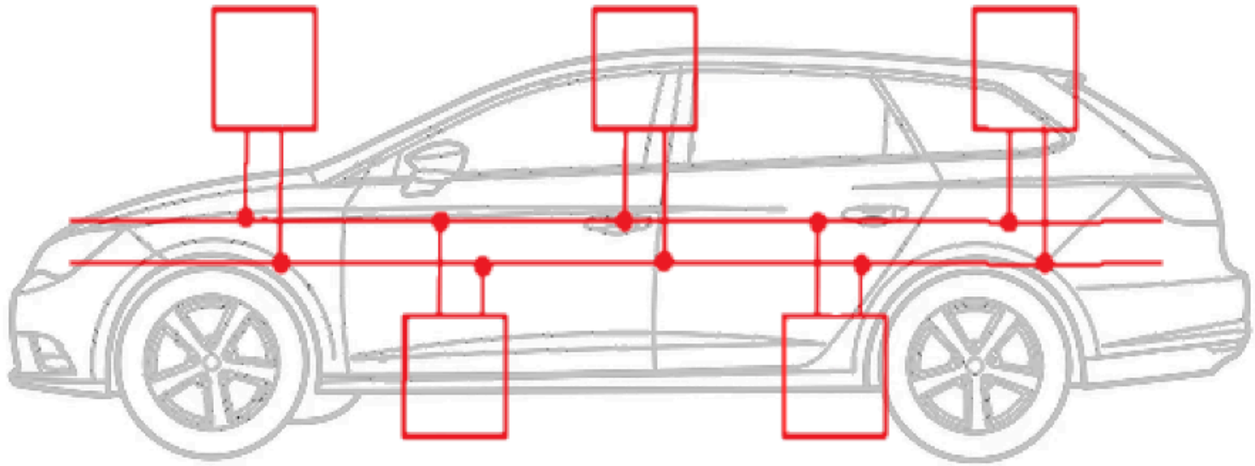
CAN PROTOCOL

Mohamed Bahae OUAADDI

mohamedbahae1402@gmail.com

Table of Content

CAN



CAN CONTROLLER AREA NETWORK

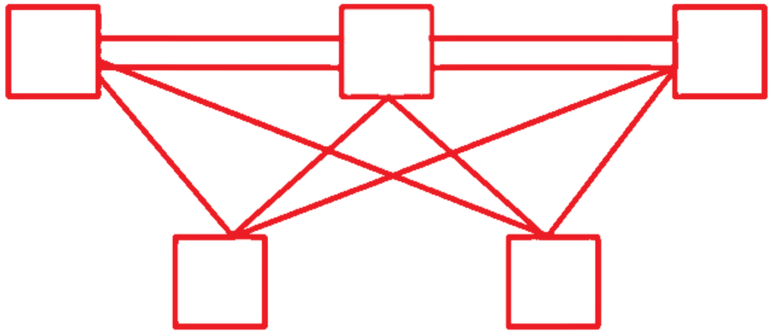
INTRODUCTION

CAN is a half-duplex asynchronous serial communication protocol developed by Robert Bosch GmbH in the early 1980s, and it was officially launched at the Society of Automotive Engineers (SAE) congress in Detroit, Michigan.
Initially designed to support robust automotive applications, the protocol was introduced to the market through a collaboration between Bosch and Intel.
The first suppliers of CAN modules at that time were Intel and Motorola. Currently almost all minor and major semiconductor manufacturers have CAN products in their portfolio.
CAN was standardized as ISO 11898 in November 1993. By 1997, 24 million CAN interfaces were produced in a single year, and just two years later, this number had already tripled. Currently, it is estimated that over a billion CAN interfaces are produced annually.

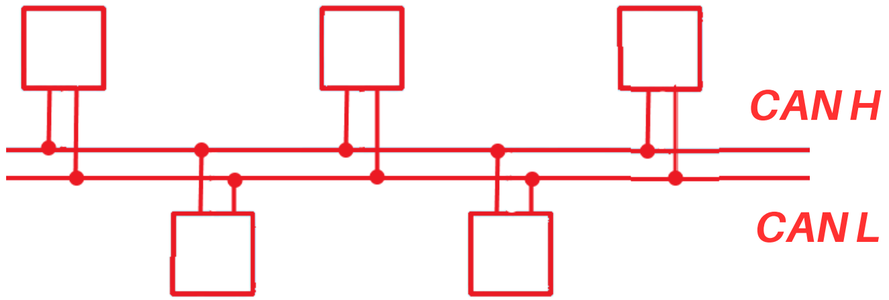
WHY INVENTING CAN ?

The evolution of the CAN bus traces back to a time when automobiles and automotive industries relied on point-to-point wiring systems, where each electronic component had its dedicated data line. As the automotive sector expanded and demanded more sophisticated features, the inclusion of additional electronic components became inevitable. However, this led to challenges such as extensive rewiring due to the rising number of components, resulting in scalability issues and heightened manufacturing costs for vehicles.

*Wiring before **CAN***



*Wiring after **CAN***



mohamedbahae1402@gmail.com

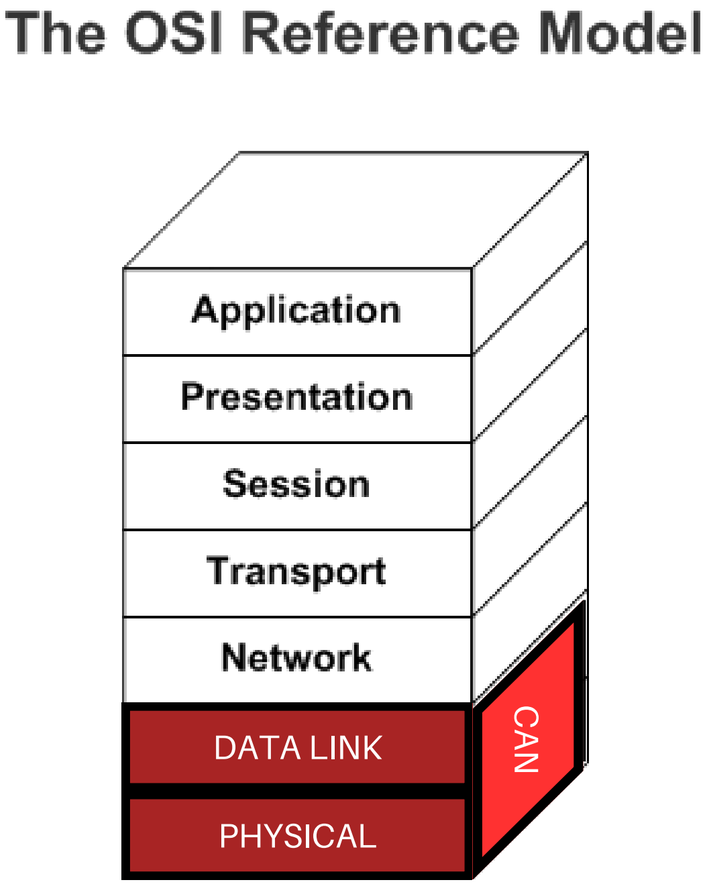
CAN FEATURES

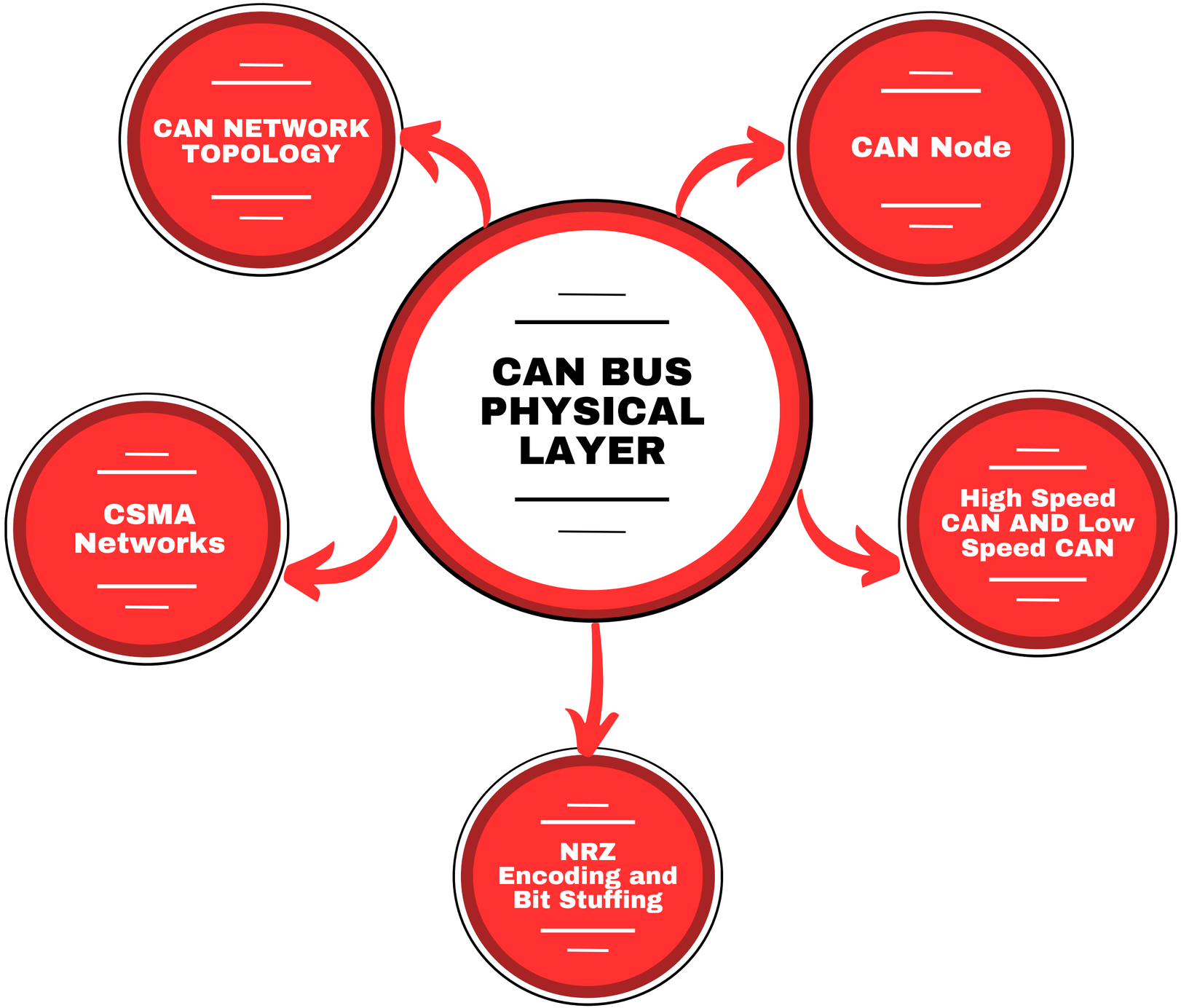
- **Half duplex Asynchronous Serial protocol**
- **Multi-Master No slave** (any node on the bus can initiate communication)
- **Message prioritization** (CAN bus uses a priority-based arbitration mechanism to determine which message gets transmitted when multiple nodes attempt to send messages simultaneously.)
- **Error Detection and high noise immunity**
- **Low Cost and Simple Implementation**
- **Scalability**
- **retransmission of faulty messages**

- **Deterministic** (the time it takes for a message to be transmitted and received can be precisely predicted. This is crucial for real-time applications where timing is critical)
- **Differential pair of wires** (twisted to prevent electromagnetic interference EMI)

CAN BUS AND OSI MODEL

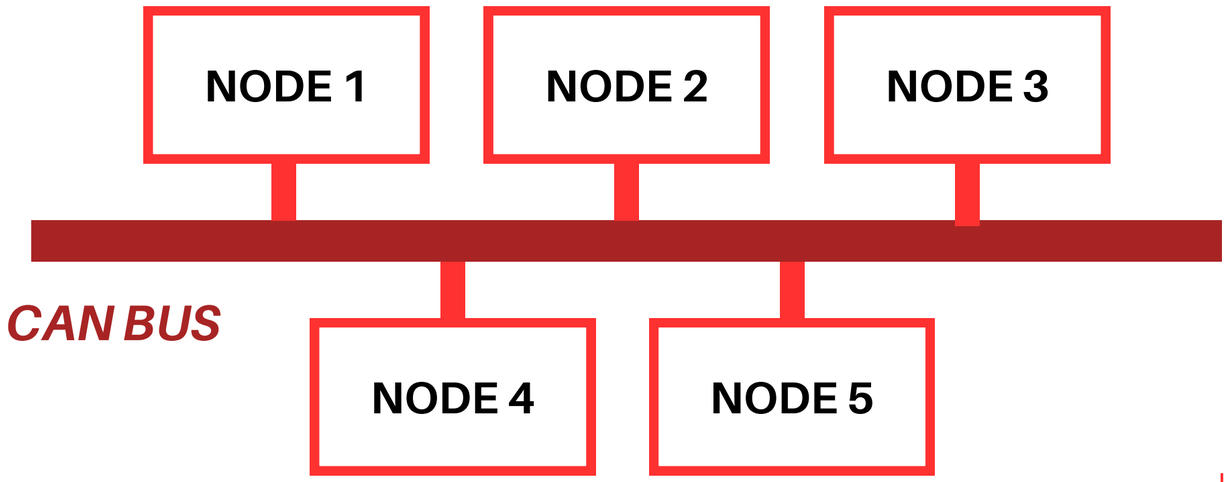
The OSI or Open Systems Interconnection Model is a conceptual framework that standardizes the functions of a Networking system into seven distinct layers. Each layer serves a specific purpose and interacts with adjacent layers to facilitate communication between different systems. It was developed by the International Organization for Standardization (ISO)




CAN BUS PHYSICAL LAYER
CAN NETWORK TOPOLOGY
CAN Node
CSMA Networks
High Speed CAN AND Low Speed CAN
NRZ Encoding and Bit Stuffing

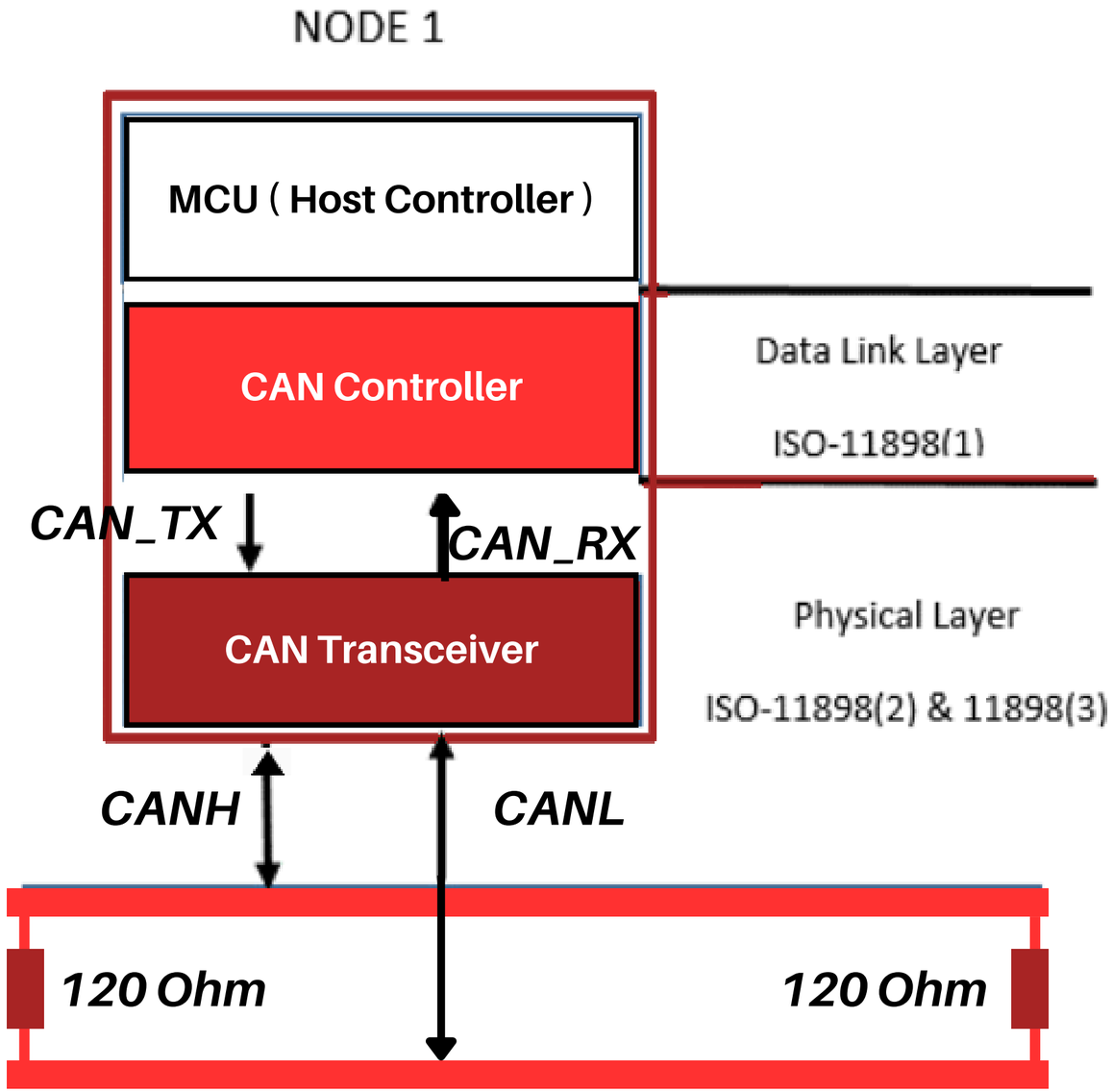
CAN NETWORK TOPOLOGY

CAN Network is a broadcast type unlike traditional networks such as I2C or USB or Ethernet, messages are sent globally in the network and all the nodes can hear the broadcasted message without any addressing



CAN NODE

- **Micro-Controller unit :** This unit decides how to deal with received data or data to be transmitted.
- **CAN Controller :** This unit applies acceptance filtering as well as it receives whole data until the complete message is not received. This unit sent data on CAN Bus serially when the bus is idle.
- **CAN transceiver :** This unit converts transmit signal from CAN controller level to CAN-BUS level and Vice-Versa .



CAN BUS Node Architecture

Why Differential signal ?

Differential signal resists electromagnetic noise compared to one conductor with an un-balanced reference.
To achieve good signal quality, it is necessary to use a twisted pair wire as the transmission medium and to protect the signal from external noise.

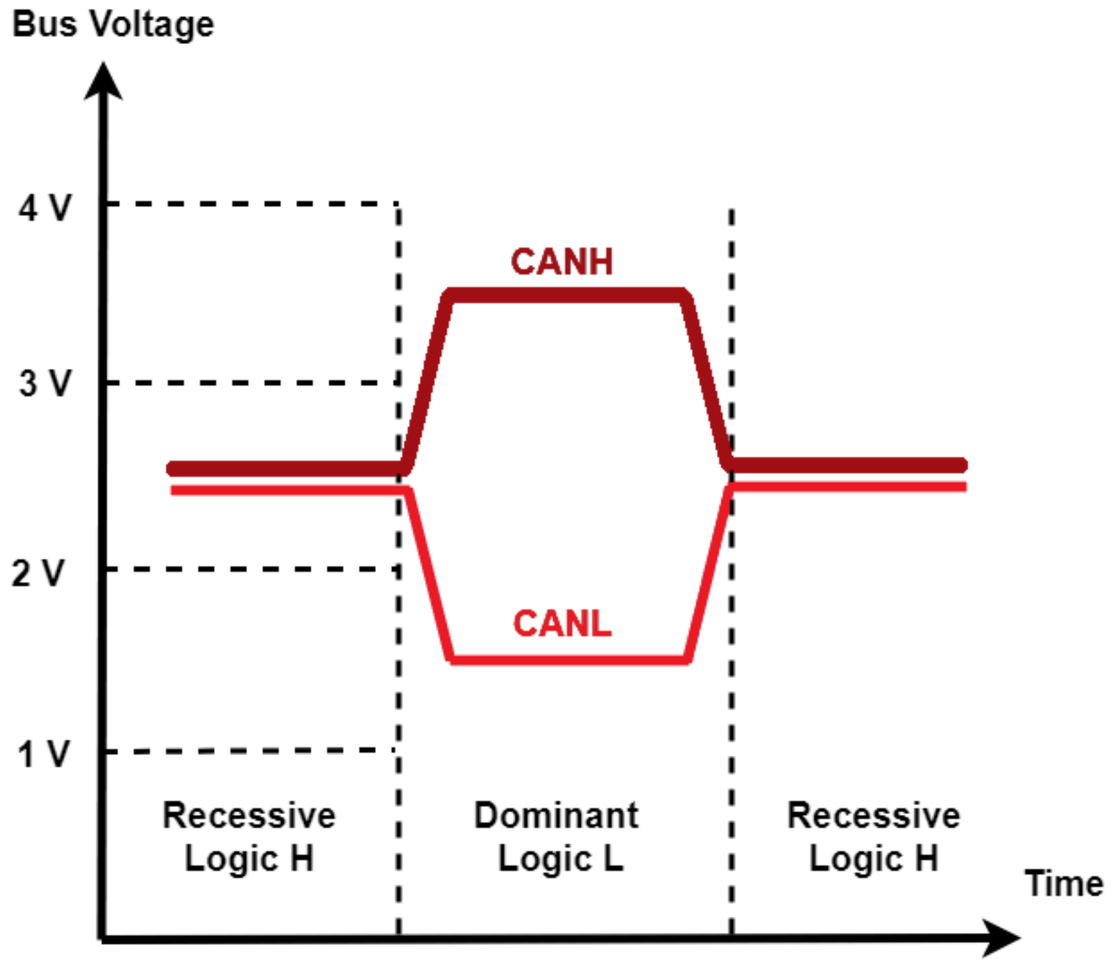
Why Termination Resistance ?

Reflexions can be particularly strong when the signal reaches the end of the cable, which is why termination adapters are placed at the ends.
The CAN Driver in every CAN node is normally completely passive when sending binary 1 the recessive level. The resistor ensures that the voltage difference returns to recessive state when the dominant signal disappears

Differential signal

- **Recessive state :** (Logical H)
  CANH = 2.5 V
  CANL = 2.5 V
  Vdiff = 0 V
- **Dominant state** : (Logical L)
  CANH = 3.5 V
  CANL = 1.5 V
  Vdiff = 2 V

Vdiff = CANH - CANL



HIGH-SPEED CAN AND LOW-SPEED CAN

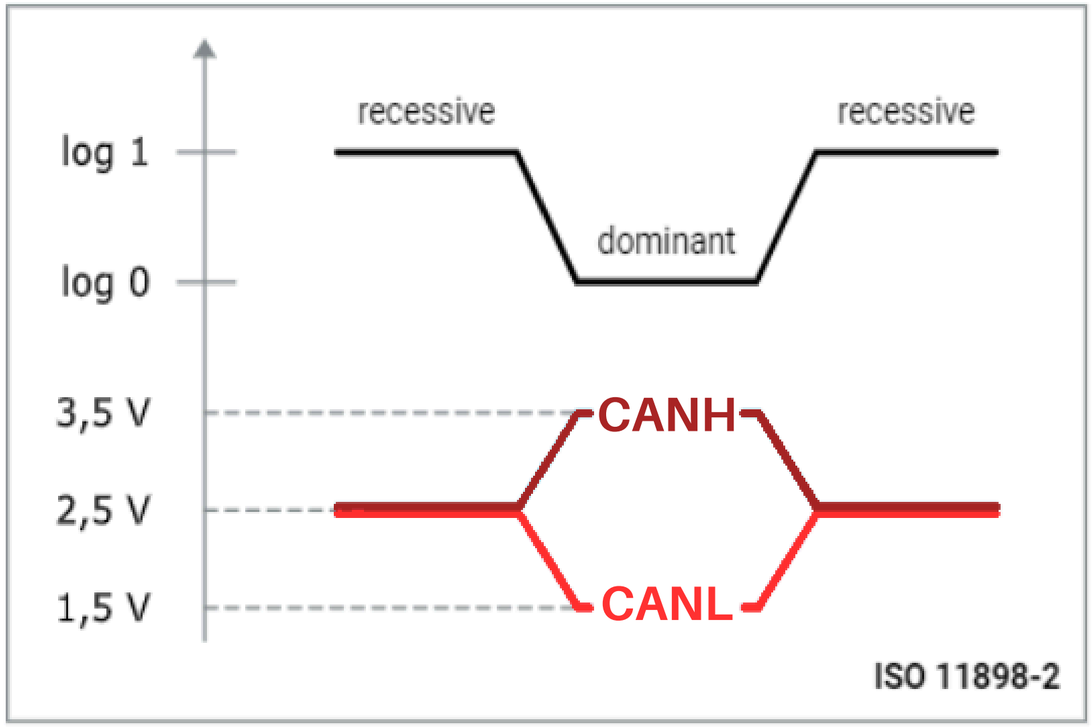
High-Speed CAN

The High-Speed CAN is the "classic" CAN bus (according to ISO 11898-2), used in many (industrial) applications. The bit rates ranging from 10 kbit/s up to 1 Mbit/s.
CAN signals are transmitted differentially. Using High-Speed CAN, the differential voltage for dominant bits is 2 V, for recessive bits 0 V.

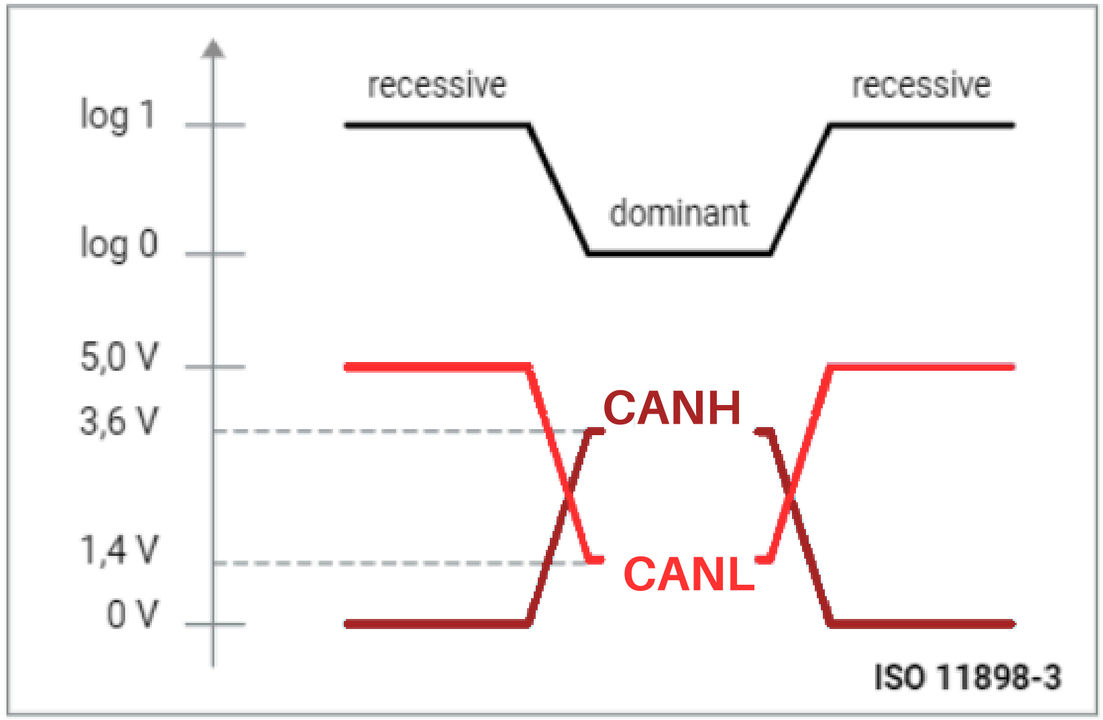
Low-Speed CAN

Low-Speed CAN, or Fault-Tolerant CAN (according to ISO 11898-3), is designed for improved durability. The recessive level of the differential signal is at -5 V, the dominant level is 2.2 V. If one of the two data lines is damaged, there is an automatic switchover to single-wire mode. This allows the system to be operated. The bit rates ranging from 40-125 kbit/s.

High-Speed CAN Bus levels



Low-Speed CAN Bus levels



CSMA (CARRIER SENSE MULTIPLE ACCESS) Networks

This method was developed to decrease the chances of collisions when two or more nodes start sending their signals. Carrier Sense multiple access requires that each node sense the bus before transmit any data. The bus has two states :

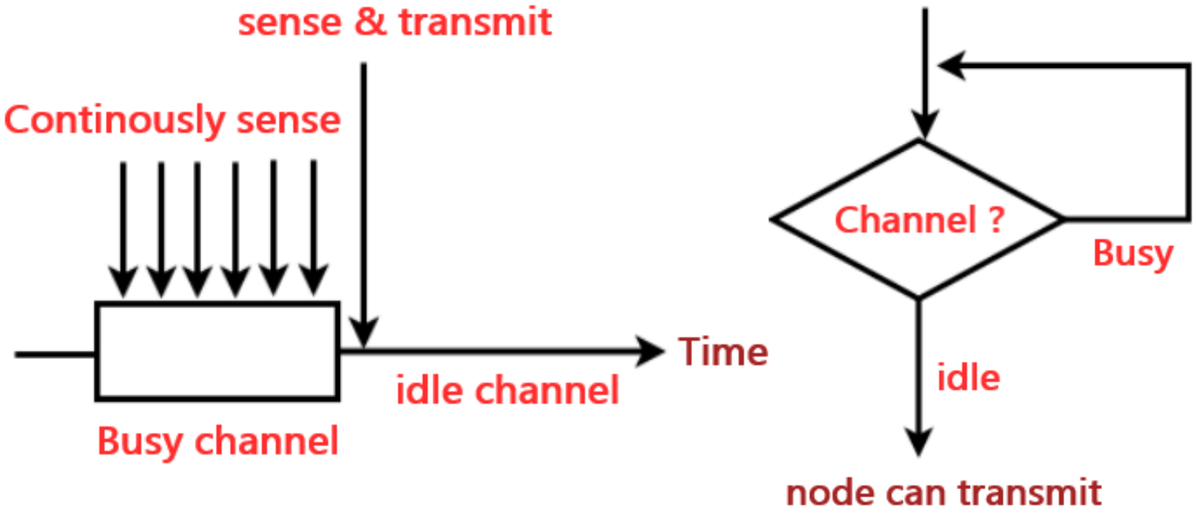
Carrier Busy = Transmission is tacking place in the bus
Carrier idle = No transmission currently taking place in bus

The possibility of collision still exists because of propagation delay.

Types of CSMA Access Modes:

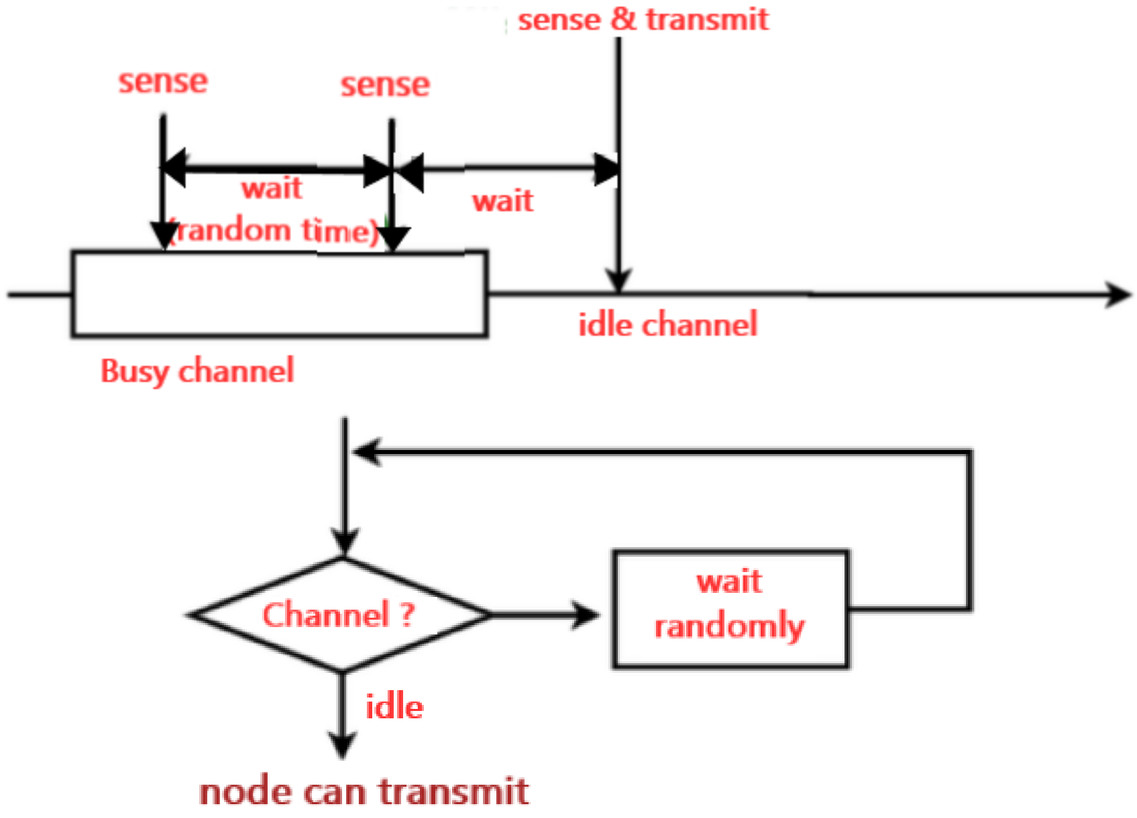
1- Persistent :

If bus is busy the node sense the bus continuously until it becomes idle



2- Non-Persistent :

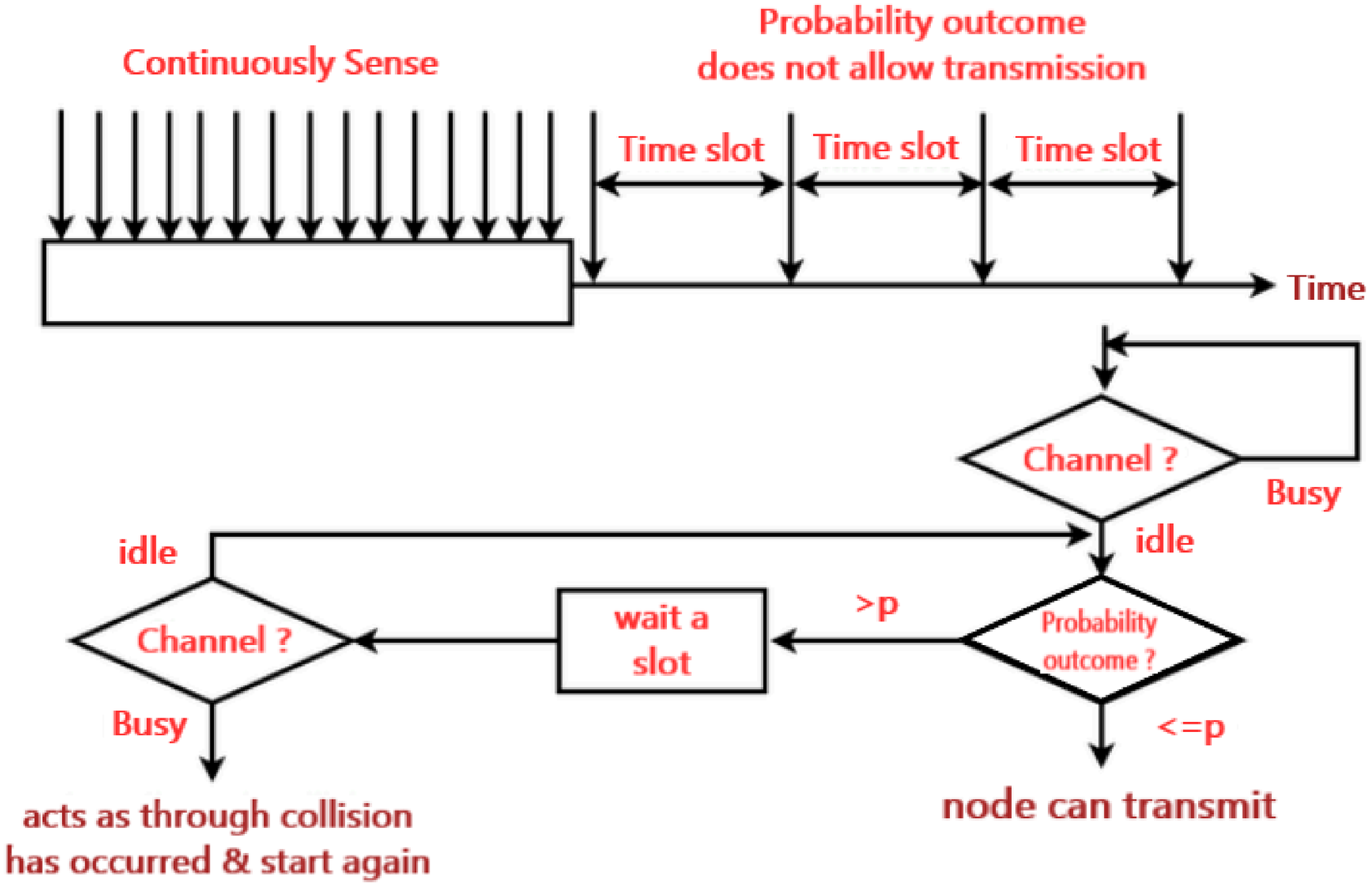
If bus is busy the node waits a random period of time and then repeat the transmission



Types of CSMA Access Modes:

3- P-Persistent :

Each node has its own time slut in the bus trimming



CSMA With Collision Avoidance

CSMA / CA :

The basic idea behind CSMA/CA is that the node should be able to receive while transmitting to detect a collision from different nodes. In wired networks, if a collision has occurred then the energy of the received signal almost doubles, and the node can sense the possibility of collision. In the case of wireless networks, most of the energy is used for transmission, and the energy of the received signal increases by only 5-10% if a collision occurs. It can’t be used by the node to sense collision. Therefore CSMA/CA has been specially designed for wireless networks.

CSMA With Collision detection

CSMA / CD :

if two nodes sense the bus to be idle and begin transmitting simultaneously, both of them will detect the collision immediately and they should stop the transmission and wait random period of time and then tries to send data again, this method used by Ethernet LAN networks.

CSMA / CD + AMP :

CSMA with collision detection and arbitration on message priority.
when two or more nodes begin sending messages at same time collision is detected and the message with the highest priority will overwrite other messages using bit wise arbitration, It means that only this highest priority message remains on the bus

CAN NETWORK IS CSMA / CD + AMP

NRZ - encoding

During signal transfer there are two types of line code used :

RZ(Return to zero) Coding :

As name explains itself “return to Zero” means in this line coding signal(1 or 0) drops to zero in between the each pulses.You can say each signal take a rest state between next signal transmission whether consecutive signals are same or different.

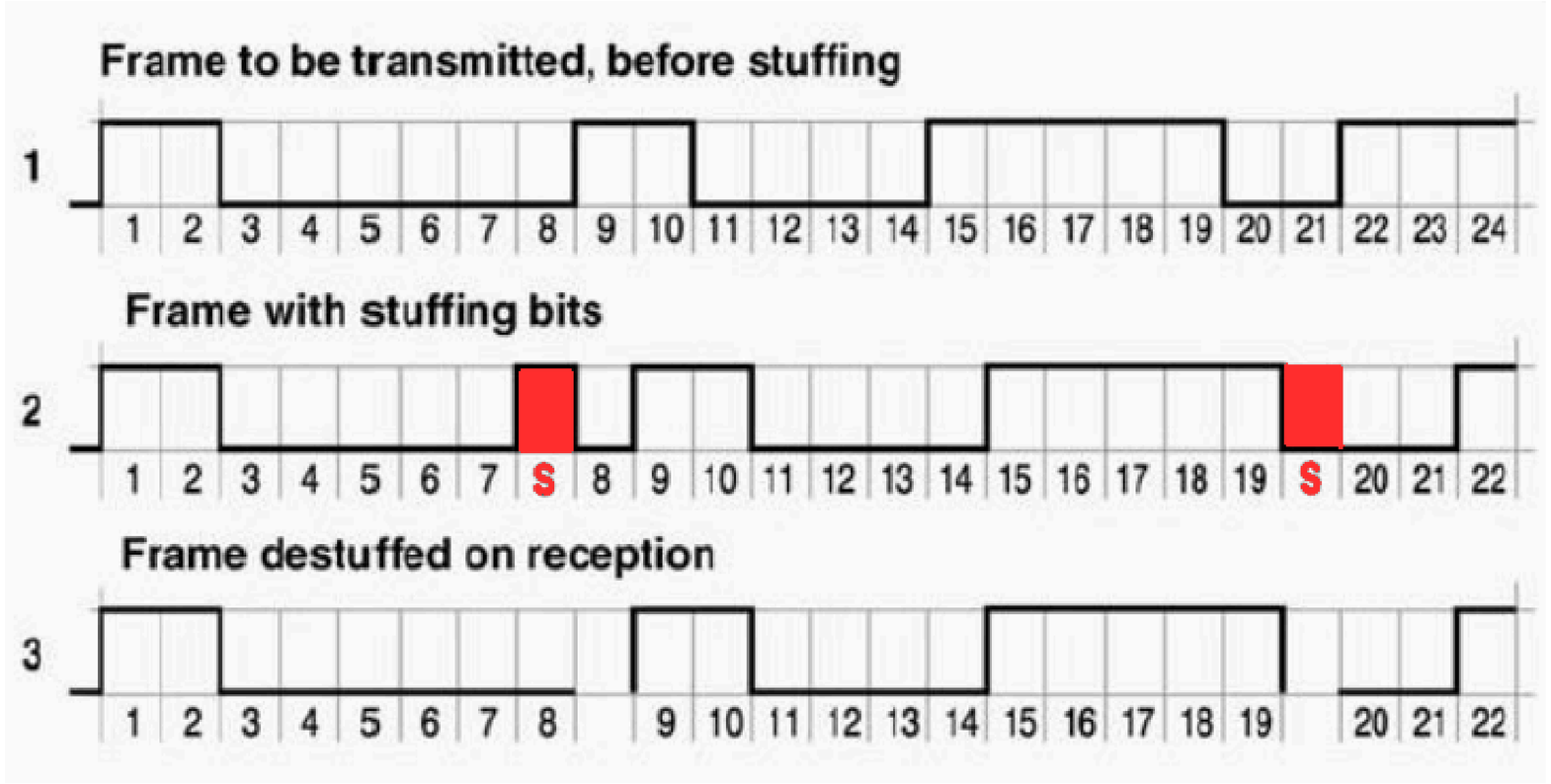
NRZ(Non-Return to Zero) code :

NRZ is “Non Return To Zero” means in NRZ line coding there is no rest state in between the pulse(it will not return to zero) as well as no change in polarity until consecutive signals are same.

CAN BUS USES NRZ ENCODING TECHNIQUE WITH BIT-STUFFING.

BIT STUFFING

Bit stuffing is a form of data communication used in various protocols, including the CAN bus, to prevent false synchronization. It involves adding non-information bits into the data stream to break up patterns that could be mistaken for frame boundaries.



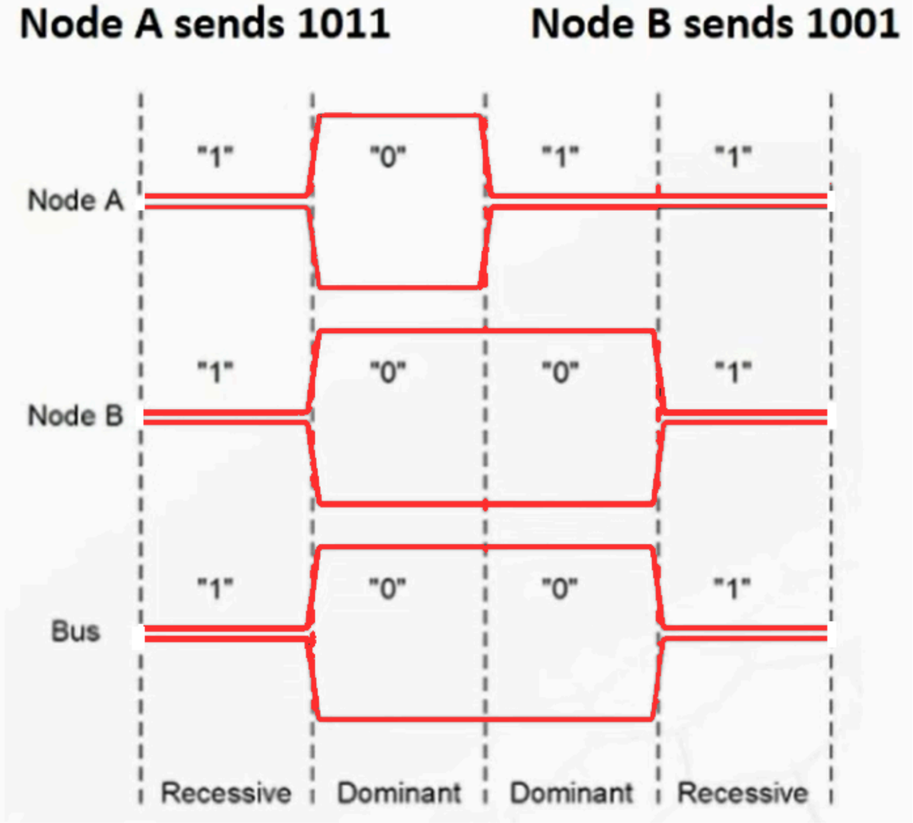
In the CAN bus protocol, bit stuffing occurs when message have more than 5 consecutive bits of the same value no falling edges or rising edges for resynchronization. The transmitter will insert on an additional bit of the opposite value into the data stream.

The primary purpose of bit stuffing is to maintain synchronization between different nodes on the CAN bus. The CAN bus protocol relies on edge transitions (the change from 0 to 1 or vice versa) to synchronize the internal clocks of different nodes. If there are too many consecutive bits of the same value, the lack of edge transitions could lead to a loss of synchronization.

BITWISE ARBITRATION

- The node or ECU, with the lowest address gets the bus, and wines arbitration.
- A dominant bit always overwrites a recessive bit on a CAN bus
- Because each node continuously monitors its own transmissions when certain node detects that the bus state does not match the bit that it transmitted. Its halts transmission while the node that win the bus continues on with its message

An example illustrates arbitration between two competing CAN nodes :



CAN BUS DATA LINK LAYER

CAN Message Types

Standard CAN vs Extended CAN

Standard CAN vs Extended CAN

There is two frame formats CAN 2.0A (CAN standard) frame and CAN 2.0B (CAN extended) frame

CAN standard defined the length of the identifier in the Arbitration field to **11 bits .** Later on, customer demand forced an extension of the standard. The new format is often called the *extended CAN* and allows no less than **29 bits** in the identifier**.** To differentiate between the two frame types, reserved bit in the control field was used (the IDE (Identifier Extension) bit).

IDE	RTR	
0	0	STANDARD DATA FRAME
0	1	STANDARD REMOTE FRAME
1	0	EXTENDED DATA FRAME
1	1	EXTENDED REMOTE FRAME

CAN Message Types

THE DATA FRAME

the DATA frame is the most common message type, It is the only frame for actual DATA transmission

THE REMOTE FRAME

The Remote Frame is just like the Data Frame, with two important differences:

• the RTR bit in the Arbitration Field is recessive

• there is no Data Field

THE ERROR FRAME

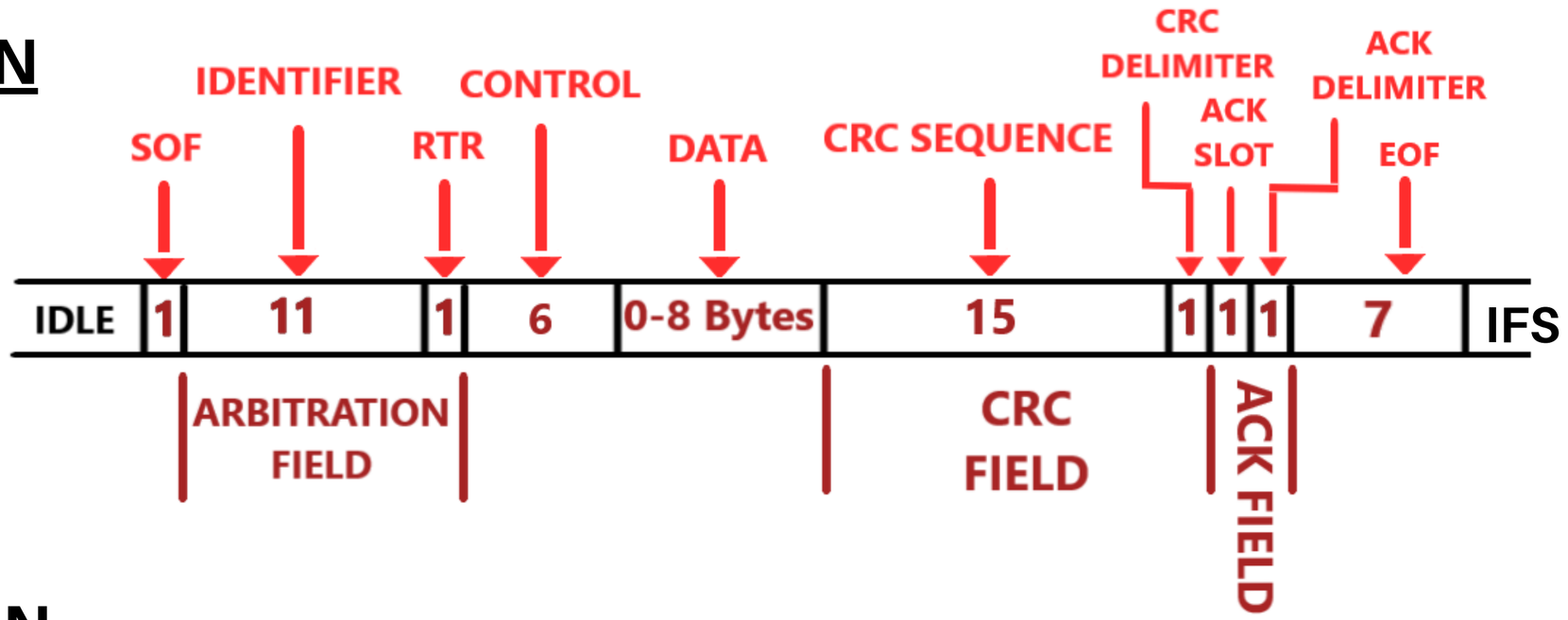
the ERROR frame is a special message It is transmitted when a node detects an error in a message

THE OVERLOAD FRAME

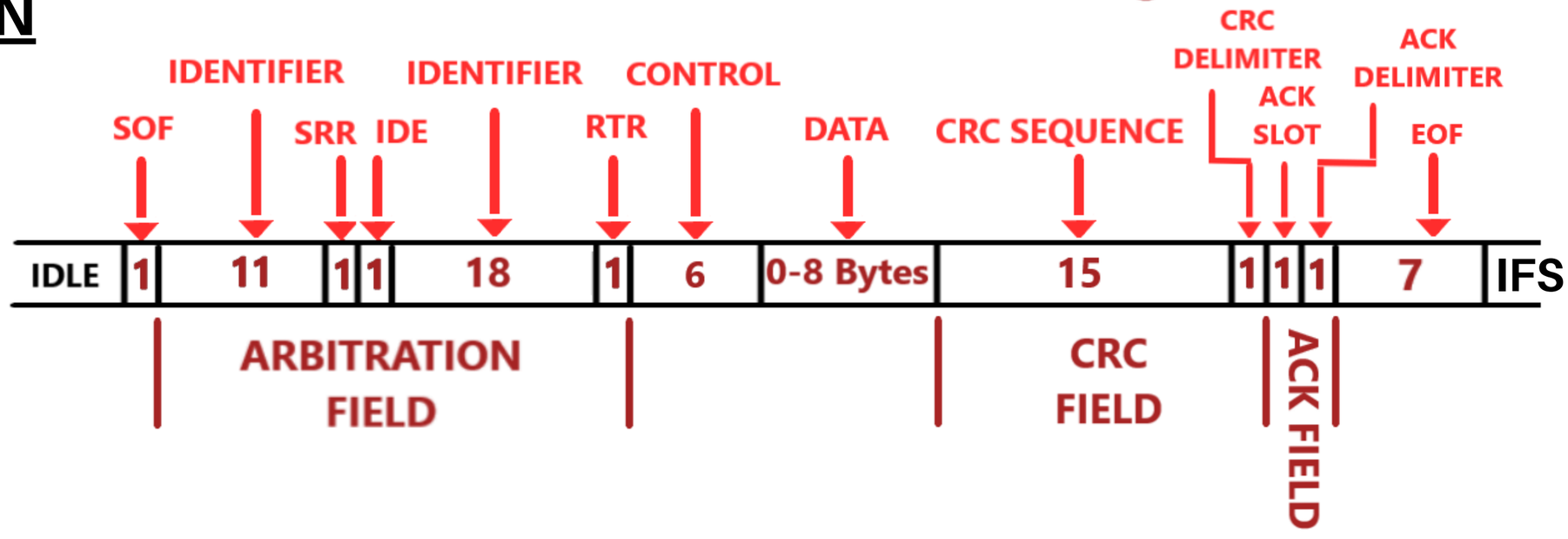
the OVERLOAD is used to provide for an extra delay between the preceding and the succeeding data or remote frame

DATA FRAME

Standard CAN

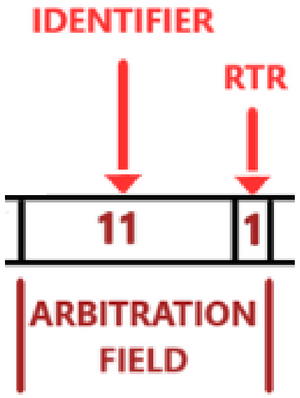


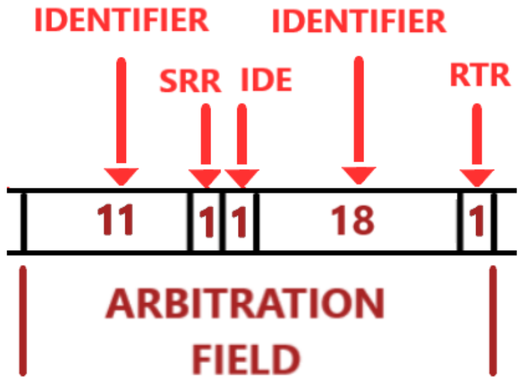
Extended CAN



SOF (Start of Frame) : Marks the beginning of data and remote Frames
Used for synchronization ,It's a single dominant bit, can start only if bus was idle

Arbitration Field :





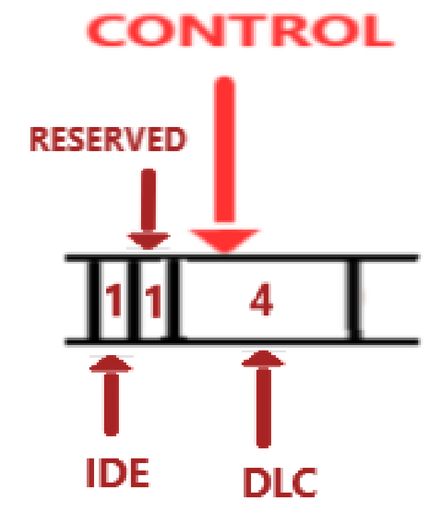
- Includes the message Identifier (consists of 11bits in CAN 2.0A and 29bits in CAN 2.0B)
- RTR (Remote Transmission Request) bit, which distinguishes data and remote frames (dominant in data frame and recessive in remote frame).

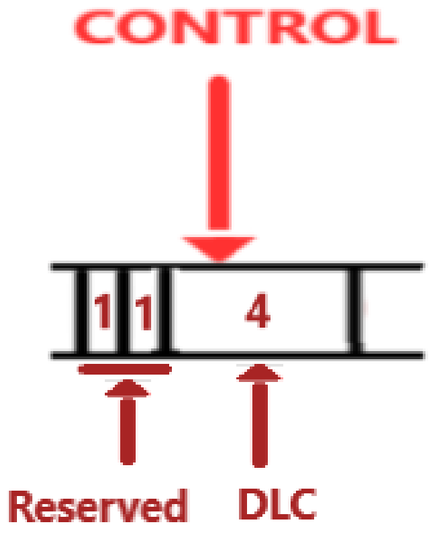
For Extended CAN

- IDE (Extended Identifier) indicates that the identifier field in the frame is using the extended format.
- SRR (Substitute Remote Request):When this bit is set to 1, it indicates that this frame is a substitute remote request, meaning the node wants to request data from another node. If set to 0, it's a data frame.

The IDE (Identifier Extension) bit belongs to:
- The Control Field of the Standard Format
- The Arbitration Field of the Extended Format

Control Field :





- IDE (Extended Identifier) : A low (dominant) IDE bit indicates an 11-bit message identifier, a high (recessive) IDE bit indicates a 29-bit identifier.
- r0, a Dominant bit reserved for future usage.
- And the DLC (Data length code) field, which consists of 4 bits indicates the number of bytes of data contained in the CAN frame. It ranges from 0 to 8 bytes, allowing for variable-length data payloads.

For Extended CAN

The Control Field was redesigned in CAN 2.0B, as shown in the picture at the bottom, in order to support the coexistence of 11-bit and 29-bit message identifiers on the same CAN bus network.

- Includes two bits, r1 and r0, reserved for future use and were kept at a low (dominant) level.
- And the DLC (Data length code) like standard CAN.

Data Field : The data field can contain upto 8 bytes.

CRC Field : Or (Cyclic Redundancy Check) field, it contains a 15-bit checksum used for error detection, followed by CRC delimiter (always recessive). It features a Hamming distance of six.

ACK Field : ACK field is 2 bits, one is the acknowledgment bit and the second is a delimiter (always recessive).
Sender writes a recessive bit in the ACK slot, and every node receiving an accurate message overwrites this recessive bit in the original message with a dominant bit, so the sender knows if at least one receiver has correctly received the message.

EOF(End of Frame) : it's a 7 recessive bits marks the end of a CAN frame (message) and disables bitstuffing, indicating a stuffing error when dominant.

IFS(Interframe space) : It's a bit field used to separate Data(or Remote) frames from preceding frames whatever type they are. It contains the bit fields INTERMISSION (3 recessive bits) and BUS IDLE.

REMOTE FRAME

The remote frame serves to request data transmission from another node, resembling the data frame but with two notable distinctions. Firstly, it's distinctly identified as a remote frame by a recessive RTR bit in the arbitration field. Secondly, unlike the data frame, it doesn't contain any data payload.

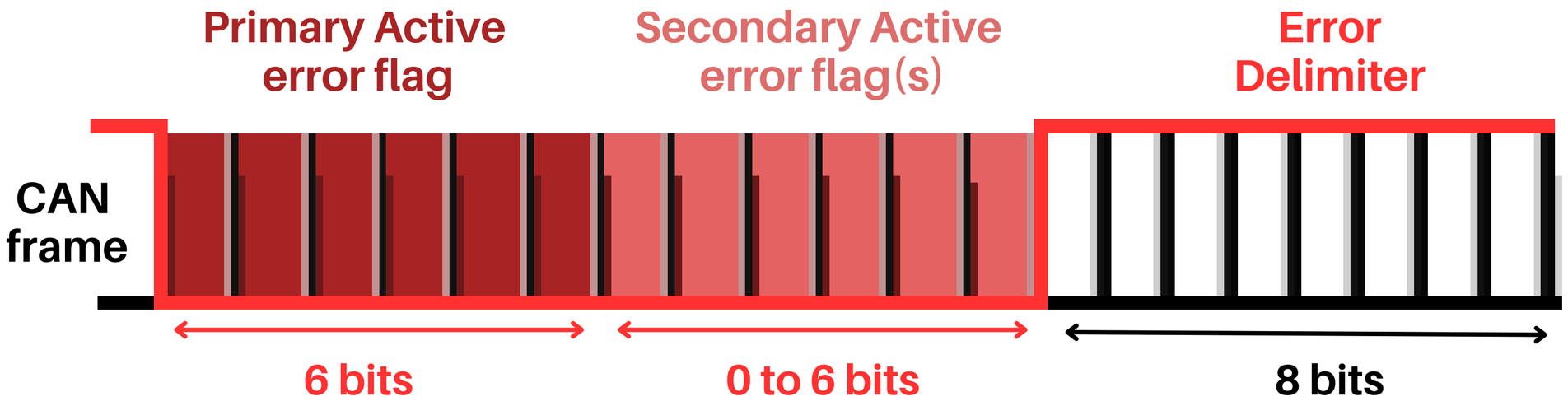
ERROR FRAME

The error frame is a special message characterized (like the End Of Frame) by a deliberate breach of the bit-stuffing rules. It's dispatched when a node detects an error within a message, prompting all other nodes in the network to send out an error frame too. Subsequently, the initial transmitter automatically initiates a retransmission of the message.

Transmission of error frame starts simultaneously with the next bit after the detection of an error (CRC error frames start after the ACK delimiter).

• **Active Error Flag**

It's a sequence of 6 dominant bits that violates bit stuffing. When it's raised by a node, the other CAN nodes will see the Active Error Flag as a Bit Stuffing Error. In response they also raise an Active Error Flag.



• **Passive Error Flag**

It's a sequence of 6 recessive bits that violates bit stuffing.

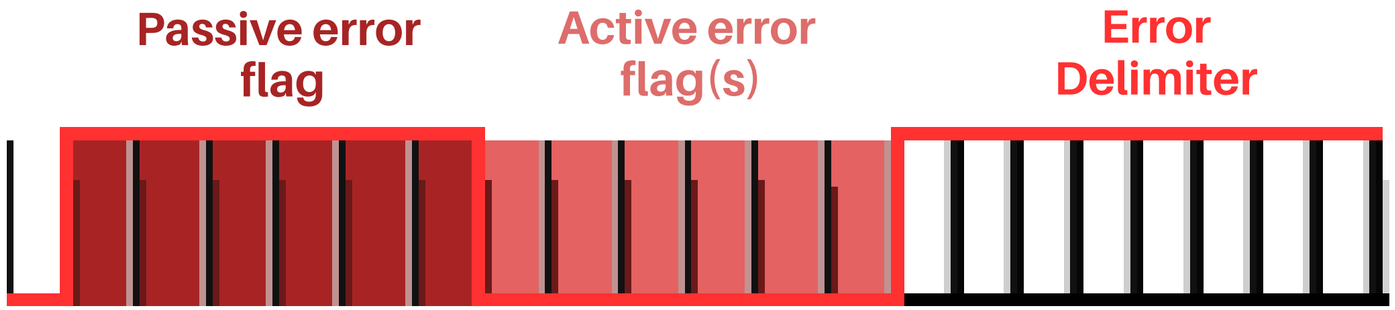
In this case we have to distinguish between a Passive Error Flag raised by a transmitting node and a receiving node :

1. *Transmitter is Error Passive :*

the other CAN nodes will see the Passive Error Flag as a Bit Stuffing Error. In response they raise an Active Error Flag.

2. *Receiver is Error Passive* :

When transmetter raises an Active Error Flag, the receiver will raise a Passive Error Flag which will be “invisible” to all CAN nodes on the bus, because any dominant bits win over the sequence of recessive bits



ERROR Types :

The error management is able to identify five different types of errors :

- **Bit-Error**

A transmitted bit is not received with the same logical value with which it was sent. Excluded are the Arbitration Field and the ACK Slot.

- **Stuff-Error**

More than five consecutive bits of the same level have been detected. Excluded are End Of Frame and Interframe Space (IFS).

- **CRC-Error**

The calculated CRC checksum does not match with the received CRC checksum

- **Form-Error**

There has been a violation of the frame format, e.g. CRC Delimiter or ACK Delimiter was not recognized as a recessive bit, or End Of Frame was disturbed

- **Acknowledgement-Error**

A transmitter did not detect a dominant bit in the ACK Slot, which means that the message was not recognized as fault-free by any other CAN node.

CAN node States：

- **Error Active**

This is the normal state of a CAN node, in which messages can be sent and received. In the event of a fault, an Active Error Flag, consisting of dominant bits, is transmitted.

- **Error Passive**

This state is reached after several errors have been detected on the CAN bus. In this state, the CAN node may continue to send and receive messages, but in the case of an error, only a Passive Error Flag is sent, consisting of recessive bits.

- **Bus Off**

In this state, the CAN node is completely disconnected from the CAN bus. It therefore cannot send or receive messages or Error Flags. This state is reached when there is a long disturbance on the CAN bus or a frequently repeated disturbance.

ERROR Counters :

The error state transitions are controlled by error counters.

Each CAN node has an error counter

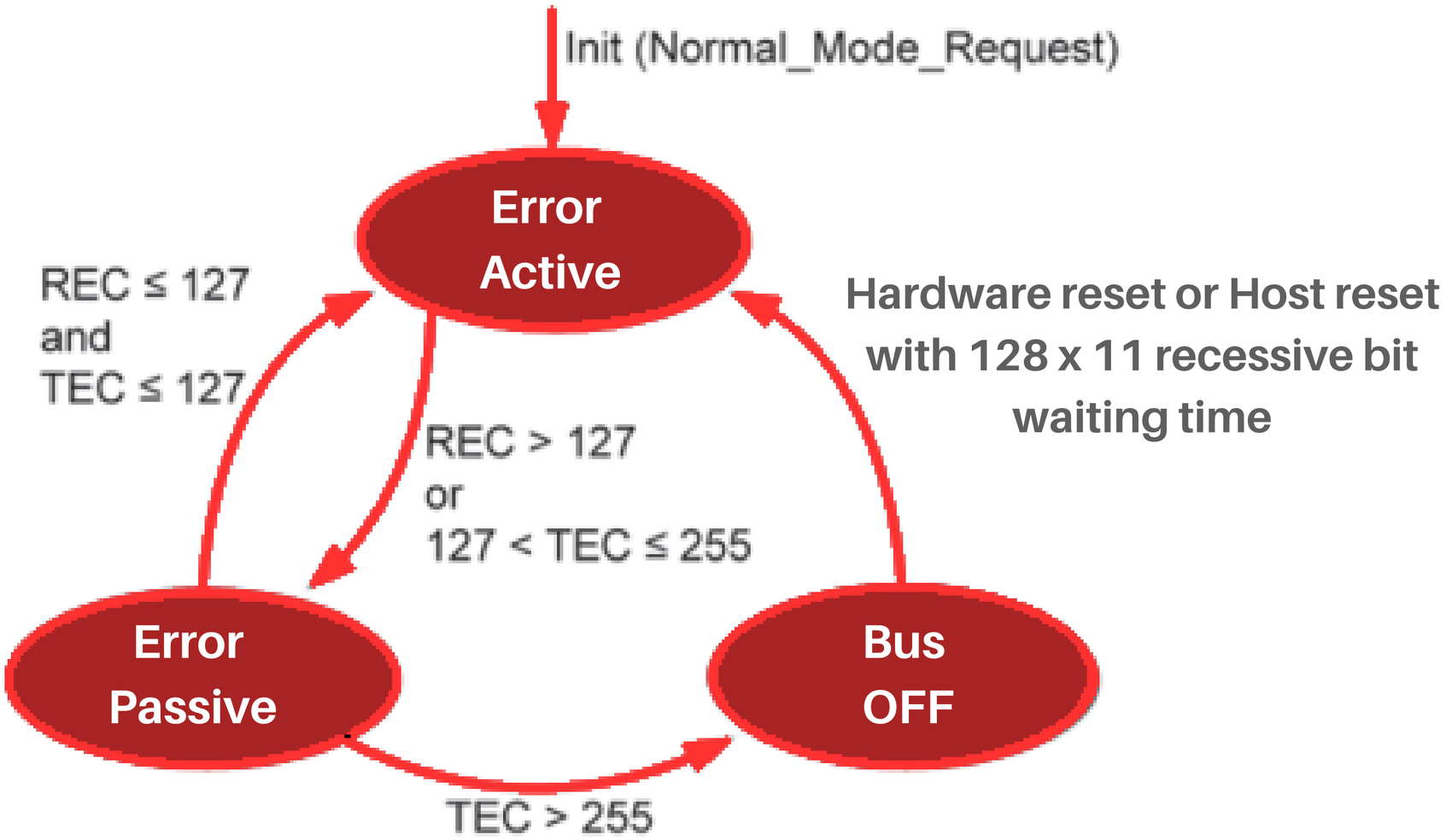
- for receive errors : REC (Receive Error Counter).
- for transmit errors : TEC (Transmit Error Counter) .

ERROR Handling :

The error handling is done in the following order :

1. An error is detected.
2. An Error Frame is sent by any CAN node that has detected the error.
3. The message currently in progress is rejected by all CAN nodes.
4. Error counters incremented.
5. The disturbed message is retransmitted

State Diagram for CAN node :



The states are characterized by the following count values：

- **Error Active:**

  The counts of the Receive Error Counter and Transmit Error Counter are both less than or equal to 127.

- **Error Passive:**

  At least one of the two error counts is greater than 127 and the Transmit Error Counter is less than 256.

- **Bus Off:**

  The Transmit Error Counter is greater than 255.

The transition from Error Active to Error Passive and vice versa is done automatically. The Bus Off state can be exited only by appropriate actions of the host controller (software or hardware reset).

Update of Counters :

Receive Error Counter :

- A receiver detects an error → **REC=REC+1.**
- A receiver detects a dominant bit after sending an Error Flag → **REC=REC+8.**
- A receiver detects a Bit-Error while sending an Active Error Flag or an Overload Flag → **REC=REC+8.**
- A receiver with REC>0 will decrement after each reception of a fault-free Data Frame or Remote Frame → **REC=REC - 1.**
- A receiver with REC=0 → **REC=0.**
- Any CAN node tolerates up to seven consecutive dominant bits after sending an Active Error Flag, Passive Error Flag, or Overload Flag. After detecting the 14th consecutive dominant bit (in case of an Active Error Flag or an Overload Flag) or after detecting the eighth consecutive dominant bit following a Passive Error Flag, and after each sequence of additional eight consecutive dominant bits, every receiver increases → **REC=REC+8.**

Transmit Error Counter :

- A transmitter starts an Error Flag → **TEC=TEC+8.**

  Exception 1: When a transmitter is in Error Passive state and detects an Acknowledgment-Error (no dominant bit in the ACK Slot) and it does not detect a dominant bit while sending its Passive Error Flag, the TEC is not changed.

  Exception 2: If a transmitter detects a Bit-Error on a recessive stuff-bit in the Arbitration Field before the RTR bit, the TEC is not changed.

- A transmitter detects a Bit-Error while sending an Active Error Flag or an Overload Flag → **TEC=TEC+8.**
- A transmitter with TEC>0 will decrement after each successful transmission of a Data Frame or a Remote Frame → **TEC=TEC−1.**
- Any CAN node tolerates up to seven consecutive dominant bits after sending an Active Error Flag, Passive Error Flag, or Overload Flag. After detecting the 14th consecutive dominant bit (in case of an Active Error Flag or an Overload Flag) or after detecting the eighth consecutive dominant bit following a Passive Error Flag, and after each sequence of additional eight consecutive dominant bits, every transmitter increases → **TEC=TEC+8.**

WARNING Level :

A node reaches the warning level when :

- **TEC > 96** **AND/OR** • **REC > 96**

When a node reaches the warning level :

- An **“Error Warning flag”** is **Set** and can be **Cleared** for **TEC <= 96** AND **REC <= 96** .
- An **“Interrupt”** might result (**Optional**).

We can use the Warning level to determine if a CAN node gets near the threshold to the **Error Passive state** by checking constantly the **Error Warning flag.**

OVERLOAD FRAME

- Overload Frames are dispatched under two conditions. Initially, when a dominant bit appears in the first two bits of the Interframe Space (IFS), an Overload Frame is sent to synchronize all CAN nodes. Additionally, a CAN controller might utilize an Overload Frame to signal to other CAN nodes that it is currently overloaded and unable to handle a new message immediately due to high workload.
- The overload frame has no effect on the error counters (REC and TEC).
- Today's CAN controllers are fast enough, they are able to handle all bus traffic without the additional delay of Overload Frames.