Real-Time Server-Based Communication With CAN

Thomas Nolte, *Student Member, IEEE*, Mikael Nolin, and Hans A. Hansson, *Associate Member, IEEE*

***Abstract*—This paper investigates the concept of share-driven scheduling of networks using servers with real-time properties. Share-driven scheduling provides fairness and bandwidth isolation between 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.

We illustrate the share-driven scheduling concept by applying it to the popular controller area network (CAN). We propose a scheduling mechanism that we call simple server-scheduled CAN (S^3-CAN), for which we also present an associated timing analysis. Additionally, we present a variant of S^3-CAN called periodic server-scheduled CAN (PS^2-CAN), which for some network configurations gives lower worst-case response-times than S^3-CAN. Also for this improvement, a timing analysis is presented. Moreover, we use simulation to evaluate the timing performance of both S^3-CAN and PS^2-CAN, comparing them with other scheduling mechanisms.

***Index Terms*—Communication systems, controller area network (CAN), protocols, real time systems, scheduling, server-based, share-driven.**

I. Introduction

IN OPTIMIZING 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, and vice-versa. Therefore, we propose the usage of *server-based scheduling techniques*, which can handle streams of both periodic and aperiodic traffic. We propose server techniques using the earliest deadline first (EDF) scheduling strategy, since EDF allows optimal resource utilization. Examples of such server-based scheduling techniques are the total bandwidth server (TBS) [1], [2] and the constant bandwidth server (CBS) [3].

In this paper, we present mechanisms for server-based scheduling of the controller area network (CAN) [4], [5], which is one of the more commonly used fieldbuses, used extensively in the automotive industry as well as in several other application domains that have real-time requirements. CAN is implementing fixed-priority scheduling (FPS) using a bit-wise arbitration mechanism in the medium access control (MAC) layer. This mechanism resolves collisions in a deterministic way and is amenable to timing analysis [6]–[8].

Using server-based scheduling techniques for CAN, fairness among users of the network is guaranteed (e.g., "misbehaving" aperiodic processes cannot adversely interfere with well-behaved processes). In contrast with other proposals to schedule CAN, aperiodic messages are not sent "in the background" of periodic messages or in separate time-slots (e.g., [9]). Instead, aperiodic and periodic messages are jointly scheduled using servers. This substantially facilitates meeting response-time requirements, both for aperiodic and periodic messages.

As a side effect of using servers, the CAN frame identifiers (which are also used as priorities during the arbitration) will not play a significant role in the frame response-time. This greatly simplifies the assignment of frame identifiers (which is often done in an ad-hoc fashion at an early stage in a project). This also allows migration of legacy systems (where identifiers cannot easily be changed) into our new framework. Also, many domain specific standards specify which identifiers to be used, further limiting the possibility to use the identifiers to tune network timing behavior [10], [11].

Share-driven scheduling for CAN has been proposed in our earlier work [12], [13]. This paper summarizes our earlier work, fully describing our approach to share-driven scheduling of the CAN bus. By providing the option of share-driven scheduling of CAN, designers are given more freedom in designing an application. The main contributions of this paper can be summarized as follows.

- We present *simple server-scheduled CAN* (S^3-CAN), a scheduling mechanism to provide bandwidth isolation between applications sending messages on a CAN bus.
- We present *periodic server-scheduled CAN* (PS^2-CAN), a modification of S^3-CAN that, under some network configurations, gives lower worst-case response-times.
- Both S^3-CAN and PS^2-CAN avoid unnecessary message passing and efficiently reclaim unused bandwidth; hence, they have very low overhead.
- Messages are allowed to have arbitrary arrival patterns, i.e., synchronous, sporadic, and asynchronous messages are supported.
- We present a worst-case response-time analysis for both S^3-CAN and PS^2-CAN, bounding the maximum delay for a server to send a message.
- We compare the worst-case response-times of S^3-CAN and PS^2-CAN with each other together with a periodic polling scheduling mechanism (PP-CAN) and a leaky bucket scheduling mechanism (LB-CAN). This comparison is done in a simulation evaluation.

The paper is organized as follows. In Section II, related work is presented. In Section III, we present S^3-CAN together with

Manuscript received May 3, 2004; revised October 1, 2004. This work was supported by the Swedish Foundation for Strategic Research (SSF) via the research programmes ARTES and SAVE, the Swedish Foundation for Knowledge and Competence Development (KK-stiftelsen), and Mälardalen University.

The authors are with the Mälardalen Real-Time Research Centre, Department of Computer Science and Electronics, Mälardalen University, SE-721 23, Västerås, Sweden (e-mail: thomas.nolte@mdh.se; mikael.nolin@mdh.se; hans.hansson@mdh.se).

Digital Object Identifier 10.1109/TII.2005.852074

its timing analysis, and in Section IV we present the modified scheduling mechanism PS^2-CAN, together with its corresponding timing analysis. In Section V the timing performance of the two scheduling mechanisms are shown in a simulation evaluation. Finally, in Section VI we conclude and present future work.

II. Background and Related Work

In this section we give an introduction to CAN and present previously proposed methods for scheduling CAN.

A. *Controller Area Network (CAN)*

The CAN [4], [5] is a broadcast bus designed to operate at speeds of up to 1 Mbps. CAN is extensively used in automotive systems, as well as in several other application domains. CAN is a collision-avoidance broadcast bus, which uses deterministic collision avoidance to control access to the bus (so called CSMA/CA). CAN transmits data in frames containing a header and 0 to 8 bytes of data.

The CAN 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.

The basis for the access mechanism is the electrical characteristics of a CAN bus. During arbitration, competing stations simultaneously put their identifiers, one bit at the time, on the bus. Bit value “0” is the dominant value. Hence, if two or more stations are transmitting bits at the same time, and if at least one station transmits a “0”, then the value of the bus will be “0”. By monitoring the resulting bus value, a station detects if there is a competing higher priority frame (i.e., a frame with a numerically lower identifier) and stops transmission if this is the case. Because identifiers are unique within the system, a station transmitting the last bit of the identifier without detecting a higher priority frame must be transmitting the highest priority active frame, and can start transmitting the body of the frame. Thus, CAN behaves as a global priority based queue, since at all nodes the message chosen during arbitration is always the active message with the highest priority.

B. *Scheduling on CAN*

In the real-time scheduling research community there exist several different types of scheduling. We can divide the classical scheduling paradigms into the following three groups.

1) Priority-driven (e.g., FPS or EDF) [14].
2) Time-driven (table-driven) [15], [16].
3) Share-driven [17], [18].

1) Priority-Driven: For CAN, the most natural scheduling method is fixed-priority scheduling (FPS) since it is the policy used by the CAN protocol. Analysis, like the FPS response-time tests to determine the schedulability of CAN message frames, have been presented by Tindell *et al.* [6]–[8]. This analysis is based on the standard FPS response-time analysis for CPU scheduling presented by Audsley *et al.* [19].

Several methods for dynamic-priority type of scheduling have been proposed for CAN. By manipulating the message identifier, and therefore changing the message priority dynamically, several approaches to mimic EDF type of scheduling have been presented [20]–[22]. However, these solutions are all based on manipulating the identifier of the CAN frame to reflect the deadline of the frame and thus reducing the number of possible identifiers to be used by the system designers. This is not an attractive alternative, since it interferes with other design activities, and is even sometimes in conflict with adopted standards and recommendations [10], [11].

2) Time-Driven: Table-driven, or time-driven, scheduling of CAN is provided by, e.g., TT-CAN [23] and FTT-CAN [9], [24]. Flexible time-triggered CAN (FTT-CAN), presented by Almeida *et al.*, supports priority-driven scheduling in combination with time-driven scheduling. In FTT-CAN, time is partitioned into elementary cycles (ECs) that are initiated by a special message, the trigger message (TM). This message contains the schedule for the synchronous traffic (time-triggered traffic) that shall be sent within this EC. The schedule is calculated and sent by a specific node called the *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 CANs native arbitration mechanism. In the second part, the synchronous window, traffic is sent according to the schedule delivered in the TM. More details of the EC layout are provided in Fig. 1, that shows two ECs during which both asynchronous messages (AM*x*) and synchronous messages (SM*x*) are being sent. In the figure, the schedule is represented by a bit field, where a “1” in position n means that synchronous message n is to be sent during the EC. The asynchronous messages compete for bus access using CANs native arbitration mechanism.

It has also been shown how to schedule periodic messages according to EDF using the synchronous part of the EC [25]. This provides greater flexibility compared to [20]–[22], since the scheduling is not based on manipulating the identifiers. Instead, there is a master node performing the scheduling of the CAN bus.

3) Share-Driven: Share-driven scheduling of CAN has been presented in our earlier work on server-based scheduling of CAN, e.g., [12], [13]. In Sections III–V, we will describe server-based scheduling of CAN in more detail. By providing the option of share-driven scheduling of CAN, designers are given more freedom in designing a distributed real-time system where the nodes are interconnected with a CAN-bus.

III. Simple Server-Scheduled CAN (S^3-CAN)

In order to provide guaranteed network bandwidth for both synchronous and asynchronous real-time messages in CAN, we propose the usage of server-based scheduling techniques. Using servers, the whole network will be scheduled as a single resource, providing bandwidth isolation as well as fairness among the users of the network. However, in order to make server-scheduling decisions, we must have information on when messages are arriving at the different nodes in the system, so that we can assign them a deadline based on the server policy in use.

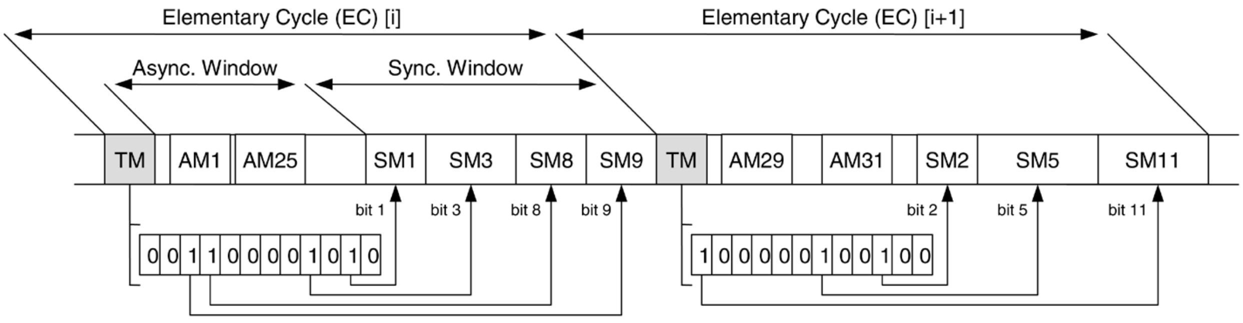

Fig. 1. EC layout and TM data contents (FTT-CAN approach).

This information should not be based on message passing, since this would further reduce the already low bandwidth offered by CAN. Our method, presented below, will provide a solution to this.

In this section, we will present our simple server-scheduled CAN (S^3-CAN), together with its associated timing analysis.

A. 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 we will have multiple servers mediating access to a single CAN bus. Each server has exclusive associated bandwidth in terms of capacity, C, and period time, T. Moreover, servers have an associated deadline which is used for scheduling decisions. Also, the servers have a local queue, in which its pending users are stored.

A server might have one or several *users*. In this paper, a user is a stream of messages, e.g., the sending of messages by an application, and these users can be of either periodic, sporadic or aperiodic nature. Since each server has exclusive right to a share of the total system bandwidth, all users sharing a server will share this portion of bandwidth. Depending on the type of queue used at the server, e.g., FIFO or priority-based, different guarantees of timeliness can be offered. Suppose a priority-based queue is used, then the users will experience a service similar to the one of an exclusive network, essentially with the only difference in the lower bandwidth. Hence, the timely behavior will be, compared to an exclusive CAN network, divided by C/T, i.e., the server's share. However, since the network is nonpreemptive, special care needs to be made in order to take the effects of blockings into consideration [9], [26]. Therefore a variant of the FPS response-time analysis could be used to calculate the timing properties.

In the scheduling literature many types of servers are described. Using fixed priority scheduling (FPS), for instance, the sporadic server (SS), is presented by Sprunt *et al.* [27]. SS has a fixed priority chosen according to the rate monotonic (RM) policy. Using EDF, Spuri and Buttazzo [1], [28] extended SS to dynamic sporadic server (DSS). Other EDF-based schedulers are the constant bandwidth server (CBS) presented by Abeni [3], and the total bandwidth server (TBS) by Spuri and Buttazzo [1], [2]. Each server is characterized partly by its unique mechanism for assigning deadlines, and partly by a set of variables used to configure the server. Examples of such variables are bandwidth, period, and capacity.

1) N-Server Characterization: Each node on the CAN bus will be assigned one or more *network servers* (N-servers). S^3-CAN is using a simplified version of TBS [1], [2]. Each N-server, s, is characterized by its period T_s, and is allowed to send a single message every server period. The message length is assumed to be of worst-case size, i.e., 8 bytes of data. 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 should not be confused with any deadline associated with a particular message. (For instance, S^3-CAN will under certain circumstances *miss* the server deadline. As we will show, however, this does not necessarily make the system unschedulable.)

2) N-Server State: The state of an N-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

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

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

$$d_s^n = d_s^{n-1} + D_s. \tag{2}$$

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

B. Medium Access (M-Server)

The native medium access method in CAN is strictly priority-based. Hence, it is not very useful for our purpose of scheduling the network with servers. Instead we introduce a *master server* (M-server) 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 [9], [24] although in this paper the master's responsibilities are a bit different. Here, 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

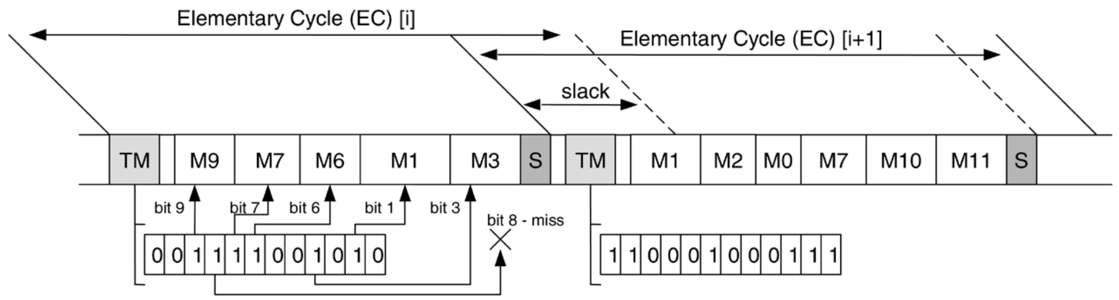

Fig. 2. EC layout and TM data contents (server approach).

that each N-server has a message to send (e.g., initially each N-server s is assigned a deadline $d_s = D_s$). Later we will see how to handle erroneous guesses, i.e., when an N-server does not have a message to send. This is done by monitoring the bus.

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 ECs, similar to the FTT-CAN approach presented in Section II-B. We use T_{EC} to denote the nominal length of an EC. 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 are allowed to send one message during the EC, i.e., the EC's schedule. 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, we do not know when inside an EC a specific message is sent. Hence, due to this uncertainty of not knowing when a message is sent within an EC, the bound on the delay of message transmissions will be proportional to the size of the 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 message (STOP) with the lowest possible priority. A small delay before sending STOP is required[1]. After sending the STOP message to the CAN controller, the M-server reads all messages sent on the bus, i.e., the M-server is monitoring the bus. When it reads the STOP message it knows that all N-servers that were scheduled to send messages in the EC have sent their messages (if they had any messages to send). After reading the STOP message the EC is over and the M-server has to update the state of the N-servers scheduled during the EC, before starting the next EC. 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, by using a STOP message we are able to use standard CAN controllers.

[1]We need to make sure that this message is not 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.

Fig. 2 presents the layout of the EC when using servers. Note that the servers that are allocated to transmit a message in the EC are indicated by a "1" in the corresponding position of the TM, and that the actual order of transmission is determined by the message identifiers, and not by the server number. In the first EC in Fig. 2 N-server "8" was allowed to send a message, however, in this example the server did not have any message to send.

1) Updating N-Server States: At this point, 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 we have two cases.

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, it updates the N-server's state according to rule 2 in Section III-A2.
2) The N-server did not send a message. In this case, the guess was incorrect and the N-server was idle. Since we do not know if a N-server has any message to send, and to be as responsive as possible, 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 III-A2.

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, for our method to be viable we need a mechanism to reclaim this bandwidth.

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 so no bandwidth is lost.

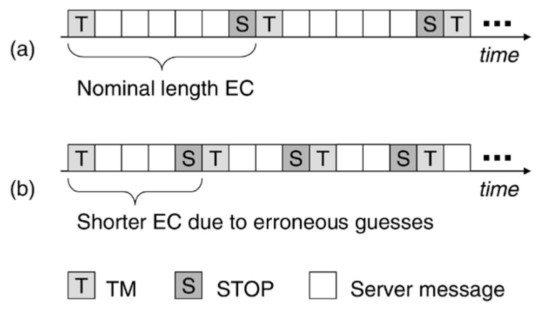

Fig. 3. Erroneous guesses increases the protocol overhead.

C. Timing Analysis

One property of S^3-CAN is that the protocol overhead increases when the network utilization is low. When the network is not fully utilized, the M-server will send more TM and STOP message than it would during full network utilization. This is illustrated in Fig. 3, where Fig. 3(a) shows a fully utilized network where all servers that were allowed to send a message did so. Fig. 3(b) shows what happens if not all servers send messages, and here we see that more TM and STOP messages are being sent. In fact, our proposed method will always consume all bandwidth of the network, regardless of the number of server messages actually existing.

This increase of protocol overhead when N-servers do not send their allocated messages makes the behavior of our scheduling mechanism more complicated than the original TBS' behavior. Moreover, the original TBS analysis is for preemptive systems. Hence, we cannot reuse the worst-case analysis of TBS to bound the worst-case message delay [1], [2].

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 zero when a message arrives to x and its deadline is set to D_x) we classify the other N-servers in the system 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, or equal to, 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 can still be scheduled for message transmission for each EC until their deadline becomes greater than D_x.

For example, consider a system of three N-servers having server periods $T_s^1 = 6$, $T_s^2 = 12$, $T_s^3 = 12$. Each N-server is allowed to send one message each server period. Each EC contains, for simplicity, only one N-server message, hence the system is fully utilized (i.e., system load $= 100\%$). After each EC, the M-server is updating the N-server deadlines accordingly. If no erroneous guesses are made and all N-servers have messages to send, the actual schedule is depicted in Fig. 4(a).

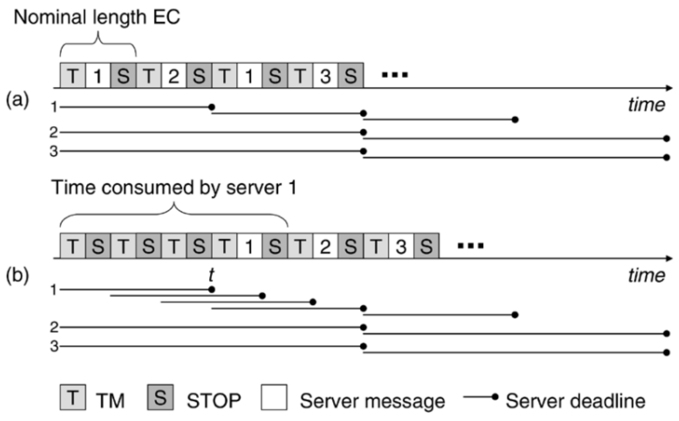

Fig. 4. Erroneous guesses causes late scheduling for S^3-CAN.

However, looking at Fig. 4(b), suppose N-server 1 has no message to send before time t. Since its server deadline is earlier compared to the other N-servers it will be scheduled for message transmission, hence blocking the other N-servers for message transmission. All erroneous guesses on N-server 1 causes an increase of protocol overhead (i.e., the number of trigger- and stop-messages (TM & STOP) increase). The result is a late scheduling of N-server 3 as depicted in Fig. 4(b).

We conclude that 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. During this interference period no server will 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.

Hence, an upper bound on the response time for a single message arriving at N-server x, R_x is

$$R_x = D_x + \left\lceil \frac{|A| + |B| + 1}{N_{EC}} \right\rceil * T_{EC} \qquad (3)$$

where N_{EC} denotes the number of message in an EC assuming worst-case message size, and T_{EC} the longest time an EC can take (including sending TM and STOP messages).

For example, consider the same set of N-servers as in Fig. 4. The analytically worst-case scenario for N-server 3 is depicted in Fig. 5, where $|A| = 1$ (i.e., N-server 1) and $|B| = 1$ (i.e., N-server 2). N-server 1 is totally blocking the network for D_3 time units, and N-server 2 has a deadline slightly before N-server 3. At time t, all N-servers have messages to send and the deadline of N-server 3 is the latest, hence N-server 3 has to wait for the other N-servers before being scheduled, as expected by (3).

Note that the implication of this formula is a worst-case response time that is (essentially) dependent on the number of N-servers in the system, and not the server deadline and periodicity. This is unfortunate, since the motivation for introducing the servers was to provide isolation between message streams.

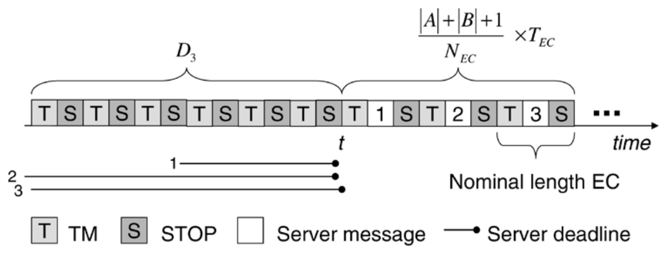
$\frac{|A|+|B|+1}{N_{EC}} \times T_{EC}$
D_3
T S T S T S T S T S T S T 1 S T 2 S T 3 S
t
time
Nominal length EC
1
2
3
T TM
S STOP
Server message
Server deadline

Fig. 5. Analytical worst-case scenario for server 3.

It is also the case that the worst-case bound requires a highly unlikely combination of server phasings and erroneous guesses from the M-server.

IV. Periodic Server-Scheduled CAN (PS^2-CAN)

The previously presented scheduling mechanism, S^3-CAN, has a worst-case response-time for a message using an arbitrary N-server x that has a large part which is independent of x's deadline (D_x). For many network configurations (e.g., configurations with many servers) this can give unreasonable worst-case response-times. In this section, we present a modification to our scheduling mechanism, where the worst-case response-time of server x instead is mainly dependent on the server deadline D_x (which is equal to the server period T_x). The modification is called periodic server-scheduled CAN (PS^2-CAN). PS^2-CAN is based on a dynamic priority version of the polling server (PS) [29].

In S^3-CAN, the increased protocol overhead caused by any erroneous guesses mostly penalize servers with long periods (since servers with short periods will still receive good service even if they are causing erroneous guesses). PS^2-CAN will instead penalize the servers which causes erroneous guesses (i.e., the servers that do not send messages when they are allocated bandwidth) by delaying them at one period (hence the name PS^2-CAN).

A. New Rules

In PS^2-CAN, the M-server will always treat a 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 III-B2 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 III-A2.

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.

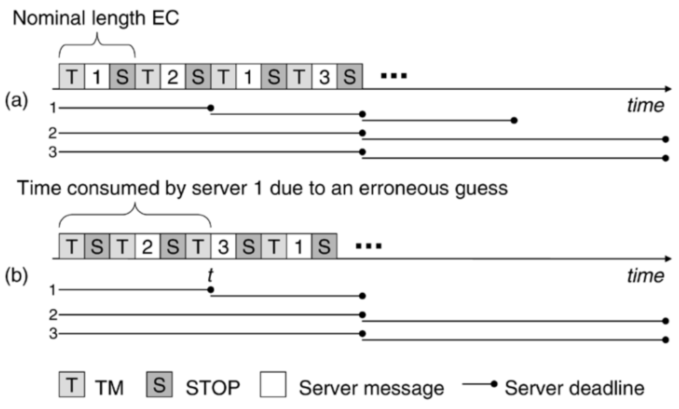

Fig. 6. Erroneous guesses for PS^2-CAN.

3) To make the scheduling mechanism work-conserving, when there are not enough eligible N-servers to fill an EC, noneligible N-servers are also selected in EDF order to be scheduled during the EC.

To illustrate the scheduling mechanism of PS^2-CAN, consider the same set of N-servers as in Fig. 4. Now when there is a erroneous guess, the N-server deadlines are updated according to the new rules. The scheduling is depicted in Fig. 6.

B. Timing Analysis

In order to provide a worst-case response-time analysis for PS^2-CAN, we will first establish the fact that our modified scheduling mechanism will behave no worse than if fully preemptive EDF task scheduling was used. Using that result we then derive the worst-case response-time.

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

a) Proof: When all N-servers send their allocated messages they will behave like periodic processes and, according to our rules, they will be scheduled in EDF. The following equation holds if the system load (including protocol overhead when all servers send their allocated message) is less than 100% [13]:

$$\sum_{\forall s}\left(\frac{M}{S \times T_s}\right) \leq 1 - \left(\frac{TM + STOP + S \times T_{sched}}{S \times T_{EC}}\right) \quad (4)$$

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/s, 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, and T_{sched} represents the computational overhead of updating the N-server deadlines and encoding the next TM after receiving the STOP message as well as the decoding of the TM in the slaves, and T_{EC} is the length of the EC. $\square$

2) 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.

b) 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 served earlier than they would have been if the N-server did send its message (and, hence, they cannot miss their deadlines). □

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

c) Proof: Lemmas 1 and 2 tell 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).

However, we only know that the second message will be scheduled no later than the EC in which its deadline is, we do not know the order of transmission within the ECs. Furthermore, we do not know when during that EC the server deadline occurs. Hence, we must pessimistically assume 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. □

V. Evaluation

To evaluate S^3-CAN and PS^2-CAN, we have performed a series of simulations. The simulation study consists of two parts. First, the timing performance of S^3-CAN and PS^2-CAN is shown in relation to their corresponding worst-case response-time analysis. Second, the timing performance of S^3-CAN and PS^2-CAN are compared with each other together with two other scheduling mechanisms [a periodic polling scheduling mechanism (PP-CAN) and a leaky bucket scheduling mechanism (LB-CAN)].

A. Timely Performance

A system consisting of 40 N-servers is simulated 1000 times for 20 s. Each N-server x has one periodic user allocated to it (with period $= T_{user} = T_x$). For each simulation, all users are given a random initial phase t_{ph}, randomly selected using a rectangular distribution with the value range $[0, T_{user})$. Moreover, each message generated by any user has a unique randomly generated message identifier. Each user has been assigned one out of five predefined periods ($1\times T_{EC}, 3\times T_{EC}, 7\times T_{EC}, 11\times T_{EC}$, and $13 \times T_{EC}$,). All the properties of the simulations are shown in Table I.

During the simulations, minimum, average, and maximum response times are measured. The results are shown in Figs. 7 and 8. For all users the messages have worst-case measured response-times less than their corresponding theoretical worst-case.

B. Comparison With Other Scheduling Mechanisms

In order to compare the performance in terms of responsiveness, S^3-CAN, PS^2-CAN, a periodic polling scheduling mechanism (PP-CAN), and a leaky bucket scheduling mechanism (LB-CAN) are compared with each other. Bandwidth isolation is enforced by all four scheduling mechanisms. Variants of periodic polling are implemented by, e.g., FTT-CAN [9], [24] and WorldFIP [30].

PP-CAN works basically the same way as PS^2-CAN, but with the slack reclaiming mechanism disabled. Hence, the duration of the ECs is fixed when using PP-CAN. PP-CAN performs similar to S^3-CAN and PS^2-CAN. However, since PP-CAN has no bandwidth reclaiming mechanism, its timing performance is somewhat worse compared to S^3-CAN and PS^2-CAN.

TABLE I
PROPERTIES OF THE SIMULATION

Network speed (bits/ms)	500
EC size (in messages)	15
EC period (including TM & STOP) (ms)	4.43
Message size (bits)	135
TM size (bits)	135
STOP size (bits)	55
T_{sched} (ms)	0
Number of N-Servers	40
Number of users (one to each N-Server)	40
Initial user phase t_{ph} (ms) (rectangular distribution)	$t_{ph} \in [0, T_{user})$
Maximum utilisation (according to eq (4))	0.9142
Utilisation in simulation (N-Servers)	0.9075
Relative utilisation of simulation (% of maximum)	99.27
Simulation time (ms)	20000
Number of simulations	1000

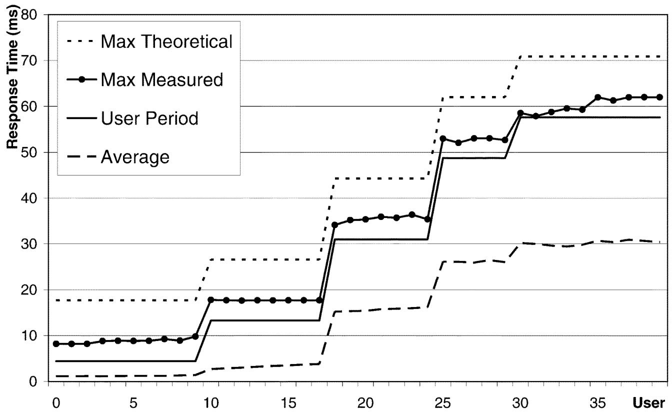

Fig. 7. Response-times of S^3-CAN.

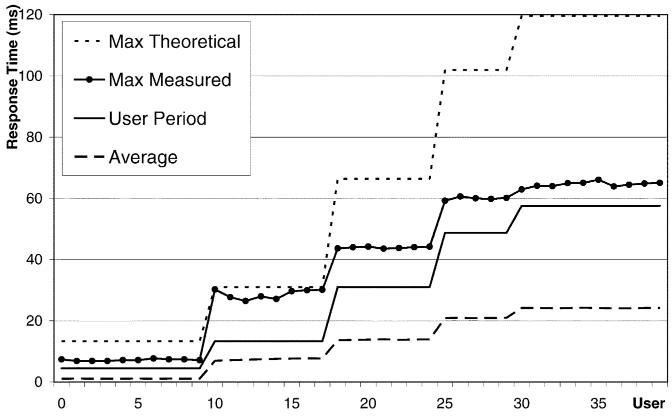

Fig. 8. Response-times of PS^2-CAN.

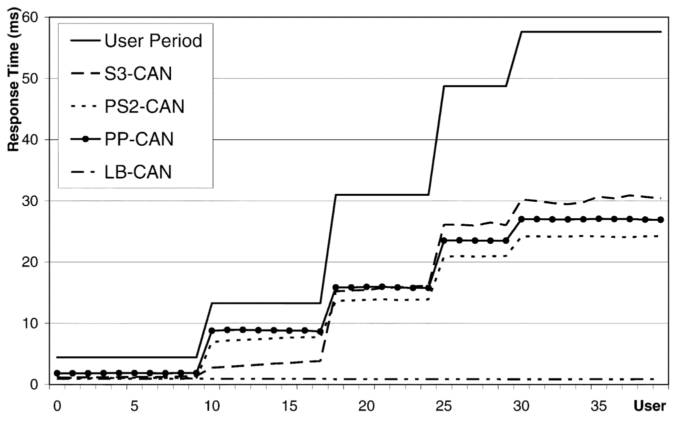

Fig. 9. Average measured response-times.

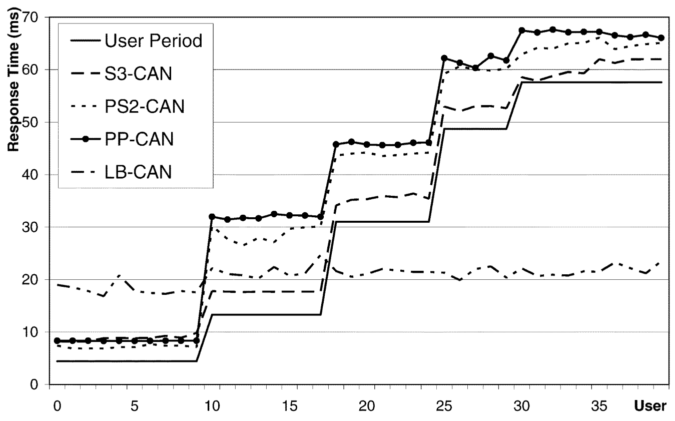

Fig. 10. Maximum measured response-times.

LB-CAN is a leaky bucket implementation on top of native CAN. Each user has its own leaky bucket, limiting the number of user messages to one message every user period. LB-CAN gives fast average responsiveness since it constantly sends messages using predefined periods without considering the rest of the users trying to send messages. In the average case, this scheduling mechanism gives small response-times, but in the worst-case a user has to wait for a potentially long time before its message is sent, due to the CAN message arbitration mechanism.

Looking at Fig. 9 (the average measured response-times), LB-CAN is the overall most responsive scheduling mechanism. PS^2-CAN is always better than PP-CAN. S^3-CAN is better than both PS^2-CAN and PP-CAN when the N-server periods are low. Elsewhere, S^3-CAN performs a bit worse in the average case compared to PS^2-CAN and PP-CAN.

However, looking at Fig. 10 (the maximum measured response-times), LB-CAN has poor performance not reflecting the periodicity of the users (i.e., users with short period does not have better performance compared to users with long period). The best worst-case measured response-times are given by S^3-CAN. This because all N-servers are sharing the protocol overhead caused by erroneous guesses. PS^2-CAN has somewhat worse performance compared to S^3-CAN, since an erroneous guess only penalizes its corresponding N-server. However, due to the slack reclaiming mechanism, the performance of PS^2-CAN is better than PP-CAN, which has the worst worst-case measured response-times in these simulations.

C. PS^2 *-CAN or* S^3*-CAN?*

If we compare just PS^2-CAN with S^3-CAN, the main difference is that with PS^2-CAN, an N-server with a short period will not have to wait for a time longer than twice its server period. Using S^3-CAN the maximum time an N-server could have to wait for is determined by the number of N-servers in the system together with the size of the EC. In that case, an N-server might have to wait for a period longer than twice its server period.

Both S^3-CAN and PS^2-CAN claim to enforce bandwidth isolation among N-servers. This is true, since the only way for a N-server to send a message is to be scheduled in a TM. This scheduling is performed by the M-server based on the N-server parameters. As long as the N-servers are not failing, the bandwidth isolation property among N-servers will be kept. However, no bandwidth isolation is provided among users sharing the same N-server, as several users might share one N-server.

If the system consists of a large set of N-servers with short server periods, PS^2-CAN is better, since the worst-case response-time for these N-servers are limited based on their periods rather than the total number of N-servers, whereas if the system consists of a few N-servers with long server period, S^3-CAN gives lower worst-case response times.

VI. Conclusions

In this paper, we have presented a server-based mechanism for scheduling of the CAN. Our method is the first server-based scheduling mechanism using dynamic priorities proposed for CAN. We use EDF scheduling to achieve high utilization. Our approach allows us to utilize the CAN bus in a more flexible way compared to other scheduling approaches, such as native CAN and FTT-CAN. Servers provide fairness among the streams of messages as well as timely message delivery.

We have presented two scheduling mechanisms (S^3-CAN and PS^2-CAN) for a centralized media access server (M-server) to schedule a set of network servers (N-servers). Both scheduling mechanisms provide fairness between different N-servers, and we have presented worst-case message response-time formulae for both scheduling mechanisms. The scheduling mechanisms differ in how they handle N-servers that do not fully use their allocated bandwidth. S^3-CAN has a worst-case response-time that is proportional to the number of N-servers in the system, whereas PS^2-CAN has a worst-case response-time that is proportional to the N-server's deadline and period.

The main strength of server-based scheduling for CAN, compared to other scheduling approaches, is that we can give good service to streams of aperiodic messages. Aperiodic messages on native CAN would make it (in the general case) impossible to give any real-time guarantees for the periodic messages sharing the bus. In FTT-CAN, the situation is better, since periodic messages can be scheduled according to EDF using the synchronous window of FTT-CAN, isolated from the aperiodic messages thus guaranteeing real-time demands. However, no fairness can be guaranteed among the streams of aperiodic messages sharing the asynchronous window of FTT-CAN.

One penalty for using the server method is an increase of CPU load in the master node (where the M-server is located), since it needs to perform the extra work for scheduling. Also, compared with FTT-CAN, we are sending one more message, the STOP message, which is reducing the available bandwidth for the system under heavy aperiodic load. However, the STOP message is of the smallest possible size and should therefore have minimal impact on the system. Also, if the CAN controller is able to detect when the bus is idle (and pass this information to the master node), we could skip the STOP message, and the overhead caused by our protocol would decrease (since this would make it possible to use our server-based scheduling without STOP-messages).

Every 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.

Using server-based scheduling, we can schedule for unknown aperiodic or sporadic messages by guessing that they are arriving, and if we make an erroneous guess, we are not wasting much bandwidth. This since the STOP message, together with the arbitration mechanism of CAN, allow us to detect when no more messages are pending so that we can reclaim potential slack in the system and start scheduling new messages without wasting bandwidth.

A future direction is to investigate whether unused bandwidth may be shared among servers. It would also be interesting to examine how the number of allowed messages to be sent within an EC, assigned to each server, can be varied in order to provide for example better response-times for the aperiodic messages.

Acknowledgment

The authors wish to express their gratitude to K.-J. Lin for all constructive discussions during the initial phase of this work, and to the anonymous reviewers for their helpful comments.

References

[1] M. Spuri and G. C. Buttazzo, "Efficient aperiodic service under earliest deadline scheduling," in *Proc. the 15th IEEE Real-Time Systems Symp. (RTSS'94)*, San Juan, PR, Dec. 1994, pp. 2–11.

[2] M. Spuri, G. C. Buttazzo, and F. Sensini, "Robust aperiodic scheduling under dynamic priority systems," in *Proc. 16th IEEE Real-Time Systems Symp. (RTSS'95)*, Pisa, Italy, Dec. 1995, pp. 210–219.

[3] L. Abeni, "Server Mechanisms for Multimedia Applications," Scuola Superiore S. Anna, Pisa, Italy, Tech. Rep. RETIS TR98-01, 1998.

[4] *Road Vehicles—Interchange of Digital Information—Controller Area Network (CAN) for High-Speed Communication*, ISO Standard-11 898, Nov. 1993.

[5] (2002) CAN in Automation (CiA), Am Weichselgarten 26, D-91 058 Erlangen. CAN Specification 2.0, Part-A and Part-B. [Online]. Available: http://www.can-cia.de/

[6] K. W. Tindell and A. Burns, "Guaranteed Message Latencies for Distributed Safety-Critical Hard Real-Time Control Networks," Dept. Comput. Sci., Univ. York, York, U.K., Tech. Rep. YCS 229, 1994.

[7] K. W. Tindell, A. Burns, and A. J. Wellings, "Calculating controller area network (CAN) message response times," *Control Eng. Pract.*, vol. 3, no. 8, pp. 1163–1169, 1995.

[8] K. W. Tindell, H. Hansson, and A. J. Wellings, "Analysing real-time communications: Controller area network (CAN)," in *Proc. 15th IEEE Real-Time Systems Symp. (RTSS'94)*, San Juan, PR, Dec. 1994, pp. 259–263.

[9] L. Almeida, P. Pedreiras, and J. A. Fonseca, "The FTT-CAN protocol: Why and how," *IEEE Trans. Ind. Electron.*, vol. 49, no. 6, Dec. 2002.

[10] (1996) CANopen Communication Profile for Industrial Systems, Based on CAL. CiA, CiA Draft Standard 301, rev 3.0. [Online]. Available: http://www.canopen.org

[11] *Design/Process Checklist for Vehicle Electronic Systems*, S. J1938, SAE Standards, May 1998.

[12] T. Nolte, M. Nolin, and H. Hansson, "Server-Based Real-Time Scheduling of the CAN Bus," in *Proc. 11th IFAC Symp. Information Control Problems in Manufacturing (INCOM'04)*, Salvador, Brazil, Apr. 2004.

[13] T. Nolte, M. Sjödin, and H. Hansson, "Server-based scheduling of the CAN bus," in *Proc. 9th IEEE Int. Conf. Emerging Technologies and Factory Automation (ETFA'03)*, Lisbon, Portugal, Sep. 2003, pp. 169–176.

[14] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *J. ACM*, vol. 20, no. 1, pp. 40–61, 1973.

[15] C. W. Hsueh and K. J. Lin, "An optimal pinwheel scheduler using the single-number reduction technique," in *Proc. 17th IEEE Real-Time Systems Symp. (RTSS'96)*, Los Alamitos, CA, Dec. 1996, pp. 196–205.

[16] H. Kopetz, "The time-triggered model of computation," in *Proc. 19th IEEE Real-Time Systems Symp. (RTSS'98)*, Madrid, Spain, Dec. 1998, pp. 168–177.

[17] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single-node case," *IEEE/ACM Trans. Networking*, vol. 1, pp. 344–357, Jun. 1993.

[18] I. Stoica, H. Abdel-Wahab, K. Jeffay, S. Baruah, J. Gehrke, and G. Plaxton, "A proportional share resource allocation algoritm for real-time, time-shared systems," in *Proc. 17th IEEE Real-Time Systems Symp. (RTSS'96)*, Los Alamitos, CA, Dec. 1996, pp. 288–299.

[19] N. C. Audsley, A. Burns, M. F. Richardson, K. Tindell, and A. J. Wellings, "Applying new scheduling theory to static priority pre-emptive scheduling," *Software Eng. J.*, vol. 8, no. 5, pp. 284–292, Sep. 1993.

[20] M. A. Livani, J. Kaiser, and W. J. Jia, "Scheduling hard and soft real-time communication in the controller area network (CAN)," in *Proc. 23rd IFAC/IFIP Workshop on Real-Time Programming*, Shantou, Taiwan, R.O.C., Jun. 1998.

[21] M. D. Natale, "Scheduling the CAN bus with earliest deadline techniques," in *Proc. 21st IEEE Real-Time Systems Symp. (RTSS'00)*, Orlando, FL, Nov. 2000, pp. 259–268.

[22] K. M. Zuberi and K. G. Shin, "Non-preemptive scheduling of messages on controller area network for real-time control applications," in *Proc. 1st IEEE Real-Time Technology and Applications Symp. (RTAS'95)*, Chicago, IL, May 1995, pp. 240–249.

[23] *Road Vehicles—Controller Area Network (CAN)—Part 4: Time-Triggered Communication*, ISO Standard-11 898-4, TT-CAN, Dec. 2000.

[24] L. Almeida, J. A. Fonseca, and P. Fonseca, "A flexible time-triggered communication system based on the controller area network: Experimental results," in *Proc. Int. Conf. Fieldbus Technology (FeT'99)*, Magdeburg, Germany, Sep. 1999.

[25] P. Pedreiras and L. Almeida, "A practical approach to EDF scheduling on controller area network," in *Proc. IEEE/IEE Real-Time Embedded Systems Workshop (RTES'01) at the 22nd IEEE Real-Time Systems Symp. (RTSS'01)*, London, U.K., Dec. 2001.

[26] L. Almeida and J. Fonseca, "Analysis of a simple model for nonpreemptive blocking-free scheduling," in *Proc. 13th Euromicro Conf. on Real-Time Systems (ECRTS'01)*, Delft, The Netherlands, Jun. 2001.

[27] B. Sprunt, L. Sha, and J. Lehoczky, "Aperiodic task scheduling for hard real-time systems," *Real-Time Syst.*, vol. 1, no. 1, pp. 27–60, 1989.

[28] M. Spuri and G. C. Buttazzo, "Scheduling aperiodic tasks in dynamic priority systems," *Real-Time Syst.*, vol. 10, no. 2, pp. 179–210, Mar. 1996.

[29] J. Lehoczky, L. Sha, and J. Strosnider, "Enhanced aperiodic responsiveness in hard real-time environments," in *Proc. 8th IEEE Real-Time Systems Symp. (RTSS'87)*, San Jose, CA, Dec. 1997, pp. 261–270.

[30] L. Almeida, E. Tovar, J. Fonseca, and F. Vasques, "Schedulability analysis of real-time traffic in worldFIP networks: An integrated approach," *IEEE Trans. Ind. Electron.*, vol. 49, no. 5, Oct. 2002.

Thomas Nolte (S'01) received the B.Eng., M.Sc., and Lic. degrees in computer engineering from the Department of Computer Science and Engineering at Mälardalen University, Västerås, Sweden, in 2001, 2002, and 2003, respectively. He is currently pursuing the Ph.D. degree in the Department of Computer Science and Electronics at Mälardalen University.

During first half of 2002, he was a Visiting Junior Specialist, Department of Electrical and Computer Engineering, University of California, Irvine (UCI), and during autumn 2004, he was a Visiting Researcher, Department of Computer Engineering and Telecommunications, University of Catania, Italy. His research interests include distributed embedded real-time systems, especially communication issues, scheduling of real-time systems, automotive and vehicular systems.

Mikael Nolin received the M.Sc. and Ph.D. degrees in 1995 and 2000 from Uppsala University, Sweden.

Since then, he has been working in both academia and in industry with embedded systems, real-time systems, and embedded communications. He is currently sharing his time between the company CC-Systems, where he is working with embedded communications solutions and architectures for distributed systems, and Mälardalen Real-Time Research Centre (MRTC), where he is an Associate Professor, focusing his research on new methods to construct software for embedded control systems in the vehicular industry. Current research topics include embedded communications, real-time analysis, software, real-time databases, and software engineering methods and tools.

Hans A. Hansson (A'01) received the M.Sc. degree in engineering physics, the Lic. degree in computer systems, the B.A. degree in business administration, and the Dr.Techn. degree in computer systems from Uppsala University, Uppsala, Sweden, in 1981, 1984, 1984, and 1992, respectively.

He is currently Professor of Computer Engineering at Mälardalen University, Västerås, Sweden. He is also Director of the Mälardalen Real-Time Research Centre. He was previously Programme Director for the ARTES national Swedish real-time systems initiative, Department Chairman and Senior Lecturer at the Department of Computer Systems, Uppsala University, Chairman of the Swedish National Real-Time Association (SNART), and Researcher at the Swedish Institute of Computer Science, Stockholm. His research interests include real-time system design, reliability and safety, timed and probabilistic modeling of distributed systems, scheduling theory, distributed real-time systems, and real-time communications networks.

Prof. Hansson is a member of the Association for Computing Machinery and SNART.