Un Meccanismo Distribuito per Migliorare l'Efficienza delle Reti CAN

Premessa

Il protocollo del Controller Area Network (CAN) è stato sviluppato, in origine, dalla compagnia tedesca Bosch nell'ambito dell'industria automobilistica, ma di recente è stato adottato dai molteplici ambienti della Automated Manufactoring (AM). Sebbene il CAN sia in grado di fornire ottime prestazioni, tuttavia presenta due lacune, le quali, anche se in situazioni non critiche possono sembrare poco importanti, potenzialmente risultano invece alquanto pericolose: 1) il CAN non è capace di effettuare una suddivisione efficiente della banda della rete tra tutte le stazioni trasmittenti; 2) inoltre non è in grado di distribuire soddisfacentemente i ritardi d'accesso dei messaggi trasmessi. Il seguente progetto vuole proporre una lieve modifica dei meccanismi base del CAN che riesca a risolvere i due problemi precedentemente descritti. Tale modifica è pienamente compatibile con il protocollo originale dello strato di controllo d'accesso al mezzo del CAN (MAC), e in tal modo anche con le componenti elettroniche che sono già state sviluppate e che possono essere facilmente reperite sul mercato.

La seguente relazione è così suddivisa: la prima parte descrive in linee generali il modello CAN con le sue principali caratteristiche; poi si passa alla descrizione del meccanismo DPQ secondo la proposta di Gianluca Cena ed Adriano Valenzano (Dipartimento di Automatica e Informatica, Politecnico di Torino); infine, nell'ultima parte viene illustrato il programma CAN Simulator 1.3, modalità di funzionamento ed implementazione.

1 Introduzione

Il CAN incorpora una lieve modifica del protocollo di base CSMA/CD ed è stato inizialmente sviluppato per essere utilizzato nell'industria automobilistica al fine risolvere i problemi di trasmissione riscontrati su alcuni tipi di veicoli.

Il maggior vantaggio nell'uso della tecnologia CAN, rispetto agli altri tipi di fieldbus disponibili, è la vasta scelta di componenti altamente economiche che implementano questo protocollo, come il controller CAN Intel 82527. Il costo è un parametro fondamentale da prendere in considerazione quando si tratta di reti che devono interconnettere device semplici ed economiche, come per esempio un sensore non-intelligente ed un attuatore. Se da questo punto di vista si confronta il CAN con le altre tecnologie disponibili sul mercato vediamo che:

- Un nodo ARINC ha un costo che si aggira intorno ai 1500 dollari
- Un nodo 1553 costa circa 450 dollari
- Un nodo CAN costa soltanto 15 dollari.

Le elevate prestazioni unite al basso costo delle sue componenti elettroniche rendono il CAN uno dei moderni controller di rete più richiesti.

Il CAN è in grado di soddisfare la maggior parte delle necessità di comunicazione degli ambienti industriali. A differenza della tecnica di accesso dello standard IEEE 802.3, infatti, il suo meccanismo MAC assicura che quando avviene una collisione, un arbitraggio non distruttivo basato sulla contesa, arresta tutte le stazioni che trasmettono, tranne quella che ha spedito il frame con priorità più alta.

Nel CAN i frame trasmessi non sono indirizzati ad una destinazione specifica, ma sono considerati come oggetti globali, ad ognuno dei quali è associato un identificatore unico. Gli identificatori (e gli oggetti) sono di tipo broadcast ed ogni nodo è in grado di leggere un oggetto se è interessato ad esso. Il CAN, implicitamente, assegna delle priorità ai differenti oggetti scambiati sulla rete, attraverso i significati dei loro identificatori. Mediante l'utilizzo di un simile meccanismo si raggiunge l'obiettivo di forzare una chiara politica di priorità tra gli oggetti scambiati, anche quando la rete è congestionata.

Come è stato detto in precedenza il CAN non è in grado di suddividere in maniera equa la banda tra tutte le stazioni, ed inoltre non distribuisce in maniera ottimale i ritardi di accesso.

Ciò accade perché i meccanismi utilizzati dal CAN sono strettamente legati alle rivalità sul bus, le quali vengono risolte per mezzo degli identificatori dell'oggetto staticamente assegnati e fissati. In queste pagine si dimostra come sia possibile soddisfare le due richieste sopra menzionate: 1) modificando leggermente il comportamento della parte del sublayer LLC (logical link control) del CAN che filtra i frame in entrata in accordo ai loro identificatori e 2) ridefinendo il significato del campo identificatore dei frame standard del CAN (il quale cambierà in maniera dinamica).

Si propone una leggera variazione del protocollo del CAN che sfrutta una coda a precedeza distribuita, in modo da migliorare quel comportamento della rete che riguarda la produzione attuale ed i ritardi riscontrati da nodi differenti nella trasmissione dei loro frame.

2 Protocollo Base del CAN

Il CAN utilizza una tecnica di accesso al canale CSMA/CD, modificata per forzare una soluzione deterministica delle collisioni su una rete che usa uno schema di priorità basato sugli identificatori degli oggetti scambiati.

Il protocollo CAN adotta una architettura stratificata che è basata sul modello di riferimento OSI, anche se non è totalmente fedele all'OSI. Come per altre reti realizzate per gli ambienti dell'AM, anch'esso si basa su di uno stack di protocollo ridotto, che consiste soltanto nei seguenti tre stati di comunicazione :

- physical layer (strato fisico) PL;
- data link layer (strato di collegamento dati) DLL;
- application layer (strato di applicazione) AL;

La nostra attenzione si deve focalizzare principalmente sul DLL: infatti la maggior parte degli aspetti del CAN che riguardano argomenti quali la condivisione della banda tra tutte le stazioni ed i ritardi di accesso che riscontrano, dipendono dai meccanismi adottati a questo livello. In accordo al ben noto modello sviluppato per le LAN, il DLL è diviso in due sottostrati denominati come:

- medium access control (controllo di accesso al mezzo MAC);
- logical link control (controllo di collegamento logico LLC);

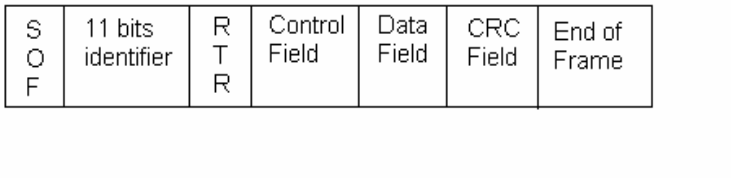

Figura 1: Formato del frame del CAN standard

La fig. 1 mostra il formato di un frame CAN standard. Il campo *start of frame* (SOF) consiste di un singolo bit dominante. Il campo *identifier* ed il bit *remote transmission request* (RTR) formano il campo *arbitration* che è utilizzato dal meccanismo MAC per risolvere le contese.

Il MAC può cominciare a trasmettere un frame non appena rileva che il bus è libero, cioè quando nessun frame è stato trasmesso ed almeno tre bit recessivi (il campo di bit *intermission*) sono stati ricevuti. Durante la fase iniziale di arbitraggio, ogni trasmettitore confronta il bit appena trasmesso con il livello osservato sul bus. Quando un bit recessivo è stato trasmesso da un nodo ed il livello dominante è rilevato sul bus, il nodo deve abortire la sua trasmissione immediatamente poiché ha perso la contesa. Viene effettuato un nuovo tentativo di spedire il frame al termine della trasmissione del frame corrente (vincente).

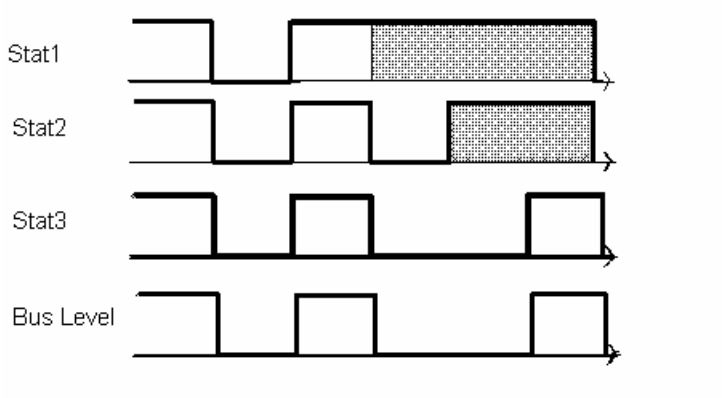

Figura 2: Fase di arbitraggio del CAN

Questa situazione è illustrata in fig. 2 dove tre stazioni tentano di spedire un messaggio nello stesso istante e causano una collisione.

Il meccanismo di arbitraggio assicura che quando avviene una collisione tutte le stazioni trasmittenti eccetto una, arrestano le proprie trasmissioni. In questo modo la contesa è risolta in maniera deterministica, e non si perdono né informazioni né tempo. Poiché il primo campo di ogni frame MAC che segue il bit di SOF è l'identificatore del messaggio scambiato, è chiaro che più basso è il valore numerico dell'identificatore, più alta sarà la priorità del messaggio.

Si dovrebbe notare come nel CAN le priorità siano utilizzate soltanto per risolvere le contese sul bus quando accade una collisione, cioè quando una o più stazioni cominciano a trasmettere allo stesso tempo. Il meccanismo di accesso al mezzo è non-preemptive, in modo che quando una trasmissione comincia in maniera vincente (vince la contesa) essa ha il permesso di inoltrare il messaggio fino all'ultimo bit.

3 Un meccanismo con coda di precedenza distribuita (DPQ).

Il CAN può implicitamente assegnare ad ogni oggetto scambiato sulla rete una priorità che corrisponde all'identificatore dell'oggetto stesso. Anche se questo meccanismo forza un arbitraggio deterministico che è capace di risolvere i conflitti che hanno luogo quando più nodi cominciano a trasmettere contemporaneamente, è abbastanza inefficiente. Infatti, in una rete CAN, un nodo, o più probabilmente un gruppo di nodi, che trasmette oggetti ad alta priorità può impossessarsi dell'intera banda tenendo così lontani gli altri nodi dalla comunicazione.

Come esempio di tale situazione, consideriamo un insieme di device connessi alla rete, ognuno dei quali ha la necessita di render noto un evento allarmante.

A causa del meccanismo adottato dal CAN, ogni messaggio di allarme è associato ad uno specifico identificatore. Sebbene tutti gli allarmi possano essere collocati allo stesso livello di priorità e tutti i nodi dovrebbero avere la stessa probabilità di trasmettere i loro allarmi, esiste un implicito ordine tra allarmi differenti, basato sugli identificatori loro assegnati. In questo modo, anche in assenza di errori della rete, alcune stazioni non saranno capaci di spedire i loro messaggi urgenti se eventi con priorità più elevata sono ripetutamente spediti da altre stazioni. Tale condizione può ovviamente portare ad una situazione potenzialmente pericolosa.

Un meccanismo davvero efficiente deve disporre di un numero di livelli di priorità che non dipenda dall'assegnazione dell'identificatore all'oggetto scambiato. Un aspetto efficiente, che per esempio forza la politica round-robin tra stazioni differenti, deve garantire a tutti gli oggetti scambiati un dato livello di priorità.

Questa caratteristica può essere ottenuta modificando leggermente la funzione di *filtraggio dell'accettazione dei frame* del sottostrato LLC. In particolare, soltanto il significato del campo dell'identificatore nei frame trasmessi deve essere modificato in qualche modo.

Il meccanismo di arbitraggio risultante è capace di forzare una politica round-robin tra le stazioni che vogliono trasmettere un messaggio sul bus, e dispone di due livelli di priorità per i servizi di trasmissione dei frame. Poco o nulla deve essere modificato a livello MAC; in questo modo è possibile riutilizzare le stesse componenti elettroniche sviluppate per l'implementazione del protocollo standard del CAN.

3.1 Principi base

L'idea che sta alla base di questo nuovo meccanismo di controllo dell'efficienza del CAN è rappresentata dall'inserimento di tutti i nodi che vogliono trasmettere sul mezzo condiviso in una coda globale (ampia secondo le caratteristiche del sistema).Secondo la ben nota strategia FIFO, quando un nodo decide di trasmettere un frame, si sposterà in fondo alla coda prima di inoltrare una nuova richiesta di trasmissione. I nodi più vicini alla testa della coda non sono certi di essere serviti prima di quelli che stanno aspettando in coda dopo di loro. Inoltre, in caso di collisione, il nodo più prossimo alla testa della coda ha la possibilità di trasmettere (diremo che ha una *precedenza* maggiore), mentre gli altri dovranno ritentare. Il protocollo della coda con precedenza distribuita (DPQ) basa se proprie operazioni su una coda all'interno della quale vengono inseriti virtualmente tutti i nodi della rete.

In pratica, abbiamo considerato due code, associate ai due differenti livelli di priorità, ed ogni nodo è inserito in entrambe e allo stesso istante. Poiché le due code sono indipendenti l'una dall'altra, possono essere gestite separatamente. La posizione di un nodo nella coda identifica il suo corrente livello di precedenza. Non appena un nodo porta a termine la sua trasmissione, si sposta in fondo alla coda, in modo da abbassare la sua precedenza al minimo. Tutti i nodi che nella coda seguono il nodo trasmittente, si muovono in avanti di una posizione, colmando il vuoto che si è appena creato. Ciò dà luogo ad una arbitraggio del mezzo di accesso che forza una politica round-robin.

La coda non è fisicamente memorizzata in qualche locazione specifica: invece è distribuita tra tutti i nodi della rete. Ogni nodo è responsabile soltanto di memorizzare ed aggiornare la propria posizione nella coda distribuita risultante, in modo che non vi sia alcuna comunicazione superiore che si riferisca a tale meccanismo.

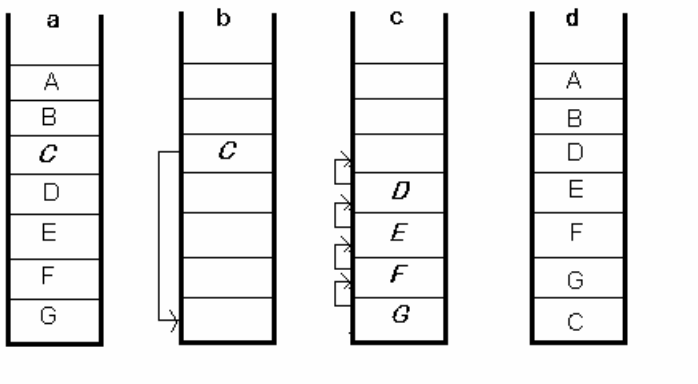

Figura 3: Il meccanismo DPQ

Per esempio, la Fig. 3 illustra una coda nella quale sono stati inseriti 7 nodi etichettati da *A* a *G* (il nodo *G* è in fondo alla coda). Supponiamo che il nodo *C* abbia la facoltà di trasmettere un frame (Fig. 3a), mentre tutti gli altri nodi controllano il frame trasmesso ed

estraggono il livello di precedenza del suo trasmettitore. I nodi da D a G sono a conoscenza del fatto che essi seguono il nodo C nella coda ed ognuno di loro avanza di una posizione in avanti (Fig 3c), raggiungendo la situazione finale della coda mostrata in Fig. 3d.

A questo punto è necessario puntualizzare la differenza tra priorità e precedenza. Per gli scopi della DPQ la priorità è assegnata ad un oggetto (che è un messaggio) quando viene trasmesso sulla rete, mentre la precedenza è sempre assegnata alla stazione. Entrambi i termini sono in relazione con la risoluzione deterministica delle collisioni sul bus, comunque la priorità DPQ (o priorità) si riferisce ad un attributo statico, scelto dall'utente come parametro (Qualità del Servizio, QoS) ogni volta che viene invocato un servizio di trasmissione. Invece la precedenza è un parametro dinamico, gestito dal meccanismo DPQ ed è invisibile all'utente. La precedenza di un nodo dipende dalla sua posizione nella coda distribuita. Poiché vi sono due DQ che corrispondono ai due livelli di priorità, in pratica ogni nodo deve amministrare due livelli di precedenza indipendenti.

La priorità del messaggio e la precedenza della stazione trasmittente sono usate in quest'ordine così da formare una priorità prefissa, che è usata come parte iniziale del campo identifier (ed in questo modo del campo di arbitraggio) in ogni frame trasmesso. In questo modo è possibile implementare un controllo più efficiente sulla risoluzione delle collisioni che si basa ancora sul meccanismo di arbitraggio convenzionale del CAN.

Come è stato già affermato, il meccanismo DPQ supporta due livelli di priorità, che sono high e low. Ovviamente le trasmissioni con priorità più alta avranno la precedenza su quelle a priorità più bassa. Inoltre il nostro meccanismo è provvisto di tre classi di arbitraggio: queued, over-queue ed under-queue. Le trasmissioni queued fanno uso del meccanismo DPQ, così sono soggette alla politica round-robin descritta sopra. Né le trasmissioni over-queue, né quelle under-queue fanno uso della DPQ. Si comportano in maniera assai simile al meccanismo base del CAN, e sono utilizzate come opzione per *bypassare* le operazioni DPQ. Tutte le trasmissioni over-queue hanno la precedenza su quelle queued alla stessa priorità, mentre si le queued che le over-queue hanno la precedenza sulle trasmissioni under-queue.

3.2 Implementazione

Il meccanismo DPQ può essere implementato senza l'introduzione di alcuna modifica al formato base dei frame del CAN: infatti è sufficiente usare una porzione del campo *identifier* in modo da codificare le informazioni di controllo.

Poiché la lunghezza di un campo *identifier* tradizionale nel CAN standard (11 bit) è troppo piccolo per contenere sia l'informazione necessaria per il meccanismo DPQ che per

l'identificatore effettivo del frame, è stato adottato il formato (*identifier*) esteso del CAN. Questo formato utilizza un identifier largo 29 bits, che diviso in un *identifier* di base ed in un *identifier* esteso come mostrato in Fig. 4.

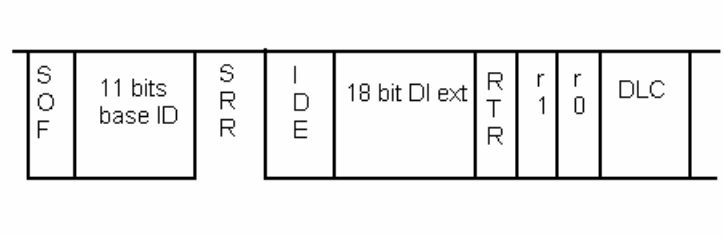

Figura 4: Formato dell'intestazione dei frame del CAN esteso

La DPQ utilizza i primi 11 bit del campo *identifier*(ID base) per le sue informazioni di controllo, mentre i 18 bit di ordine inferiore rimanenti (ID ext) sono utilizzati per memorizzare l'identificatore effettivo dell'oggetto scambiato (EID).

Il ruolo del campo EID nel meccanismo DPQ è identico a quello dell'identifier utilizzato nelle reti CAN convenzionali, il quale identifica senza ambiguità l'oggetto scambiato. Comunque, a differenza del campo identifier del CAN, il valore del campo EID sotto normali condizioni operazionali non è usato dalla DPQ per risolvere contese dovute a collisioni sul bus. La Fig. 5 illustra il formato dei frame DPQ, e specificatamente l'uso fatto del campo identifier del CAN.

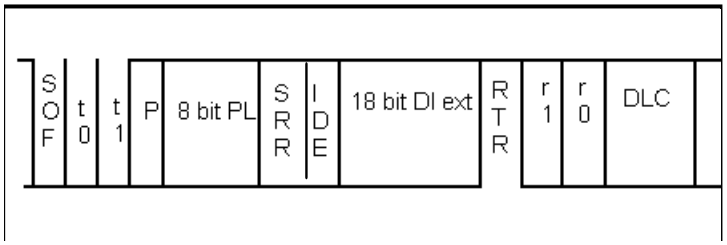

Figure 5: Formato dell'header dei frame DPQ.

I primi due bit (t0 e t1) devono essere settati al valore logico zero per i frame trasmessi utilizzando il meccanismo DPQ. Ciò rende la DPQ compatibile con i meccanismi convenzionali del CAN, cioè che le stazioni DPQ e le stazioni CAN possono essere connesse alla stessa rete ed allo stesso istante. Infatti, la DPQ è pienamente compatibile con le operazioni del sottostrato MAC del CAN quando viene adottato il formato del frame convenzionale. In un certo modo è compatibile anche col formato esteso dei frame, ma soltanto fin quando nella trasmissione sono utilizzati identificatori i cui primi due bit non sono entrambi zero. In questo modo la trasmissione dei frame convenzionali avrà una priorità più bassa rispetto alle

trasmissioni DPQ. Ciò è abbastanza ragionevole, poiché se non è così, il traffico convenzionale può sfruttare l'intera lunghezza di banda rendendo inutile il meccanismo DPQ.

Il bit (P) di priorità, specifica se il frame deve essere trasmesso come un frame ad alta priorità (P=0), oppure come frame a bassa priorità (P=1). I successivi 8 bit rappresentano il livello di *precedenza* (PL) del frame. Il livello di precedenza dei frame trasmessi che usa il queued mode è identico alla posizione nella Queue (QP) della stazione che sta trasmettendo il frame (questa posizione è identica alla precedenza della stazione), e varia da 1, quando il nodo è davvero nella prima posizione nella coda distribuita, a 254 quando si trova in coda. Un valore del livello di precedenza pari a 0 significa che il frame è stato trasmesso utilizzando la modalità over-queue di trasmissione, mentre il valore 255 è utilizzato per la modalità di trasmissione under-queue. E' chiaro da questa scelta che i frame over-queue avranno la precedenza su tutti i frame trasmessi alla stessa priorità, a differenza delle trasmissioni under-queue che avranno la precedenza minore.

I due bit riservati t0 e t1, il bit di priorità ed il campo per il livello di precedenza formano insieme la priorità prefissa (PP) che, sotto normali condizioni operazionali, è il solo campo utilizzato dalla DPQ per risolvere le contese sul bus quando avviene una collisione.

3.3 Descrizione del meccanismo DPQ

Le operazioni del meccanismo base DPQ sono molto semplici. Ogni nodo deve gestire soltanto due variabili locali addizionali che sono utilizzate per memorizzare la posizione dello stesso nodo sia nella coda distribuita ad alta priorità che in quella a bassa priorità. Queste due variabili possono essere indicate come QP_H e QP_L rispettivamente, e poiché le operazioni svolte sono le medesime per entrambe le code e poiché le due code sono indipendenti è sufficiente considerare la generica coda X. La posizione variabile relativa a questa coda è indicata come QP_X.

Inizialmente assumiamo che due nodi differenti non possano avere lo stesso valore per la variabile QP_X o, in altre parole, si assume che ogni nodo attivo occupi una posizione differente nella coda distribuita. Questo implica che al più 254 nodi che adottano il meccanismo DPQ possono essere attivi allo stesso tempo nella rete.

Inoltre, si assume che la Queue è riempita dalla testa alla coda, senza che nessuna posizione tra due qualsiasi stazioni adiacenti possa rimanere vuota. Se per esempio vi sono 10 nodi attivi, essi occuperanno le posizioni nella coda che variano da 254 a 245 incluso. Assicurate le due precedenti condizioni, passiamo alle procedure che hanno il compito di regolare la modalità secondo la quale un nodo entra in una DQ oppure la abbandona.

Sia F un frame e sia indicato il suo campo utilizzando la notazione convenzionale adottata: per esempio F.P è il campo di priorità del frame. In accordo all'algoritmo DPQ, quando un nodo i chiede di trasmettere un frame F alla priorità X, si muoverà in coda alla relativa Queue, effettuando le seguenti operazioni:

```
F.P=X;

F.PL=QP_Xi;

Send_Frame(F);

QP_Xi=254;
```

Se avviene qualche errore durante la trasmissione, la funzione Send_Frame ritrasmette il frame F finché non sia spedito correttamente: in questo modo la precedenza è abbassata soltanto quando il frame è stato trasmesso con successo. D'altro canto, tutti i nodi che seguono il nodo trasmittente nella coda devono spostarsi in avanti di una posizione verso la testa. In pratica, tutti i nodi che rilevano che il frame F sia stato trasmesso correttamente, cioè ogni nodo j tale j!=i, devono controllare il campo PL del frame in ricezione ed aggiornare la sua variabile locale QP_X opportunamente:

```
F=ReceiveFrame();

X=F.P;

if (QP_Xj >= F.PL) AND (QP_Xj > 1)

QP_Xj=QP_Xj - 1;
```

Queste sono le due principali modifiche che devono essere incluse nella funzione di filtraggio di accettazione di frame di ogni stazione CAN.

3.4 Entrata ed uscita dalla coda distribuita

In accordo al meccanismo DPQ sopra descritto, se un nuovo nodo fosse capace di collocarsi casualmente nella coda e di cominciasse le sue trasmissioni, si potrebbe avere una posizione di conflitto. Allo stesso modo, se un nodo sparisse semplicemente, verrebbe a crearsi un vuoto nella DQ.

Sebbene il meccanismo DPQ sia capace di correggere entrambe le situazioni, le procedure designate per permettere ai nodi di entrare e di abbandonare correttamente le DQ possono migliorare le *preformance* dell'intera rete. Specificatamente la procedura di entrata nella rete è in grado di evitare conflitti di posizione, mentre la procedura di uscita può prevenire la creazione di buche nelle DQ.

L'aspetto base della DPQ può essere migliorato adottando la seguente procedura Enter. Un nodo i, che vuole accedere alla DQ, manda una frame ad alta priorità usando la classe di arbitraggio over-queue. L' *identifier* esteso del frame viene settato al valore specifico ID_Enter_Queue (è la priorità più elevata con cui un frame può essere spedito). Questo nodo si sposta, perciò, in coda sia nella coda ad alta che in quella a bassa priorità. Tutti gli altri nodi devono soltanto spostarsi di una posizione in avanti in entrambe le code.

```
Enter {

F.P = H;

F.PL = PL_Over_Queue;

F.EID = ID_Enter_Queue;

Send_Frame(F);

QP_Hi = 254;

Qp_Li = 254;

    }
```

Allo stesso tempo, la procedura Leave è utilizzata da una stazione per informare tutti gli altri nodi DPQ attivi del suo desiderio di lasciare la coda distribuita. In questo caso, essa spedisce sulla rete un frame ad alta priorità il cui identifier effettivo è settato ad un certo valore ID_Leave_Queue e il cui campo date è settato al valore della sua variabile QP_Li col risultato che ogni nodo che precede il nodo uscente deve spostarsi verso il basso in entrambe le code così da colmare la hole appena formatasi.

```
Leave {

F.P = H;
```

```
F.PL = QP_Hi;

F.DATA = QP_Li;

F.EID = ID_Leave_Queue;

Send_Frame(F);

    }
```

4 Vantaggi della DPQ

Quando il carico sulla rete è basso, la tecnica CSMA/CD adottata dal CAN offre un vantaggio significativo rispetto altri meccanismi di accesso al mezzo per le reti in tempo-reale come il passaggio di token nel quale il ritardo riscontrato da un nodo che vuole trasmettere un frame è molto breve.

Quando il carico sulla rete cresce, comunque, il CAN tende a divenire inefficiente. Infatti, si può facilmente dimostrare che alcuni oggetti ad alta priorità potrebbero sfruttare l'intera larghezza di banda. Di conseguenza, in una rete CAN pesantemente caricata, alcuni nodi (gli altri) potrebbero riscontrare ritardi impredicibilmente lunghi per la trasmissione di oggetti urgenti.

In alcune circostanze, la natura sia delle applicazioni che corrono sulla rete che del loro traffico generato possono richiedere che la banda tra due diverse stazioni sia divisa efficientemente, oppure che sia limitato il ritardo che essi riscontrano nelle loro trasmissioni. Quando vengono usati sevizi di trasmissione DPQ queued, è possibile dimostrare che il ritardo riportato da un nodo nella trasmissione di frame ad alta priorità non è mai più lungo del tempo impiegato da tutti gli altri nodi per la trasmissione di ogni frame, a patto che nessun frame ad alta priorità DPQ over-queue o frame CAN convenzionali sia trasmessi nel frattempo.

Infatti, anche se un nodo appare in coda alla Queue e tutti gli altri nodi tentano ripetutamente di trasmettere i loro frame, dopo che tutti gli altri nodi hanno avuto la loro opportunità di trasmettere (e si sono spostanti in coda alla Queue) il nodo in ritardo raggiunge la head della Queue. Si dovrebbe notare che quando la rete ha un carico leggero, la DPQ si comporta in maniera simile al CAN e non peggiora in alcun modo i bassi ritardi di trasmissione tipici di questo protocollo. In questo modo si può dimostrare che quando una

rete ha un carico pesante, la DPQ divide uniformemente tra tutte le stazioni la banda legata al traffico queued.

Un altro aspetto che deve essere preso in considerazione quando si incontra una rete DPQ caricata pesantemente è l'uso delle priorità: in alcune situazioni, infatti, la DPQ assicura che tutte le richieste di trasmissione di frame ad alta priorità saranno soddisfatte prima che qualsiasi frame a bassa priorità sia trasmessa. Il carattere di efficienza è comunque garantito anche per le trasmissioni a bassa priorità.

5 Il programma CAN simulator 1.3

Il programma CAN simulator 1.3 è stato realizzato mediante C++ in ambiente Windows.

In tutte le fasi di implementazione del progetto, ho cercato sempre di mantenere un certo livello di realismo nella simulazione del CAN. Cioè ho voluto fare in modo che un utente particolarmente curioso, possa effettivamente vedere cosa accade in ogni istante della corrente sessione del bus, secondo le caratteristiche da lui imposte alla rete in esame. In questo modo è possibile quindi rilevare quale stazione stia iniziando a trasmettere ed in quale istante; se si è verificata una collisione e quali sono le stazioni coinvolte, insieme ovviamente a tutte le normali informazioni quali l'identificatore della stazione, il numero di byte da trasmettere, la corrente posizione della stazione trasmittente nella rispettiva coda distribuita (se si utilizza l'opzione *CAN with DPQ*), la deadline del frame trasmesso etc...

5.1 Modalità di funzionamento

Mediante CAN simulator 1.3 è possibile effettuare una simulazione abbastanza fedele di una rete CAN.

Le opzioni della barra menu allo start del programma impongono all'utente di selezionare una versione del CAN tra quelle disponibili, cioè il *CAN 1.0* (standard, 11 bit identifier), il *CAN 2.0* (extended, 29 bit identifier) ed infine il *CAN 2.0 with DPQ* (extended, 18 bit identifier).

Una volta selezionato il modello CAN , si può passare allora alla definizione delle caratteristiche della rete da simulare mediante l'opzione *Config_net.* In effetti, sappiamo che il numero di stazioni che possono prendere parte ad una rete CAN è limitato solo dalla velocità del bus, ma poiché lo scopo principale di questo programma è quello di mettere a

confronto il CAN con DPQ ed il CAN extended standard, ho preferito fissare a 254 il numero massimo di stazioni che possono essere selezionate (essendo 254 la dimensione massima delle code DPQ). Inoltre il numero massimo di frame che possono essere trasmessi è fissato in 500, ma solo per una questione di memoria: modificando opportunamente il valore MAXFRAME nel programma tale limite può essere tranquillamente adatto alle specifiche necessità.

I passi successivi consistono nel selezionare, se si vuole, il comando *Output*, il quale consente di stampare i dati della corrente simulazione sulla standard printer, ovvero di scriverli più comodamente sul file denominato "Output.dat". Verranno quindi descritti: il numero di frame trasmessi, il numero di

stazioni, il modello CAN adottato, il livello di collisione (vedi dopo), lo scheduling dei frame trasmessi, il tempo trascorso nella simulazione, il numero di collisioni, il ritardo di accesso totale. Per default, se non viene selezionato nulla, nessun tipo di stampa verrà effettuata.

L'opzione *Collision level high/low* permette all'utente di impedire o meno un elevato numero di collisioni. Per default il livello è settato ad *high.*

Per mezzo del comando *Mode* si può imporre al programma di interagire (*Interact*) con l'utente, permettendogli di osservare gli eventi più importanti. Per default *Mode* è settato ad *Auto*, cioè la simulazione si arresterà soltanto quando l'ultimo frame sarà stato trasmesso.

Le impostazioni correnti, nonché il traffico generato, possono essere salvati su file per essere rivisti successivamente.

Rispettando ciò che accade nella realtà, per quanto riguarda la simulazione, il modello *CAN 2.0* ed il modello *CAN 2.0 with DPQ* sono totalmente intercambiabili (cioè la simulazione effettuata col primo modello può essere ripetuta col secondo mostrandone in tal modo le differenze, che è l'obiettivo di questo programma). La simulazione del CAN 1.0 non gode ovviamente di questi requisiti.

Al termine di ogni simulazione un *Message-Box* chiederà di illustrare o meno, mediante un istogramma, i dati della simulazione. L'asse delle ascisse indica il ritardo di accesso medio riscontrato per ogni stazione trasmittente. L'asse delle ordinate descrive le stazioni ordinate secondo priorità (valore dell'identificatore).

Il programma, inoltre, crea e aggiorna un file denominato "Excel2.dat", all'interno del quale vengono immessi i dati relativi ai ritardi riscontrati e la distribuzione degli stessi negli intervalli di tempo. Tali informazioni possono essere comodamente importate da Excel per realizzare grafici esplicativi come quelli in coda alla presente relazione.

5.2 Implementazione

Per quanto riguarda l'implementazione del *CAN simulator 1.3* mi riferirò soltanto alla parte relativa alla simulazione, non tratterò quindi l'implementazione dell'interfaccia grafica.

Le due più importanti procedure del programma sono senza dubbio le funzioni *Canstart* e *Canstart2*, che si occupano rispettivamente della simulazione del meccanismo DPQ per il modello proposto e del meccanismo CSMA\CD per il modello standard.

Il vettore di stazioni CAN ,di dimensione massima *MAXSTATION*, che come è stato detto in precedenza è fissato a 254, contiene tutte le informazioni relative alla rete selezionata Ogni stazione è una struttura formata dal suo identificatore, dal numero di byte da trasmettere, dal suo stato di attività, dall'inizio e dalla fine delle sue trasmissioni, dal suo campo di priorità P, e dai campi relativi alle due code DPQ.

La funzione *Trafficmaker* ha il compito di generare tutto il traffico della rete. Si intende cioè che il traffico è generato in maniera *off-line*: ogni frame generato viene inserito in una coda per poi essere preso in esame soltanto durante la simulazione, dando in questo modo, all'utente, la sensazione di un traffico prodotto in maniera *on-line*.

5.2.1 Il meccanismo CSMA\CD e la funzione Canstart2

Il file header *list.h* contiene le funzioni che permettono di simulare il meccanismo di risoluzione delle contese sul bus CSMA\CD. La funzione *Canstart2* provvede ad estrarre dalla coda un frame, che viene denominato come *Old_Frame*, il quale è candidato alla trasmissione. Si estrae, successivamente, un secondo frame *New_Frame*, ed *Old_Frame* viene confrontato con quest'ultimo. Se *Time_in* di *Old_Frame* coincide con il valore del bus_time ed inoltre il valore *Time_in* di *New_Frame* è maggiore (dovrà essere trasmesso successivamente) allora il frame *Old_Frame* può essere inoltrato senza problemi. Altrimenti se il valore del *Time_in* di *Old_Frame* è maggiore del bus_time, il bus è libero e rimarrà tale fin quando non coinciderà finalmente col valore *Time_in*. Infine se *Time_in* di *Old_Frame* e *New_Frame* coincidono, siamo

di fronte ad una collisione: il frame con identificatore più basso viene inserito in una lista concatenata. Prima di continuare la trasmissione di *Old_Frame*, viene estratto un nuovo *New_Frame*, e l'operazione si ripete fin quando non vi è più alcuna collisione. Il frame che esce sconfitto dalla contesa, viene inserito, come detto, in una lista concatenata, ordinata secondo il valore dell'identificatore crescente. Nella realtà, infatti, se un gruppo di stazioni ad alta priorità ha perso la contesa, allora la stazione con priorità più elevata tra queste avrà assicurata la possibilità di inoltrare il proprio messaggio nella successiva fase del bus (mostrando in tal modo che il CAN standard non distribuisce equamente i ritardi di accesso tra tutte le stazioni). Non appena il bus è libero, la *Canstart2* controlla se per caso vi sia almeno un frame nella lista concatenata. Se ciò è vero, elimina il frame della lista e lo trasmette. Allo stesso modo non appena una stazione ha terminato la propria trasmissione, si confrontano le informazioni relative al frame *Old_Frame* (il *New_Frame* precedente) e al primo nodo della lista (che come detto risulterà senz'altro il vincitore del gruppo in attesa).

5.2.2 Il meccanismo DPQ e la funzione Canstart

L'implementazione del meccanismo DPQ segue a grandi linee quella del meccanismo precedentemente descritto. Le funzioni che lo implementano sono contenute nel file header *listdpq.h.*

Le osservazioni più interessanti riguardano comunque la generazione del traffico e la risoluzione delle contese.

Ho detto in precedenza che la funzione *Trafficmaker* ha il compito di generare il traffico, ma nella simulazione del CAN con DPQ si preoccupa anche di effettuare l'operazione di enter di una stazione nelle due code distribuite, aggiornando quindi i valori *QP_L* e *QP_H* secondo quanto spiegato in 3.4.

Seguendo sempre la stessa alternanza *Old_Frame*/*New_Frame* della *Canstart2*, le contese vengono risolte in questo modo. Si controlla il valore della priorità P, e il frame con priorità più bassa viene inserito ovviamente nella lista concatenata ordinata secondo PL, relativa alla QP_L. Se i valori di P sono identici, si prende in esame il campo PL e il frame verrà inserito nella relativa lista (se P è 0 in QP_H, altrimenti in QP_L).

Al termine della trasmissione di ogni frame viene confrontato il frame *Old_Frame* (il candidato) con il primo della lista QP_H se non è vuota, altrimenti col primo della QP_L ; il frame con priorità più elevata e quindi con precedenza maggiore sarà inoltrato, mentre glia altri dovranno essere inseriti nelle liste dovute.

Tutte le volte un cui una stazione trasmette il proprio messaggio (Send_frame), viene modificato il suo campo QP_X, il quale viene settato a 254; dopodiché (Receive_Frame) vengono decrementati di una unità tutti i valori relativi al campo QP_X delle stazioni che seguono la stazione trasmittente (cfr. 3.3).

La procedura *Leave* viene implementata modificando i valori QP_L e QP_H secondo quando descritto in 3.4. Non appena una stazione termina la trasmissione del frame corrente, se ha ultimato le sue trasmissione (ha trasmesso un numero di frame pari al valore del campo *sent_bytes*), essa deve comunicare a tutte le altre stazioni la sua uscita dalle due code: ciò verrà fatto, mediante un ciclo while incrementando di un'unità il valore dei campi QP_L e QP_H di tutte le stazione attualmente attive che precedono la stazione uscente nelle rispettive code.

5.3 Analisi dei risultati

Potrebbe essere lecito domandarsi se possa risultare davvero conveniente modificare un modello quasi perfetto, per convenienza e prestazioni, come il CAN, per un modello nuovo, tutto da scoprire.

Traducendo in termini di prestazioni quello che ci chiediamo è :- I ritardi di accesso riscontrati con il modello DPQ superano quelli del CAN standard?-. Dopo molteplici prove su configurazioni di rete differenti, ho potuto constatare che le prestazioni del CAN con DPQ (dal punto di vista dei ritardi, ovviamente), sono assai simili a quelle del CAN 2.0, leggermente migliori o leggermente peggiori.

I tre grafici che seguono mostrano i risultati ottenuti considerando una rete con 25 stazioni trasmittenti, un carico variabile da 80 fino a 500 frame ed una velocità di 1Mbit/sec.

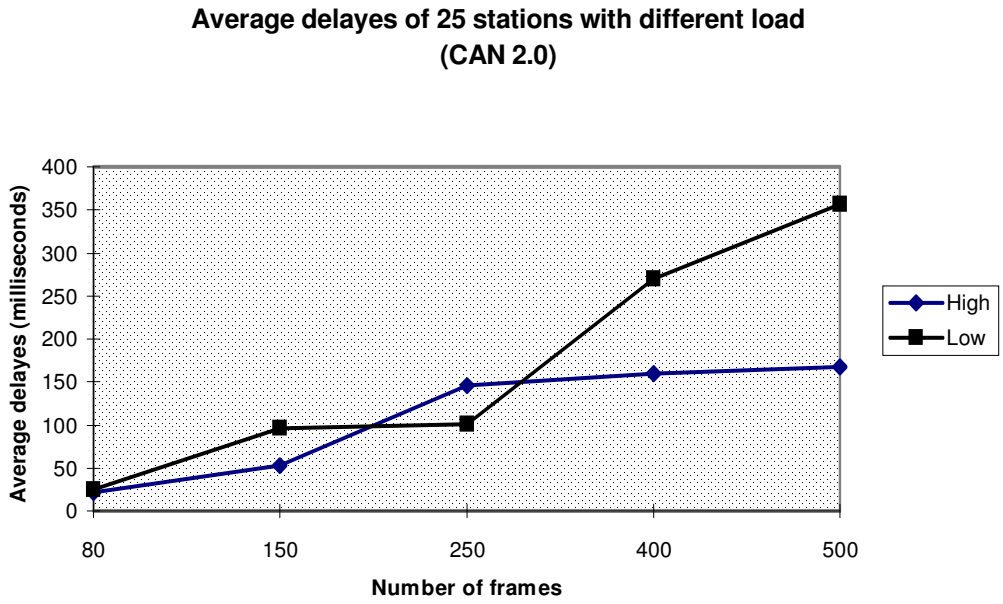

Grafico 1.

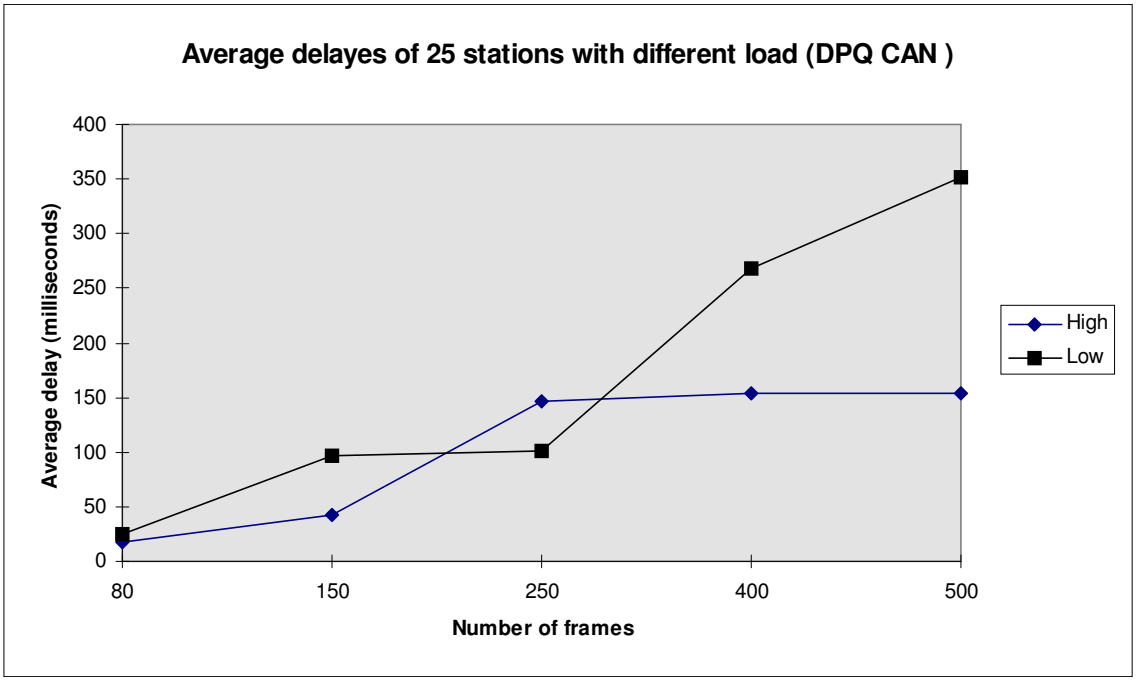

Grafico 2.

Il grafico 1 ed il grafico 2 mostrano il comportamento delle due versioni di CAN nelle diverse situazioni di carico, sia per quanto riguarda il traffico ad alta che a bassa priorità. Si può facilmente notare come il loro comportamento sia quasi del tutto speculare; ciò dimostra, quindi, che il DPQ CAN è candidato a risolvere i problemi sopra menzionati, ma risulta comunque ancora un CAN affidabile.

Il grafico 3., che segue, confronta infine le due versioni del CAN secondo i dati precedenti, considerando il ritardo totale (alta e bassa priorità) : possiamo reputare tale grafico, come la risposta in cifre alla domanda che ci siamo posti all'inizio di questo paragrafo.

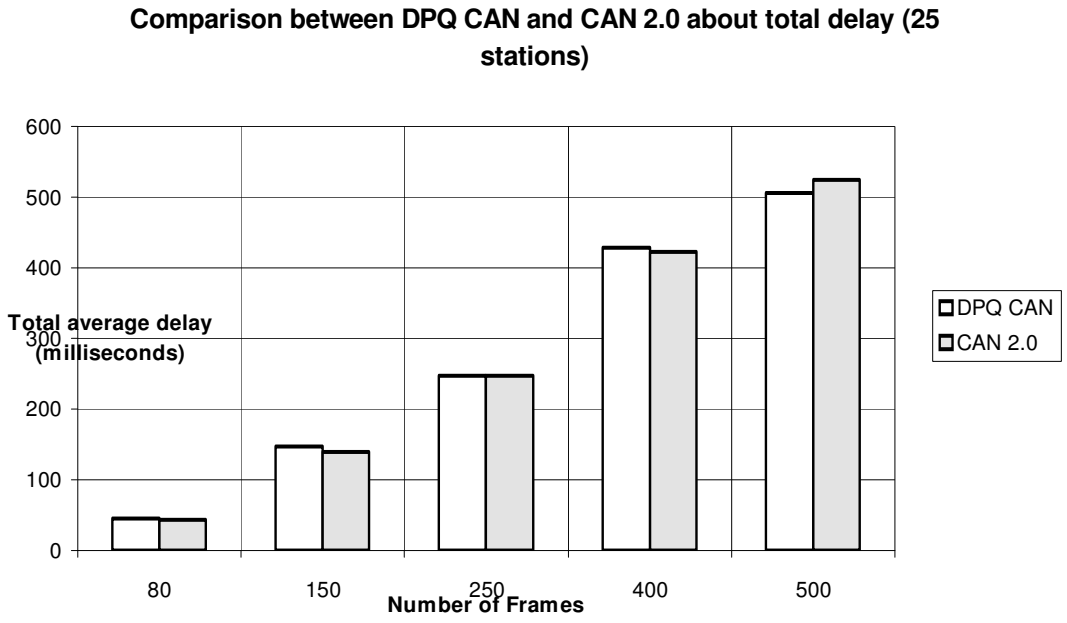

Grafico 3.

5.3.1 Distribuzione dei ritardi e ritardi di accesso medio.

Prendiamo in esame due delle situazioni descritte in precedenza dal punto di vista del ritardo totale, e analizziamole, adesso, secondo la distribuzione dei ritardi e il ritardo medio di ogni stazione trasmittente.

5.3.1.2 Prova 1 : 25 stazioni e 80 frame..

Consideriamo una rete CAN con 25 stazioni trasmittenti, 80 frame trasmessi ed una velocità di 1Mbit al secondo

Il grafico che segue mostra la distribuzione dei ritardi dei frame (high e low) riscontrati utilizzando il CAN 2.0. Osservando la curva relativa al frame ad High priority, vediamo come vi siano molti frame concentrati in alcuni intervalli e pochi in altri; ciò produce numerosi picchi.

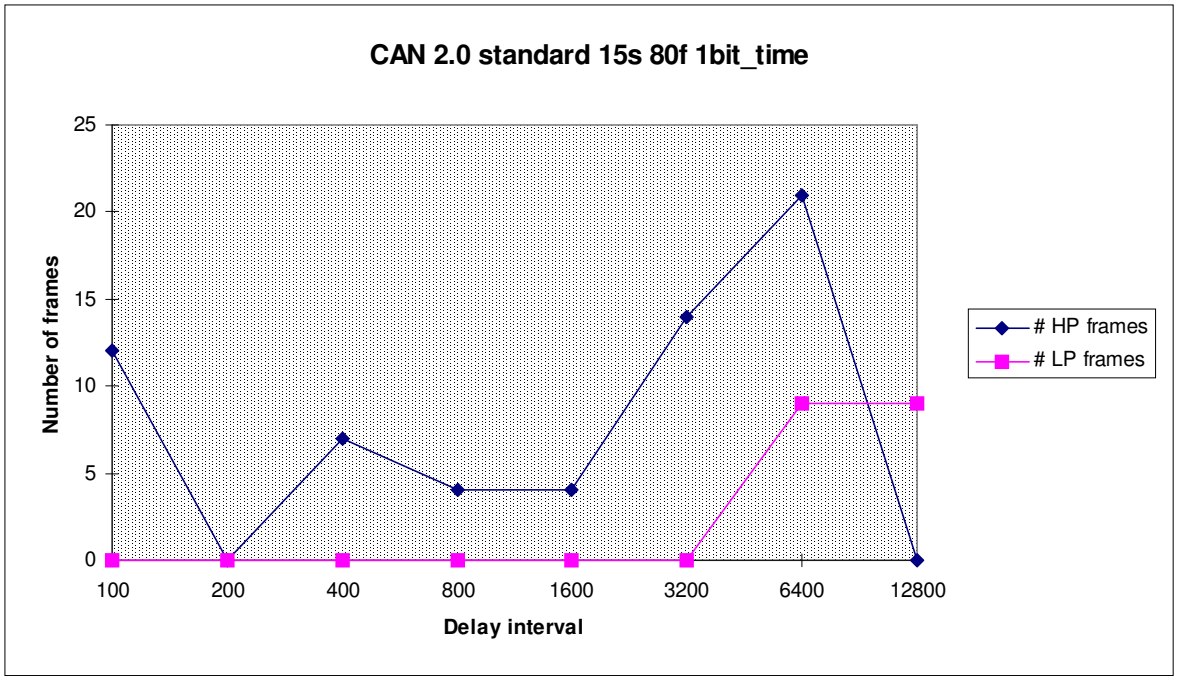

Grafico 2.1.

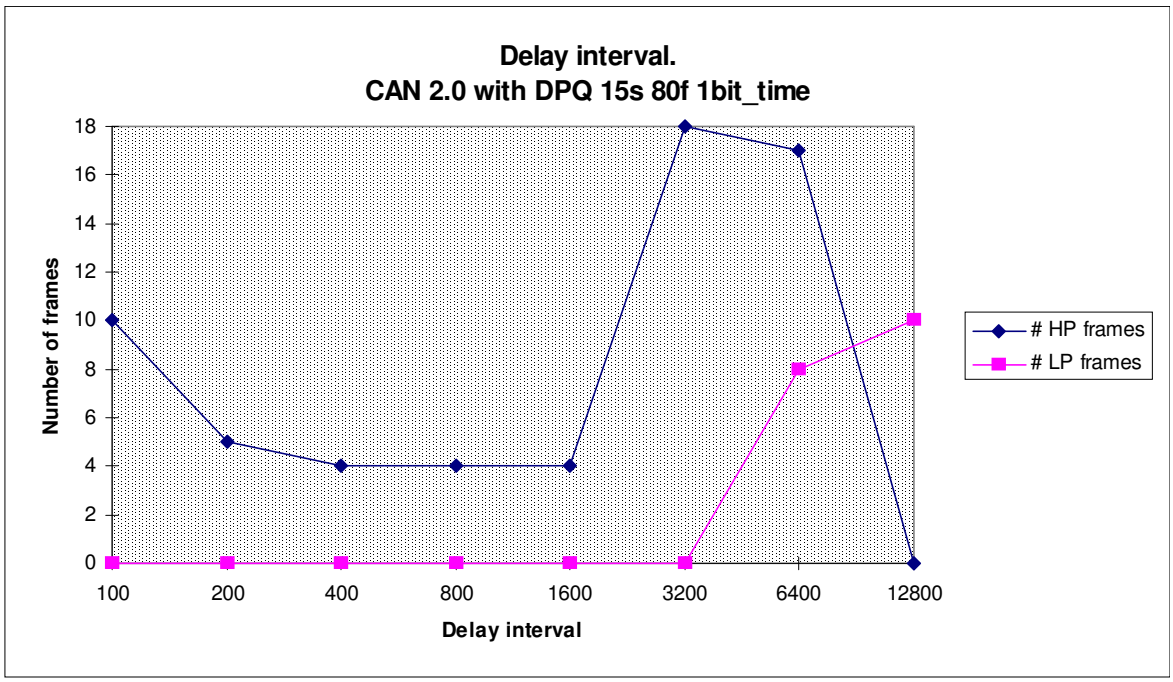

Grafico 2.2.

Il precedente grafico rappresenta l'equivalente al Grafico 2.1, dal punto di vista DPQ. Vediamo come non vi siano più quei picchi e i ritardi vengono distribuiti più uniformemente. Ciò che accade, quindi, è che non vi sono più alcune stazioni che riportano ritardi molto bassi ed altre che vanno incontro a lunghi periodi di attesa; invece, il CAN con DPQ, cerca di distribuire uniformemente i ritardi tra tutte le stazioni, evitando in tal modo la situazione descritta nel grafico 2.1.

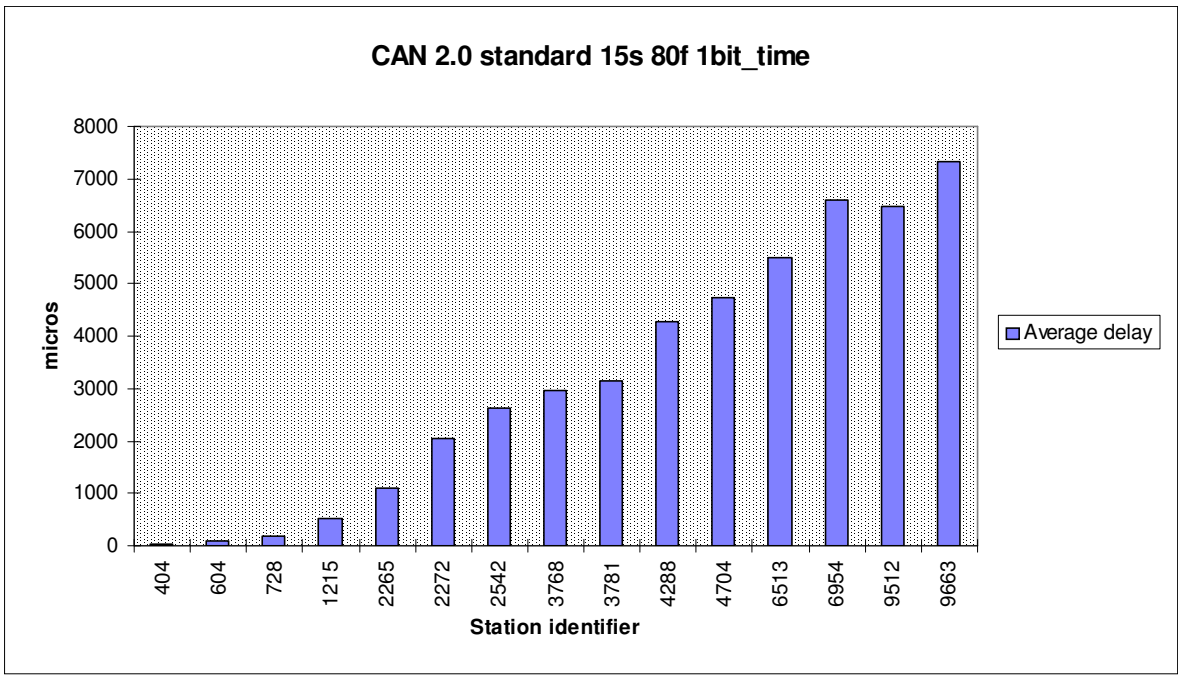

Grafico 2.3.

Il grafico 2.3 illustra, per ogni stazione, in ordine di priorità, il ritardo di accesso medio riscontrato. Rispettando la strategia CSMA/CD del CAN, si ha una variazione del ritardo medio che cresce al crescere del valore dell'identificatore.

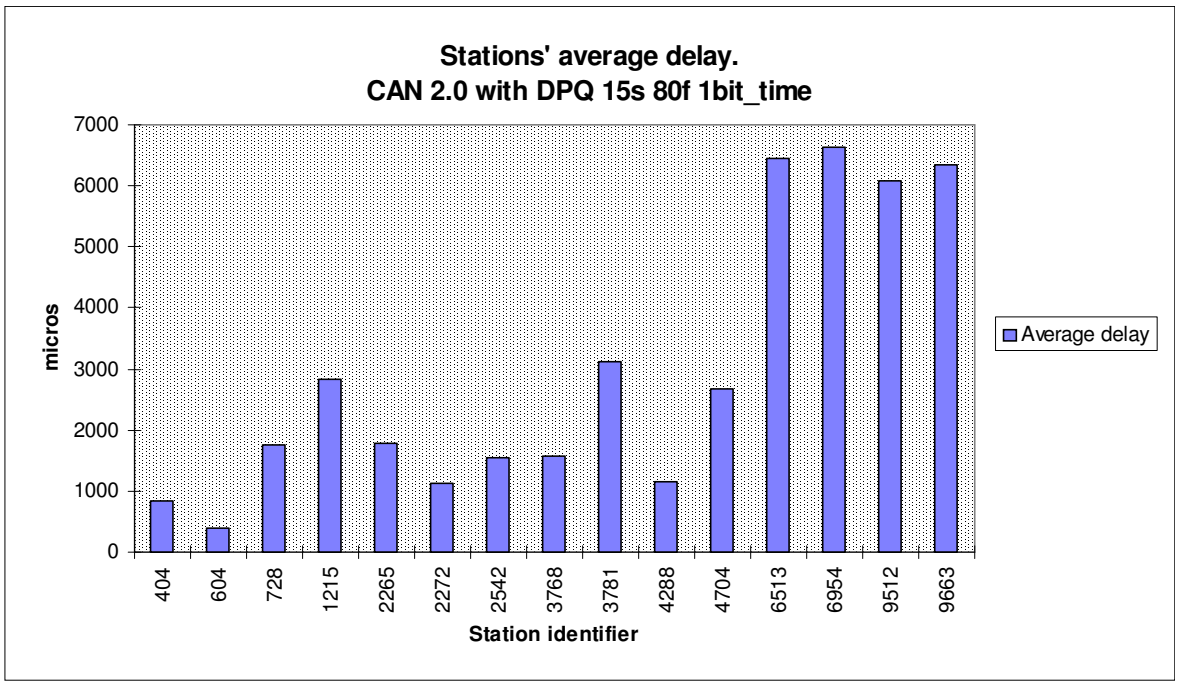

Grafico 2.4.

Il grafico 2.4, relativo al DPQ CAN, mostra invece il comportamento del meccanismo DPQ, che tende a suddividere uniformemente la banda tra tutte le stazioni. Ovviamente, rispettando la strategia DPQ, i ritardi non variano più al variare del valore dell'identificatore. In effetti si può notare, come il ritardo minore riportato con il DPQ CAN, si maggiore del ritardo più basso del CAN 2.0 e, analogamente, il ritardo più elevato con il DPQ CAN sia inferiore al corrispettivo con il CAN 2.0. Questa osservazione appare più evidente analizzando il grafico 2.5.

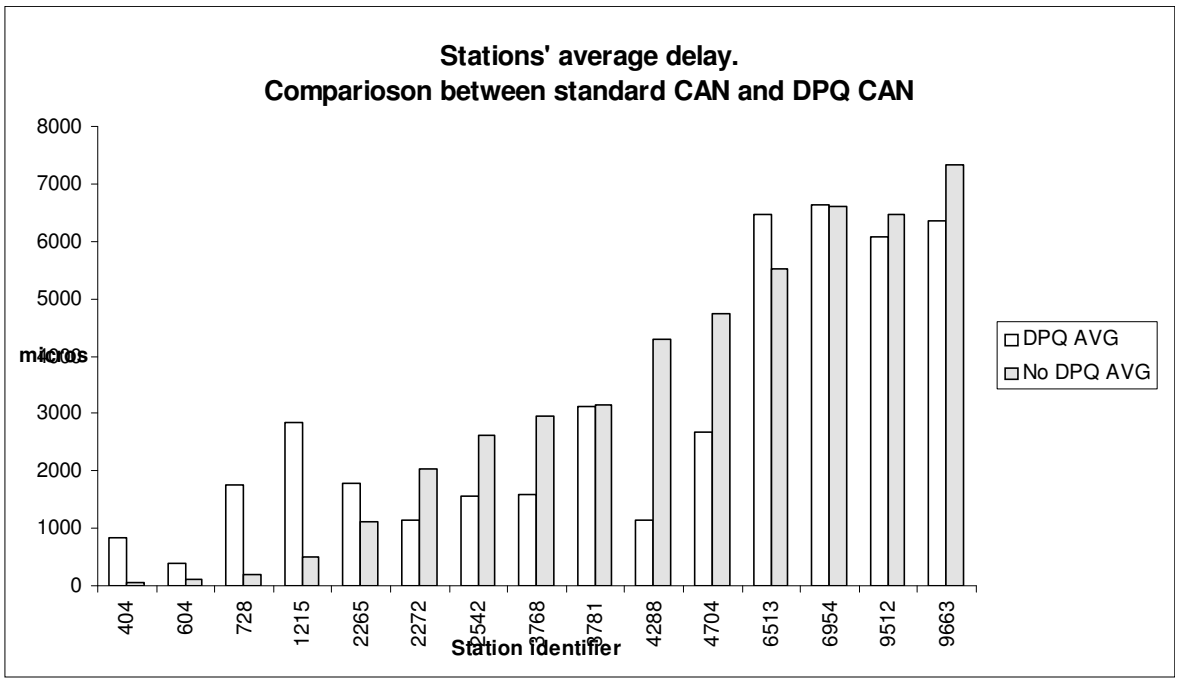

Grafico 2.5

5.3.1.3 Prova 2 : 200 stazioni e 400 frame

Consideriamo infine il caso un po' più particolare di 200 stazioni che trasmettono in totale 400 frame ad una velocità di 1 Mbit al secondo.

Le osservazioni fatte in precedenza vengono riconfermate anche con configurazioni di rete più complesse.

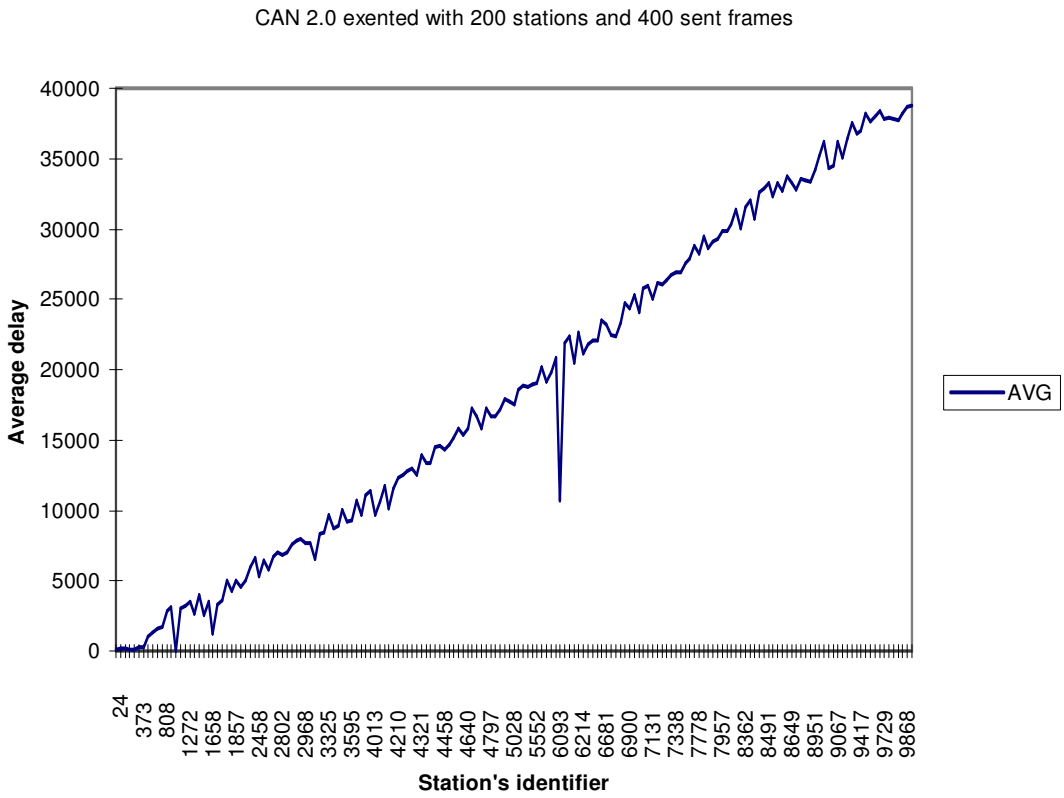

Grafico 3.1

Infine il Grafico 3.2 mostra la netta suddivisione nella manipolazione dei frame, ad alta e bassa priorità, effettuata dal DPQ CAN. Non si ha una funzione crescente come in precedenza, ma una funzione che tende ad essere constante all'interno di ciascun gruppo di priorità.

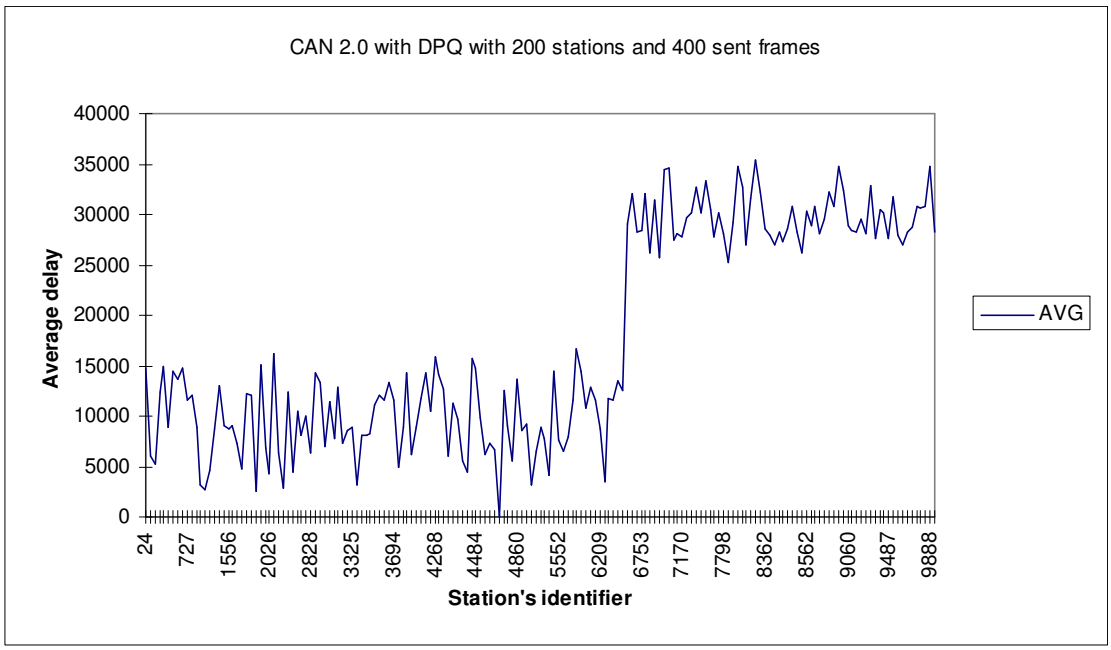
CAN 2.0 with DPQ with 200 stations and 400 sent frames
Average delay
40000
35000
30000
25000
20000
15000
10000
5000
0
24
727
1556
2026
2828
3325
3694
4268
4484
4860
5552
6209
6753
7170
7798
8362
8562
9060
9487
9888
Station's identifier
AVG

Grafico 3.2