitter Characteristics**
3. **Receiver Characteristics**

The ***Frequency Hopping Spread Spectrum*** is used for the transmission. The **Spectrum Spreading** is obtain by hopping between 79 frequency segments distributed between $2.402GHz$ and $2.480GHz$ ($1MHz$). The transmitter and receiver stay on one of this channels for a certain time and then hop to another channel. This system implement **Frequency Division Multiplexing** (**FDM**) and **Time Division Multiplexing**

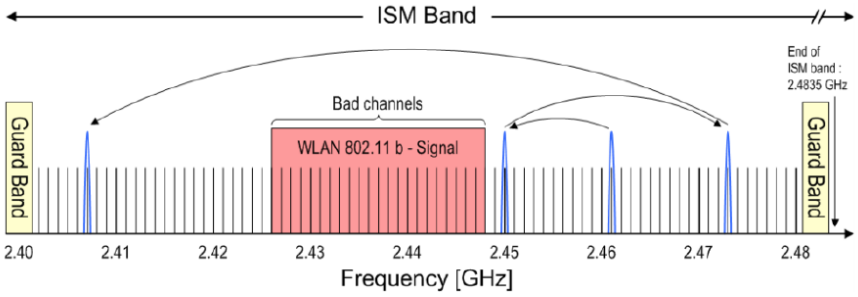
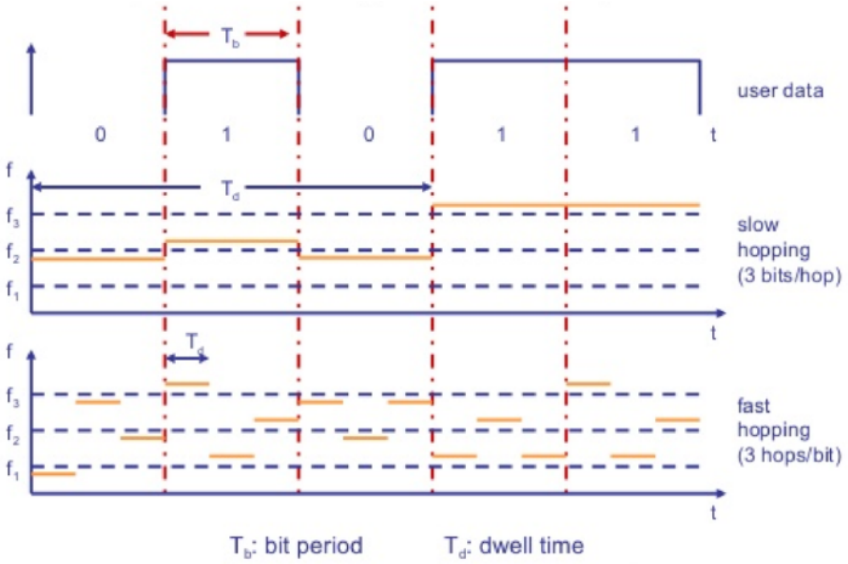


Figura 3.5: Frequency Hopping Spread Spectrum

The changes of frequency it could be performed in two different way:



- ***slow hopping***: the transmitter uses one frequency for several bit period:
  - typically cheaper (relaxed sync constraints, more time per slot)
  - not immune to narrowband interferances
- ***fast hopping***: the transmitter uses several frequencies for one period:
  - typically expensive (hard sync constraints, more slot jumps and high resync frequency)
  - almost immune to narrowband
  - rate 1600 hops/sec: $625\mu s$ of slot time

Baseband Layer

While the **radio layer** is controlled by the transceiver, the **baseband layer** is implemented by the controller of a device how want to transmit in bluetooth. The *baseband* is the **physical layer** of the bluetooth and is used to manages physical channel and

links, but also *error connection*, *data whitening*, *hop selection* and *bluetooth security*. Is is used to managed **synchronous** and **asynchronous** links. It handles packet and perform the *inquiry* and the *paging* for create connection with nearby device. The baseband layer applies a **time division duplex** (**TDD**) scheme to alternating transmitter and receiver. Baseband handles two types of links:

1. ***Synchronous Connection Oriented*** link (***SCO***): is a symmectric point-to-point link between a master and a single slave in the piconet. The master maintains the *SCO* link by using reserved slot at regular intervals:
   - *SCO* is commonly used for the transmission of voice information.
   - master can support up to three simultaneously *SCO* links, instead, slave only two (sometimes three).
   - *SCO* packets are never re-transmitted, normally used for $64kB/s$ speech transmission.
2. ***Asynchronous Connection Less*** link (***ACL***): is point-to-multipoint between the master and all the slave present in the piconet. In the slot not reserved for the *SCO* links, the master can establish an *ACL* link to any slave
   - slave alredy into a *SCO* communication are included.
   - only a single *ACL* link can exist.
   - for most of *ACL* packets, re-transmission is applied.

These means that *SCO* does not need a connection, there is not an acknoledgement (like **UDP**), instead the *ACL* need the *ACK* to check if the message needs to be re-transmitted (like **TCP**).

Bluetooth Packet

bits	72	54	0 - 2745
description	**access code**	**header**	**payload**

1. ***access code***: is used for *time synchronization*, *offset compensation*, *paging* and *inquiry*.
2. ***header***: contains information for *packet acknoledgement*, *packet numbering* for out-of-order packet reordering, *flow control*, *slave address* and *error check* for header.

3. ***payload***: can contain either voice field, data field or both. The payload will also contain a **payload header**.

Other Baseband Function

The baseband makes other features available like:

1. ***error correction***:
   - $\frac{1}{3}$ **rate FEC** (**Forward Error Correction**): every bit is repeated three times (**rendundancy**), in this way the receiver can discard up to two bits and validate the transmission.
   - $\frac{2}{3}$ **rate FEC**: it use a *polynomial generator* to **encode** ten bits code into fiftheen bits code.

     > If i want to send 1010001111 that is ten bits long, i divide into two pieces: 10100 and 01111 and i will compute the XOR: $10100 \oplus 01111 = 11011$. I will send the three concatenated values: $m = [10100, 01111, 11011]$. The receiver it could receive wrong the second value, but it can rebuild it performing the XOR between the first and third value. It allow only one "wrong" value, if not it can not reconstruct the original message.

   - **ARQ scheme** (**Automatic Repeat Request**): uses *ACK*, *NACK*, *RTO*, etc.
2. ***flow control***: use **FIFO** queues both in *ACL* and *SCO* links for transmission and receive. In the case that the **RX FIFO** queues are full, **flow control** is used to avoid **congestion** and **packet drop**. The receiver send a **stop** indication, the transmitter **blocks** its **TX FIFO** queues. When the queues of the receiver are ready it sends an **go** signal to resume the transmission.
3. ***synchronization***: the piconet is sync using the master clock and it is needed for the transmission at least three information:
   - **channel hopping sequence**
   - **phase of the sequence**
   - **channel access code** to place on the packets (piconet code)
4. ***security***: at link layer, security is maintained by authentication of the peers and encryption of the information. For this basic security we need a public

address which is unique for each device, two secret key (authentication keys and encryption keys) and a random number generator.

Link Manager Layer

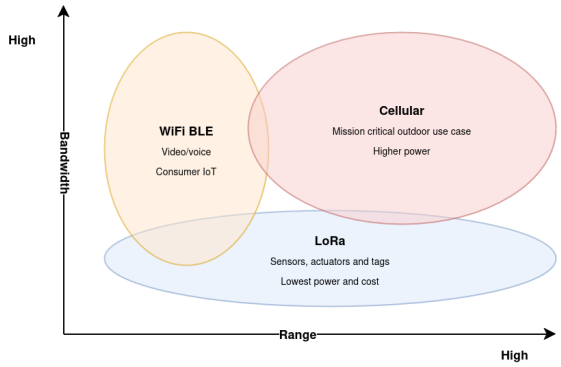
On top that layer it is used the ***Link Manager Protocol*** (***LMP***) that carries out link setup, authentication, link configuration and other protocol. The Link-Controller/Baseband provides services to **LMP** likes: *authentication*, *pairing*, *encryption*, *change link key*, *slot/clock offset request*, *switch master/slave role*, *hold/sniff/park role* and *quality of service*.

Automotive & Bluetooth:

- **in-car infortainment system**: permit hands-free audio, calling and application control.
- **remote keyless system**: smart phone are the new key, that use *proximity detection* for locking and unlocking car.
- **in-vehicle wearables**: permit to monitoring the driver health.
- **under-the-hood & connected maintenance**: allow to transfer diagnostic information.

	bluetooth	**bluetooth low energy**
freq. band	2.4GHz ISM Band	2.4GHz ISM Band
no. of channel	79, one 1MHz	40, one 2MHz
power consumption	low	less
data rate	between 1$Mbps$ and 3$Mbps$	1$Mbps$
latency	$\approxeq$ 100ms	$\approxeq$ 6ms
range	$<$ 30m	50m (in open area: 150m)
topology	peer-to-peer (1:1)	peer-to-peer (1:1) star (many:1) broadcast (1:many) mesh (many:many)
device pairing	required	not required
voice capable	yes	no
node/active slave	7	unlimited
security	64bits or 128 bits	128 bits AES
smartphone compatibility	100%	100%
use cases	streaming application like audio streaming, file transfer and headset	location beacons, smart home application, medical deivces, industrial monitoring and fitness trackers

3.3 LoRa: Long Range



LoRa has an unlicenced frequency band equal to: $868MHz$ (in Europe) and very long range coverage: up to **10 km**. It is composed by two part:

1. **LoRa**: the *physical layer* that is **proprietary**.
2. **LoRaWAN**: **Long Range Wide Area Network** that defines the upper layers (in particular MAC layer).

3.3.1 LoRa Stack

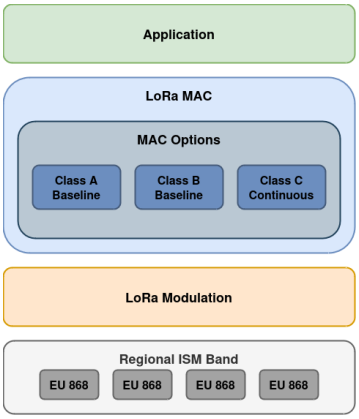


Figura 3.6: LoRa Stack

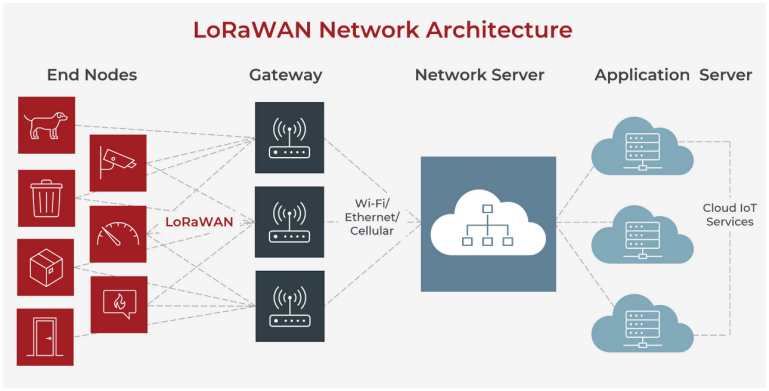


Figura 3.7: LoRaWAN network architecture

LoRaWan defines the *MAC Layer*, the point-to-point network, the communication protocol and the system architecture standard (Fig. 3.7). It, also, managed the *frequency*, *data rate* and *power device*.

- **end device, node** an embedded object with low-power communication constraints.
- **gateway** receive broadcast data and send data from/to the *end device*.
- **network server** route message from *end device* to the right application, and back.

Addressing: to each *device* is assign an **unique identifier** (**DevEUI**) of 64 bits, instead to each *application* is attribute **distinctive fingerprint** (**AppEUI**) of 64 bits, moreover after the access inside a network of a new device it receives a **dynamic** *non-unique* address of 32 bits (**DevAddr**).

Frame Counter prevent **replay attack**, to prevent this each nodes (both *end device* and *server*) reject message that contain frame counter that is lower than the expected ones.

The *end device* could be classified into **three different classes**:

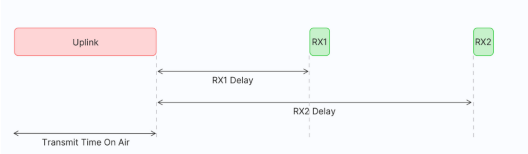


Figura 3.8: **Class A**

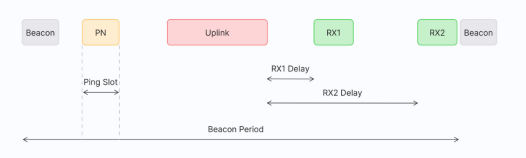


Figura 3.9: **Class B**

Support bidirectional communication (from/to gateway), transmission messages can be sent in any time (random), two possible receive windows at specific time (after 1s and after 2s) after a transmission message, the gateway can reply in one of this two windows.

Extend *class A* by adding scheduled receive windows for receiving message from the gateway/server. It use time-sync beacons transmitted by the gateway, the *end device* periodically open receive window.

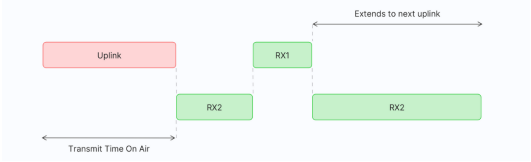


Figura 3.10: **Class C**

Extend *class A* by keeping the receive window open unless they are transmitting. This allow **low latency** communication, but it consume more energy than *class A* device.

3.3.2 Secuirty

LoRaWAN specifies three different class of **security keys**: ***NwSKey***, ***AppSKey*** and ***AppKey***, all have the lenght of 128 bits. The algorithm use for the encryption is ***AES-128***.

When a device joins the network (called **activation** or **join**), an *application session key*

(**AppSKey**) and *network session key* (**NwSKey**) are generated that will be used for the duration of the session. The **AppSKey** is the *private key*, instead the **NwSKey** is the *public key*. The **Network Session Key** is used for the interaction between the node and the server and it is used to validate the integrity of each message using the **Message Integrity Code** (**MIC**). The **MIC** is similar to checksum expect that it prevents intentional tampering with message (**AES-CMAC**), it is also used to map the **DevAddr** to both **AppEUI** and **DevEUI**. The **Application Session Key** is used for encryption and decryption of the payload.
The **Application Key** (**AppKey**) is only know by the device and the application. The **NwSKey** and **AppSKey** are derived from this key. If you dinamically activate the device (**Over the Air Activation - OTAA**) the two sessione key are regenerate.

3.3.3 MAC Commands

The *end device* and the *server* use **MAC Commands** for configure the transmission and for the communication.

- checking connectivity
- requesting the status of the device
- adapting the data rate of the device
- modify channel settings

Pros	Cons
long range	not realtime data (packet each minutes)
low power	not use for domotic house (ZigBee or Bluetooth)
low cost $\approx$ 20\$ per node	watch video (WiFi)
low bandwidth between $250bit/s$ and $11kbit/s$	
secure: 128 bits end-to-end encryption	

3.4 V2X

Intelligent Transportation System (***ITS***) add information and communications technology to transport infrastructures and vehicles in effort to improve their safety, relaibility, efficiency and quality → avoid congestion (easy work for the drivers).
European Telecommunication Standard Institute (***ETSI***) standardize some new messages that allows to be transmitted into the network: ***CAM*** and ***DENM*** to permit safety, a new way to send broadcast message into the network, efficiency and enabling the **QoS** in the frame.

V2X	=	***IVC*** (***Inter-Vehicles Communication***)	=	***VANET*** (***Vehicle ad-Hoc Network***)

Different name, but same thing. Today the most used and the standardize one is the **V2X**, that indicates multiples thigs that can be connect with the vehicle:

- **V2V**: vehicular-to-vehicular
- **V2I**: vehicular-to-infrastructure network
- **other**: it depends on top of which domain are created

It is possible identify two different **application** for **V2X**:

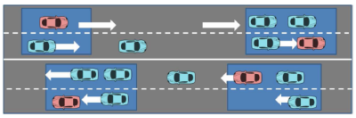
Non Safety Purpose (≅ Infotainment): many messages and high data-rate, low latency and low reliability demand.	***Safety Purpose*** (≅ X-by-Wire): few message (sporadic) and small packet size, high latency and high reliability demand. It is not possible to tollerate false positive or true negative messages.

Based on the requirements of the domain where we want deploy **V2X** we can have different application with different characteristics (latency, reliability, area and persistence).

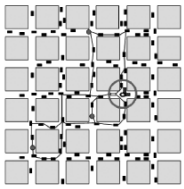
Timeline

Traditional Network	→ ***Mobile ad-Hoc Network (MANET)***	→ ***Vehicular ad-Hoc Network (VANET)***
• wired • non-moving nodes • static configuration	• wireless • mobile nodes • dynamic configuration • infrastructure (optinal): WiFi and Mobile cellular line.	• dynamic topologies (could be bus or star) • infrastructure(less) • broadcast or unicast messages

It is necessary to change the name from **MANET** to **VANET** because there is two main different purpose of the network utilization moreover **VANET** could change its own topologies dynamically. It is possible to distinguish **VANET** in two different scenario:



With only "one direction" for the movement we can have a *stable connection* (if the vehicles are going in same the way) or *unstable connection*, if the vehicles are not going in the same way (few second of connections). **Autonomous Drive** is drammatic easy on freeway (no cross road or obstacles)



There are "two direction" for movement. The circumstances can change based on the environment: if the vechicle is driving it can have few neighboard (multiple hops to reach someone faraway), instead in a parking area the number of neighboard it can be easily increase. In the urban scenario it can be possible to have *obstacles*: **GPS** not always work.

Firmware Over The Air (***FOTA***)

Stationary Support Units (***SSU***): it is the name give to a nodes that is more important than the other and can have multiple scope:

- provide connectivity to other nodes.
- manage the radio use for connectivity purpose.
- unlimited power supply.
- is not connected to internet: is part of the jungle, but more important.

RoadSide Unit (***RSU***): it is an **SSU** with the internet connection, you can act as a piece of the infrastructure and can enable the traffic to the ***Traffic Information Center*** (***TIC***).

It is possible identify two different **infrastructure** for **V2X**:

infrastructure

(problem alredy solve):

- resource management.
- add latency (multiple hops).
- high load on the *core network*.
- high throughput.

infrastructureless:

- self organized: channel access and authentication are the biggest problem.
- low latency (no multiple hops).
- low datarate: wireless environment in which multiple nodes can communicate simultaneously.

In the **infrastructureless** scenario it is necessary to define by your own the rules that the nodes must be compliant to, to allow communication. Moreover there are an heterogeneous scenario to support where the communication can be *point-to-point* or *broadcast*.

CHALLENGES		
	Communication	The channel condition dynamically varying based on network topology and the communication type (unicast or broadcast). If multiple nodes want to communicate in the same time it could be happen **congestion** or **starvation** on the medium channel
	Networking	The flow information that was send from a node A to a node B can be unicat or multicast (following access algorithm of **CSMA/CA**). In the same network it can be present multiple different nodes: **heterogeneity**
	Mobility	Not only for the high speed but for the dynamically change of the topology. It can be predict.

3.5 Wireless Communication

1. **Broadcast Media** (one way): frequency are used for business purposes, you can only receiveing data, like police radio.

   Traffic Message Channel (***TMC***)
   Radio Distrinbution System (***RDS***)

2. **Cellular Lane**: it is possible to put bottom in **V2X**. Divide the world into cell (***User Equipment - UE***), each served by base station (**3GPP** have *license* spectrum frequency).

   Frequency Divide Multiple Access (***FDMA***)
   Radio Network Core (***RNC***)

3. **IEEE 802.11p**: new communication protocol based on the requirements of the **V2X**.

3.6 C-V2X

It is the acronym for **Cellular V2X** and try to embrace the characteristic of **V2X** on the public system, this type of implementation leads to some problem, like:

1. using an infrastructure alredy used for other purpose it is possible to have *delay*.
2. if the number of the nodes in the infrastructure double (the ones alredy present and all the vehicles plus the RSUs) there are a lot of nodes in the infrastructure. This can lead to *capacity* problems
3. if we consider that one autonomous car use $4000Gb$ of data per day, it can involve some *bit-rate* problem.
4. it is *expensive*.

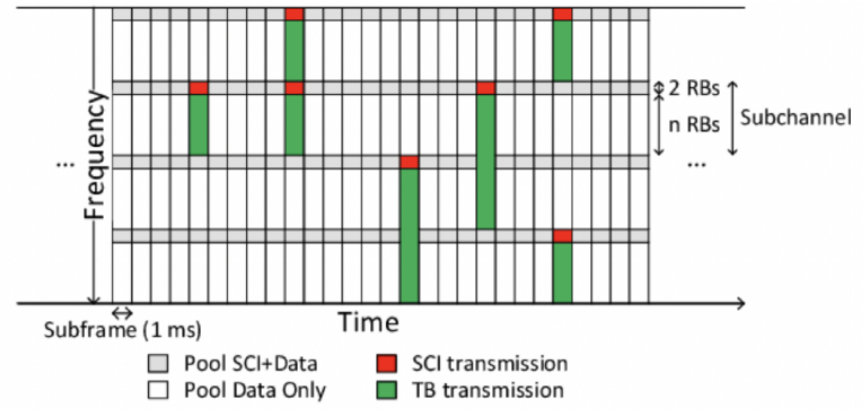
Since the cellular line infrastructure it was create for other purpose, instead the **V2X**, the downlink and uplink are **unbalance**, so the communication channel could be:

1. **downlink**: from the server to the device the delay is on the order of $10ms$. ***Forward Access Channel*** (***FACH***).
2. **uplink**: from the device to the server, it can have a delay approximately of $50ms$ (if we use **LTE** it can be lowered until $25ms$). ***Random Access Channel*** (***RACH***).
3. there is also a dedicated channel with a delay approximately between $250ms$ and $2s$. ***Dedicated Transport Channel*** (***DCH***).

It is possible to classified the **C-V2X** based on the **radio interfaces** used: **LTE-Uu** each node is attach to the infrastructure and for communicate each other a node A upload the information on the server and the node B read throught the server (*license spectrum*) or the **LTE-PC5**, where there is not infrastructure.
In the **LTE-PC5** if you are in the same area of coverage of an antenna you can use the support of the infrastructure (never implemented), instead if you are out of coverage it is possible to use point-to-point commnuication using **LTE** at $5.9Ghz$ frequency ($\simeq$ 802.11p).

The **physical layer** of **C-V2X** is divide into ***Resource Block*** (***RB***) with a width of $180MHz$ each one. Every **RB** is split into ***Time Unit*** (***TU***) each one of $1ms$ long. In this way is performed a matrix where on the y-axis there are the **frequency** and on the x-axis the **time**. If a node wants to transmit something it needs to take possession of a **RB** for a certain amount of time (m **TU**), for each transmission there are two different type of message classes:

- ***Transport Block*** (***TB***) where is allocated the data.
- ***Sidelink Control Information*** (***SCI***) that permit to manage the communication: rent a **RB** for a certain amount of time. In this type of message is included the ***Modulation and Coding Scheme*** (***MCS***).



TB and **SCI** are transmitted in adjacent **RBs**, the **SCI** requires two adjacent **RB**. The idea behind the medium access is similar to bluetooth, where we change the frequency each time we want to transmit a piece of information, in this case we take a frequency for a certain amount of time for the entire duration of the transmission $\cong$ ***Time Division Multiple Access*** (***TDMA***) it is mandatory to have synchronize all nodes $\rightarrow$ need **GPS** and **GNSS** for the synchronization.

3.7 802.11p

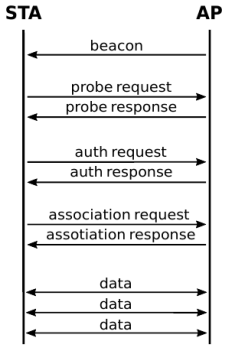


Figura 3.11: 802.11 data flow

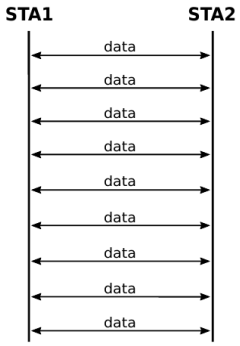


Figura 3.12: 802.11p data flow

IEEE 802.11

WiFi: high latency and used by everyone → lot of pollution in the world spectrum. It is created for the internet navigation have completly different requirements by **V2X**. There is no **QoS** by default.

Basic Service Set (***BSS***): there is a node that orchestrate the network (authentication, routing and iP assignement) contain the **BSSID**. The Fig. 3.11 define how a client joining the communication. The server sent a **beacon** that contain the frequency for that access point and propagate the **SSID**.

IEEE 802.11p

bandwidth: $10MHz$

throughput: $3 - 27Mbit/s$

range: up to $1km$

speed: $200km/h$

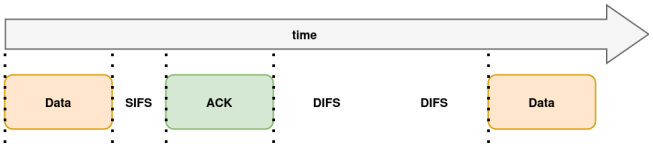
It is created knowing the requirements for the **V2X** use the *MAC Layer* of **802.11a** plus extension:

1. random **MAC address**
2. use of ***Enhanced Distributed Channel Access*** (***EDCA***) for prioritizing the message on the medium access (**QoS**)
3. multi-frequency and multi-radio capabilities

The base idea is of **802.11p**: there is no need of an iP you can listen and talk on a particular frequency. (Fig. 3.12), is named: ***Outside The Context of a BSS*** (***OCB***). **802.11p** is the technologies adopt for the implementation of **V2X** also known as ***Wireless Access in Vehicular Environment*** (***WAVE***). Two new tipe of node: ***RoadSide Unit*** (***RSU***) and ***On-Board Unit*** (***OBU***).

3.7.1 Physical Access

There are two mainly actor in the **802.11p** network: the *provider* and the *user*, but there is no technical difference between them. The main task of the **provider** is to send a ***On-Deamand Beacon*** ($\simeq$ beacon in **802.11**) that contain the information and parameter to join the network. The **physical access** is **wireless**.



Distributed Coordinator Function (***DCF***) of **802.11** two main intervals (it is a time that a station have to wait before transmit):

1. ***SIFS*** - ***Short Interframe Space***: before the acknoledgement.
2. ***DIFS*** - ***Data Interframe Space***: before the data.

In a wireless environment it is not possible to anticipate what the other are going to do: **Carrier Sense** getting a look at the medium listen until is idle, **Avoid Collision** if the medium is congestionated you can increase the intervals.

3.7.2 Quality of Service

Before continue the physical access part on the **802.11p** small introduction on what is the **QoS**. Differently from module 1, in this part of course we have **logical link**, but **channel/medium wireless**, in this way the priority it cannot be written in the protocol, but must be written in the message and we have to treat according to its rules (written in the msg).
$\rightarrow$ give me a packet I will control and I will decided what to do according to the service that you want.

QoS is the overall performance of a telephony or computer network (important for traffic with special requirements $\rightarrow$ different protocols means different requirements). We need to have a **dedicated QoS** for each **TCP/iP level**.

1. ***H2N*** (include *physical layer* and *data link layer*) 3-bit field called ***Priority Code Point*** (***PCP***) identify the priority of that frame using these 3-bit:
   - the type of modulation that we have on the medium → **bitrate**.
   - bandwidth

PCP	**priority**	**traffic type**
1	0 (lowest)	background
0	1	best effort
2	2	excellent effort
3	3	critical application
4	4	video $< 100ms$ latency and jitter
5	5	voice $< 100ms$ latency and jitter
6	6	internal control
7	7 (highest)	network control

2. ***Network (iP)*** is used for **packet schedule** and **routing protocol**:
   - ***IntServ***: rent a certain resourse for all the router on the path between you and the end node (impossible to actuate except for *bank domain*).
   - ***DiffServ***: every single hop you try to do your best. If there is a packets more important than one other the router try to forward first the most important one. The **problem** is that this type of control affected the end-to-end performance with all the router performance (the scheduling complexity is not linear).

     Different Services Code Point (***DSCP***) is the method used in networking to classify and manage network traffic by assigning 6.bit value in the iP header, allowing for prioritization of different type of data. $2^6 = 64$ different class. The mainly used are the ***Per-Hop Behaviour*** (***PHB***):
     - (default) **best-effort**
     - ***Expedited Forwarding*** (***EF***): low loss and latency traffic.
     - ***Assured Forwarding*** (***AF***): assurance of delivery.
     - **class selector PHBs**: backword compatibility

We cannot have a one-to-one conversion from 3-bit of **H2N** to the 6-bit of **iP** so it was define a mapping between **PCP** and **DSCP**.

Lv2	Lv3		**Application**
PCP	**DSCP**	**PHB**	
0	0	0	best effort
1	8	CS1	torrent
1	10	AF11	bulk data
2	16	CS2	network management
2	18	AF21	transactional data
3	24	CS3	call signaling
3	26	AF31	mission-critical data
4	32	CS4	streaming video
4	34	AF41	video conferancing
5	46	EF	voice
6	48	CS6	routing
7	56	CS7	network control

3. ***Transport***:
   - **TCP**: congestion control, fairness among flows and friendliness among TCP algorithm.
   - **UDP**: no congestion control, problem delegated at layer three.
4. ***Application***: it became **Quality of Experience**

3.7.3 Physical Access pt.2

802.11 uses **DCF** for the medium access but for the **V2X** purpose there are some limitation:

- there is no congestion control: many station means many collision that produce lower bandwith and have not enough bandwidth means congestive collapse.
- no QoS guarantees: there is no difference between packet with higher priority instead of the lower priority.

In some cases **DCF** is replaced by ***Point Coordination Function*** (***PCF***) but need an infrastructure, that in **V2X** can be present or not and uses a *centralized function* (present in the **AP**) for enabling the QoS mechanism.

802.11p uses a ***Hybrid Coordination Function*** (***HCF***) like policy for the medium access.

HCF details:

- include the **802.11e** amendment for the definition of a set of QoS enhancements for wireless LAN application through modification to the media access control layer.
- convert the **DCF** into ***Enhanced Distributed Channel Access*** (***EDCA***).
- convert the **DIFS** into the ***Arbitration Interframe Space*** (***AIFS***): it is close to what we have seen in CANBus, permit to associated a number which change according to the priority of the packet that you have to transmit.
- defines ***Traffic Categories*** (***TC***).

The core idea is force an application to use an **AIFS** shorter or larger according to is priority: modify the **CSMA/CA** using shorter **AIFS** for higher priority packets.

Enhanced Distributed Channel Accass (***EDCA***)
It is a ***TCMA*** protocol (***Tiered Contention Multiple Access***) that define the **AIFS**. It classify user data in four **Access Categories** (**AC**): from **AC0** (lowest priority) to the **AC3** (highest priority).
It permit to enable **Burst Mode** using the **Transmission Opportunity** (***TXOP***): is a bounded time interval during which a station can send many frames as possible, during this period the station does not need to use standard **EDCA** to access the channel.
Each **ACs** has different ***Contention Window*** (***CW***), **AIFS** and **TXOP**.

In **802.11p** te four access categories have its own queue and each of then compete

in an indipendent way (Fig. 3.13) to access to the medium (with its own algorithm), instead of **802.11** we have the *scheduler*, but in the **11p** is not enough because a device A have to transmit a new packet with higher priority than a packet that the device b have to transmit it can collide with the last one. In this protocol the **access** is more important tha the **backbone**.

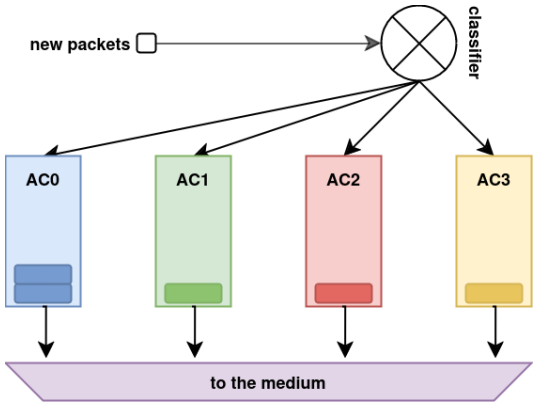


Figura 3.13: Medium Access - EDCA

HCF maps eight user priorities with four access categories → mapping:

	802.11p			**802.11p QoS**	
priority	**PCP**	**Acronym**	**Traffic Type**	**AC**	**Disignation**
lowest	1	BK	background	AC_BK	background
	2	-	spare	AC_BK	background
	0	BE	best effort	AC_BE	best effort
	3	EE	excellent effort	AC_BE	best effort
	4	CL	controlled load	AC_VI	video
	5	VI	video	AC_VI	video
	6	VO	voice	AC_VO	voice
highest	7	NC	network control	AC_VO	voice

Another important table is the definition of the **EDCA** parameter for the **802.11p** communication for every type of **AC**

AC	CW_{min}	CW_{max}	AIFS	Max TXOP
AC_BK	15	1023	9	0
AC_BE	15	1023	6	0
AC_VI	7	15	3	3.008ms
AC_VO	3	7	2	1.504ms
legacy DCF	15	1023	2	0

3.7.4 802.11p Upper Layer

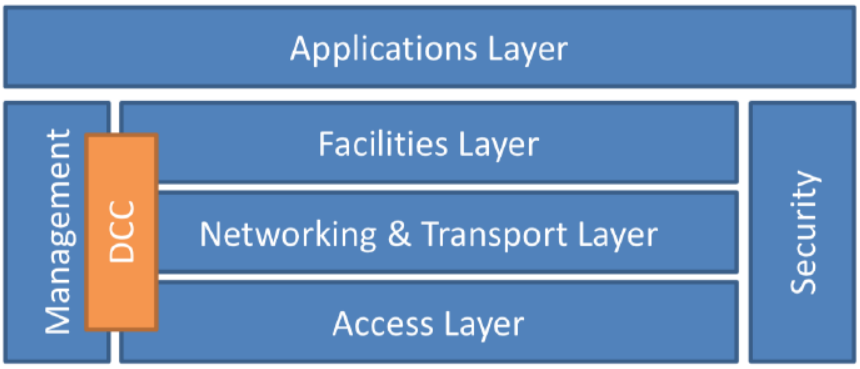


Figura 3.14: European Standard for 802.11p

Decentralize Congestion Control (***DCC***): the environment change based on where we are and so accordin with the environment and the application there are multiple parameter to continuosly read for avoid congestion. It is used an ***Finite State Machine*** (***FSS***) to know how to set the **802.11p** parameter for the transmission. There are three different state:

1. **relaxed**: low data rate and high range
2. **active**: neutral
3. **restrictive**: high data rate and low range

To relax the state (from restrictive to relaxed) the same *maxChannelLoad* must be in the correct range at least for five seconds, instead to limit the state (from relaxed to

restrictive) the same *minChannelLoad* must be in the correct range at least for one second, this difference between the transiction direction is to avoid the **isteresis** that means jumping from a state to another continuosly, polluting the transmission.

Input: will allow us to understand uf we have to change the state, normally is based on the **channel load** or the **measured received power** (**RSS**).

Output: each output have an associated value for each state:

- **Transmission Power**: is the bulk movement of electrical energy from a generating site
- **Minimum Packet Interval**: how fast modulate
- **BitRate**: how dealing the contention

3.7.5 ETSI Message

Cooperative Awareness Message (***CAM***)	***Decentralize Environmental Notification Message*** (***DENM***)
Very based information according to the awareness which is used to increase safety (position, speed and heading → where the vehicle is pointing geographically speaking). Is a periodic message, where the period is between $[100ms, 1s]$ if something happen in that interval there is no way of notification. The **CAM** check if $\Delta angle > 4°$ or $\Delta position > 5m$ or $\Delta speed > 1m/s$ if nothing of this param change significantly the **CAM** msg is sent every second.	decentralize because is something transmitted by who want, environment notification because is caused by event (**event-driven**) caused by vehicular sensor. Event like: hand braking, accident, construction work, imminente collision, low visibility, high wind, icy roads, etc. Have local scope based on area, road topology and driving direction

CAM message is **strongly related** with **GPS**.

3.8 Broadcast & Flooding