# Using Servers to Provide Bandwidth Isolation on the Controller Area Network

Thomas Nolte, Mikael Sjödin, and Hans Hansson
Mälardalen Real-Time Research Centre
Department of Computer Science and Engineering
Mälardalen University, Västerås, SWEDEN
http://www.mrtc.mdh.se

## Abstract

*We present a new share-driven server-based method for scheduling messages sent over the Controller Area Network (CAN) [8]. Share-driven methods are useful in many applications, since they provide both fairness and bandwidth isolation among the users of the resource. Our method is the first share-driven scheduling method proposed for CAN, and it is based on Earliest Deadline First (EDF), which allows higher utilization of the network than CAN's native fixed-priority scheduling (FPS). We use simulation to show the performance and properties of server-based scheduling for CAN. The simulation results show that the bandwidth isolation property is kept, and that with our method a good Quality-of-Service (QoS) is provided, where virtually all messages are delivered within their deadline.*

## 1 Introduction

The Controller Area Network (CAN) [8] is widely used in automotive and other real-time applications. CAN uses a fixed-priority based arbitration mechanism that provides timing guarantees and that is amenable to timing analysis [11].

Today, distributed real-time systems become more and more complex and the number of micro controllers attached to CAN buses continue to grow. CAN's maximum speed of 1 Mbps remains, however, fixed; leading to performance bottlenecks. This bottleneck is further accentuated by the growing computing power of CPUs. Studies have shown that CAN's priority mechanism allows for lower network utilization than the Earliest Deadline First (EDF) mechanism [6]. Hence, in order to reclaim some of the scarce bandwidth forfeited by CAN's native scheduling mechanism, novel approaches to scheduling CAN are needed.

In optimising the design of a CAN-based communication system (and essentially any other real-time communication system) it is important to both guarantee the timeliness of periodic messages and to minimize the interference from this traffic on the transmission of aperiodic messages. Therefore, in this paper we propose the usage of an EDF server-based scheduling technique, which improves on existing techniques, since: (1) Fairness among the messages is guaranteed (i.e., "misbehaving" aperiodic processes cannot starve well-behaved processes), and (2) in contrast with other proposals, aperiodic messages are not sent "in the background" of periodic messages or in separate time-slots [5]. Instead, all messages are jointly scheduled using servers. This substantially facilitates meeting response-time requirements for both aperiodic and periodic messages.

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, (2) Time-driven, and (3) Share-driven. For CAN, *priority-driven* scheduling is the most natural scheduling method since it is supported by the CAN protocol, and FPS response-time tests for determining the schedulability of CAN message frames have been presented by Tindell *et al.* [11]. This analysis is based on the standard fixed-priority response-time analysis for CPU scheduling presented by Audsley *et al.* [3]. TT-CAN [7] provides *time-driven* scheduling for CAN, and Almeida *et al.* present Flexible Time-Triggered CAN (FTT-CAN) [2], which supports priority-driven scheduling in combination with time-driven scheduling. The server-based scheduling presented in this paper provides the first share-driven scheduling approach for CAN. By providing the option of share-driven scheduling of CAN, designers are given more freedom in designing an application.

As a side effect, by using servers, the CAN identifiers assigned to messages will not play a significant role in the message response-time. This greatly simplifies the process of assigning message 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.

Outline: In Section 2 we present the server scheduling, and in Section 3 we discuss an approach to analysis. We evaluate the server scheduling in Section 4 and conclude in Section 5.

## 2 Server-Based Scheduling on CAN

In order to provide bandwidth isolation for CAN, we propose the usage of server-based scheduling techniques.By 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 server 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 communicated by message passing, since this would further reduce the already low bandwidth offered by CAN. Our method, presented below, will provide a solution to this.

## 2.1 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 we will have multiple servers mediating access to a single CAN bus.

A server has one or more *users*. A user is typically a process or a task that requires access to the resource associated with the server. In this paper, a user is a stream of messages that is to be sent on the CAN bus. Typically, messages are associated with an arrival pattern. For instance, a message can arrive periodically, aperiodically, or it can have a sporadic arrival pattern. The server associated to the message handles each arrival of a message.

In the scheduling literature many types of servers are described. Using FPS, for instance, the Sporadic Server (SS) is presented by Sprunt *et al.* [9]. SS has a fixed priority chosen according to the Rate Monotonic (RM) policy. Using EDF, Spuri and Buttazzo [10] extended SS to Dynamic Sporadic Server (DSS). Other EDF-based schedulers are the Constant Bandwidth Server (CBS) presented by Abeni [1], and the Total Bandwidth Server (TBS) by Spuri and Buttazzo [10]. 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.

In this paper we will describe a general framework for server scheduling of the CAN bus. As an example we will use a simplified version of TBS. A TBS, $s$, is characterized by the variable $U_s$, which is the server utilization factor, i.e., its allocated bandwidth. When the *n*th request arrives to server $s$ at time $r_n$, it will be assigned the deadline $d_n = \max(r_n, d_{n-1}) + \frac{C_n}{U_s}$ where $C_n$ is the resource demand (can be execution time or, as in this paper, message transmission time). The initial deadline is $d_0 = 0$.

### 2.1.1 Server Characterization

Each node on the CAN bus will be 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 assumed 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.

### 2.1.2 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 is according to the following 3 cases: (1) when an *idle* server receives message $n$ at time $r_n$ it becomes *active* and the server deadline is set to $d_s^n = \max(r_n + D_s, d_s^{n-1})$, (2) when an *active* server sends a message and still has more messages to send, the server deadline is updated to $d_s^n = d_s^{n-1} + D_s$, and (3) when an *active* server sends a message and has no more messages to send, the server becomes *idle*.

## 2.2 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 [6], although in this paper the master’s responsibilities are a bit different. Here the M-Server is responsible for keeping track of the state of each N-Server and for allocating 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 (e.g., initially each N-Server $s$ is assigned a deadline $d_s = D_s$). Later we will see how to handle erroneous guesses.

In fact, the N-Servers’ complete state is contained within the M-Server. Hence, the other nodes do not maintain N-Server states, they only have 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, and 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 that are allowed to send one message during the EC. Upon reception of a TM, the N-Servers that were granted permission will send their message. The messages will 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, from the reception of a granting TM, the delay of message transmissions will be bounded by 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. One way of doing this is to send a stop message (STOP) with the lowest possible priority. After sending STOP to the CAN controller, the M-Server reads all messages sent on the bus. When it reads STOP it knows that all N-Servers have sent their messages, since by the CAN arbitration mechanism all higher priority messages will be sent before STOP.

After reading STOP 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. The following section describes how this is done.

#### 2.2.1 Updating N-Server States

At this point it is possible for the M-Server to verify whether or not its initial guess that all 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 2.1.2, and (2) the N-Server did not send a message. In this case the guess was incorrect and 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 2.1.2.

#### 2.2.2 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, and (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.

## 3 Approach to Analysis

The server-based scheduling proposed in this paper provides a high level of Quality-of-Service (QoS), in the sense that an N-Server, $s$, almost always delivers its messages within the bound $T_s + T_{EC}$. A condition for providing this QoS is that the N-Servers in the system have a total utilisation that is less than the theoretical upper bound for this protocol as presented in [4].

### 3.1 Message Delivery

Since the deadline of an N-Server may not be on the boundary between ECs and we have no control of message order within an EC it may be the case that an N-Server misses its deadline. Thus, even if an N-Server is scheduled within the EC where its deadline is, it may be the case that the N-Server misses its deadline with as much as $T_{EC}$.

Also affecting the message delivery time is the effectiveness of the M-Server's guesses about N-Server states. When the system is not fully utilised (e.g., when one or more N-Servers do not have any messages to send), the EC will terminate prematurely and cause a new EC to be triggered. This, in turn, increases the protocol overhead (since more TM and STOP messages are being sent). However, it should be noted that this increase in overhead only occurs due to unutilised resources in the system. Hence, when the system is fully utilised no erroneous guesses will be made and the protocol overhead is kept to a minimum.

However, when a system goes from being under-utilised to being fully utilised (for instance when a process that was sleeping is woken up and starts to send messages to its server) we may experience a *temporary* overload situation due to the protocol overhead. If the total system utilisation is less than the theoretical upper bound, then we are guaranteed that this overload will eventually be recovered. However, during the time it takes for the overload to be recovered the M-Server may be unable to schedule each N-Server in the EC where its deadline is. Hence, occasionally an N-Server may miss its deadline with as much as $2 \times T_{EC}$.

## 4 Evaluation

In order to evaluate the performance of our server approach we have performed simulations. We chose to perform two different experiments and, for each experiment, investigate three different scenarios. We have investigated both close to maximum usage of the bandwidth, and somewhat lower than maximum usage of the bandwidth. Hence, the difference between the two experiments is the total bandwidth usage by the N-Servers. For the details and figures from the experiments, please consult [4].

The results of the simulations are good, and the protocol manages erroneous guesses very well, especially when the network is not fully utilised. Since deadlines may occur inside an EC, it is natural that some messages are delivered at a time of $T_s + T_{EC}$, since we never know exactly when a specific message is sent inside an EC (as discussed Section 3.1). Therefore, occasionally, a message is the last message delivered within an EC, even though its corresponding N-Server's deadline is earlier.

With close to maximum usage of the bandwidth, some messages are delivered at a time of $T_s + 2 \times T_{EC}$. This is caused by bandwidth overload due to erroneous guesses, and messages have to be scheduled in a later EC than the one containing their N-Server's deadline (Section 3.1). However, this is a rare phenomenon and did in our experiments not occur with somewhat lower than maximum usage of the bandwidth. The worst result in our simulations were that 20 out of 13477 messages were delivered at a time of $T_s + 2 \times T_{EC}$. This result was obtained when the bandwidth utilisation was set very close to maximum.

## 5 Conclusions

In this paper we have presented a new approach for scheduling of the Controller Area Network (CAN). The difference between our approach and existing methods is that we make use of server-based scheduling (based on EDF). Our approach allows us to utilize the CAN bus in a more flexible way 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.

The strength of server-based scheduling for CAN, compared to other scheduling approaches, is that we can cope with 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, 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, since it needs to perform the extra work for scheduling. Also, compared with FTT-CAN, we are sending one more message, STOP, which is reducing the available bandwidth for the system under heavy aperiodic load. However, STOP is of the smallest size possible and therefore it should have minimal impact on the system. However, if the CAN controller is able to detect when the bus is idle (and pass this information to the master node), we could skip STOP, and the overhead caused by our protocol would decrease (since this would make it possible to use our server-based scheduling without STOP).

As we see it, 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 and aperiodic messages (although no fairness among aperiodic messages) and hard real-time, 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.

## 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

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

[2] L. Almeida, J. Fonseca, and P. Fonseca. 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)*, Magdeburg, Germany, September 1999.

[3] 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 Engineering Journal*, 8(5):284–292, September 1993.

[4] T. Nolte, M. Sjödin, and H. Hansson. Server-Based Scheduling of the CAN Bus. Technical report, ISSN 1404-3041 ISRN MDH-MRTC-99/2003-1-SE, Mälardalen Real-Time Research Centre, Mälardalen University, Sweden, April 2003.

[5] P. Pedreiras and L. Almeida. 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)*, pages 67–75, Porto, Portugal, September 2000. IEEE Industrial Electronics Society.

[6] P. Pedreiras and L. Almeida. A Practical Approach to EDF Scheduling on Controller Area Network. In *Proceedings of the IEEE/IEE Real-Time Embedded Systems Workshop (RTES'01) at the* $22^{nd}$ *IEEE Real-Time Systems Symposium (RTSS'01)*, London, England, December 2001.

[7] Road Vehicles - Controller Area Network (CAN) - Part 4: Time-Triggered Communication. International Standards Organisation (ISO). ISO Standard-11898-4, December 2000.

[8] Road Vehicles - Interchange of Digital Information - Controller Area Network (CAN) for High-Speed Communication. International Standards Organisation (ISO). ISO Standard-11898, Nov 1993.

[9] B. Sprunt, L. Sha, and J. Lehoczky. Aperiodic Task Scheduling for Hard Real-Time Systems. *Real-Time Systems*, 1(1):27–60, 1989.

[10] M. Spuri and G. Buttazzo. Efficient Aperiodic Service under Earliest Deadline Scheduling. In *Proceedings of the* $15^{th}$ *IEEE Real-Time Systems Symposium (RTSS'94)*, pages 2–11, San Juan, Puerto Rico, December 1994. IEEE Computer Society.

[11] K. W. Tindell, H. Hansson, and A. J. Wellings. Analysing Real-Time Communications: Controller Area Network (CAN). In *Proceedings of* $15^{th}$ *IEEE Real-Time Systems Symposium (RTSS'94)*, pages 259–263, San Juan, Puerto Rico, December 1994. IEEE Computer Society.