Server-Based Scheduling of the CAN Bus

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

In this paper we present a new share-driven server-based method for scheduling messages sent over the Controller Area Network (CAN). 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. Our server-based scheduling is based on Earliest Deadline First (EDF), which allows higher utilization of the network than using CAN's native fixed-priority scheduling approach.

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 they show that our method provides a Quality-of-Service (QoS), where virtually all messages are delivered within a specified time.

1 Introduction

The Controller Area Network (CAN) [18, 5] is widely used in automotive and other real-time applications. CAN uses a fixed-priority based arbitration mechanism that can provide timing guarantees and that is amenable to timing analysis [24, 25, 26]. However, studies have shown that CAN's fixed-priority scheduling (FPS) allows for lower network utilization than Earliest Deadline First (EDF) scheduling [10, 16].

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 steadily growing computing power of CPUs. 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 periodic traffic on the transmission of aperiodic messages.

Therefore, in this paper we propose the usage of *server-based scheduling techniques* (based on EDF) such as Total Bandwidth Server (TBS) [20, 22], or Constant Bandwidth Server (CBS) [1], which improves existing techniques since: (1) Fairness among users of a resource 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 [15]. Instead, aperiodic and periodic messages are jointly scheduled using servers. This substantially facilitates meeting response-time requirements, for both aperiodic and periodic messages.

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.

The paper is organized as follows: In Section 2 related work is presented. In Section 3 we present the server-based CAN network. Section 4 presents an approach to analysis, and in Section 5 the proposed method is evaluated using simulation. Finally, in Section 6 we conclude and present future work.

2 Background and Related Work

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

2.1 The Controller Area Network

The Controller Area Network (CAN) [18, 5] 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). 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 are simultaneously out-putting 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 one station transmit 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 priority based queue since, at all nodes, the message chosen during arbitration is always the active message with the highest priority.

2.2 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) [9].
2. Time-driven (table-driven) [8, 6].
3. Share-driven [14, 23].

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.* [24, 25, 26]. This analysis is based on the standard fixed-priority response-time analysis for CPU scheduling presented by Audsley *et al.* [4]. TT-CAN [17] provides *time-driven* scheduling for CAN, and Almeida *et al.* present Flexible Time-Triggered CAN (FTT-CAN) [2, 3], which supports priority-driven scheduling

in combination with time driven-scheduling. FTT-CAN is presented in more detail below. However, *share-driven* scheduling for CAN has not yet been investigated. 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.

2.2.1 Flexible Time-Triggered Scheduling

Pedreira and Almeida present a method to combine event-triggered traffic with time-triggered [15]. The approach is based on FTT-CAN (Flexible Time-Triggered communication on CAN) [2, 3]. 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, the aperiodic messages are sent, and in the second part, the synchronous window, traffic is sent according to the schedule delivered by the TM. More details of the EC layout are provided in Figure 1.

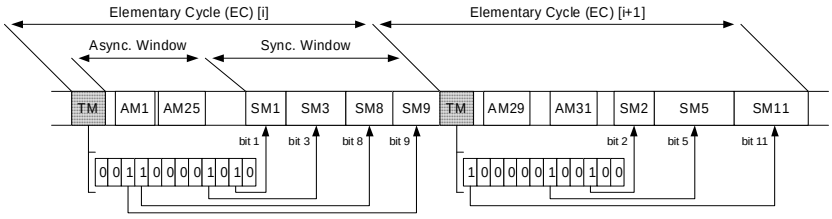

Figure 1. EC layout and TM data contents (FTT-CAN approach).

2.2.2 EDF Scheduling

As an alternative to the fixed-priority mechanisms offered by CAN, an approach for EDF was developed by Zuberi *et al.* [27]. They propose the usage of a Mixed Traffic Scheduler (MTS), which attempts to give a high utilization (like EDF) while using CAN's 11-bit identifiers for arbitration. Using the MTS, the message identifiers are manipulated in order to reflect the current deadline of each message. However, since each message is required to have a unique message identifier, they suggested the division of the identifier field into three sub-fields.

Other suggestions for scheduling CAN according to EDF include the work by Livani *et al.* [10] and Di Natale [11]. These solutions are all based on manipulating the identifier of the CAN 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 [7].

Using FTT-CAN, Pedreiras and Almeida [16] show how it is possible to send periodic messages according to EDF using the synchronous window of FTT-CAN. Their method is based on periodic message sets with fixed deadlines. Pedreiras and Almeida have also developed a method for calculating the worst-case response-time of the messages using the asynchronous window [15]. Using their approach, greater flexibility is achieved since the scheduling is not based on manipulating the identifiers. Instead, there is a master node performing the scheduling of the CAN bus.

The drawback of all these methods, for achieving EDF, is that they require each message to have a fixed, *a priori* known, deadline. Thus, using these methods, it is impossible to implement a server-based scheduler, since, when

using a server, the message deadlines are not *a priori* known, but assigned by the server at the time of message arrival.

3 Server-Based Scheduling on CAN

In order to provide guaranteed network bandwidth for real-time messages, we propose the usage of server-based scheduling techniques instead of the previously proposed methods described above. We have previously studied CBS end-to-end system design [13], and we also gave a brief presentation of a CBS-based scheduling approach for CAN in [12]. In this paper we will take a more general approach to server-based scheduling, and describe the basic mechanisms in detail.

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 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.

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 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.* [19]. SS has a fixed priority chosen according to the Rate Monotonic (RM) policy. Using EDF, Spuri and Buttazzo [20, 21] 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 [20, 22]. 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 a deadline according to

$$d_n = \max(r_n, d_{n-1}) + \frac{C_n}{U_s} \tag{1}$$

where C_n is the resource demand (can be execution time or, as in this paper, message transmission time). The initial deadline is $d_0 = 0$.

3.1.1 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. (For instance, our scheduling method, presented below, will under certain circumstances *miss* the server deadline. As we will show, 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 is 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
$$d_s^n = \max(r_n + D_s, d_s^{n-1}) \tag{2}$$
2. When an *active* server sends a message and still has more messages to send, the server deadline is updated according to
$$d_s^n = d_s^{n-1} + D_s \tag{3}$$
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 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 [16], 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 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 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. 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 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, 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. One way of doing this is to send a stop[1] message (STOP) with the lowest possible priority[2]. 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. Figure 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.

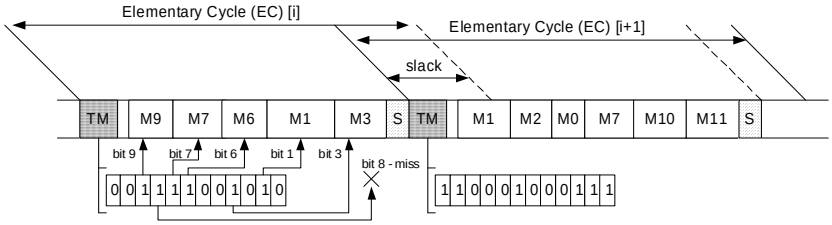

Figure 2. EC layout and TM data contents (server approach).

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.
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

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 3.1.2.
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 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] A small delay before sending STOP is required. 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.

[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, by using a STOP message we are able to use standard CAN controllers.

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.

4 Approach to Analysis

The server-based scheduling proposed in this paper provides a high level of Quality-of-Service (QoS), in the sense that N-Servers, s, almost always deliver their 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 fulfils inequality 4. We can not allocate N-Servers with a total utilisation higher than inequality 4, since such an allocation of N-Servers could cause the system to be overloaded. Hence, the total utilisation of the system is not allowed to exceed

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

where S is the network speed in bits/second, M is the length of a message (typically worst-case which is 135 bits), T_s is the period of the N-Server, and T_{EC} is the length of the EC in seconds. TM and STOP are the sizes of the TM and the STOP messages in bits, typically 135 and 55 bits.

4.1 Message Delivery

Due to the nature of scheduling with ECs, we never know exactly when inside an EC the message is delivered. This is because all messages allowed to be sent within an EC will be sent to the CAN controllers, where the CAN arbitration mechanism decides the order in which the messages will be delivered.

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 inequality 4 holds for the system 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 $2 \times T_{EC}$.

5 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.

	Experiment 1	Experiment 2
Network speed (bits/ms)	125	125
EC size (in messages)	5	4
EC period (including TM & STOP) (ms)	6.92	5.84
Message transmission time (ms)	1.08	1.08
TM transmission time (ms)	1.08	1.08
STOP transmission time (ms)	0.44	0.44
Number of N-Servers	15	15
Maximum utilisation (inequality 4)	0.780347	0.739726
Utilisation of simulation (N-Servers)	0.614244	0.727838
Utilisation of simulation (% of maximum)	78.71	93.27
Simulation time (ms)	20000	20000

Table 1. Properties of the two experiments.

Each simulation was executed for 20000 milliseconds (ms) and all message response times were measured. The properties of the simulations are summarised in Table 1.

5.1 Scenario 1

In this scenario only 2 of the totally 15 N-Servers are having messages to send. N-Server 14 has one message to send every server period. N-Server 1 also has one message to send every server period from time 7500 to time 12500. Both N-Servers deliver their messages within their periods. The result of the first experiment is shown in Figure 3, where the N-Server periods (ms) are $T_1 = 11.68, T_{14} = 46.72$, and the result of the second experiment is shown in Figure 4, where the N-Server periods (ms) are $T_1 = 13.84, T_{14} = 55.36$.

What we see in this scenario is that even though we have a huge amount of erroneous guesses (since 13 of the N-Servers have no messages to send, the M-Server will always make an erroneous guess for them), the N-Servers which have messages to send are being served as intended, i.e., allowed to send a message every server period. Hence, the measured response-times never exceed the N-Server period. Note that the response-time for N-Server 14 decreases when N-Server 1 has messages to send. This is due to that the number of erroneous guesses is less, decreasing the overhead of the protocol.

5.2 Scenario 2

In this scenario all N-Servers, except N-Server 8, have a message to send in each server period. N-Server 8 has one message to send each server period from time 7500 to time 12500. The result of the first experiment is shown in Figure 5, where the period (ms) of N-Servers 1, 8, and 14 are $T_1 = 11.68, T_8 = 29.2, T_{14} = 46.72$. The result of the second experiment is shown in Figure 6, where the period (ms) of N-Servers 1, 8, and 14 are $T_1 = 13.84, T_8 = 34.6, T_{14} = 55.36$. For simplicity, only N-Servers 1, 8, and 14 are shown in the graph.

What we can see in the first experiment, is that even though the bandwidth usage is quite high, the bandwidth isolation property is kept and all three N-Servers deliver their messages within their respective T_s. Looking at the

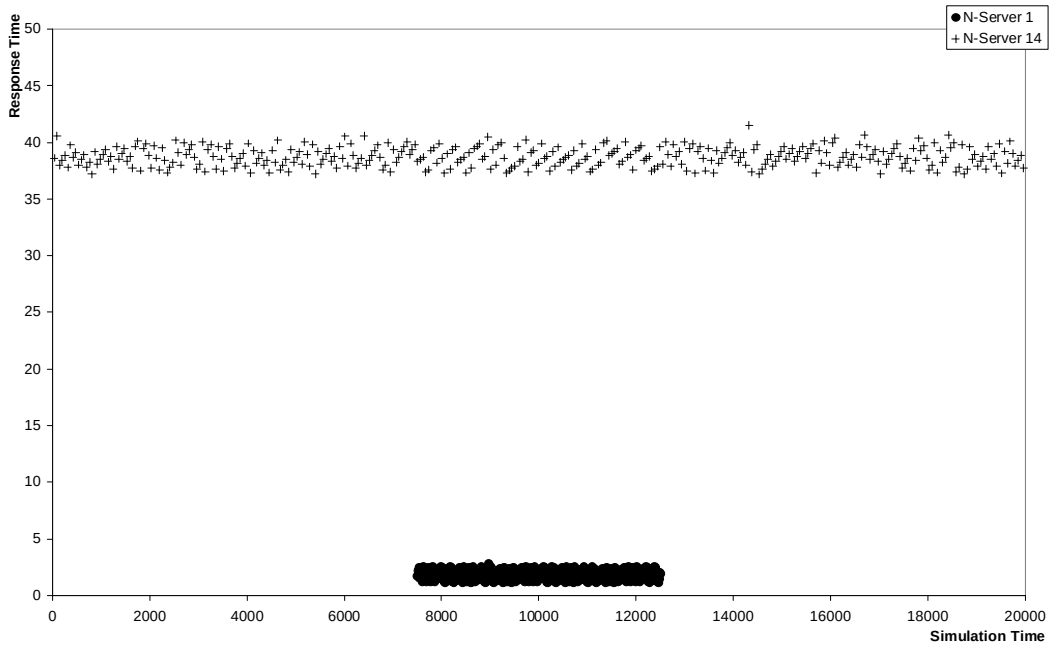

Figure 3. Experiment 1 (medium bandwidth usage) – scenario 1.

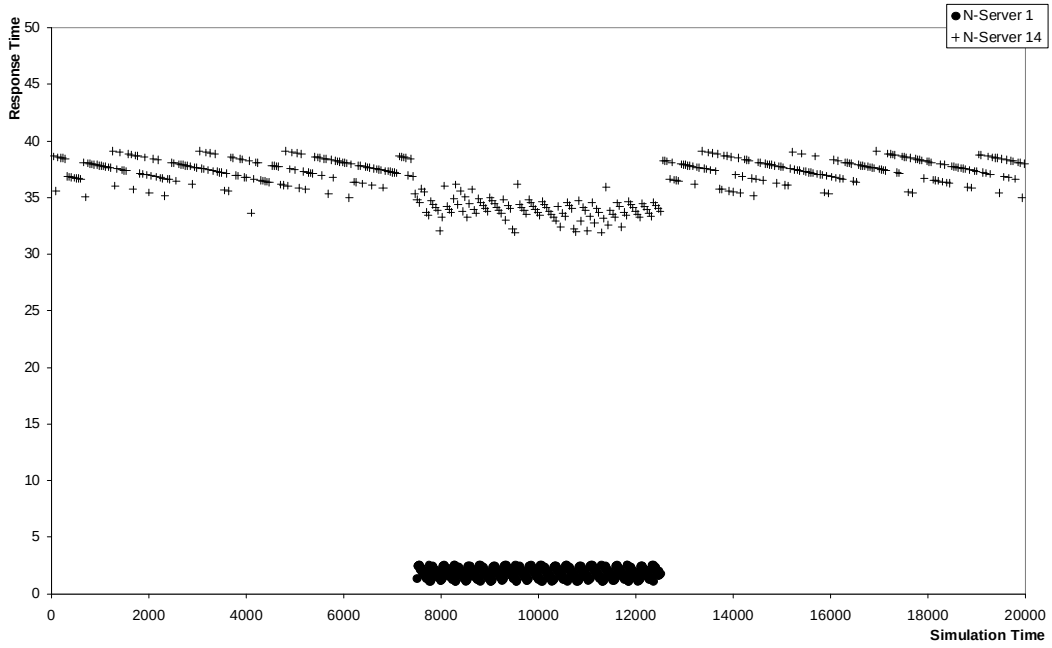

Figure 4. Experiment 2 (high bandwidth usage) – scenario 1.

whole set of N-Servers, only 3 of the servers deliver a total number of 5 messages (N-Server 5 and N-Server 7) as late as $T_s + T_{EC}$. In the whole simulation a total of 10947 messages were sent.

When running the second experiment, with the close to maximum bandwidth requirement, all N-Servers deliver messages as late as $T_s + T_{EC}$, and one of the servers (N-Server 4) deliver one message in $T_s + 2 \times T_{EC}$. We

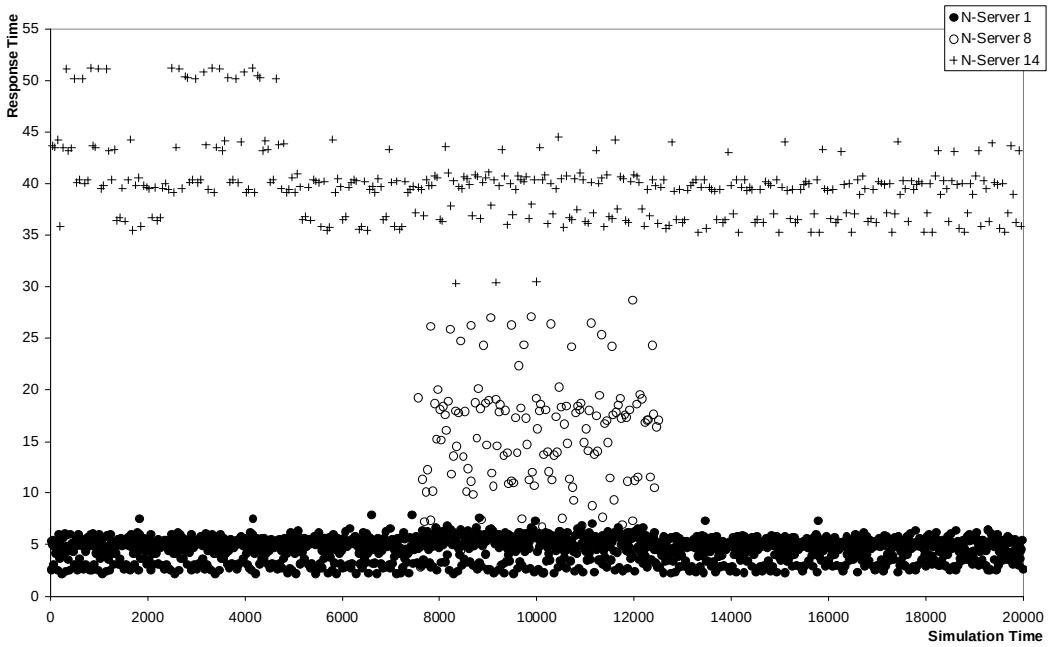

Figure 5. Experiment 1 (medium bandwidth usage) – scenario 2.

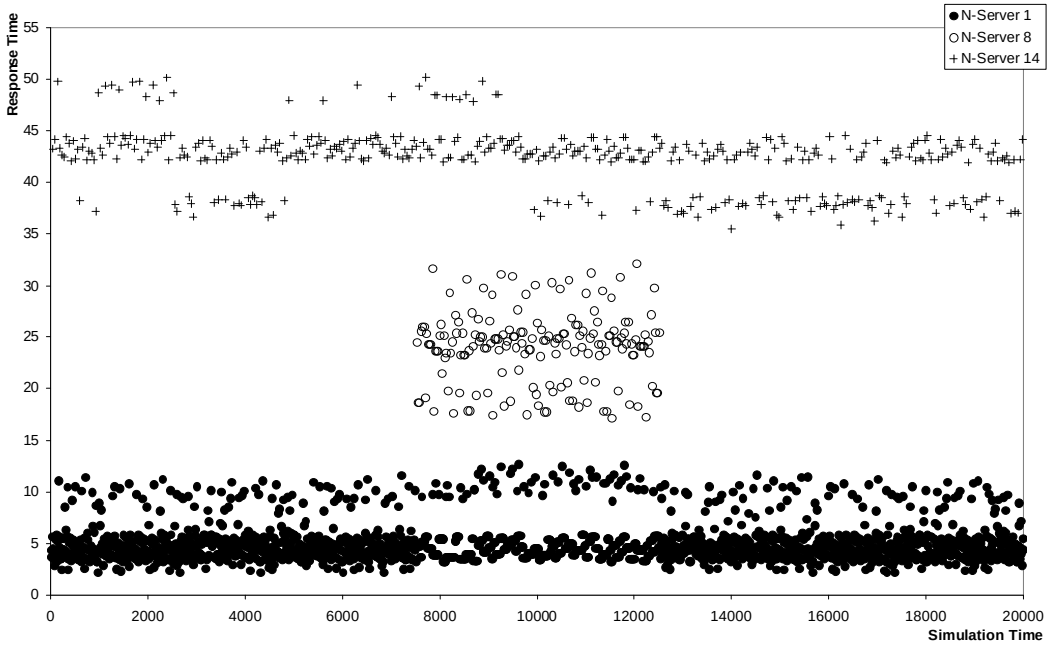

Figure 6. Experiment 2 (high bandwidth usage) – scenario 2.

believe this is due to the increased protocol overhead caused by erroneous guesses done by the M-Server regarding message readiness, as described in Section 4. Note that only 1 message from a total of 12964 messages was delivered as late as $T_s + 2 \times T_{EC}$.

5.3 Scenario 3

In this scenario all N-Servers have a message to send in each server period. In the first experiment, which consisted of a total of 11380 messages, only 3 messages (all from N-Server 10) were delivered at a time later than T_s. The 3 late messages were all delivered in $T_s + T_{EC}$, i.e., the EC following their server's deadline.

When running the second experiment, with the close to maximum bandwidth requirement, all N-Servers have some messages delivered in an EC following the server's deadline. Also, a total of 20 messages (the simulation had a total of 13477 messages) deliver a message in $T_s + 2 \times T_{EC}$.

5.4 Discussion

The data obtained from all 3 scenarios under both experiments is presented in Table 2, where S is the scenario (1-3), T_s is the N-Server period, WCR is the worst-case measured response-time, BCR is the best-case measured response-time, n is the number of messages sent through the N-Server, "$+1$" is the number of messages which were delivered in $T_s + T_{EC}$, and "$+2$" is the number of messages which were delivered in $T_s + 2 \times T_{EC}$.

Since deadlines 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 in Section 4.1). Therefore, occasionally, a message is the last message delivered within an EC, even though its corresponding N-Server's deadline is earlier.

For the second experiment, 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 (as discussed in Section 4.1). However, this is a rare phenomenon that only occurs in the second experiment when the bandwidth demand by the N-Servers is near to the theoretical maximum expressed by inequality 4.

6 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 Earliest Deadline First (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, the STOP message, which is reducing the available bandwidth for the system under heavy aperiodic load. However, the STOP message 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 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).

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

		Experiment 1						Experiment 2					
N-Server	S	T_s	WCR	BCR	n	$+1$	$+2$	T_s	WCR	BCR	n	$+1$	$+2$
0	1	13.84	0	0	0	0	0	11.68	0	0	0	0	0
	2	13.84	9.00	3.24	1446	0	0	11.68	12.56	3.24	1713	3	0
	3	13.84	9.04	3.24	1446	0	0	11.68	12.52	4.36	1713	23	0
1	1	13.84	2.84	1.12	362	0	0	11.68	2.60	1.12	429	0	0
	2	13.84	7.92	2.16	1446	0	0	11.68	12.68	2.16	1713	11	0
	3	13.84	7.96	2.16	1446	0	0	11.68	12.72	3.28	1713	48	0
2	1	13.84	0	0	0	0	0	11.68	0	0	0	0	0
	2	13.84	6.84	1.08	1446	0	0	11.68	12.64	1.08	1712	16	0
	3	13.84	6.88	1.08	1446	0	0	11.68	12.76	2.20	1712	57	0
3	1	20.76	0	0	0	0	0	17.52	0	0	0	0	0
	2	20.76	10.72	1.64	964	0	0	17.52	18.40	1.12	1142	1	0
	3	20.76	11.40	2.16	964	0	0	17.52	24.20	1.16	1141	262	14
4	1	20.76	0	0	0	0	0	17.52	0	0	0	0	0
	2	20.76	11.84	1.08	964	0	0	17.52	23.68	4.84	1141	257	1
	3	20.76	10.84	1.08	964	0	0	17.52	25.08	5.60	1141	319	2
5	1	27.68	0	0	0	0	0	23.36	0	0	0	0	0
	2	27.68	28.12	1.76	723	4	0	23.36	27.88	9.92	856	87	0
	3	27.68	27.60	2.76	723	0	0	23.36	29.88	7.08	856	146	3
6	1	27.68	0	0	0	0	0	23.36	0	0	0	0	0
	2	27.68	27.04	4.68	723	0	0	23.36	29.00	12.52	856	51	0
	3	27.68	26.52	1.68	723	0	0	23.36	28.84	10.24	856	110	0
7	1	34.60	0	0	0	0	0	29.20	0	0	0	0	0
	2	34.60	34.68	8.44	578	1	0	29.20	33.24	15.04	685	21	0
	3	34.60	34.20	9.24	578	0	0	29.20	34.76	12.80	685	34	0
8	1	34.60	0	0	0	0	0	29.20	0	0	0	0	0
	2	34.60	28.76	2.88	145	0	0	29.20	32.16	17.20	172	16	0
	3	34.60	33.12	8.16	578	0	0	29.20	35.00	15.68	685	71	0
9	1	41.52	0	0	0	0	0	35.04	0	0	0	0	0
	2	41.52	40.80	25.00	482	0	0	35.04	39.64	22.36	570	27	0
	3	41.52	40.84	26.60	482	0	0	35.04	39.32	20.80	570	36	0
10	1	41.52	0	0	0	0	0	35.04	0	0	0	0	0
	2	41.52	39.72	25.76	482	0	0	35.04	40.12	24.96	570	52	0
	3	41.52	43.80	25.52	482	3	0	35.04	39.68	24.48	570	75	0
11	1	48.44	0	0	0	0	0	40.88	0	0	0	0	0
	2	48.44	47.52	21.96	413	0	0	40.88	44.72	27.48	489	15	0
	3	48.44	47.60	23.08	413	0	0	40.88	47.08	24.60	489	18	1
12	1	48.44	0	0	0	0	0	40.88	0	0	0	0	0
	2	48.44	46.44	22.6	413	0	0	40.88	44.12	29.24	489	23	0
	3	48.44	46.52	22.00	413	0	0	40.88	46.00	27.20	489	45	0
13	1	55.36	0	0	0	0	0	46.72	0	0	0	0	0
	2	55.36	52.32	29.44	361	0	0	46.72	49.16	32.92	428	2	0
	3	55.36	52.36	30.36	361	0	0	46.72	51.40	30.32	428	17	0
14	1	55.36	41.52	37.2	361	0	0	46.72	39.12	31.92	428	0	0
	2	55.36	51.24	30.36	361	0	0	46.72	50.20	35.52	428	28	0
	3	55.36	51.28	29.28	361	0	0	46.72	50.32	34.08	428	51	0

Table 2. Summary of simulation results.

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.

However, the approach presented in this paper is not suitable for handling background traffic, since all bandwidth is allocated for the proposed protocol. Traditionally, background traffic could be assigned priority lower than real-time traffic. Hence, traffic without real-time demands could use unused bandwidth without interfering with the real-time traffic.

One future direction is to provide an upper bound on message delivery. Moreover, we want to investigate whether unused bandwidth may be shared among servers. Also it would be interesting to see 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.

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. Flexible Time-Triggered Communication on a Controller Area Network. In *Proceedings of the Work-In-Progress Session of the* 19^{th} *IEEE Real-Time Systems Symposium (RTSS'98)*, Madrid, Spain, December 1998. IEEE Computer Society.

[3] 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.

[4] 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.

[5] CAN Specification 2.0, Part-A and Part-B. CAN in Automation (CiA), Am Weichselgarten 26, D-91058 Erlangen. http://www.can-cia.de/, 2002.

[6] C.-W. Hsueh and K.-J. Lin. An Optimal Pinwheel Scheduler Using the Single-Number Reduction Technique. In *Proceedings of the* 17^{th} *IEEE Real-Time Systems Symposium (RTSS'96)*, pages 196–205, Los Alamitos, CA, USA, December 1996. IEEE Computer Society.

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

[8] H. Kopetz. The Time-Triggered Model of Computation. In *Proceedings of the* 19^{th} *IEEE Real-Time Systems Symposium (RTSS'98)*, pages 168–177, Madrid, Spain, December 1998. IEEE Computer Society.

[9] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *Journal of the ACM*, 20(1):40–61, 1973.

[10] M. Livani and J. Kaiser. EDF Consensus on CAN Bus Access for Dynamic Real-Time Applications. In *Proceedings of the* 6^{th} *International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS'98)*, Orlando, Florida, USA, March 1998.

[11] M. D. Natale. Scheduling the CAN Bus with Earliest Deadline Techniques. In *Proceedings of the* 21^{st} *IEEE Real-Time Systems Symposium (RTSS'00)*, pages 259–268, Orlando, Florida, USA, November 2000. IEEE Computer Society.

[12] T. Nolte, H. Hansson, and M. Sjödin. Efficient and Fair Scheduling of Periodic and Aperiodic Messages on CAN Using EDF and Constant Bandwidth Servers. Technical Report ISSN 1404-3041 ISRN MDH-MRTC-73/2002-1-SE, Mälardalen Real-Time Research Centre, Mälardalen University, Sweden, May 2002.

[13] T. Nolte and K.-J. Lin. 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)*, pages 355–360, Taipei, Taiwan, ROC, December 2002. IEEE Computer Society.

[14] A. Parekh and R. Gallager. A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case. *IEEE/ACM Transactions on Networking*, 1(3):344–357, June 1993.

[15] 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.

[16] 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.

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

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

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

[20] 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.

[21] M. Spuri and G. Buttazzo. Scheduling Aperiodic Tasks in Dynamic Priority Systems. *Real-Time Systems*, 10(2):179–210, March 1996.

[22] M. Spuri, G. C. Buttazzo, and F. Sensini. Robust Aperiodic Scheduling under Dynamic Priority Systems. In *Proceedings of the* 16^{th} *IEEE Real-Time Systems Symposium (RTSS'95)*, pages 210–219, Pisa, Italy, December 1995. IEEE Computer Society.

[23] 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 *Proceedings of* 17^{th} *IEEE Real-Time Systems Symposium (RTSS'96)*, pages 288–299, Los Alamitos, CA, USA, December 1996. IEEE Computer Society.

[24] K. W. Tindell and A. Burns. Guaranteed Message Latencies for Distributed Safety-Critical Hard Real-Time Control Networks. Technical Report YCS 229, Dept. of Computer Science, University of York, York, England, June 1994.

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

[26] 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.

[27] K. Zuberi and K. Shin. 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)*, pages 240–249, Chicago, IL, USA, May 1995. IEEE Computer Society.