Vehicular communications

Ollari Ischimji Dmitri

24 ottobre 2023

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduction to Vehicular Communications

1.1 Principles and challenges

L'obiettivo principale è quello dell'**Intelligent Transportation System** (ITS), ovvero un sistema che permetta di migliorare la sicurezza, la mobilità e l'efficienza del trasporto che sono composti di hardware, software e communication technologies.

Alcuni esempi di ITS sono:

- **Infotainment Sustainability**
- **Tolling**
- **Emergency Call**
- **Crash Imminent Safety**

Solitamente le ITS Applications sono isolate e non permettono il passaggio di dati da una app all'altra, mentre le C-ITS(Cooporative ITS) sono applicazioni che permettono la cooperazione tra veicoli e infrastrutture.

Tutte le informazioni raccolte dai sensori del veicolo vengono raccolti e inviati alle **Engine Control Units(ECUs)** che sono sistemi integrati che controllano ona o più funzioni del veicolo.

Si utilizzano i sistemi BUS per effettuare il passaggio di informazioni fra le ECUs(che ora si chiamano Electronic Control Units) e i sensori del veicolo.

Applicazioni visionarie basate sulla comunicazione fra veicoli:

- Lane assistant
- Lateral collision
- Accident reporting
- Intersection assistace
- cooperative driving

Le principali difficoltà della comunicazione sono:

- **Basic**

  - latenza
  - throughput
- **Communication in veichles**
  - robustezza
  - costo
- **Communication between veichles**
  - privacy
  - sicurezza
  - reachability
  - interoperabilità

1.2 Standardization and open issues

Il gruppo di lavoro **ISO/TC 204** per le standardizzazioni è responsabile per la maggiorparte degli aspetti architetturali per gli ITS, ha prodotto il concetto **Communications Access for Land Mobiles(CALM)** e **ETSI TC ITS** basato sui recenti C-ITS europei.

1.2.1 CALM

Soluzione a strati che permette comunicazioni:

- **V2V**: Veichle to Veichle
- **V2I**: Veichle to Infrastructure
- **I2I**: Infrastructure to Infrastructure

Sfrutta la comunicazione multichannel, capace di connettersi a tanti gestori e utilizza IPv6. Tecnologia che è in grado di selezionare il mezzo di comunicazioni migliroe per l'occasione.

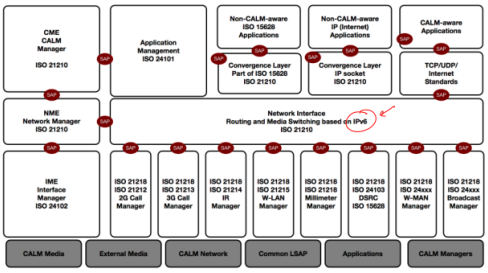

Figura 1.1: CALM architecture

1.2.2 Visione Europea

L'Europa è leader mondiale in ITS(probabilmente per industria dell'automotive) ed è conscia delle quantità di denaro, energia e impatto di CO2 causate dalle congestioni stradali.

L'obiettivo europeo è quello di raggiungere **zero incidenti, zero ritardi, meno impatto sull'ambiente, persone informat, servizi accessibili, rispetto della privacy e sicurezza.**

Lo sviluppo è portato avanti dalle aziende del territorio.

1.2.3 Visione Americana

In america lo sviluppo è mandato avanti da fondi statali, vengono elargiti programmi di 5 anni per lo sviluppo di ITS.

L'approccio americano porta all'utilizzo sia di tecnologie **non-DSRC(Dedicated Short Range Communications)** che di tecnologie **DSRC** per l'utilizzo rispettiamente di segnali a lunga distanza come la rete cellulare oppure di segnali a corto raggio come le onde radio.

1.3 ITS Architecture

Data la granded quantità di persone nel mondo pronte a sviluppare ITS, si è introdotto il concetto di **ITS-S(ITS station)** che rappresenta un **set di istruzioni** per lo sviluppo di un ITS.

Da un punto di vista architetturale il ITS-S è simile al modello OSI, nasce cosi l'architettura **ITSC** che ruota attorno alle ITS station, una unità computazionale modulare.

Questo porta alla realizzazione di diversi ITS:

- Vehicle ITS
- Roadside ITS
- Central ITS
- Personal ITS

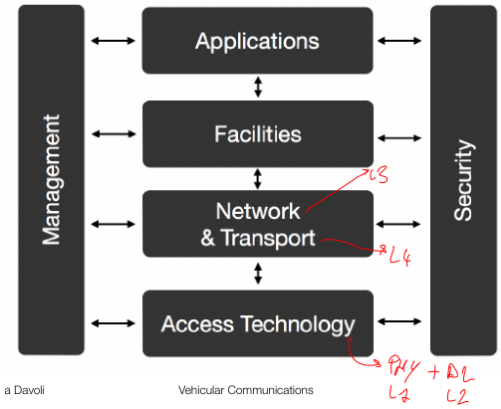

Figura 1.2: ITS architecture

L'host ITS-S è composto da:

- 4 layer logici orizzonatali
  - **Access technologies** layer: unisce layer fisico e layer data link del sistema ISO/OSI
  - **Networking e transport** layer: corrisponde ai layer Networking e layer Transport del sistema ISO/OSI
  - **Facilities layer**: corrisponde al layer Session, Presentation e Application del sistema ISO/OSI
  - **Application** layer: composto dalle applicazioni ITS-s e costruito sui precedenti tre layer

- 2 layer verticali

L'**architettura wave** è un sotto gruppo di funzionalità proposte dall'architettura Figura 1.2 e omette l'implementazione per lo strato **Facilities** e **Application**.

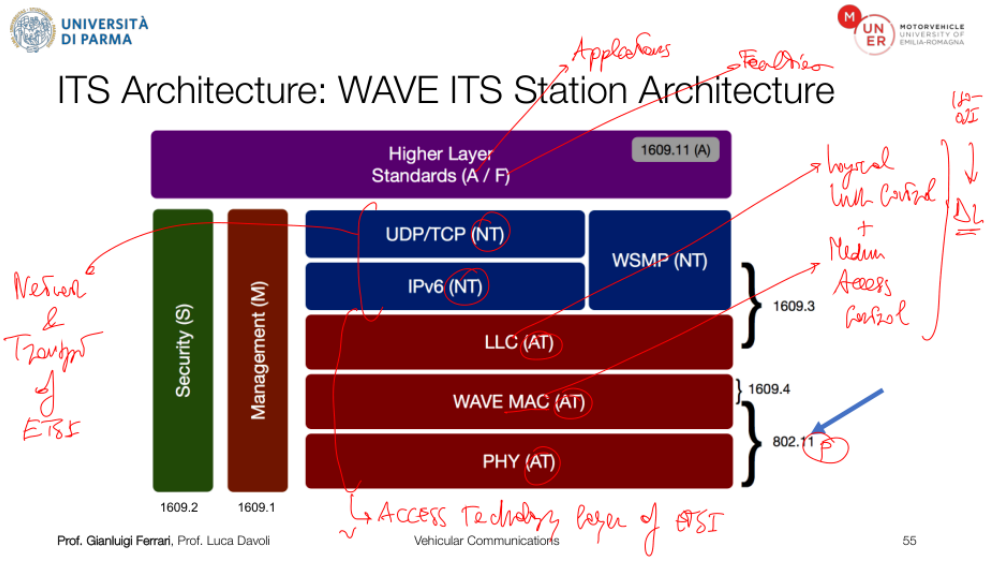

Figura 1.3: WAVE architecture

La comunicazzione nell'architettura di tipo WAVE è categorizzabile in due:

- data-plane: per funzioni data driven come infotainment
- management-plane: per funzioni come la frenata, ecc

WAVE ha un suo protocollo per la comunicazione che prende il nome di **WSMP(WAVE Short Message Protocol)** e serve per effettuare le comunicazioni tra veicoli in movimento.

1.3.1 Applicazioni ITS

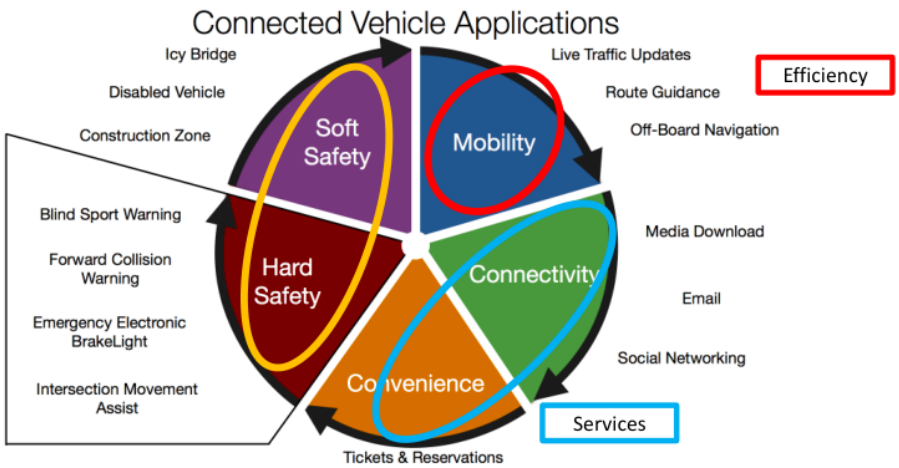

Figura 1.4: ITS applications

1.4 Autonomous driving

1.4.1 NHSTA Levels - USA

- **Level 0**: no automation
- **Level 1**: Funzioni individuali sono automatizzate(frenata, stabilità ecc)
- **Level 2**: Almeno 2 o più controlli sono automatizzati e comunicano fra di loro(adaptive cruise control, lane keep assist)
- **Level 3**: Il veicolo è in grado di gestire tutte le funzioni di guida in determinate condizioni
- **Level 4**: Il veicolo è in grado di gestire tutte le funzioni di guida e l'umano non deve prende mai il controllo

1.4.2 SAE Levels - Europe

- **Level 0**: no automation
- **Level 1**: hands on(adaptive cruise control, parking assistances)
- **Level 2**: hands off(il sistema prende controllo totale e se non riesce a gestire la situazione da il controllo all'umano)
- **Level 3**: eyes off(guidatore presente al posto di guida per necessita di emergenze ma può fare altro come leggere ecc)
- **Level 4**: mind off(umano non deve fare niente)
- **Level 5**: no driver(no volante)

Capitolo 2

Telecomunicacion network basics

2.1 The OSI and Internet models

L'architettura **Open System Interconnetion(OSI)** punta a collegare sistemi eterogenei fra di loro, la sua specifica è la **ISO 7498** ed è un modello costituito di 7 *strati*.

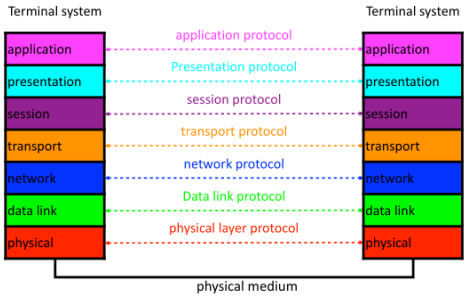

Figura 2.1: Architettura OSI

2.1.1 Application Layer

Livello del modello OSI dove le applicazioni accedono ai servizi di rete, permette ad esempio di trasferire un file, connettersi a database, uso di mail, ecc.

2.1.2 Presentation Layer

Livello del modello OSI adibito alla trasmissione di dati, traduce differenti formati di dati presenti nell'Application Layer in uno standard per gli strati inferiori.

Fornisce servizi per la trasmissione sicura ed efficiente dei dati come:

- data encryption
- data compression
- altre

2.1.3 Session Layer

Livello del modello OSI che permette due computer diversi di **creare**, **usare** e **finire** una sessione, utile per trasmissione di dati e accesso in remoto.

Introduce:

- **Controllo di dialogo**: per regolare la trasmissione e la durata delle stesse
- **Gestione dei token e sincronizzazione**

2.1.4 Transport Layer

Livello del modello OSI adibito alla gestione dei pacchetti da trasmettere:

- Divide pacchetti grandi in più piccoli
- Riordina i pacchetti nell'ordine correto all'arrivo

Gestisce inoltre il riconoscimento degli errori e il loro recupero:

- Ricezzione di pacchetti di confermo dell'arrivo (ACK)
- Reinvia pacchetti persi

2.1.5 Network Layer

Livello del modello OSI adibito alla gestione dell'instradamento dei dati attraverso le sottoreti:

- Determina il percorso di instradamento per arrivare alla destinazione
- gestisce problemi di congestione

2.1.6 Data Link Layer

Impacchetta bits in frames per il layer fisico, provvede a fornire un metodo di trasmissione dei frames affidabile.

2.1.7 Physical Layer

Trasmette bits da pc a pc, regola la trasmissione dello stream di bits sul mezzo fisico.

Definisce come i cavi sono collegati ai pc, come i segnali sono codificati e come i cavi sono collegati ai pc.

2.1.8 Servizi del sistema OSI

Nel sistema OSI ogni layer fornisce un servizio al layer superiore e consuma il servizio del layer inferiore.

Gli strati posono offrire servizi connessioneless o connection-oriented.

Gli strati possono fornire servizi affidabili per evitare la perdita di dati o non affidabili.

I **Service** è il set di primite fornite da un layer ad un'altro layer e definisce cosa un layer è capace di fare.

Protocol è un set di regole che definisce il formato e il significato dei messaggi scambiati tra entità del livello.

2.1.9 Internet Protocol Vs OSI

L'internet protocol è formato di 7 strati, il modello OSI è formato di 4 strati(perchè tcp e ip sono unificati in un unico livello):

- **Application**: Application, Presentation, Session
- **TCP(reliable)/UDP(non reliable)**: Transport
- **IP**: Network
- **Network Access**: Data Link, Physical
- **Hardware**: Physical

2.2 Communication models

Le tipologie di comunicazione sono dipendenti dalle entità coinvolte:

- **end-to-end e relayed**: location
- **unicast, multicast, broadcast**: comunicazione tra una sorgente e un gruppo di destinazioni(number)
- **client-server, peer-to-peer**: comunicazione tra un server e un client o tra due entità terminali(role)

2.3 Delimitation

Delimita le Unita Informative(UI o in inglese IU) da trasmettere in bit o byte, step necessario per permettere ai layer sottostanti di trasmettere i dati.

Implementa metodi per delimitare le UI:

- bit/byte counting
- inserimento di start e stop bit

2.3.1 Bit/Char stuffing

Dopo una ripetizione di un valore binario per 5 volte, si aggiunge un valore opposto per delimitare la UI. Esempio: Avendo una UI 0000000000, si aggiunge un 1 per delimitare la UI e diventa 000001000001.

I vantaggi introdotti sono quelli inerenti alla robustezza e alla sincronizzazione. Mentre gli svantaggi sono legati alla ridondanza delle informazioni e alla perdita di efficienza.

2.4 Sequence control

2.4.1 Fragmentation and aggregation

Capacità di dividere dati in blocchi più piccoli per la trasmissione e di aggregare più blocchi in un unico blocco.

2.4.2 Risequentialization

Necessaria per riordinare i pacchetti nell'ordine corretto all'arrivo.

2.5 Error management

Il controllo dell'errore ha 3 possibili soluzioni:

- **Error detection**: rilevazione dell'errore
- **Error correction**: correzione dell'errore
- **Error recovery**: recupero dell'errore

2.5.1 Error Detection

La error detection si ottiene mediante l'aggiunta di ridondanza ai dati da trasmettere come ad esempio il parity check, aggiunge un bit a 1 se la sequenza ha un numero dispari di 1 e 0 se la sequenza ha un numero pari di 1.

Se applicato questo processo ad un blocco di 2 dimensioni si ottiene la block parity check, che permette di rilevare errori.

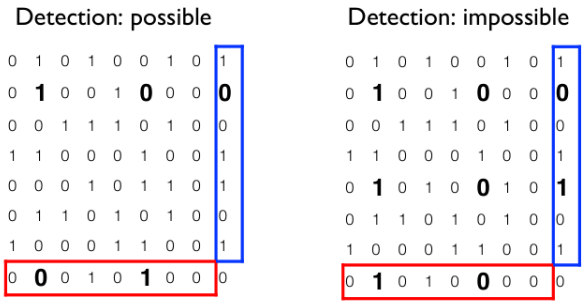

Figura 2.2: Parity block

2.5.2 Complement sum

Quando si riceve il pacchetto, si calcola il **checksum** dei dati ricevuti(come in Figura 2.3) e lo si confronta al checksum allegato al pacchetto ricevuto, nel caso di checksum differente si deve ritrasmettere il pacchetto.

Other codes

Polynomial codes conosciuti anche come **Cyclic Redundancy Check(CRC)**, usano moltiplicazioni tra polinomi per effettuare il checksum.

2.5.3 Error Correction

Con la **block parity check** si possono recuperare errori ma solo se presente un errore di 1 bit.

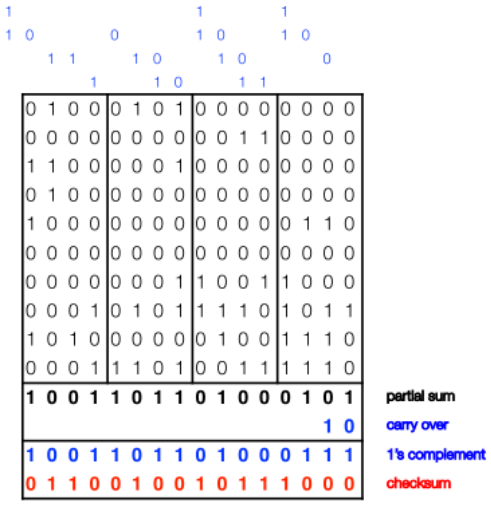

Figura 2.3: Complement sum

Vengono quindi introdotte tecniche **Forward Error Correction(FED)**(ad esempio Algoritmo di Viterbi) che permettono di capire la presenza di un errore mediante algoritmi di ricostruzione.

Con FED si ricorre a ridondanza per eliminare errori(pochi in numero), non sono necessari messaggi di corretta ricezione, che torna molto utile nel caso di comunicazione unidirezionale.

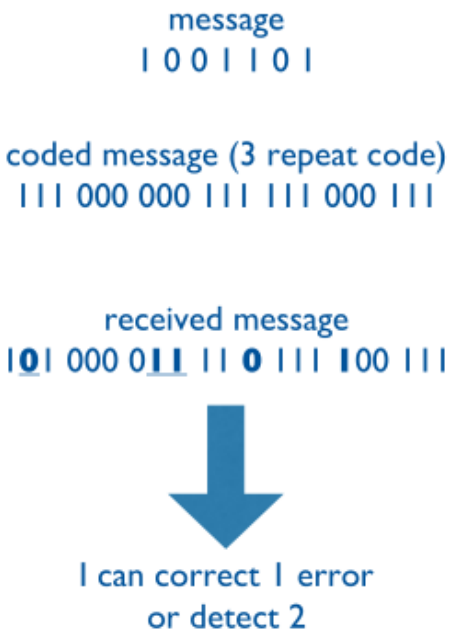

Figura 2.4: Repetition code

2.5.4 Error Recovery

Quando si parla di comunicazione in reti di comunicazioni, si ricade nel richiedere automaticamente un pacchetto che non risulta corretto al ricevitori, esistono approcci automatici come **Automatic Repeat Request, ARQ**.

Esistono inoltre differenti meccanismi di ritrasmissione che come punto focale hanno:

- Error detection
- Acknowledgements
- timers
- IU identifiers

Le procedure ARQ cambiano in base alla dimensione delle finestra:

- **Stop and wait**: finestra di dimensione 1, si attende l'ack prima di inviare il pacchetto successivo
- **Sliding window, go-back-N**: finestra di dimensione N, si inviano N pacchetti prima di attendere l'ack(non ha un selettore per il resending e invia tutto il blocco)
- **Sliding window, selective repeat**: finestra di dimensione N, si inviano N pacchetti prima di attendere l'ack(ha un selettore per il resending e invia solo il pacchetto corrotto)

Stop and Wait

Il pacchetto **ACK(acknowledgement)** solitamente è molto corto per evitare correzzioni nel pacchetto che conferma la corretta ricezione.

È necessario stabilire un tempo limite entro il quale si da per scontato la *scomparsa* del pacchetto, solitamente si basa sul **Round Trip Time(RTT)** che dipende dalla congestione della rete e ne misura i ritardi per arrivare da punto A a punto B.

Altro fattore chiave è capire quali dati sono stati inviati e quali no, per evitare duplicazioni. Per questo problema di è scelto di indicizzare i pacchetti con una sequenza che prende il nome di **SeQuence Number(SQN)** per identificare univocamente quali pacchetti da ritrasmettere.

Si può parlare anche di ACK comulativi mediante l'uso di SQN consecutivi.

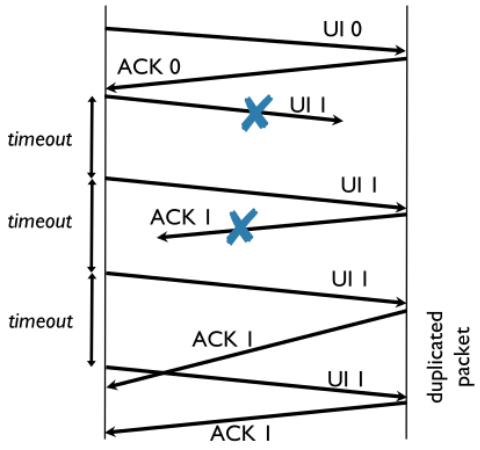

Figura 2.5: Esempio comunicazione stop and wait senza SQN

Stop and wait performance

I tempi considerati sono:

- T_U: tempo di trasmissione di un pacchetto, misurato in s/IU

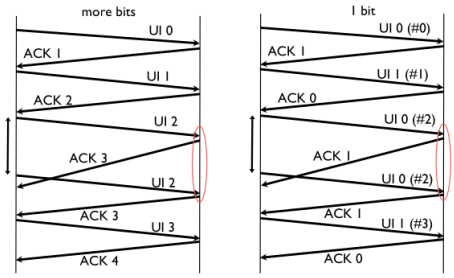

Figura 2.6: Esempio comunicazione stop and wait con SQN

- T_P: tempo di propagazione di un pacchetto, misurato in s/IU
- T_A: tempo di trasmissione di un ACK, misurato in s/IU

Il tempo totale per inviare un'unità informativa(caso ideale):

$$T_{tot} = T_U + 2T_P + T_A \tag{2.1}$$

Il massimo grado di utilizzo di un canale di comunicazione nel caso di **assenza di errore**:

$$\rho_0 = \frac{T_U}{T_{tot}} \tag{2.2}$$

$$= \frac{T_U}{T_U + 2T_P + T_A} \tag{2.3}$$

$$= \begin{cases} \frac{1}{2+2\frac{T_P}{T_U}} & \text{se} \quad T_U = T_A \\ \frac{1}{2\frac{T_P}{T_U}+1} & \text{se} \quad T_U >> T_A \\ 0 \quad \text{se} & T_P >> T_U \end{cases} \tag{2.4}$$

Nel caso di **presenza di errore**, non viene ricevuto l'ACK dal trasmettitore, devo fare alcune assunzioni:

- Indipendenza statisticamente dei pacchetti informativi
- perdita di pacchetti ACK

Indico con p la probabilità di perdita del pacchetto.
Il tempo per l'arrivo di un pacchetto con presenza di errori diventa:

$$\bar{T_1} = (N_t - 1)T_0 + T_1 \tag{2.5}$$

$$= \frac{p}{1-p}T_0 + T_1 \tag{2.6}$$

$$\simeq \frac{T_1}{1-p} \tag{2.7}$$

Dove N_t è desscritta come:

$$N_t = \sum_{k=1}^{\infty} kp^{k-1}(1-p) = \frac{1}{1-p} \tag{2.8}$$

Quindi l'utilizzazione del canele diventa:

$$\rho = (1-p)\rho_0 \tag{2.9}$$

Sliding window

La ricezione con **sliding window** può essere non sequenziale ma non può superare il **timeout** che invalida il pacchetto.

Esistono due strategie per il reinvio dei dati persi con la tecnica dello sliding window:

- **go-back-N**: torna indietro di un numero N di unità informative
- **selective repeat**: reinvia solo i pacchetti effetivamnete persi

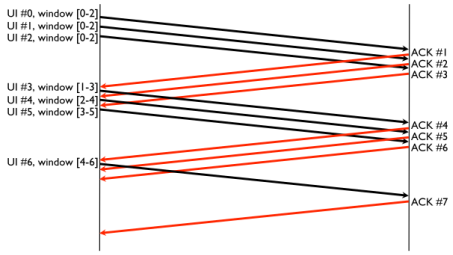

Figura 2.7: Sliding window base

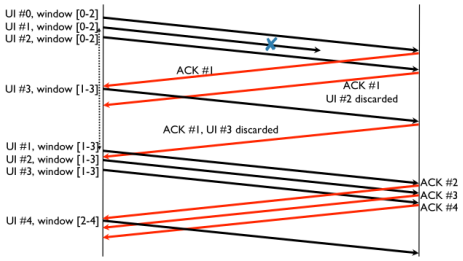

Figura 2.8: Sliding window go back N

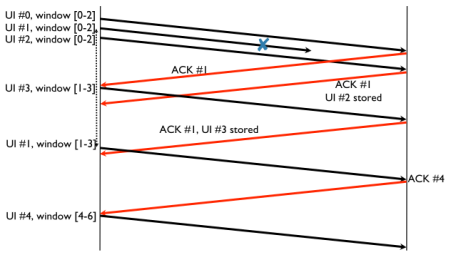

Figura 2.9: Sliding window selective repeat

Capitolo 3

Intra-vehicle Communications

3.1 Bus Systems

3.1.1 Perchè usare i bus?

Un bus che collega tutti i componenti al posto di avere una topologia a grafo completo ha i seguenti vantaggi:

- **Riduzione dei costi**: meno cavi e meno connettori
- **Riduzione del peso**: meno cavi
- **Riduzione del volume**: meno cavi
- **Alta modularità**: modifica veicoli
- **Alta modularità**: cooperazione con OEM
- **Modularità**: riuso di moduli
- **Standardizzazione**: standardizzazione dei componenti e dei protocolli (meno errori scemi)

3.1.2 Casi d'uso per intra-vehicle communications

- Driveline: Engine and transmission control
- Active Safety: Electronic Stability Programme (ESP)
- Passive Safety: Air bag, belt tensioners
- Comfort: Interior lighting, A/C automation
- Multimedia and Telematics: Navigation system, CD changer

La geolacaliuazione è fornita da protocolli di navigazione satellitare. I più famosi sono:

- GPS: USA
- Galileo: EU
- Glonass: Russia
- Beidou: China

Altre soluzioni sono RTK(Real Time Kinematic)!!

3.1.3 Classificazione: On-board-communcation

- Complex control and monitoring tasks: trasmissione dei dati tra ECUs(Engine Control Unit) e MMI(Man Machine Interface simile a HMI che sta per human machine interface)
- Simplification of wiring: rimpiazzare i fili di rame con bus per ridurre la complessità dei cablaggi
- Multimedia bus systems: Trasmette un sacco di dati per i sistemi di intrattenimento

3.1.4 Classificazione: Off-board-communication(OBD connector)

- Diagnostics: diagnosi del veicolo
- Flashing: aggiornamento del software
- Debugging: debug del software

3.1.5 Classificazione per casi d'uso e importanza

Application	Message Length	Message rate	Data rate	Latency	Robustness	Cost
Control and monitoring		2	2	3	3	2
Simplified wiring				1	2	1
Multimedia	1	2	3	1	1	3
Diagnosis						1
Flashing	2		2		1	
Debugging		1	1	2		

Tabella 3.1: Classificazione per casi d'uso e importanza

3.1.6 Classificazione SAE(Society of Automotive Engineers)

3.1.7 Network Topologies

- **Repeter**: amplificazione del segnale a livello fisico
- **Bridge**: medium/timing adaptation, unfiltered forwarding a livello data link
- **Router**: medium/timing adaptation, filtered forwarding a livello network
- **Gateway**: medium/timing adaptation, filtered forwarding, protocol translation a livello application

Class	Data rate	vantaggio	Dispositivi
A	$10kBit/s$	Economico	Diagnosi
B	$64kBit/s$	Correzione errori	Networking ECUs
C	$1MBit/s$	Comunicazione in tempo reale	Drive train
D	$10MBit/s$	Bassa latenza	X-By-Wire

Tabella 3.2: Classificazione SAE

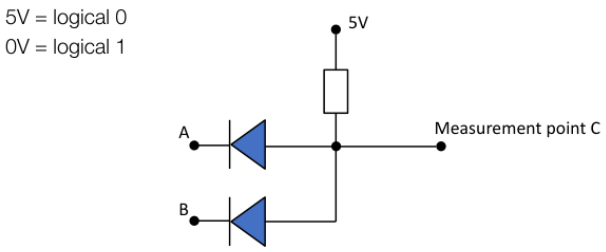

Figura 3.1: Wired OR

Wired OR

Wired AND

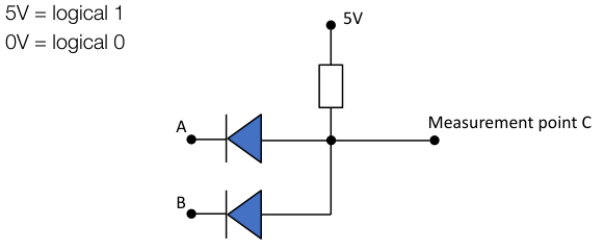

Figura 3.2: Wired AND

Wave Effect

Ad alte velocità entrano in gioco degli effetti non volueti a causa dell'alta velocità.

$$c_0 = 3 \cdot 10^8 \frac{m}{s} \tag{3.1}$$

$$c \approx \frac{1}{3} c_0 = 10^8 \frac{m}{s} \tag{3.2}$$

Il segnale nei veicoli ha un delay di circa $200ns$ quindi il problema insorge quando il tempo per bit è:

$$t_{bit} < 2000ns = 2\mu s \tag{3.3}$$

3.2 Bit coding

Esistono due tipologie di encoding dell'informazione:

- Non Return to Zero (NRZ)
- Manchester

Nella tipologia **NRZ** il valore logico 0 è caratterizzato da un segnale basso, mentre il segnale logico 1 è identificato da un segnale alto.

Nell'encoding di tipo **Manchester** si pone attenzione al cambio di livello per attribuire il valore logico, il valore logico 0 è identificato dal passaggio da basso livello ad alto livello e il valore logico 1 è identificato dal passaggio di stato da alto livello a basso livello.

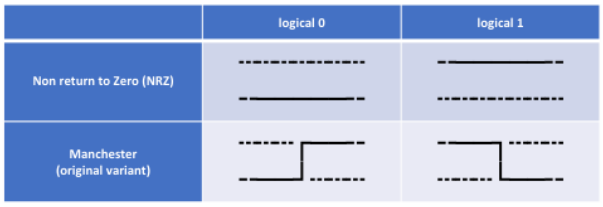

Figura 3.3: Bit coding

3.2.1 Reducing ElectroMagnetic Interference(EMI)

- Aggiungere schermatura ai fili
- usare fili twistati per coppie di fili(annullano effetti di elettromagnetismo a vicenda)
- Ridurre la ripidità del segnale
- usare usare NRZ che ha pochi cambi di stato

3.2.2 Clock drift

Il clock drift è causato dalla costruzione fisica del quarzo usato per il clock, che diffferensce leggermente da altri clock, questo fenomeno causa **desincronizzazione**.

3.2.3 Bit stuffing

Il problema associato all'utilizzo della codifica NRZ è che inviando una serie di bit costanti, in presenza di piccoli ritardi, i dati vengono ricevuti in maniera sbagliata.

Una soluzione proposta è quella del **Bit Stuffing** ed inserisce un bit extra dopo n bit consecutivi.

- Se ci sono 3 uni di fila, agiunge uno zero
- se ci sono 3 zeri di fila, aggiunge un uno

3.3 Classification according to bus access

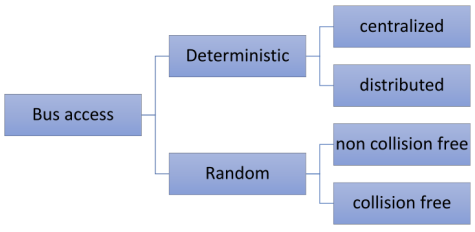

Figura 3.4: Classification according to bus access

3.3.1 Deterministic

Centralized

Accesso al bas di tipo **master-slave**(similare ad un sistema pooling)

Decentralized

Protocolli basati su **token**, tutti collegati a cerchio e si invia il messaggio con un token (che è tipo il bastone della parola) al ricevitore, il ricevitore manda il suo messaggio dopo nella catena insieme altoken e cosi via.

Solo il nodo con il token poò inviare il suo pacchetto di informazioni.

L'altro approccio è il **TDMA**(Time-division multiple access), si identificano i client attaccati al mezzo di comunicazione e si divide la comunicazioni a slot temporali e si può capire chi invia guardando il riferimento al clock

Grande problema di TDMA è la sincronizzazione.

3.3.2 Random

Non Collision Free

CSMA/CA(Carrier Sense Multiple Access)/(Collision Avoidance) misura l'energia del bus e invia quando l'energia è sotto un certo *livello di riferimento*.

CSMA senza CA, se sente il canale occupato, seleziona un tempo random che chiama backoff e aspetta, dopo di che riprova ad ascolare il bus.

CSMA con CA ogni nodo conta il tempo che il noda che sta comunicando finisca, i nodi posso comunicare in qualsiasi momento e quindi ogni conteggio sarà diverso, dopo che il nodo ha finito di comunicare, ogni nodo aspetta il tempo che ha contato il precedenza.

se mentre i nodi stanno aspettando il tempo contato per trasmettere uno dei nodi finisce, si salva il tempo avanzato agli altri nodi e si utilizza quello per il prossimo check di chi tocca.

Questo sistema non è *giusto* e può portare pacchetti a rimanere nella coda per tempi lunghi.

CSMA/CD(Carrier Sense Multiple Access)/(Collision Detection) se più nodi comunicano l'energia sul bus è maggiore del solito e i nodi si rendono conto del problema.

I nodi si fermano(per risparmiare risorse) e mandano un segnale **jamming** che è una sequenza di bit di alto livello per comunicare agli altri nodi il problema e di non comuniicare per un po.

Si applicano ai nodi dei tempi di backoff mediante strategie di backoff e si riprova.

CSMA/CR(Carrier Sense Multiple Access)/(Collision Resolution):

1. **Arbitration phase**: si compete per avere il canale
2. **data**: il nodo che ha vinto il canale comunica

E si itera questo processo ogni volta che un nodo vuole comunicare.

3.3.3 Typical structure of an ECU

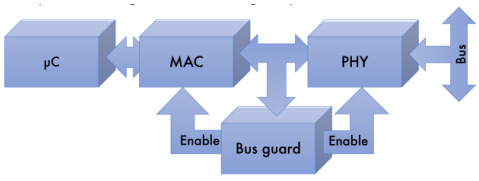

Figura 3.5: Struttura ECU

3.4 Protocols

3.4.1 Il Bus K-Line

Il Bus K-Line è un protocollo standard dell'industria degli anni '80, successivamente standardizzato come ISO 9141. Esistono numerose varianti del protocollo, soprattutto a livello di Link Layer. Tuttavia, ci concentriamo principalmente sull'ISO 14230, noto come KWP 2000 (Keyword Protocol), che specifica i dettagli dei livelli fisici e di collegamento. Questo bus è bidirezionale e consente la comunicazione su un singolo filo, noto come la "linea K".

Facoltativamente, può essere presente anche una "linea L" unidirezionale, che consente reti miste. Il Bus K-Line è principalmente utilizzato per collegare unità di controllo elettroniche (ECU) a strumenti diagnostici (tester), anche se è possibile utilizzarlo per la comunicazione diretta tra ECU in casi rari. I livelli logici sono relativi alla tensione a bordo del veicolo, con valori inferiori al 20% che rappresentano "0" e valori superiori all'80% che rappresentano "1". La trasmissione di bit segue uno schema compatibile con UART (Universal Asynchronous Receiver Transmitter), comprensivo di un bit di start, otto bit di dati, un bit di stop e un bit di parità opzionale. La velocità di trasmissione dei bit può variare da 1,2 kBit/s a 10,4 kBit/s, ma è determinata dall'ECU, non dal bus. Di conseguenza, il master deve essere in grado di gestire più velocità di bit.

Protocollo del Bus K-Line

Il protocollo del Bus K-Line prevede una procedura di stabilimento della connessione che esiste in due varianti: "5 Baud init" e "Fast init". Nella prima variante, il master invia l'indirizzo di destinazione utilizzando una velocità di 5 bit al secondo. L'ECU risponde con il valore 0x55 (01010101), che rappresenta il byte basso della chiave di comunicazione, e il byte alto con la velocità di dati desiderata. Il master deduce la velocità di bit dal modello e invia un bit di "Echo" corrispondente all'alto byte invertito. L'ECU risponde con un altro bit di "Echo" corrispondente all'indirizzo di destinazione invertito.

La seconda variante, "Fast init", ha una procedura più rapida (100 ms) e una velocità di bit costante di 10,4 kBit/s. In questa variante, il master invia un modello di "sveglia" (25 ms basso, 25 ms di pausa) seguito da una richiesta di "Start Communication" che include l'indirizzo di destinazione. L'ECU risponde con una "chiave" entro un massimo di 50 ms, la quale codifica le varianti di protocollo supportate.

La comunicazione sul Bus K-Line è sempre inizializzata dal master, che invia una richiesta a cui l'ECU risponde con una risposta. L'indirizzamento può essere a livello fisico (per identificare specifiche ECU) o a livello funzionale (per identificare classi di ECU come motore, trasmissione, ecc.). La durata di una singola trasmissione a 10,4 kBit/s varia da 250 ms nel miglior caso a 5,5 s nel caso peggiore, il che implica un tasso di dati di livello di applicazione inferiore a 1 KB/s.

Intestazione del Protocollo del Bus K-Line

L'intestazione di un messaggio K-Line comprende un "byte di formato" che codifica la presenza e il significato dei byte di indirizzo. Inoltre, la lunghezza del pacchetto può essere codificata nel byte di formato, consentendo di omettere il byte di lunghezza. L'intestazione include l'indirizzo di destinazione, l'indirizzo di origine, la lunghezza del messaggio e il payload, che può contenere fino a 255 byte. Il primo byte del payload rappresenta l'Identificatore del Servizio (SID), seguito da un byte di checksum che è la somma di tutti i byte del messaggio, calcolata in modulo 256.

Service Identifiers (SIDs) del Bus K-Line

Il Bus K-Line utilizza Service Identifiers (SIDs) per definire il tipo di servizio richiesto o fornito. Tra i SIDs standard troviamo quelli per l'inizializzazione e la terminazione della sessione, come "0x81h Start Communication Service Request" e "0x82h Stop Communication Service Request". Altri SIDs possono essere definiti dai produttori e passati invariati al livello di applicazione. Inoltre, è comune utilizzare due SIDs per ciascun tipo di messaggio: il primo SID rappresenta una risposta positiva, mentre il secondo indica una risposta negativa.

Gestione degli Errori del Bus K-Line

La gestione degli errori sul Bus K-Line prevede che, se arriva un segnale errato, l'ECU ignora il messaggio. Il master, d'altro canto, rileva la mancanza di un'accettazione e ripete il messaggio. Se dati non validi vengono inviati, il livello di applicazione può rispondere con una risposta negativa. In questo modo, sia il master che l'ECU possono reagire in modo appropriato agli errori.

Utilizzo nella Diagnostica On Board (OBD)

Il Bus K-Line è comunemente impiegato nella Diagnostica On Board (OBD). Il pin 7 del connettore OBD è dedicato al Bus K-Line. Nella diagnosi OBD, viene utilizzata una variante più rigorosa del protocollo. La velocità di bit è fissa a 10,4 kBit/s e non vi sono modifiche nei tempi. L'intestazione del messaggio non è più variabile e il byte di lunghezza non è mai incluso. L'indirizzo è sempre incluso e la lunghezza massima del messaggio è limitata a 7 byte. Il protocollo OBD deve utilizzare l'indirizzamento logico da parte del tester e l'indirizzamento fisico da parte delle ECU.

Riassunto

Principalmente utilizzato per scopi di diagnostica, il Bus K-Line utilizza la segnalazione UART per la trasmissione dei dati. La comunicazione su questo bus segue uno schema di tipo Richiesta-Risposta, in cui un dispositivo invia una richiesta e un altro dispositivo risponde con una risposta corrispondente. Questa modalità di comunicazione è ampiamente utilizzata nell'ambito della diagnostica per interagire con unità di controllo elettroniche (ECU) e ottenere informazioni o effettuare regolazioni.

3.4.2 CAN - Controller Area Network

Protocollo realizato nel 1986 da Bosh, la topologia di rete è quella del **Bus**.

Il protocollo riesce a mantenere contemporaneamente fino a 110 nodi e i segnali possibili sono 2:

- **LOW**: segnale dominante
- **HIGH**: segnale recessivo

CAN ha una lunghezza massima del BUS di massimo $500m$ ad una velocità di $125kBit/s$. Successivamente nello standard **ISO 11898** si è descritto due velocità:

- LOW speed CAN: fino a $125kBit/s$
- High speed CAN: fino a $1MBit/s$

Il protocollo CAN interessa solo i layer 1 e 2 dell'ISO/OSI stack e le sue caratteristiche principali sono:

- random access
- collision free
- message oriented
- no indirizzi(solo broadcast o multicast)

Layer fisico

Il protocollo CAN può usare due configurazioni:

- **HIGH SPEED CAN**:
  - fino a $500kBit/s$
  - 2 cavi arrotolati per ridurre interferenze da campi elettromagnetici
  - cavi di collegamento ai nodi massimo di $30m$ (branch)
  - Segnale tra 0 e $2V$
  - resistore terminale di 120Ω
  - l'errore deve essere scoperto entro il tempo di 1 BIT, quindi la lunghezza è vincolata a Equazione 3.5
- **LOW SPEED CAN**:
  - fino a $125kBit/s$
  - 2 cavi per ridurre le interferenze elettromagnetiche
  - nessuna restrizione sui cavi di branch che collegano i nodi al bus
  - segnale tra 0 e $5V$
- **SINGLE WIRE CAN**:
  - velocità fino a $83kBit/s$
  - 1 solo cavo e massa come riferimento
  - segnale tra 0 e $5V$

Avendo il la velocità alla quale si vuole trasmettere i dati, si possono ricavare i metri massimi del cavo, ad esempio avendo una velocità di $R = 500kBit/s$ i metri massimi del filo sono calcolabili come:

$$\frac{1}{R} \geq 2l \cdot 10^{-8} \tag{3.4}$$

$$l \leq \frac{1}{2R \cdot 10^{-8}} \tag{3.5}$$

Comunicazione senza indirizzo

- I messaggi portano un identificativo del messaggio di 11 bit (CAN 2.0A) o 29 bit (CAN 2.0B).
- Le stazioni non hanno un indirizzo, e i frame non ne contengono uno.
- Le stazioni utilizzano l'identificativo del messaggio per decidere se un messaggio è destinato a loro.
- L'accesso al mezzo avviene utilizzando CSMA/CR con arbitrato bit per bit.
- Il livello di collegamento utilizza 4 formati di frame: Dati, Richiesta (Remote), Errore, Sovraccarico (controllo di flusso).
- Il formato dei dati nel protocollo CAN segue lo schema Figura 3.6

Figura 3.6: Formato dei dati CAN

CSMA/CR con arbitrato bit per bit

- Evita collisioni tramite l'accesso al bus controllato dalla priorità.
- Ogni messaggio contiene un identificativo corrispondente alla sua priorità.
- L'identificativo codifica "0" dominante e "1" recessivo: la trasmissione simultanea di "0" e "1" produce un "0".
- Riempimento dei bit: dopo 5 bit identici, viene inserito un bit di riempimento invertito (ignorato dal ricevitore).
- Quando nessuna stazione sta trasmettendo, il bus legge "1" (stato recessivo).
- La sincronizzazione avviene a livello di bit, rilevando il bit di inizio della stazione trasmittente.
- Attendere la fine della trasmissione corrente.
- Attendere 6 bit recessivi consecutivi.
- Inviare l'identificativo (mentre si ascolta il bus).
- Osservare una discrepanza tra il livello di segnale trasmesso e rilevato.
- Questo indica che si è verificata una collisione con un messaggio di priorità superiore.
- Ritirarsi dall'accesso al bus e riprovare in seguito.
- Realizzazione di uno schema di priorità non pre-emptive.

Figura 3.7: Esempio di arbitrato bit per bit

- Garanzie in tempo reale per i messaggi con priorità più alta, ad esempio, messaggi con il più lungo prefisso "0".

Esempio di arbitrato bit per bit(Figura 3.7):
In questo caso il client 2 si rende conto che c'è un problema e fa **back off** lasciando libero il bus.

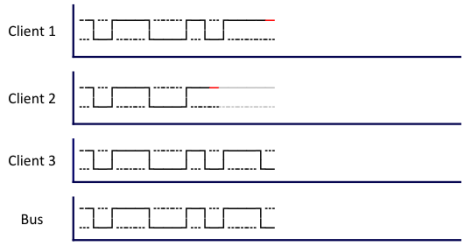

Figura 3.8: Esempio di arbitrato bit per bit

In Figura 3.8 il client 1 si rende conto del bus impegnato da una comunicazione con priorità maggiore e fa **backoff** che porta a mantenere sul bus la comunicazione con priorità maggiore(client 3) e poi segue la trasmissione del dato che il client 3 deve trasmettere.

TTCAN(Time triggered CAN)

un problema del CAN è quello della temporizzazione e del mantenimento dei clock fra i client.

Per risolvere questo problema è stato creato il TTCAN, che ha un nodo dedicato che prende il nome di **time master** è che periodicamente manda un segnale **basic cycles** a tutti i nodi.

Con **basic cycle** si intende un numero definito di slot occupabili che vengono popolati da una fase di organizzazione per priorità all'inizio della stessa.

CAN non si può usare per real time communications.

Message Filtering

I messaggi vengono accettati in base al loro identificativo mediante l'uso di due registri:

Bit	10	9	8	7	6	5	4	3	2	1	0
Acceptance Code Reg.	0	1	1	0	1	1	1	0	0	0	0
Acceptance Mask Reg.	1	1	1	1	1	1	1	0	0	0	0
Resulting Filter Pattern	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**X**	**X**	**X**	**X**

Figura 3.9: Registri di filtraggio CAN

Data Format

Il formato di dati segue:

- NRZ
- bit stuffing
- il frame inizia con un bit significativo(0)
- il message idenfier è 11 Bit(CAN 2.0A) oppure adesso 29Bit (CAN 2.0B)
- Bit di controllo idenficano il tipo di messaggio e la lunghezza
- payload: massimo 8 Byte, trasmesso a $500kBit/s$
- i dati interessanti del frame sono pochi e quindi il data rate effettivo è $30kBit/s$

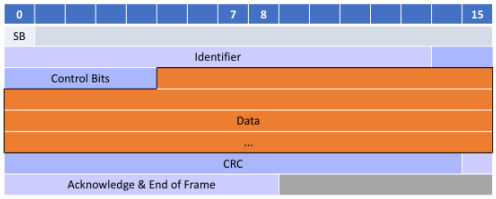

Figura 3.10: Data format

Error detection LOW LEVEL

- Il mittente verifica la presenza di livelli di segnale inattesi sul bus.
- Tutti i nodi monitorano i messaggi sul bus.
- Tutti i nodi verificano la conformità del protocollo dei messaggi.
- Tutti i nodi controllano il riempimento dei bit.
- Il ricevitore verifica il CRC.
- Se uno qualsiasi dei nodi rileva un errore, trasmette un segnale di errore. (6 bit dominanti senza bit stuffing)
- Tutti i nodi rilevano il segnale di errore e scartano il messaggio.

Error detection HIGH LEVEL

- Il mittente verifica la ricezione di un riconoscimento. (Il ricevitore trasmette un bit dominante "0" durante il campo di ACK del messaggio ricevuto)
- Ripetizione automatica delle trasmissioni fallite.
- Probabilità residua di fallimento circa 10^{-11}.

Transoport Layer

Utile alla gestione del flusso, e gestioen dei frammenti di un singolo messaggio divisi in più data frames.

I due protocolli sono:

- ISO-TP
- TP 2.0

ISO-TP

Figura 3.11: ISO-TP

L'**HEADER** introduce un byte opzionale per l'adressing e da 1 a 3 byte **PCI**(Protocol Control Information) che identificano il tipo di messaggio e byte specifici al messaggio.

Single-frame è identificato da 1 byte PCI, l'high-niggle(4bit) è 0, e il low-nibble identifica i bytes in payload, non è possibile fare controllo di flusso.

First-frame: identificato da 2 bytes PCI, high-nibble è 1 e low-nibble con aggiunta di 1 byte definisce la dimensione del payload.

Dopo il primo frame il sender aspetta il **flow control frame**.

Consecutive-frame: 1 byte PCI, high nibble è 2, low nibble è un numero d isequenza **SN** che parte da 1

Flow control-frame: 3 bytes pci, high nibble è 3 e low nibble specifica lo stato del flusso **FS**

- FS 1: clear to send
- FS 2: wait

Il byte 2 specifica la block size BS e il byte 3 indica ST(separation time).

TP 2.0

Protocollo simile a TCP:

- connection oriented
- comunicazione basata su canali
- specifica fasi di: setup, configurazione, trasmissione e chisura
- ogni ECU ha un'indirizzo

Broadcast

- ripetizione per 5 volte per evitare perdita
- Byte 0: adress of desctination ECU
- Byte 1: operation code(broadcast response or request)

- Byte 2, 3, 4: Service ID(SID)
- Byte 5, 6: Response (si alterna 0x5555 e 0xAAAA per dire che non si vuole aspettare risposta)

Channel setup

- Byte 0: adress destination ECU
- Byte 1: Operation code(Channel request, positive response e negative response)
- Byte 2, 3: RX ID(presente se validity nibble di Byte 3 è 0 altrimenti non settato)
- Byte 4, 5: TX ID(presente se validity nibble di Byte 5 è 0 altrimenti non settato)
- Byte 6: application type

Impostazione dei parametri del canale Nel contesto del protocollo TP 2.0 del Bus CAN, sono previste operazioni di impostazione dei parametri del canale. Il Byte 0 indica l'opcode, che può essere 0xA0 per la "Richiesta di configurazione del canale" (parametri per il canale verso l'iniziatore) o 0xA1 per la "Risposta di configurazione del canale" (parametri per il canale inverso). Il Byte 1 specifica la dimensione del blocco, ovvero il numero di messaggi CAN che il mittente deve attendere prima di ricevere un ACK. I Byte 2, 3, 4 e 5 definiscono i parametri di temporizzazione, ad esempio il tempo minimo tra due messaggi CAN.

Gestione generale e chiusura del canale Nel contesto del protocollo TP 2.0 del Bus CAN, sono previste operazioni di gestione generale e chiusura del canale. Il Byte 0 specifica l'opcode, che può essere 0xA3 per il "Test" (che riceverà una risposta con la "Risposta di configurazione della connessione"), 0xA4 per il "Break" (in cui il ricevitore scarta i dati dall'ultimo ACK in poi) o 0xA5 per la "Disconnessione" (in cui il ricevitore risponde con una disconnessione).

Trasmissione di dati tramite canali Nel protocollo TP 2.0 del Bus CAN, la trasmissione di dati avviene tramite canali. Il Byte 0, con il nibble alto, specifica l'opcode. Quando il bit più significativo (MSB) è 0, indica un payload. Quando /AR è 0, indica che il mittente sta ora aspettando un ACK. Quando EOM è 1, indica che si tratta dell'ultimo messaggio di un blocco. Quando il MSB è 1, il messaggio è solo un ACK, senza payload. Quando RS è 1, il canale è pronto per il prossimo messaggio, è un meccanismo di controllo del flusso. Il nibble basso del Byte 0 rappresenta il numero di sequenza. I Byte da 1 a 7 contengono il payload.

CAN - main takeaways

- BUS standard nei veicoli
- message oriented
- CSMA con bitwise arbitration
- error detection
- Transport layer: ISO-TP VS TP 2.0

3.4.3 LIN: Local Interconnect Network

Il goal principale del LIN bus è quello di essere poco costoso di low speed can, la specfica riguarda lo strato fisoc e data link layer.

LIN è simile a K-Line e sfrutta il concetto di master-slave, il nodo master solitamente fa parte di un bus CAN, infatti il lin bus prende il nome di sub bus solitamente.

Il LIN bus è formato di un solo filo e offre una comunicazione bidirezionale fino a velocità di $20kBit/s$.

La trasmissione dei bit è compatibile con UART(1 start bit basso, 8 data bit e 1 stop bit alto) e la comunicazione è del tipo message oriented, quindi non ci sono indirizzi.

L'errore è percepito mediante l'ascolto dell'energia del canale di comunicazione, non è presente una metodo di correzione dell'errore.

I messaggi sono scanditi da cicli temporali definiti dal master.

I segnali inviati dal master sono:

1. sync break: 13 bit bassi e 1 bit alto
2. sync byte: 0x55(01010101)
3. LIN Identifier: 6 bit(I0 fino a I5) + 2 bit di parity
    - 0x00 fino a 0x3b sono tipi di dati definit dall'applicazione
    - 0x3c fino a 0x3d sono messaggi per la diagnosi
    - 0x3e è un messaggio definito dall'applciazione
    - 0x3f è un messaggio riservato
    - i bit di parity sono codficati come

I segnali inviati dallo slave sono risposte con 8 byte di dati, con codifica LSB first(Least Significant Bit) prende anche il nome di **Little Endian**.

La lunghezza è identificata dal'identificatore LIN.

I frame di dati finiscono con il checksum che è costruito come somma di tutti i bytes e dei loro riporti.

Types of requests

Gli **unconditionl frame** sono i frame più semplici, progettati per *pooling periodico* e sono uno slave risponde. Un esempio è: "la porta davanti a destra ha cambiato stato?".

Gli **event triggered frame** viene effettuato il **pooling** da tanti slaves, spesso questa richiesta viene avviata dal can. Può portare a collissione che il master riconosce dai dati corrotti e richiede dati individualemnte agli slaves.

Gli **sporadic frame** inviato dal master solo quando necessario

Un esempio di schedule table è:

Per fare la diagnosi esterna e interrogare il bus lin si procede in due modi:

- si interroga il master collegato al can protocol che risponde impersonando i vari nodi della sua sotto rete
- si interroga direttamente il nodo lin e quindi il master deve fare tunnellign dei dati(schefezza ma più sofisticato)

Tabella 3.3: Schedule table

1	unconditionl	ac
2	unconditionl	rain sensor
3	unconditionl	tire pressure
4	event triggered	door status
5	sporaic	temperature

Main takeaways

3.4.4 FlexRay

Le motivazioni per lo sviluppo di flexray sono:

- tutto fatto by wire (X-By-Wire)
- can bus non ha ridondanza ed è facile che fallisca
- can bus è lento
- can bus non è deterministico

Le soluzioni principali sono state introdotte da OEM(Original Equipment Manufacturer) e le proposte sono state TTCAN, TTP/TTA ecc.

Alla fine BMW, VW, Bosch e altri hanno fatto un consorzio e realizzato un nuovo bus chiamato felxray.

La topologia di rete è:

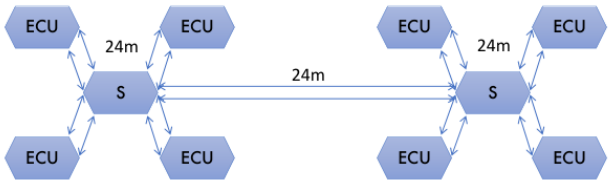

Figura 3.12: FlexRay network

si usano solitamente due canali per comunicare cosi da avere due comunicazioni parallere la $10MBit/s$ che si sommano e danno $20MBit/s$.

La trasmissione necessita di sincronizzazione tra sender e receiver.

Non viene utilizzato il bit stuffing per mantenere la lunghezza dei dati deterministica.