Admission Control and Overload Handling in FTT-CAN

F. Bertozzi, M. Di Natale
Scuola Superiore S. Anna - Pisa, Italy
cisco@gandalf.sssup.it marco@sssup.it

L.Almeida
Universidade de Aveiro - Aveiro, Portugal
lda@det.ua.pt

Abstract

The paper presents new protocols for admission control and handling of overload conditions when scheduling real-time messages on CAN networks. The proposed solution builds incrementally on the FTT-CAN protocol, which provides guaranteed scheduling for cyclic (periodic) and aperiodic message streams as defined at design time and best effort scheduling for signalling (including admission requests) and scheduling of dynamically arriving message streams. FTT-CAN does not specify any particular admission control or overload management methods but provides mechanisms that support their implementation. This work exploits such mechanisms and provides the description of protocols related to admission control, guarantee of dynamic message streams, temporary overload management and fault containment. This work provides the description of protocols and implementation related issues that allow for guaranteed bandwidth and bounded response time for admission requests, and graceful recovery from temporary overload of event-based messages. The proposed approach has been implemented and validated by means of simulation.

1. Introduction

The CAN bus [1] is an industry standard for communication of computer- or microcontroller-based applications in cars, manufacturing plants and medical devices. Low cost, standardization, reliability and possibility of guaranteeing a worst case message transmission time by means of a predictable MAC-level protocol are among the reasons for its success. Predictable scheduling of real-time messages on the CAN bus is made possible by a number of scheduling algorithms. These can be classified according to the following:

- *Asynchronous scheduling*: these protocols typically use message identifiers to encode message priorities and exploit the standard contention resolution mechanism of CAN to implement priority-based scheduling. Both static and dynamic priority scheduling methods have been proposed. In [6] implementation and analysis of fixed priority (Rate Monotonic) scheduling is presented. Later works [8, 9] feature proposals for encoding dynamic (earliest deadline) priorities in message identifiers and analysing the timing correctness of the system.

- *Synchronous scheduling*: protocols belonging to this class divide time in cycles. Nodes can be statically assigned transmission slots in the context of each cycle by means of dispatch tables (as in TT-CAN [7]). Alternatively, a master node dynamically assigns the rights to transmit to each node (message) on a cycle by cycle basis. The scheduling strategy is implemented by the master node and communicated to the slave nodes at the beginning of each cycle [2, 4]. Methods belonging to the last subclass leave the opportunity for scheduling changes at run-time.

Schemes belonging to the second class have the practical advantage of allowing a strict separation between the allocation of IDs and the actual message scheduling performed at the MAC level. This separation brings along an extra flexibility and eases application development (according to recommendations and application developers' practice message IDs should be under the application designer's control).

FTT-CAN is among the best representatives of the second class. It enables guaranteed scheduling of periodic (time-based) and possibly aperiodic (event-based) message streams. The sheduling policy is implemented in the master node, which is responsible for allocating the transmission of individual message instances into the elementary cycle. FTT-CAN provides predictable scheduling of recurrent (i.e. periodic) and sporadic load when message streams can be exactly characterized at design time. On the other hand, guaranteeing dynamic load or handling requests for changes in either the periodic or aperiodic message flows requires adequate admission control and possibly overload management capabilities that, despite supported, are left unspecified so that they can be tailored later for specific classes of applications.

The additional flexibility that comes from supporting real-time periodic as well as non real-time traffic and the possibility of changing the timing requirements of real-time message streams or possibly even changing the set of active streams at any time is a desirable property for industrial systems. The most obvious requirement for changing the rate of real-time messages comes from mode changes,

0-7803-8734-1/04/$20.00 ©2004 IEEE.

such as those that occur when switching from normal environmental scanning to higher speed sampling because of the occurrence of an event (for example in target tracking applications). Furthermore, recent developments in control theory, such as logarithmic quantization, show how control applications may benefit from dynamic changes in the rate of the samples (hence the messages delivering them). Finally, in [10], the authors show how dynamic optimization of the quality of the controlled system response can be obtained by flexible control task timing constraints. Unfortunately, this flexibility often contrasts with the need for predictability and deterministic system behaviour.

This paper contributes to the combination of flexibility and predictability by proposing a set of methods for admission control and overload management that can be implemented within FTT-CAN. In particular, it contains description of:

- a new, simpler, schedulability test for synchronous and asynchronous message streams scheduled in FTT-CAN. The new schedulability formula can easily checked at run-time for guaranteeing admission to dynamic real-time streams;
- an admission control scheme which provides guaranteed bandwidth and a bounded access time to asynchronous admission requests sent to the master node without interfering with the scheduling of other messages;
- a protocol for recovery from a possible overload situation for soft-type real-time tasks.

The proposed solution is correct and stable under overload or selected fault conditions. The performance of the scheduling algorithm is evaluated by simulation on a number of randomly generated message sets based on the standard SAE Automotive Benchmark [6].

The next Section provides an introduction to our reference architecture, including the description of the problem and the definitions used throughout the paper. Section 2 also features an introduction to FTT-CAN. Section 3 contains a description of the standard mechanisms for guaranteeing real-time message streams in FTT-CAN. Sections 4 and 5 describe our admission control scheme and our new simpler schedulability test. Section 6 presents a protocol for recovery from overload situations. Finally, Section 7 contains the results of our experiments and the Conclusions section ends the paper.

2. Reference architecture

2.1. Introduction

The system architecture assumes a CAN bus with adapters implementing the standard arbitration protocol. The message traffic consists of both time- and event- triggered real-time messages. The time triggered traffic includes periodic message streams only, characterized by their period attribute. Event-triggered messages are characterized by a worst case interarrival time. We assume a centralized message management and scheduling executed by a *master node* that retains knowledge of all real-time messages and the corresponding allocated bandwidth. The configuration of message traffic is predefined at designed time, but it can be dynamically changed by adding or removing streams (periodic or aperiodic) such as in the case of a mode change or any other kind of dynamic request. Requests for dynamic modification of the real-time load must be guaranteed to arrive in a bounded time. We assume that message streams are divided into hard-type streams (periodic or sporadic), which are known at design time and should be guaranteed under any condition and firm-type streams, which can be dynamically requested and conditionally admitted into the system, provided their deadlines can be guaranteed by the admission control algorithm. Acceptance of hard-type streams is subject to an a-priori guarantee from the master node. Requests for additional periodic or sporadic streams can be guaranteed even at risk of temporary overloads. The system must react to an overload condition by adjusting the load from real-time messages.

2.2. Problem Definition

The terms and definitions used in the paper are the following:

$n_{\max}$ the maximum number of nodes in the network;
n_i the i-th node in the network;
N_s the number of real-time message streams;
N_{syn} the number of periodic (synchronous) message streams;
N_{asy} the number of sporadic (asynchronous) message streams.

A real-time message stream is further characterized by the following parameters:

P_i the period of consecutive messages in the i-th stream for time-based streams or
mit_i the minimum inter arrival time between any two consecutive instances of the i-th event-based stream;

D_i the deadline that applies to all the message of the i-th stream.

Attributes and parameters referring to the hard or soft message subset bring an additional superscript such as, for example, N^H_{syn} (number of hard-type synchronous streams.) Furthermore, as it is common in CAN-based applications, we require that the scheduling algorithm has minimal impact on the selection of the message identifiers by the application developer. Since our proposed solution is based on the FTT-CAN algorithm, the next section provides a short introduction to FTT-CAN.

2.3. FTT-CAN

The FTT-CAN protocol (please refer to [2] for a detailed description) divides time into an infinite sequence of Elementary Cycles (ECs) with fixed duration. Each elementary cycle is further divided into two phases (windows) where time- and event-triggered traffic is scheduled with temporal isolation. The first window is used to

transmit event-triggered or asynchronous messages. The second window is used to transmit time-triggered or synchronous message streams.

All nodes are synchronized at the start of each EC by the reception of a trigger message (TM), which is sent by a *master node*. The trigger message contains a bit mask where the i-th bit, if set, enables transmission of a periodic message from the i-th node inside the EC (master/multislave control mode). Collisions between (enabled) slave messages inside the EC are handled by the native distributed arbitration of CAN. Hence, the schedule issued by the master node only gives access rights to the EC but does not enforce the in-cycle transmission order (1). The event-triggered traffic is scheduled in its window by using the standard CAN arbitration mechanism. Slaves with pending aperiodic requests simply try to transmit immediately within the asynchronous phase of each elementary cycle and the master has no knowledge neither control of event-triggered requests.

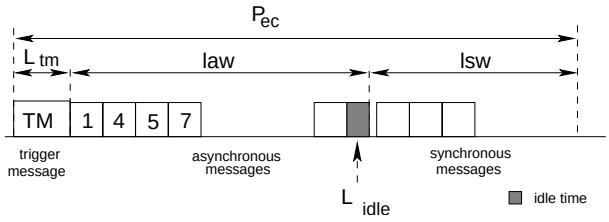

Figure 1. Elementary cycle in FTT-CAN.

Since the synchronous message window is placed at the end of the EC, the trigger message also conveys its relative starting instant. The asynchronous window fills the remaining cycle time. The reason why the asynchronous window precedes the synchronous one is twofold: giving nodes enough CPU time for decoding the EC trigger message and avoiding priority inversions among asynchronous messages. In fact, the beginning of the asynchronous window must be synchronized at the bit level in all nodes in order to continue with any hanging arbitration process from the previous EC without extra blocking (except one lower priority message when the message is first activated). Thus, the asynchronous window is open for transmission during the transmission of the TM, and all nodes enter into arbitration right after the TM, without any blocking. FTT-CAN reserves some identifier bits for its internal use: only the last 6 bits (out of 11) can be used as true message identifiers, the others are used for protocol management purposes (see Figure 2).

Type	*TX_ND*	*Message ID*	*Data field*
[0110]	[0/1]	[0..64]	Application dependent
CAN ID Field			CAN Data Field
b10...b7	b6	b5...b0	1 to 8 bytes

Figure 2. Identifier bits in the standard FTT-CAN proposal.

In detail, the first four bits encode the message type (for example, 0001 for the trigger message) and the fifth bit defines the message *liveness*. Figures 2 and 3 describe the bit content of the identifier field of all messages and the trigger message. The first row of the table is the name of the subfields; the second row defines the possible values for each field and the last row the order of the bits such as, for example, *b0* is the rightmost bit and *b10* is the eleventh bit of the identifier.

The meaning of the bits composing the trigger message (TM) is shown in Figure 3.

Type	*Master ID*	*New Plan*	*Sequence number*	*Synch. Window length*	*Message Trigger Field*
[0001]	[0..7]	[0/1]	[0..7]	[0..255]	Bitmap
CAN ID field				CAN Data Field	
b10...b7	b6...b4	b3	b2...b0	MSB	1 to 7 bytes

Figure 3. Trigger message in the standard FTT-CAN proposal.

The protocol allows for the definition of a maximum length for the synchronous windows (LSW) and correspondingly a maximum bandwidth for that type of traffic. Consequently, a minimum bandwidth can be guaranteed for the asynchronous traffic. The synchronous traffic is protected from the interference of asynchronous requests by preventing the start of transmissions that could not complete within the respective window. This is achieved by removing from the network controller transmission buffer any pending request that cannot be served up to completion within that interval. Consequently, a short amount of idle time may appear at the end of the asynchronous window. The policy adopted by the master to schedule periodic requests can be Rate Monotonic (RM), Earliest Deadline (EDF) [3] or possibly any other algorithm that allows for predictable scheduling of periodic requests. Sporadic messages are scheduled by the native MAC protocol of CAN. Therefore, fixed priority scheduling must be considered for the purposes of analysis. Further definitions are required for explaining schedulability analysis of messages in FTT-CAN. We assume (Figure 1):

- the transmission of the trigger message takes L_{tm} (constant) time units;
- the width of the synchronous window inside the n-th elementary cycle is $lsw(n)$ time units and
- the width of the asynchronous window is $law(n)$;
- L_{idle} is the worst case length of the idle time separating the transmission of asynchronous messages from synchronous messages (provably equal to the worst case message transmission time).
- P_{ec} the period of the elementary cycle;

3 Schedulability of real-time traffic in FTT-CAN

In [2] Almeida and Pedreira show how RM and EDF can be adapted in order to schedule periodic message traf-

fic in FTT-CAN. Periodic message streams can be guaranteed if:

$$\begin{array}{ll} \text{(RM)} & U = \Sigma_{i=1}^{N_s}(\frac{C_i}{P_i}) < N_s(2^{\frac{1}{N_s}} - 1)(\frac{L - L_{idle}}{P_{ec}}) \\ (EDF) & U = \Sigma_{i=1}^{N_s}(\frac{C_i}{P_i}) < (\frac{L - L_{idle}}{P_{ec}}) \end{array} \quad (1)$$

Furthermore, a timeline analysis approach allows an even more accurate schedulability assessment for fixed priorities scheduling. Given the timing attributes of asynchronous messages, it is possible to upper bound the worst case response time to event-based requests.

The schedulability analysis presented in [2] requires a priori knowledge of synchronous and asynchronous message parameters. The latest transmission time for an asynchronous message stream of index i is computed by evaluating the earliest time instant (from the critical instant t=0) when the cumulative bus demand of the asynchronous traffic $H_i(t)$ from higher priority streams equals the bus availability function (cumulative bus time available) for the asynchronous messages $A(t)$. The equation $H_i(t) = A(t)$ is solved iteratively by letting $t^0 = H_i(0)$ and $t^{m+1} = A^{-1}(H_i(t^m))$ until a value $t^{m+1} = t^m$ is found or t^{m+1} becomes greater than the limit value that allows the message to be transmitted before its deadline (please refer to [2] for the details). Scheduling of dynamically arriving asynchronous requests is only best-effort, since there is no mechanism to perform online admission control of this type of traffic.

The proposed formula requires evaluating an upper bound of the periodic message traffic inside each EC. Any change in the set of periodic messages implies recomputing the worst case response times of sporadic messages. Furthermore, the iterative schedulability formula requires non negligible computing times, preventing its use at runtime on inexpensive controllers. As a matter of fact, experimental figures show computation times of a few hundreds of ms for small 8-bit microcontrollers and hundreds of microseconds for an FPGA implementation. In low-end systems, the formula is clearly meant to be computed offline. If a dynamic admission scheme guaranteeing newly arriving requests for real-time streams is needed, then a new (simpler) analysis formula is required.

4. From admission control to overload recovery

The implementation of an admission control scheme on FTT-CAN requires a non-trivial set of subprotocols. First, we need to define a protocol that allows slave nodes to send control messages to the master node in a predictable way, that is, we provide a bounded worst case transmission time for slave requests without compromising the schedulability of the other messages.

Requests for dynamic changes in the real-time message streams cannot be guaranteed (in many systems) by using the timeline analysis in [2]. Hence, a new simpler (unfortunately also pessimistic) analysis is needed. Such an analysis is provided in subsection 5.2.

Finally, our admission control schemes allows transmission of new real-time streams or modification of the parameters of the existing streams as long as they do not affect schedulability of hard type streams. This may result in temporary overloads for non hard type streams. Therefore, an overload recovery mechanism is provided to let slave nodes drop lower priority messages that cannot be guaranteed or negotiate lower bandwidth requirements for some of the existing streams.

5. Admission control

The need for an admission control policy arises from the need of handling requests for addition and/or removal of message streams and from the need of changes in the time parameters of one or more streams. The admission control messages issued by slave nodes in the network is an event triggered traffic. In the original FTT-CAN proposal it is quite difficult (or impossible) to bound the transmission delay of control messages or their impact on the other asynchronous streams since an adequate transmission policy for control messages is simply missing. To this purpose, we propose a new scheme, which allows the transmission of only one control message per cycle in a round robin fashion, therefore introducing a possibly higher but bounded delay. The control message (*Ctrl*) is sent at the beginning of the elementary cycle, right after the trigger message (Figure 4). Its transmission time is bound by L_{ctrl}

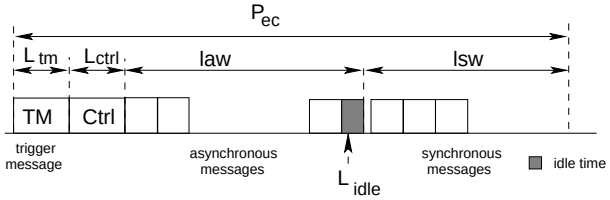

Figure 4. Elementary cycle in FTT-CAN with control message.

5.1. Transmission priority

For each elementary cycle, only one node is allowed to send a control message. The master node defines the priority rank for the transmission of the control messages in the current EC. Each node has a position in the rank and the highest ranked node with an outgoing control message gets the rights to transmit. The priority rank is encoded into the *node* field of the trigger message (Figure 6) by using the following rule: Each slave node n_i is assigned a unique identifier n_id_i ($n_{\max} \leq 64$ is the maximum number of nodes in the system). The master node selects each of the slave nodes at each round as the first node in the

rank in a round robin fashion and sends its identifier n_sel in the trigger message. The i-th slave node (n_i) computes the *node* subfield, hence the priority (identifier) of its outgoing control message according to the following rule:

$$node\_subf_i = |n\_id_i - n_{sel}|_{n_{\max}}$$

If we suppose for sake of clarity that $n\_id_i = i$, then the situation at the beginning of each round is represented in Figure 5. The ranking algorithm clearly works for any set of slave identifiers n_id_i.

control msg identifiers	cycle 0 $n_{sel} = 0$	cycle 1 $n_{sel} = 1$	cycle 2 $n_{sel} = 2$
n_id_0	0	4	3
n_id_1	1	0	4
n_id_2	2	1	0
n_id_3	3	2	1
n_id_4	4	3	2

Figure 5. Control message identifiers.

In a network with *n* slave nodes, each node can send a control message (at worst) every *n* elementary cycles. Control messages are sent in single shot transmission mode, this means that in case arbitration is lost, all nodes retrieve (i.e., do no retransmit) their control message.

Admission requests parameters are encoded in the fields of the control message as follows. The type of the stream (periodic/sporadic) is encoded in the *p/s* field. The request type (admit new stream, remove stream, modify stream attributes) is encoded in the 2 bits of the *op* field. The identifier of the (new, removed or modified) message stream is mapped in the corresponding *msg_ident* field. Finally, message attributes, such as length and period or worst case interarrival time, are encoded in the remaining bits. Please note that the message categories of Figure 6 include both hard and firm type messages in the same class (hard message must have higher priority identifiers).

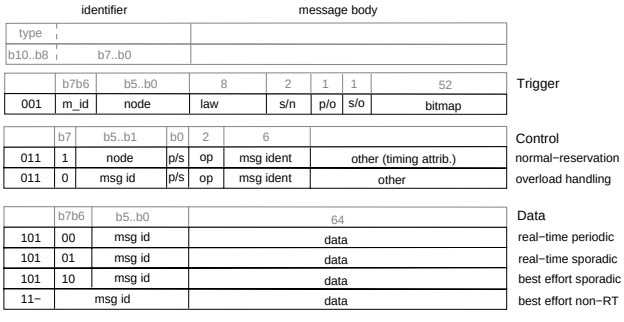

Figure 6. Message formats.

5.2 Schedulability analysis

In [2] the authors perform schedulability analysis of both periodic and aperiodic messages by exploiting timeline analysis on a cycle by cycle basis. This results in an iterative schedulability test, which can hardly be used at runtime. A new and faster admission test can easily be obtained by simplifying the schedulability analysis with the assumption that the worst case length of the periodic message schedule does not depend on the cycle instance. To this purpose, we assume the synchronous window to be of constant size LSW and the asynchronous window of a corresponding worst case size LAW. Such an assumption has been previously exploited for scheduling real-time messages in FIP buses by Pedro and Burns [5]. Our experiments will give an estimate of the additional pessimism introduced by this assumption in the FTT-CAN case.

We assume that Earliest Deadline (EDF) is used for scheduling periodic streams by the master node and sporadic streams (if admitted) are scheduled in the asynchronous window based on their identifiers, which implies a fixed priority scheduling mode. Both schedulability analysis and the overload management protocol have been defined based on these assumptions.

It is possible to compute the minimum width of the synchronous window LSW^H required for scheduling the N^H_{syn} hard type periodic messages by solving the EDF equation in (1) for the window size L as

$$LSW^H = P_{ec} * \Sigma_{i=1}^{N^H_{syn}} \frac{C_i}{P_i} + L_{idle}$$

The minimum lenght of the synchronous window that is required for scheduling all periodic messages is:

$$LSW = P_{ec} * \Sigma_{i=1}^{N_{syn}} \frac{C_i}{P_i} + L_{idle}$$

When considering the worst case interference from higher priority messages, it is possible to compute the minimum required length for the asynchronous window allowing the transmission of hard type traffic (LAW^H) and the minimum length for sending all asynchronous traffic before the deadlines.

LAW^H can be computed as follows. Suppose a message belonging to the i-th asynchonous stream becomes ready at $t = 0$. First, we compute the maximum number of elementary cycles α_i that can be safely awaited on the bus before the i-th asynchronous message is transmitted without violating its deadline.

$$\alpha_i = \left\lfloor \frac{D_i - C_i}{P_{ec}} \right\rfloor$$

The number of higher priority requests that can possibly prevent the transmission of the i-th message stream is given by all higher priority messages arriving in the interval $[-L_{tm} - L_{ctrl} - 2L_{idle} - LSW^H, \alpha_i P_{ec}]$, where the term L_{idle} needs to be doubled because asynchronous messages can be blocked (at most) once, for the time it takes to send a lower priority message (L_{idle} in the worst case). Therefore, the number of higher priority requests is

$$\left\lceil \frac{\alpha_i * P_{ec} + L_{tm} + L_{ctrl} + 2L_{idle} + LSW^H}{mit_j} \right\rceil \quad (2)$$

If these requests must be served in at most α_i cycles, then the minimum width of the asynchronous window is

$$LAW_i^H = \frac{\Sigma_{j \epsilon hp_i} \left\lceil \frac{\alpha_i * P_{ec} + L_{tm} + L_{ctrl} + 2L_{idle} + LSW^H}{mit_j} \right\rceil C_j}{\alpha_i} \quad (3)$$

The limit value LAW_i^H is obtained by considering the timing requirements of the i-th stream only. The worst case constant value of LAW that can possibly satisfy the requirements of all hard type streams is

$$LAW^H = \max_{i=1..N_{asyn}^H} (LAW_i^H)$$

If the idle interval is included in the expression of LAW, then it is

$$LAW^H = \max_{i=1..N_{asyn}^H} (LAW_i^H) + L_{idle}$$

The width of the asynchronous window for all aperiodic (hard and soft type) tasks can be approximated by an expression, which does not depend from LSW. In order to obtain a simple formula that can be quickly computed we need to eliminate the term LSW from formula (3). Since our objective is to find the guarantee condition assuming a constant size for the asynchronous and synchronous windows LAW and LSW, the LSW term can be easily eliminated:

$$LSW = P_{ec} - L_{tm} - L_{ctrl} - LAW$$

Furthermore, in order to allow solving the equation in closed form, we need to drop the ceiling expression from it. This can be easily obtained by maximizing equation 2 with

$$\frac{\alpha_i * P_{ec} + L_{tm} + L_{ctrl} + 2L_{idle} + LSW}{mit_j} + 1$$

and, substituting LSW

$$\frac{(\alpha_i + 1) * P_{ec} + 2L_{idle} - LAW + mit_j}{mit_j}$$

Solving equation 3 for LAW, after the substitutions that account for our pessimistic and simplifying assumptions, we obtain the admission test for each firm-type aperiodic messages stream i at run-time.

$$\forall i \; LAW_i = \frac{\Sigma_{j \epsilon hp_i} \frac{P_{ec}(\alpha_i + 1) + 2L_{idle} + mit_j}{mit_j} C_j}{\alpha_i + \Sigma_{j \epsilon hp_i} \frac{C_j}{mit_j}} \quad (4)$$

Hence, LAW can be obtained as

$$LAW = \max_{i=1..N_{asyn}} (LAW_i)$$

and again, if the idle interval is included in the expression of LAW then it is

$$LAW = \max_{i=1..N_{asyn}} (LAW_i) + L_{idle}$$

In order to guarantee all hard type messages it must always be (guaranteed offline):

$$LAW^H + LSW^H + L_{tm} + L_{ctrl} \leq P_{ec}$$

A similar test can be performed on-line in order to check if an overload condition is in progress for firm type message streams. If the requested synchronous and asynchronous windows for hard type messages can be guaranteed by the system, the corresponding values define the guaranteed synchronous and asynchronous windows as LSW^{gua} and LAW^{gua} respectively.

5.2.1 Admission rules

Considering the schedulability tests in the previous section, the proposed admission protocol can be outlined as follows. LAW^{gua} and LSW^{gua} are the guaranteed minimum width for the asynchronous and synchronous window respectively. Additional requests may arrive at runtime for both synchronous and asynchronous streams. If accepted, those requests increase the minimum width of both synchronous and asynchronous windows respectively at LSW^{req} and LAW^{req} (Figure 7). The new requirements may or may not be safely guaranteed.

1. when a periodic message request arrives, the corresponding new minimum synchronous window LSW^{req} and the available time for synchronous requests LSW^{avail}([1]) are computed. If $LSW^{req} > LSW^{avail}$ there is an overload situation on periodic messages;
2. if a new sporadic message arrives, LAW^{avail}([2]) is similarly computed. If $LAW^{req} > LAW^{avail}$ there is an overload situation in progress on sporadic messages;

6. Overload management

Dynamically admitting new firm type streams in the system and/or dynamically changing the time parameters of guaranteed streams, implies the possibility of an overload situation. Hence, an overload management protocol must necessarily complement our dynamic admission control strategy.

Our choice is to let slave nodes manage (and possibly recover from) the possible overload rather than having

[1] $LSW^{avail} = P_{ec} - L_{tm} - L_{ctrl} - LAW^{req}$

[2] $LAW^{avail} = P_{ec} - L_{tm} - L_{ctrl} - LSW^{req}$

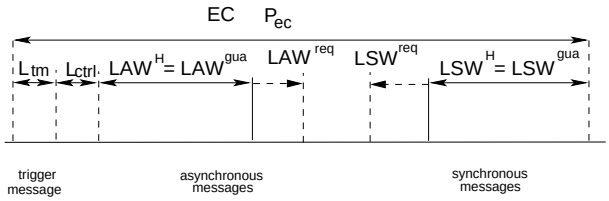

Figure 7. Extending the asynchronous and the synchronous windows beyond the guaranteed amount.

a centralized and possibly complex strategy for overload handling in the master. The master node is only responsible for signalling slaves of an existing overload situation by using the trigger message (*p/o* field for overload on periodic streams, *s/o* for overloads on sporadics).

The selection of the stream to be removed requires a simple agreement protocol among slave nodes. Once again, the standard CAN arbitration mechanism is exploited to this purpose. Slave nodes perform recovery from overload by removing some of their streams from the master tables. Control messages are used to signal the master the decision of dropping one of the streams sent by the slave. Among all the slaves that have firm-type streams, our algorithm selects the lowest priority stream that, once removed, can possibly restore schedulability.

Selection of the stream to be removed depends upon the type of overload, since periodic streams are scheduled according to an earliest deadline policy by the master and sporadic messages are scheduled according to their (static) priorities by the native CAN protocol.

If the system is experiencing an overload on periodic streams, then recovery from overload can be obtained by simply lowering the periodic load, starting from highest identifier messages (which is the simplest way for lowering the system load in caes of EDF scheduling). If, however, the overload occurs on sporadic messages, then it is necessary to compute the highest priority message that can possibly miss its deadline. Formula (4) is iteratively applied for increasing values of the priority index *i*, until the priority threshold i_{ov} is obtained. Since removing sporadic streams with priority lower than the computed threshold would not affect schedulability of the stream with priority i_{ov} (sporadic messages are scheduled with fixed priorities), the slave nodes must be informed of the lowest priority of the sporadic streams that must be considered for rejection. In this case, the trigger message that signals the occurrence of overload, also contains indication of the lower bound on the priority of the messages considered for rejection (*node* field).

The selection of the lowest priority stream is made by using control messages. As soon as the master node detects an overload and signals its occurrence with the trigger message, all the slaves with firm-type streams encode the lowest priority (higher than i_{ov} in case of an overload on sporadic messages) among their firm-type messages $low\_firm\_id_i$ in the message identifier of the next control message: The lowest message priority (highest identifier) is encoded in the *msg_id* subfield and results in the lowest control message identifier (highest CAN priority), which wins the contention.

$$msg\_id\_subf_i = n_{\max} - low\_firm\_id_i$$

If an overload is detected and signalled (by the master node) in the trigger message, then it makes no sense to allow control messages from slave nodes in the same elementary cycle. To prevent such messages, the master node sends a dummy control message, with priority higher than any other possible control message from the slaves, right after the trigger message (in the same elementary cycle - see also the example Figure 8). Please remember that control messages are always sent in single shot transmission mode (i.e. no retransmission on error or arbitration loss).

Removal of a single message might not be sufficient for recovery. After the recovery cycle, all slaves assume that the overload situation is still in progress and prepare for contention and removal of the next message by repeating the procedure in the following elementary cycle. If this assumption is correct, then the master node keeps the overload bit set in the following cycle and a new recovery iteration is performed.

However, it may also be that normal schedulability conditions are recovered after just one cycle or the priority threshold that slaves must consider for rejection is changed. In the first case, the overload bit is reset in the following trigger message and the master node sends the dummy preventing transmission of control messages from slaves. In case the overload is still active, but the priority threshold is changed, then the master keeps the overload bit set and signals the new priority threshold in the trigger. At this point, slave nodes need to recompute their control messages, based on the new priority threshold. Once again, an extra cycle is granted for this activity by sending a dummy control message. A simpler protocol is also possible. In this case, slave nodes assume normal schedulability is recovered after each cycle. It is the master responsibility to inform slaves in case the overload situation is still active, by keeping the overload bit set in the following cycle and sending the dummy control right after. In this case, one message is dropped every two elementary cycles, until a guaranteed scheduling for all firm-type messages is obtained. The first protocol has been selected for implementation in our experiments.

The following example explains the sequence of actions that take place in a sample recovery procedure. The CAN network consists of four nodes, S_1, S_2, S_4, S_6. Node S_1 outputs (periodic) message streams with identifiers 2,5,19,23; messages with identifiers 4,15 are sent by node S_2; 11,30 by node S_4 and 20,21,25 by node S_6.

The corresponding lowest priorities are: message with id=23 for S_1, id=15 for S_2, id=30 for S_4, and id=25 for S_6. If messages are labeled with the corresponding identifiers,

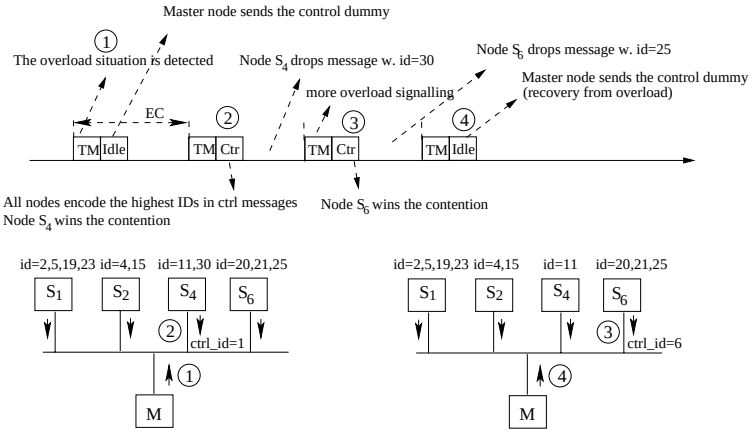

Figure 8. System overload management.

then $P_{m_{15}} > P_{m_{23}} > P_{m_{25}} > P_{m_{30}}$ (lower id means higher priority). If an overload situation is detected, all nodes try to transmit a control message that corresponds to an attempt at removing the corresponding stream: $msg\_id\_subf_1(m_{23}) = 9$, $msg\_id\_subf_2(m_{15}) = 17$, $msg\_id\_subf_4(m_{30}) = 2$, $msg\_id\_subf_6(m_{25}) = 7$. The situation is represented in Figure 8 where node S_4 wins the contention with the control message (characterized by a $msg\_id\_subf = 2$) corresponding to the stream with identifier 30 (lowest priority among all firm-type streams). Removal of message with $id = 30$ is not sufficient for recovery from overload, hence in the following elementary cycle the master keeps the overload bit set in the trigger message and the slave nodes send another set of control messages in order to determine the next message to be removed, with node s_6 winning the contention for message with $id = 25$, which is removed in the following cycle. After this contention round, the overload is cleared and the master signals recovery from overload by clearing the corresponding bit in the following trigger message and sending the dummy control message right after it.

7. Experiments

This section contains the results of two sets of experiments (performed by simulation on a purposely written software tool). The first set aims at showing the behavior of the algorithm in overload conditions, proving that it successfully restores a schedulable condition when requests for new timing attributes of messages or new message streams arrive. The second set of experiments provides an evaluation of the additional pessimism introduced in the schedulability analysis formula by the approximations that simplify its evaluation at run-time.

7.1 Handling overload conditions

The following experiments have been performed to show the behavior of our admission protocol in overload conditions. The first case features an application example where the minimum interarrival and deadline attributes for

id	size	mit/dline	alt. dline 1	alt. dline 2
1	135	32/5		
2	135	32/12	16/9	
3,4	135	32/12		
5	135	11/11		
6	135	20/11		
7,8,9	135	24/11		
10,12,13,14				
31,32,33,36 (*)	135	48/9	32/16	24/24

Table 1. Message set in the first example.

the newly arrived streams are subject to negotiation with the schedulability manager in the master node. The abstract message set (represented in Table 1) is loosely inspired by the set in the SAE benchmark. Message stream 2 and streams with identifiers from 10 to 33 feature alternate possible values for the minimum interarrival time and deadline attributes. Our admission control protocol takes advantage of this opportunity by admitting new streams with relaxed timing constraints and missing no deadline. In the experiment (Figure 9) the starting message set includes messages with identifiers from 1 to 9. The load of the system is progressively increased by letting message stream change their attributes (such as stream 2 at time 64) or new streams enter the system (message streams 12, 31, 32, 33 at time 96 and message streams 11, 13, 14, 36 at time 192).

As shown by the graph in Figure 9 where time is on the X-axis and the sum of the reciprocal of the deadlines, computed over all the messages in the set (an indication of the system load/criticality) is on the Y-axis, our algorithm negotiates relaxed interarrival/deadline constraints or rejects new streams altogether and keeps the system schedulable at all times. Please note how new streams are admitted one at a time, with a delay of (at least) one elementary cycle, because of the need for asking admission with a control message sent to the master node. The load of the system increases until the maximum guaranteed value (dotted line) is obtained. On the contrary, if the streams had been just sent in a FTT-CAN implementation without admission control, 27 deadline miss would have

id	size	mit/dline	arrival time
1	135	32/5	
2	135	16/9	
3,4	135	32/12	
5	135	11/11	
6,7,11,14	135	24/9	96
20(*)	135	20/11	
21,25,26(*)	135	24/11	
22,30(*)	135	24/11	192
27(*)	135	24/9	192

Table 2. Message set in the second example.

occurred.

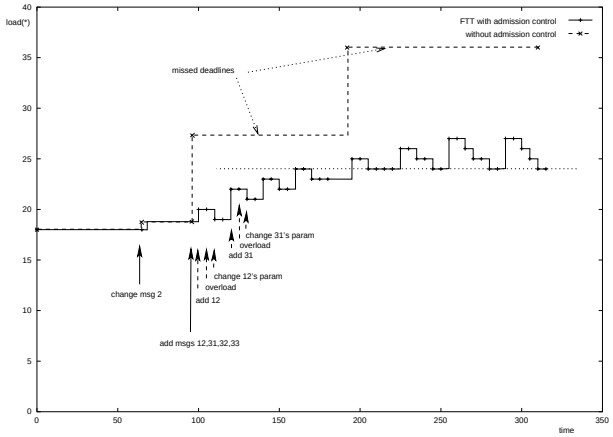

Figure 9. Handling overloads caused by changes in the timing attributes of messages.

In the second experiment (Table 2 and Figure 10), we tested the behavior of the admission control scheme against the arrival of new streams with fixed timing attributes. Message streams 1 to 5 and 20, 21, 25, 26 were assumed as the base load and new message streams were added at time 96 and 192. In this case there was no possibility of adjusting the timing oparameters, therefore, rejected streams were simply dropped from the system. Once again, the graph shows how the algorithm keeps the system load under the guarantee condition (at the price of some pessimism). Had the messages been sent without an admission test, 15 deadline miss would have occurred after time t=96 units.

7.2. Evaluation of the pessimistic assumptions

We performed four sets of experiments comparing our algorithm with FTT-CAN without admission control in order to evaluate the additional pessimism introduced in the guarantee analysis by the approximations in our method. All message sets have been constructed by varying the average bus utilization U defined as $U = C_i/T_i$. The first two sets have been constructed as follows:

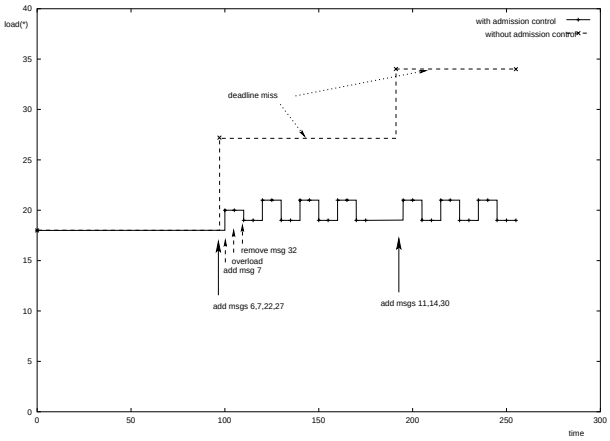

Figure 10. Handling overloads caused by new message streams.

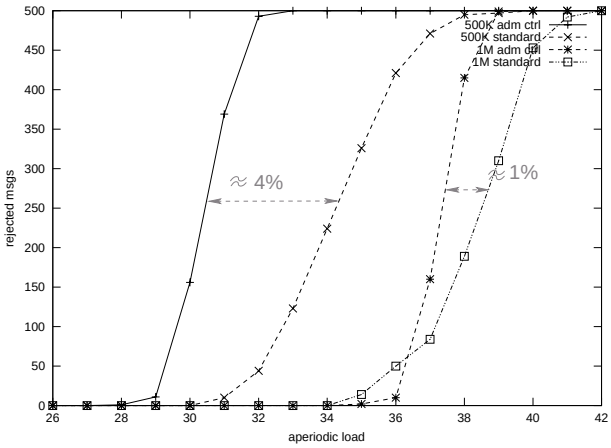

Figure 11. Overload management: synchronous load=30%.

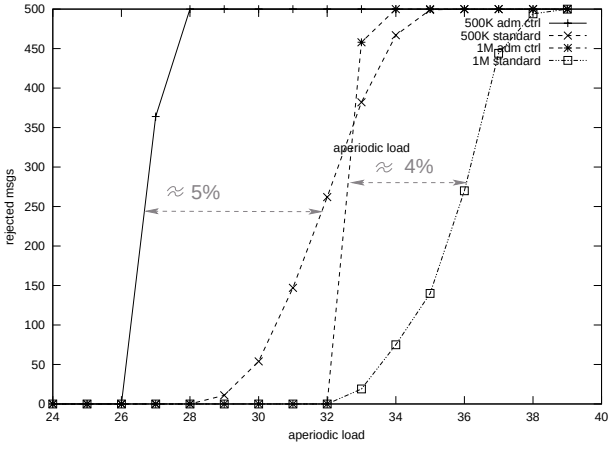

Figure 12. System overload management.

- The periodic load consists of a set of periodic streams with deadlines equal to the period. This load amounts for a bus utilization U_{syn}. The synchronous utilization is $U_{syn} = 0.3$ and the asynchronous utilization U_{asy} (plotted on the X axis) goes from 0.2 to 0.5 with steps of 0.05. Message periods are randomly chosen between 2 and 10 elementary cycles for the first graph (Figure 11) and between 11 and 20 EC for the second graph (Figure 12). The transmission times have been randomly assigned by selecting a message size between 55 and 135 bits (except for the last message, which was truncated at the value resulting in the desired U). For each step 500 experiments have been performed;
- the message set is ideally scheduled by EDF (deadline ordering);
- the limit value for LAW is computed by using our approximate (on-line) algorithm and the exact algorithm defined in [2];
- a sporadic set is generated with deadline equal to the minimum interarrival time and utilization U_{asy};
- the asynchronous set is tested with the standard admission control algorithm (labelled as standard) and our algorithm (labelled as dynamic) checking if the set is schedulable.

The third graph (Figure 13) has been obtained by only changing the synchronous utilization and setting it to $U_{syn} = 0.4$ (periods between 2 and 10 elementary cycles, similar results have been obtained for the case of longer periods). As shown by the graphs, the impact of using a simpler and more pessimistic admission control is relatively small, ranging from 1% in the best case to about 7% of less asynchronous utilization in the worst case. On the other hand, the benefits in terms of execution time (of the admission algorithm) are substantial, to the point of allowing their execution on-line, in low-end micro-controllers. The pessimism is lower for higher bit rates, which can be explained by the fact that, for the same value of P_{ec}, the relative proportion of message durations, and consequently of L_{tm}, L_{idle} and L_{ctrl}, are lower, leading to reduced pessimism.

8 Conclusions

The paper presents a new protocol for admission control of firm type messages and handling of overload conditions, based on the FTT-CAN proposal. The admission control protocol is based on a worst-case message guarantee condition that allows for a fast implementation on inexpensive microcontrollers. Together with an overload recovery scheme, it allows for guaranteed bandwidth and bounded response time for admission requests, and graceful recovery from temporary overload of event-based messages. The proposed approach has been implemented and validated by means of experiments showing the correct behavior of the approach in spite of some pessimism (between 1% and 7% in the utilization of asynchronous messages.)

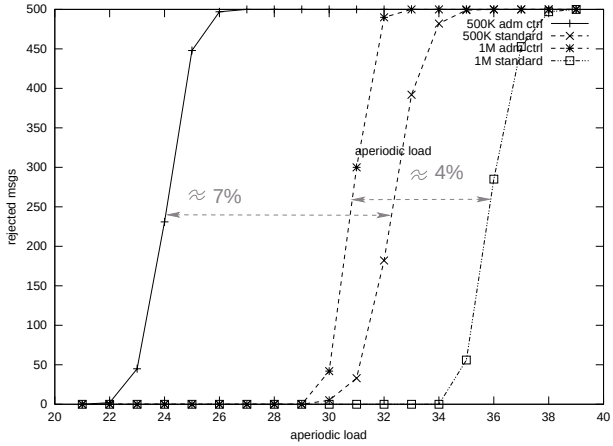

Figure 13. System overload management.

References

[1] Robert Bosch GmbH. CAN Specification Version 2.0, September 1991.

[2] L. Almeida, P. Pedreiras, J.A. Fonseca. "The FTT-Can Protocol: Why and How". *IEEE Transaction on Industrial Electronics, Vol. 49, No. 6, December 2001, pages 1189-1201.*

[3] C.L. Liu and J. Layland, Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment, J. Assoc. Comput. Machinery 10(1), 174-189 (1973)

[4] T. Nolte, M. Nolin, H. Hansson. "Server-based Scheduling of the CAN bus", *in 9th IEEE International Conference on Emerging Technologies and Factory Automation*, Lisbon, Portugal , September 2003.

[5] P. Pedro, A. Burns "Worst Case Response Time Analysis of Hard Real-Time Sporadic Traffic in FIP Networks", in Proceedings of 9th Euromicro Workshop on Real-time Systems, Toledo, Spain, pp. 5-12, 1997

[6] K. Tindell and A. Burns, "Guaranteeing Message Latencies in Controller Area Networks", in Proceedings of the 1st International CAN Conference, Maiz, Germany, Sept. 1994

[7] ISO/WD11898-4. Road Vehicles - Controller Area Network (CAN) - Part 4: Time-Triggered Communication, December 2000.

[8] M. Di Natale, "Scheduling the CAN Bus with Earliest Deadline Techniques". Proceedings of the IEEE Real-Time Systems Symposium. 2000.

[9] Zuberi, Khawar, Kang Shin. "Scheduling Messages on Controller Area Network for Real-Time CIM Applications". IEEE Transactions on Robotics and Automation, vol.13. 1997

[10] P. Mart, J. M. Fuertes, G. Fohler, K. Ramamritham: "Improving Quality-of-Control Using Flexible Timing Constraints: Metric and Scheduling Issues." IEEE Real-Time Systems Symposium 2002: