SERVER-BASED REAL-TIME COMMUNICATION ON CAN

Thomas Nolte, Mikael Nolin, and Hans Hansson

Mälardalen Real-Time Research Centre
Deptartment of Computer Science and Engineering
Mälardalen University, Västerås, SWEDEN
Email: {thomas.nolte, mikael.nolin, hans.hansson}@mdh.se

Abstract: This paper presents a share-driven scheduling protocol for networks with real-time properties. The protocol provides fairness and bandwidth isolation among predictable as well as unpredictable streams of messages on the network. The need for this kind of scheduled real-time communication network is high in applications that have requirements on flexibility, both during development for assigning communication bandwidth to different applications, and during run-time to facilitate dynamic addition and removal of system components.
The share-driven scheduling protocol is illustrated by applying it to the popular Controller Area Network (CAN). Two versions of the protocol are presented together with their associated timing analysis.

Keywords: fieldbus networks, real-time, scheduling, CAN, controller area network

1. INTRODUCTION

In optimising the design of a real-time communications system it is important to both guarantee the timeliness of periodic messages and to minimize the interference from periodic traffic on the transmission of aperiodic messages. Therefore, the usage of *server-based scheduling techniques* is proposed to handle streams of both periodic and aperiodic traffic. The servers proposed in this paper are using the Earliest Deadline First (EDF) scheduling strategy, since EDF allows optimal resource utilisation. Examples of such server-based scheduling techniques are the Total Bandwidth Server (TBS) (Spuri and Buttazzo, 1994; Spuri *et al.*, 1995), and the Constant Bandwidth Server (CBS) (Abeni, 1998).

In this paper, server-based scheduling is applied to the Controller Area Network (CAN) (CAN, 1993), which is one of the more common communication networks used today in the automotive industry as well as in several other application domains that have real-time requirements. CAN is implementing fixed-priority scheduling using a bit-wise arbitration mechanism in the Medium Access Control (MAC) layer. This mechanism avoids collisions in a deterministic way and is amenable to timing analysis (Tindell *et al.*, 1995).

Using server-based scheduling techniques for CAN, fairness among users of the network is guaranteed (e.g., "misbehaving" aperiodic processes cannot aversively interfere with well-behaved processes). This is desirable in, e.g., an open architecture where the adding and removing of network users is controlled by an online admission control. In contrast with other proposals for CAN scheduling, aperiodic messages are not sent "in the background" of periodic messages or in separate time-slots (Pedreiras and Almeida, 2000). Instead, aperiodic and periodic messages are jointly scheduled using servers. This substantially facilitates meeting response-time requirements, both for aperiodic and periodic messages.

Share-driven communication on CAN is presented previously (Nolte *et al.*, 2003). The main contributions of this paper are:

- A worst-case response-time analysis for the previously proposed protocol, bounding the maximum delay for a server to send a message.
- An improvement of the previously proposed protocol that, under many network configurations, can give lower worst-case response-times.
- A worst-case response-time analysis for the improved protocol.
- A comparison of the worst-case response-times of the two protocols.

The paper is organized as follows: Section 2 presents related work. In Section 3, server-based CAN is presented together with its corresponding timing analysis, and in Section 4 the modified mechanism is presented together with its corresponding timing analysis. Finally, conclusions are presented in Section 5.

2. BACKGROUND AND RELATED WORK

In this section, CAN and previously proposed CAN scheduling methods are presented.

2.1 The Controller Area Network

The Controller Area Network (CAN) (CAN, 1993) is a broadcast bus designed to operate at speeds of up to 1Mbps. CAN is extensively used in automotive systems, as well as in other applications.

CAN is a collision-avoidance broadcast bus, which uses deterministic collision resolution to control access to the bus (so called CSMA/CA). The frame identifier is required to be unique, in the sense that two simultaneously active frames originating from different sources must have distinct identifiers. Besides identifying the frame, the identifier serves two purposes: (1) assigning a priority to the frame, and (2) enabling receivers to filter frames.

CAN guarantees that the highest priority active frame will be transmitted. Hence, CAN behaves like a priority-based queue.

2.2 Scheduling on CAN

Several different types of scheduling methods exists for CAN, where FPS scheduling is the most natural scheduling method since it is supported by the CAN protocol. Response-time tests for determining the schedulability of CAN frames under FPS have been presented (Tindell *et al.*, 1995). This analysis is based on the standard fixed-priority response-time analysis for CPU scheduling (Audsley *et al.*, 1993). TT-CAN (TT-CAN, 2000) provides *time-driven* scheduling for CAN, and Flexible Time-Triggered CAN (FTT-CAN) (Almeida *et al.*, 1999) supports priority-driven scheduling in combination with time-driven scheduling. FTT-CAN is presented in more detail below. The share-driven approach has for CAN been introduced earlier (Nolte *et al.*, 2003). Later, in section 3 and 4, server-based scheduling of CAN is presented in detail.

2.2.1. Flexible Time-Triggered Scheduling To combine event-triggered traffic with time-triggered traffic in CAN, FTT-CAN (Flexible Time-Triggered communication on CAN) is proposed (Almeida *et al.*, 1999; Pedreiras and Almeida, 2000). In FTT-CAN, time is partitioned into Elementary Cycles (ECs) which are initiated by a special message, the Trigger Message (TM). This message contains the schedule for the synchronous traffic that shall be sent within this EC. The schedule is calculated and sent by a master node. FTT-CAN supports both periodic and aperiodic traffic by dividing the EC in two parts. In the first part, the asynchronous window, a (possibly empty) set of aperiodic messages are sent using CAN's native arbitration mechanism. In the second part, the synchronous window, traffic is sent according to the schedule delivered in the TM.

2.2.2. EDF Scheduling Several suggestions for scheduling CAN using EDF have been presented (Livani *et al.*, 1998; Natale, 2000; Zuberi and Shin, 1995). These solutions are all based on manipulating the identifier of the CAN frame to reflect the deadline of the frame, and thus they reduce the number of possible identifiers to be used by the system designers. Restricting the use of identifiers is often not an attractive alternative, since it interferes with other design activities, and is even sometimes in conflict with adopted standards and recommendations (CiA, 1996; J1938, 1998). FTT-CAN can also schedule messages according to EDF, using the synchronous window.

3. SERVER-BASED SCHEDULING ON CAN

In order to provide guaranteed network bandwidth for both synchronous and asynchronous real-time messages in CAN, server-based scheduling techniques provide a solution.

Server-based scheduling of CAN has been presented earlier (Nolte and Lin, 2002; Nolte *et al.*, 2003). However, in this paper response-time analysis is presented for (Nolte *et al.*, 2003) together with modifications that, for some system configurations, limits the worst-case response-time while retaining fairness. Furthermore, equations that bound the worst-case response-time for each server are presented.

Using servers, the whole network will be scheduled as a single resource, providing bandwidth isolation as well as fairness among the users of the servers. However, in order to make server-scheduling decisions, the scheduler must have information on when messages are arriving at the different nodes in the system, so that it can assign them a deadline based on the server policy in use. This information should not be sent to the scheduler using message passing, since this would further reduce the already low bandwidth offered by

CAN. The method, presented below, will provide a solution to this.

3.1 Server Scheduling (N-Servers)

In real-time scheduling, a *server* is a conceptual entity that controls the access to some shared resource. Sometimes multiple servers are associated with a single resource. For instance, in this paper multiple servers are mediating access to a single CAN bus.

The server used in this paper is a simplified version of the Total Bandwidth Server (TBS) (Spuri and Buttazzo, 1994; Spuri *et al.*, 1995).

3.1.1. Server Characterization Each node on the CAN bus is assigned one or more *network servers* (N-Servers). Each N-Server, s, is characterized by its period T_s, and it is allowed to send one message every server period. The message length is allowed to be of worst-case size. A server is also associated with a relative deadline $D_s = T_s$. At any time, a server may also be associated with an absolute deadline d_s, denoting the next actual deadline for the server. The server deadlines are used for scheduling purposes only, and are not to be confused with any deadline associated with a particular message. (For instance, the scheduling method presented below will under certain circumstances *miss* the server deadline. However, this does not necessarily make the system unschedulable.)

3.1.2. Server State The state of a server s is expressed by its absolute deadline d_s and whether the server is *active* or *idle*. The rules for updating the server state are as follows:

(1) When an *idle* server receives message n at time r_n it becomes *active* and the server deadline is set so that

$$d_s^n = \max(r_n + D_s, d_s^{n-1})$$

where $d_s^0 = 0$.

(2) When an *active* server sends a message and still has more messages to send, the server deadline is updated according to

$$d_s^n = d_s^{n-1} + D_s$$

(3) When an *active* server sends a message and has no more messages to send, the server becomes *idle*.

3.2 Medium Access (M-Server)

The native medium access method in CAN is strictly priority-based. Hence, it is not very useful when scheduling the network with servers. Instead a *master server* (M-Server) is introduced, which is a piece of software executing on one of the network nodes. Scheduling the CAN bus using a dedicated “master” has been previously proposed (Almeida *et al.*, 1999; Pedreiras and Almeida, 2000). In this paper, the M-Server has two main responsibilities:

(1) Keep track of the state of each N-Server.

(2) Allocate bandwidth to N-Servers.

The first responsibility is handled by *guessing* whether or not N-Servers have messages to send. The initial guess is to assume that each N-Server has a message to send (i.e., initially each N-Server s is assigned a deadline $d_s = D_s$). How to handle erroneous guesses is shown later.

The N-Servers’ complete state is contained within the M-Server. Hence, the code in the other nodes does not maintain N-Server states. The code in the nodes only has to keep track of when bandwidth is allocated to them (as communicated by the M-Server).

The M-Server divides time into Elementary Cycles (ECs), similar to the FTT-CAN approach presented in Section 2.2.1. T_{EC} denotes the nominal length of an EC. Moreover, T_{EC} is the temporal resolution of the resource scheduled by the servers, in the sense that N-Servers can not have their periods shorter than T_{EC}. When scheduling a new EC, the M-Server will (using the EDF principle based on the N-Servers’ deadline) select the N-Servers that are allowed to send messages in the EC. Next, the M-Server sends a Trigger Message (TM). The TM contains information on which N-Servers that are allowed to send one message during the EC. Upon reception of a TM, the N-Servers allowed to send a message will enqueue a message in their CAN controllers. The messages of the EC will then be sent using CAN’s native priority access protocol. Due to the arbitration mechanism, it is unknown when inside an EC a specific message is sent. Hence, the bound on the delay of message transmissions will be a multiple of the length of an EC.

Once the TM has been sent, the M-Server has to determine when the bus is idle again, so the start of a new EC can be initiated. The M-Server does this by sending a stop[1] message (STOP) with the lowest possible priority. After sending the STOP message to the CAN controller, the M-Server reads all messages sent on the bus. When it reads the STOP message it knows that all N-Servers have sent their messages[2].

After reading the STOP message the EC is over and the M-Server has two tasks to complete before starting the next EC:

(1) Update the state of the N-Servers scheduled during the EC.

[1] A small delay before sending STOP is required. This message is not allowed to be sent before the other nodes have both processed the TM (in order to find out whether they are allowed to send or not), and (if they are allowed to send) enqueued the corresponding message.

[2] Another way of determining when the EC is finished would be that the CAN controller itself is able to determine when the bus becomes idle. If this is possible, there is no need for the STOP message. However, using a STOP message is suitable with standard CAN controllers.

(2) Decide how to reclaim the unused bandwidth (if any) during the EC.

The following two sections describe how these tasks are solved.

3.2.1. Updating N-Server States By observing the actual transmissions during an EC it is possible for the M-Server to verify whether or not its guess that N-Servers had messages to send was correct, and to update the N-Servers' state accordingly. For each N-Server that was allocated a message in the EC two cases exists:

(1) The N-Server sent a message. In this case the guess was correct and the M-Servers next guess is that the N-Server has more messages to send. Hence, the N-Server's state us updated according to rule 2 in Section 3.1.2.
(2) The N-Server did not send a message. In this case the guess was incorrect since the N-Server was idle. The new guess is that a message now has arrived to the N-Server, and the N-Server state is set according to rule 1 in Section 3.1.2.

3.2.2. Reclaiming Unused Bandwidth It is likely that not all bandwidth allocated to the EC has been completely used. There are three sources for unused bandwidth (slack):

(1) An N-Server that was allowed to send a message during the EC did not have any message to send.
(2) One or more messages that were sent was shorter than the assumed worst-case length of a CAN message.
(3) The bit-stuffing that took place was less than the worst-case scenario.

To not reclaim the unused bandwidth would make the M-Server's guessing approach of always assuming that N-Servers have messages to send extremely inefficient. Hence, a mechanism to reclaim this bandwidth is needed.

In the case that the EC ends early (i.e., due to unused bandwidth) the M-Server reclaims the unused bandwidth by reading the STOP message and immediately initiating the next EC to make sure that no bandwidth is lost.

3.3 Timing Analysis

One property of the above method is that the protocol overhead increases when the network utilisation is low. When the network is not fully utilised, the M-Server will send more TM and STOP message than it would during full network utilisation. This is illustrated in Fig.1, where Fig.1(a) shows a fully utilised network where all servers that were allowed to send a message did so. Fig.1(b) shows what happens if not all servers send messages. Note the increase of TM and STOP messages. In fact, the proposed method will always consume all bandwidth of the network, regardless of the number of server messages actually sent.

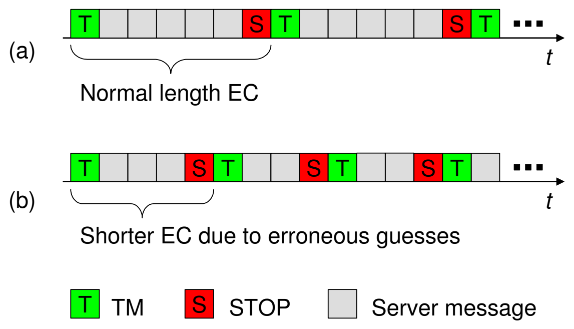

Fig. 1. Erroneous guesses increases the protocol overhead.

This increase in overhead when no N-Server messages are sent makes the analysis more complicated than in the original TBS, and the worst-case analysis of TBS cannot be reused.

As the scheduling mechanism used in this paper is based on EDF, looking at an N-Server x, the server might have to wait for other servers to be scheduled before it will be scheduled. Since the scheduling is completely based on guessed server-deadlines, x might be the only server in the system having messages to send, yet x may have to wait for all other servers to be scheduled due to their earlier deadlines. In order to derive an upper bound on the worst possible delay x can experience (setting time to 0 when a message arrives to x and its deadline is set to D_x) the other N-Servers in the system are put into two sets:

- A – all servers with periods shorter than x's:

$$A = \{s : D_s < D_x\}$$

- B – all servers with periods equal to or longer than x's:

$$B = \{s : s \neq x \wedge D_s \geq D_x\}$$

If the number of servers in A (denoted by $|A|$) is greater than the number of messages fitting in one EC, x may potentially not be scheduled for the duration of its deadline. For instance, if servers in A have no messages to send they will still be scheduled for message transmission for each EC until their deadline becomes greater than D_x.

Hence, N-Server x may not be scheduled until the EC which overlaps D_x. Since, at this point each of the other N-Servers (in sets A and B) may have deadlines arbitrary close to (and before) D_x, x may have to wait for each of the other servers to send one (1) message. No server will after time D_x be scheduled twice, since the first time a server is scheduled (after D_x) its new deadline will be set to a time greater than D_x. Hence, the message from server x may in the worst-case be the $|A| + |B| + 1$-th message to be sent after D_x.

Thus, an upper bound on the response time for N-Server x, R_x is:

$$R_x = D_x + \left\lceil \frac{|A| + |B| + 1}{N_{EC}} \right\rceil * T_{EC}$$

where N_{EC} denotes the number of messages in an EC and T_{EC} the longest time an EC can take (including sending TM and STOP messages).

4. PROTOCOL FOR LOWER RESPONSE-TIMES

The previously presented protocol has a worst-case response-time for a server x that has a large part which is independent of x's deadline (D_x). This can for many network configurations (e.g. configurations with many servers) give unreasonable worst-case response-times. In this section a modification to the previous protocol is presented, based on a dynamic priority version of the Polling Server (PS) (Lehoczky *et al.*, 1997). Using this new approach, the worst-case response-time of server x is mostly dependent on the server deadline D_x (which is equal to the server period T_x).

In the previous protocol, the increased protocol overhead caused by erroneous guesses mostly penalise servers with long periods (since servers with short periods will still receive good service even if they are causing erroneous guesses). The proposed modification will instead penalise the servers which causes erroneous guesses (i.e. the servers that do not send messages when they are allocated bandwidth).

4.1 New Rules

In the modified protocol, the M-Server will always treat an N-Server that has been allocated bandwidth as if the N-Server actually did send a message (regardless whether it did send any message). Hence the rules of Section 3.2.1 will be changed to a single rule:

- Treat the N-Server as if it sent a message. The next guess is that the N-Server has more messages to send. Hence it updates the N-Server's state according to rule 2 in Section 3.1.2.

In order for an N-Server not to cause more than one erroneous guess during its period, the M-Server will apply the following rules when deciding the schedule for the EC starting at time t:

(1) Only servers that are within one period from their deadline will be eligible for scheduling. Formally the following condition must hold for a server x to be eligible for scheduling:

$$d_x - t \leq T_x$$

(2) Among the eligible N-Servers, select N-Servers in EDF order to be scheduled during the EC.

(3) To make the protocol work-conserving, when there are not enough eligible N-Servers to fill an EC, also non-eligible N-Servers are selected in EDF order to be scheduled during the EC.

4.2 Timing Analysis

In order to provide a worst-case response-time analysis, the first step is to establish that the modified protocol will behave no worse than if pure EDF scheduling was deployed. Using that result the worst-case response-time can be derived.

Lemma 1: If all N-Servers that are allocated messages do send messages, no erroneous guesses will be made and the protocol will emulate pure EDF (and all deadline will be met if load is less than 100%).

Proof: When all N-Servers send their allocated messages they will behave like periodic processes and according to the scheduling mechanism they will be scheduled in EDF. The following equation is true if the system load (including protocol overhead when all servers send their allocated message) is less than 100% (Nolte *et al.*, 2003):

$$\sum_{\forall s}\left(\frac{M}{S \times T_s}\right) \leq 1 - \left(\frac{TM + STOP + S \times T_{sched}}{S \times T_{EC}}\right)$$

where s is an N-Server in the system, M is the length of a message (typically worst-case which is 135 bits), S is the network speed in bits/second, T_s is the period of the N-Server. TM and STOP are the sizes of the TM and the STOP messages in bits, typically 135 and 55 bits, T_{sched} represents the computational overhead of updating the N-Server deadlines and encoding the next TM after receiving the STOP message, and T_{EC} is the length of the EC. □

Lemma 2: If an N-Server that is allocated a message does not send a message, no other N-Server will miss its deadline because of this.

Proof: When an N-Server does not send a message it has been allocated, the EC will be terminated early (as can also happen if not all messages are of maximum length). However, since the N-Server will not be scheduled again until its next period it cannot cause more overhead (in terms of TM and STOP messages being generated). And, since the current EC is terminated early, subsequent messages from other N-Servers that are to be scheduled in forthcoming ECs will be server earlier than they would have been if the N-Server did send its message (and, hence, they cannot miss their deadlines). □

Theorem: The worst-case response-time for N-Server x is not greater than $2 * T_x + T_{EC}$.

Proof: Lemmas 1 and 2 tells us that N-Server x will be scheduled at least once during its period T_x. There is however no restriction on when, during T_x, the server is scheduled. In the worst-case, the distance between two consecutive occasions of scheduling of x then becomes $2 * T_x$ (if the first message is scheduled immediately in its T_x and the second message is scheduled as late as possible in its T_x).

The second message will be scheduled no later than the EC in which its deadline is. However, the order of transmission within that ECs is unknown. Hence, the pessimistic assumption is that the message is scheduled as the last message in the EC and that the deadline is at the start of the EC. This will incur an extra maximum delay of T_{EC} for the message. □

4.3 *Comparison of Mechanisms*

The main difference between the new mechanism and the old one is that with the new mechanism a server with a short period will not have to wait for a time longer than twice its server period. Using the old mechanism the maximum time a server could have to wait for is determined by the number of servers in the system together with the size of the EC. In that case a server might have to wait for a period longer than twice its server period.

If the system consists of a large set of servers with short server periods, the new mechanism is better, since the worst-case response-time for these servers are limited by their periods rather than the total number of servers, whereas if the system consists of a few servers with long server period, the old mechanism is giving lower worst-case response times.

5. CONCLUSIONS

In this paper a new mechanism for scheduling of the Controller Area Network (CAN) is presented. The difference between the new mechanism and existing methods is the usage of server-based scheduling techniques. Furthermore, the mechanism is based on Earliest Deadline First (EDF) scheduling to achieve high utilisation. The approach allows a more flexible way to utilize the CAN bus compared to other scheduling approaches such as native CAN and Flexible Time-Triggered communication on CAN (FTT-CAN). Servers provide fairness among the streams of messages as well as timely message delivery.

Two different server-based scheduling mechanisms are presented, suitable for different system configurations. It is also shown how to derive upper bounds for the response-times for both mechanisms.

The main strength of server-based scheduling for CAN, compared to other scheduling approaches, is the good service to streams of aperiodic messages. On native CAN, streams of aperiodic messages can starve other streams of messages. In FTT-CAN the situation is better, since periodic messages can be scheduled according to EDF using the synchronous window of FTT-CAN, thus separating periodic streams from aperiodic ones, guaranteeing real-time demands of the latter. However, no fairness can be guaranteed among the streams of aperiodic messages sharing the asynchronous window of FTT-CAN.

However, each scheduling policy has both good and bad properties. To give the fastest response-times, native CAN is the best choice. To cope with fairness and bandwidth isolation among aperiodic message streams, the server-based approach is the best choice, and, to have support for both periodic messages with low jitter, and aperiodic messages (although no fairness among aperiodic messages), FTT-CAN is the choice.

ACKNOWLEDGEMENTS

The work presented in this paper was supported by the Swedish Foundation for Strategic Research (SSF) via the research programme ARTES, the Swedish Foundation for Knowledge and Competence Development (KK-stiftelsen), and Mälardalen University.

REFERENCES

Abeni, L. (1998). Server Mechanisms for Multimedia Applications. Technical Report RETIS TR98-01. Scuola Superiore S. Anna. Pisa, Italy.

Almeida, L., J. A. Fonseca and P. Fonseca (1999). A Flexible Time-Triggered Communication System Based on the Controller Area Network: Experimental Results. In: *Proceedings of the International Conference on Fieldbus Technology (FeT'99)*.

Audsley, N. C., A. Burns, M. F. Richardson, K. Tindell and A. J. Wellings (1993). Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling. *Software Engineering Journal* **8**(5), 284–292.

CAN (1993). Road Vehicles - Interchange of Digital Information - Controller Area Network (CAN) for High-Speed Communication. *International Standards Organisation (ISO)*.

CiA (1996). CANopen Communication Profile for Industrial Systems, Based on CAL. CiA Draft Standard 301, rev 3.0.

J1938, SAE (1998). Design/Process Checklist for Vehicle Electronic Systems. *SAE Standards*.

Lehoczky, J.P., L. Sha and J. Strosnider (1997). Enhanced Aperiodic Responsiveness in Hard Real-Time Environments. In: *Proceedings of* 8^{th} *IEEE Real-Time Systems Symposium (RTSS'87)*. San Jose, California, USA. pp. 261–270.

Livani, M. A., J. Kaiser and W. J. Jia (1998). Scheduling Hard and Soft Real-Time Communication in the Controller Area Network (CAN). In: *Proceedings of the* 23^{rd} *IFAC/IFIP Workshop on Real-Time Programming*. Shantou, ROC.

Natale, M. Di (2000). Scheduling the CAN Bus with Earliest Deadline Techniques. In: *Proceedings of the* 21^{st} *IEEE Real-Time Systems Symposium (RTSS'00)*. Orlando, Florida, USA.

Nolte, T. and K. J. Lin (2002). Distributed Real-Time System Design using CBS-based End-to-end Scheduling. In: *Proceedings of the* 9^{th} *IEEE International Conference on Parallel and Distributed Systems (ICPADS'02)*. Taipei, Taiwan, ROC.

Nolte, T., M. Sjödin and H. Hansson (2003). Server-Based Scheduling of the CAN Bus. In: *Proceedings of the* 9^{th} *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2003)*. Calouste Gulbenkian Foundation, Lisbon, Portugal. pp. 169–176.

Pedreiras, P. and L. Almeida (2000). Combining Event-triggered and Time-triggered Traffic in FTT-CAN: Analysis of the Asynchronous Messaging System. In: *Proceedings of the* 3^{rd} *IEEE International Workshop on Factory Communication Systems (WFCS'00)*. Porto, Portugal. pp. 67–75.

Spuri, M. and G. C. Buttazzo (1994). Efficient Aperiodic Service under Earliest Deadline Scheduling. In: *Proceedings of the* 15^{th} *IEEE Real-Time Systems Symposium (RTSS'94)*.

Spuri, M., G. C. Buttazzo and F. Sensini (1995). Robust Aperiodic Scheduling under Dynamic Priority Systems. In: *Proceedings of the* 16^{th} *IEEE Real-Time Systems Symposium (RTSS'95)*.

Tindell, K. W., A. Burns and A. J. Wellings (1995). Calculating Controller Area Network (CAN) Message Response Times. *Control Engineering Practice* **3**(8), 1163–1169.

TT-CAN (2000). Road Vehicles - Controller Area Network (CAN) - Part 4: Time-Triggered Communication. *International Standards Organisation (ISO)*.

Zuberi, K. M. and K. G. Shin (1995). Non-Preemptive Scheduling of Messages on Controller Area Network for Real-Time Control Applications. In: *Proceedings of the* 1^{st} *IEEE Real-Time Technology and Applications Symposium (RTAS'95)*. Chicago, IL, USA. pp. 240–249.