Mälardalen University Licentiate Thesis
No.10

Reducing Pessimism and Increasing Flexibility in the Controller Area Network

Thomas Nolte

May 2003

Department of Computer Science and Engineering
Mälardalen University
Västerås, Sweden

ISBN 91-88834-48-4
Printed by Arkitektkopia, Västerås, Sweden
Distribution: Mälardalen University Press

Abstract

The Controller Area Network (CAN) is a widely used real-time communication network for automotive and other embedded applications. As new applications continue to evolve, the complexity of distributed CAN based systems increase. However, CAN's maximum speed of 1 Mbps remains fixed, leading to performance bottlenecks. In order to make full use of this scarce bandwidth, methods for increasing the achievable utilisation are needed.

Traditionally, real-time scheduling theory has targeted hard real-time systems, which most of the time are safety critical. Since these systems (by definition) are not allowed to have any timing flaws, analysis techniques need to take all possible scenarios of execution combinations and execution times of the system into consideration. This will result in a system that is configured for the worst possible scenario. Whether this scenario is likely, or even possible, in the real system is not considered. Hence, the result may be an unnecessarily expensive system, with potentially overly provisioned resources.

In this thesis we address two issues. In the first part, we investigate how to loosen up pessimistic real-time analysis in a controlled way, thereby allowing the designer to make well-founded trade-offs between the level of real-time guarantee and the system cost. Specifically, we investigate and model the bit-stuffing mechanism in CAN in order to retrieve representative distributions of stuff-bits, which we then use in the response time analysis instead of the worst-case values normally used. We evaluate the validity of these stuff-bit distributions in two case studies, and we integrate this representation of message frame length with the classical CAN worst-case response-time analysis.

In the second part of the thesis, we propose a novel way of scheduling the CAN. By providing server-based scheduling, bandwidth isolation between users is guaranteed. This increases the flexibility of CAN, by providing efficient handling of sporadic and aperiodic message streams. Server-based scheduling also has the potential to allow higher network utilisation compared to CAN's native scheduling. The performance and properties of server-based scheduling of CAN is evaluated using simulation. Also, the server-based scheduling is applied in an end-to-end analysis.

Keywords: controller area network, CAN, real-time communication, real-time analysis, reliability trade-off analysis, bit-stuffing, server-based scheduling

To Karin and Jürgen

Preface

The journey that has resulted in, among other things, this thesis started on a hot summer day in Västerås, back in 2000. I was currently doing my Bachelors degree project at Adtranz (now Bombardier Transportation), evaluating Java in VxWorks as a suitable HMI platform. Going on my daily lunch down-town, I ran into my supervisor at the time, Christer Norström. During lunch together he presented the idea of a continuation of my studies towards a Ph.D. degree. At that time the idea of postgraduate studies had never crossed my mind, but since that day, almost 3 years ago, I know that I could not have made a better choice. Thank you Christer, for giving me this opportunity.

The work presented in this thesis would not have been possible without the help of my supervisors Hans Hansson and Christer Norström. You have both believed in me, supported my work, and inspired me throughout these years. Thank you Hans for all help and your impressive work capacity with short notice. Thanks also to Mikael Sjödin for a huge amount of help with structuring and reading of this thesis.

Thank you Kwei-Jay Lin for all help and support during my stay at University of California, Irvine. I had a really nice time during my five-month period there. I learned a lot and met many nice friends: Sue, Angelo, Aki, and Andy. Also, thank you Kwei-Jay Lin for arranging and for taking care of me in Taiwan. Taipei is one of the most interesting places I have visited up until today.

Also, I must thank some colleagues that I have met at various conferences throughout the world. Mille grazie Lucia Lo Bello, for constructive reviewing and help on my initial paper. Others who have inspired me a lot are Luís Almeida, Alan Burns, and Iain Bate.

A big thank you goes to my study-partner Dag Nyström, who still manages

to share office space with me. We started together in 1997 with very different backgrounds, ending up studying daily together for 4 years, starting as Ph.D. students together, and now, another 2 years later, defending our licentiate theses the same day.

It has been a pleasure working at IDt, with all colleagues, especially Joel Huselius, Anders Pettersson, Damir Isović, Daniel Sundmark, Tomas Lennvall, Radu Dobrin, and Jonas Neander. Thank you Harriet Ekwall for support in all sorts of practical issues. When I was constructing this list of people that I wish to thank, I was putting virtually all names of the people working here at IDt on it, together with a bunch of other people, so, instead of giving the complete list, Thank you all!

Finally, and most importantly, I thank my family and my friends for supporting me through these years, and I thank Carla for her love.

This work has been supported by the Swedish Foundation for Strategic Research (SSF) via the research programme ARTES, the Swedish Foundation for Knowledge and Competence Development (KK-stiftelsen), LM Ericsson's Research Foundation, and Mälardalen University.

Tusen tack!

Thomas Nolte, Västerås, April 2003.

Contents

List of Publications

The following articles are included in this licentiate[1] thesis:

A. *Using Bit-Stuffing Distributions in CAN Analysis*, Thomas Nolte, Hans Hansson, Christer Norström, and Sasikumar Punnekkat[2], In Proceedings of IEEE/IEE Real-Time Embedded Systems Workshop (RTES'01), London, England, December 2001.

B. *Minimizing CAN Response-Time Jitter by Message Manipulation*, Thomas Nolte, Hans Hansson, and Christer Norström, In Proceedings of the 8^{th} IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'02), San Jose, CA, USA, September 2002.

C. *Probabilistic Worst-Case Response-Time Analysis for the Controller Area Network*, Thomas Nolte, Hans Hansson, and Christer Norström, In Proceedings of the 9^{th} IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'03), Toronto, Canada, May 2003.

D. *Server-Based Scheduling of the CAN Bus*, Thomas Nolte, Mikael Sjödin, and Hans Hansson, Technical Report ISSN 1404-3041 ISRN MDH-MRTC-99/2003-1-SE, Mälardalen Real-Time Research Centre, Mälardalen University, Sweden, April 2003.

E. *Distributed Real-Time System Design using CBS-based End-to-end Scheduling*, Thomas Nolte and Kwei-Jay Lin[3], In Proceedings of the 9^{th} IEEE International Conference on Parallel and Distributed Systems (ICPADS'02), Taipei, Taiwan, ROC, December 2002.

[1] A licentiate degree is a Swedish graduate degree halfway between M.Sc. and Ph.D.

[2] Vikram Sarabhai Space Centre, Trivandrum, India.

[3] Department of Electrical and Computer Engineering, University of California, Irvine, CA 92697, USA.

Besides the above articles, I have (co-)authored the following scientific papers:

I. *Integrating Reliability and Timing Analysis of CAN-based Systems*, Hans Hansson, Thomas Nolte, Christer Norström, and Sasikumar Punnekkat, In IEEE Transactions on Industrial Electronics, 49(6), December 2002.

II. *Effects of Varying Phasings of Message Queuings in CAN-based Systems*, Thomas Nolte, Hans Hansson, and Christer Norström, In Proceedings of 8^{th} International Conference on Real-Time Computing Systems and Applications (RTCSA'02), Tokyo, Japan, March 2002.

III. *Modeling and Analysis of Message-Queues in Multi-Tasking Systems*, Thomas Nolte and Hans Hansson, In Proceedings of ARTES Real-Time Graduate Student Conference, Lund, Sweden, March 2001.

IV. *Probabilistic Worst-Case Response-Time Analysis for the Controller Area Network*, Thomas Nolte, Hans Hansson, and Christer Norström, Technical Report ISSN 1404-3041 ISRN MDH-MRTC-77/2002-1-SE, Mälardalen Real-Time Research Centre, Mälardalen University, Sweden, November 2002.

V. *Efficient and Fair Scheduling of Periodic and Aperiodic Messages on CAN Using EDF and Constant Bandwidth Servers*, Thomas Nolte, Hans Hansson, and Mikael Sjödin, Technical Report ISSN 1404-3041 ISRN MDH-MRTC-73/2002-1-SE, Mälardalen Real-Time Research Centre, Mälardalen University, Sweden, May 2002.

VI. *Reducing Pessimism in CAN Response-Time Analysis*, Thomas Nolte, Technical Report ISSN 1404-3041 ISRN MDH-MRTC-51/2002-1-SE, Mälardalen Real-Time Research Centre, Mälardalen University, Sweden, March 2002.

VII. *Minimizing CAN Response-Time Jitter by Message Manipulation*, Thomas Nolte, Hans Hansson, and Christer Norström, Technical Report ISSN 1404-3041 ISRN MDH-MRTC-52/2002-1-SE, Mälardalen Real-Time Research Centre, Mälardalen University, Sweden, March 2002.

VIII. *Effects of Varying Phasings of Message Queuings in CAN-based Systems*, Thomas Nolte, Hans Hansson, and Christer Norström, Technical Report ISSN 1404-3041 ISRN MDH-MRTC-44/2001-1-SE, Mälardalen Real-Time Research Centre, Mälardalen University, Sweden, December 2001.

I

Thesis

Chapter 1

Introduction

"Real-time systems are defined as those systems for which the correctness depends not only on the logical result of the computation, but also on the time at which the results are produced", Stankovic [44].

The Controller Area Network (CAN) [10, 37] is a widely used fieldbus in automotive and other real-time applications. CAN uses a fixed-priority based arbitration mechanism that can provide real-time guarantees and that is amenable to timing analysis.

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 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, methods for increasing the achievable utilisation are needed, e.g., novel analysis methods that allow increased utilisation while guaranteeing timing requirements to be fulfilled, and novel approaches to schedule CAN.

1.1 Problem Formulation

In this thesis we address the following two issues:

1. Probabilistic modelling of the bit-stuffing mechanism of CAN – to make timing analysis of CAN less pessimistic.

2. Server-based scheduling of CAN – to increase the flexibility and utilisation when scheduling CAN.

The first problem is addressed by investigating and modelling the bit-stuffing mechanism in CAN in order to retrieve some representative distributions of stuff-bits. By using these distributions, instead of the worst-case values traditionally used, the level of pessimism in the analysis is significantly lowered. Hence, it will be possible to make trade-offs between the level of pessimism and reliability.

To understand the motivation of this kind of trade-off, consider traditional real-time scheduling theory, which targets safety-critical hard real-time systems. Since these systems (by definition) are not allowed to have any timing flaws, analysis techniques need to take all possible scenarios of execution combinations and execution times into consideration. This results in a system that is configured for the worst possible scenario. Whether this scenario is likely, or even possible, in the real system is not considered. Given that the model assumptions are correct, this is fine from a strict hard real-time perspective, but will result in a more expensive system (in terms of resources and hardware), compared to if we can allow a small probability of a deadline miss.

In this thesis we will investigate how to loosen up the real-time analysis in a controlled way, thereby allowing the designer to make well-founded trade-offs between the level of real-time guarantee and the resource needs (system cost). There are several motivations for this work, including:

- By also considering reliability, there is always some small probability of system failure, regardless of the rigor of the schedulability analysis, i.e., it is fair to say that “there is no such thing as a hard real-time system”.

  Reliability is defined as the probability that a system can perform its intended function, under given conditions, for a given time interval. Missing a deadline in a hard-real time system is, just as a hardware failure, a violation of the intended behaviour. Hence, the probability of missing a deadline is for such a system an important aspect to consider when calculating the overall system reliability. The aspects that need to be considered in calculating the overall reliability, e.g., for a communication system (like CAN), are depicted in Figure 1.1.

  The presented link between timing guarantees and reliability forms a basis for making trade-offs between the two, i.e., we could allow some deadline misses as long as their effect on the system reliability does not invalidate the overall system reliability requirement.

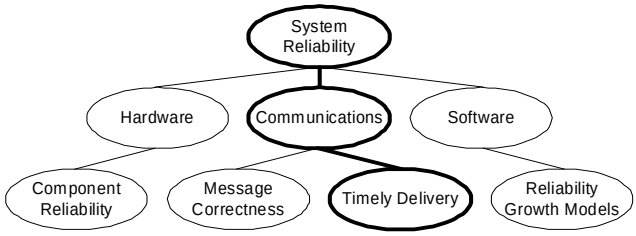

Figure 1.1: System reliability: a top-down view.

- More and more applications have real-time demands, although not as safety-critical as traditional real-time systems, e.g., audio and video transmissions. These new applications open the door to loosen up timing requirements a bit, thus allowing some deadlines to be violated on rare occasions. By using probability distributions of values instead of worst-case values, designers are given more freedom in terms of arguments for making well-founded trade-offs between, e.g., resource requirements and timeliness.

For the second part of the thesis we consider server-based scheduling. By providing server-based scheduling, bandwidth isolation between users of a resource is guaranteed. This allows for effective handling of sporadic and aperiodic message streams. However, in order to implement server-based scheduling on CAN, new mechanisms to provide information to servers are needed. When the server is scheduling the messages, it must know when they are arriving at the different nodes in the distributed system in order to make the correct scheduling decisions. This information, however, should not be communicated by message passing, since this would further reduce the already low bandwidth offered by CAN. Instead, we present a novel and efficient approach that is based on the characteristics of CAN message frame arbitration.

1.2 General Applicability

In this thesis we address two issues. The first issue is related to the estimation of an overall system reliability. As presented in Figure 1.1, several parts contribute to the overall reliability of a system. We propose a solution to the communication part, applied to CAN. However, using distributions of values

instead of worst-case values is applicable not only to CAN and communication systems. It can be used generally to more accurately capture the behaviour of essentially any system. Thereby allowing for greater degree of trade-offs between timing and reliability.

For the second issue addressed in this thesis (to increase the flexibility of CAN scheduling), the mechanisms proposed can be directly applied to any priority-queue based system or subsystem. However, to stay focused, we only implement these scheduling mechanisms on CAN.

1.3 Outline

The outline of this thesis is as follows: Chapter 2 starts by explaining and presenting basic real-time concepts that will be used throughout this thesis, and in Chapter 3 CAN is described together with relevant and related work. The technical contribution of this thesis is presented in Chapter 4, and in Chapter 5 we present our conclusions and suggestions for future work. Finally, included papers are appended in chapters 6 - 10.

Chapter 2

Basic Concepts

In real-time systems, not only the correct logical result of a computation is of importance. Real-time systems are additionally looking into at which time the results of these computations are produced. Over the last decades, various analysis techniques have been developed to determine at what time, in a worst-case scenario, a specific computation is completed.

Violating the predetermined properties in terms of *timeliness* can result in a bad or sometimes catastrophic scenario. For example, consider an airbag in a car. The purpose of an airbag system is to inflate a bag of gas that will catch the driver of a car in case of a collision. The airbag is not to be inflated too early or too late, since this could cause injury to the driver. A too early inflation will allow too much gas to leave the airbag before the driver is thrown into it. Moreover, a too late inflation will cause the driver to smash into the steering wheel before the airbag is exploding in his face. Only the correct timing will cause the airbag to work as intended.

In the area of *real-time computing*, we have a real-time system running one or several *real-time programs*. The real-time system is usually connected to some *input devices*, and based on the input from these devices it is *controlling a system*. A real-time program consists of a set of *tasks* that are executing in some resource-constrained environment. Several tasks may share the same processor. The goal of the real-time system is to function as intended in all possible scenarios, most importantly, also during peak-load.

The real-time system can either be executing on a single CPU or it can be distributed over several computing nodes. The latter type of real-time system is the main focus of this thesis. We will assume a system in which each node

is executing a number of tasks that are communicating using a fieldbus[1] type of communication network. We define these systems as Distributed Computer Control Systems (DCCS). An overview of key issues in a DCCS is presented in Figure 2.1 (Figure taken from Tovar [51]). This figure identifies the 9 components that contribute to the worst-case response-time of a task in network node A (task 1). The role of this task is to obtain data from a process sensor located in the network node B. To do that, firstly task 1 has to be scheduled together with the other tasks on host processor A. Once task 1 is scheduled for execution, and it has executed until completion (1), its message is queued (2) in the communications adapter A before it is sent on the shared broadcast bus (3). When the message is received on node B, the communications adapter B is putting the message in the arrival-queue (4) before it can be consumed by task 3 (which must be scheduled (5) together with the other tasks on host processor B). Then, the procedure is repeated (6-9) for the response.

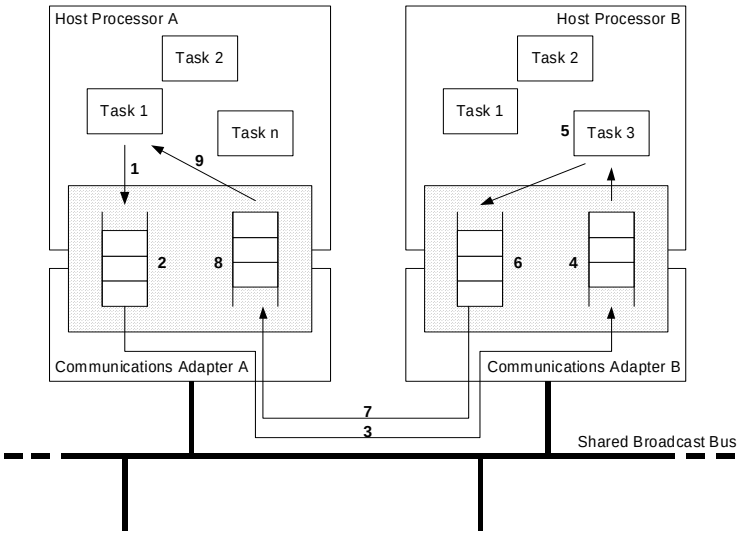

Figure 2.1: A Distributed Computer Control System (DCCS).

[1] Examples of fieldbuses include Controller Area Network (CAN), Process Network (P-NET), Process Field Bus (PROFIBUS), and Factory Instrumentation Protocol (FIP, later WorldFIP).

2.1 Real-Time

A real-time system consists of a set of real-time programs, which in turn consists of a set of *tasks*. These tasks are executing in an environment with limited resources. The tasks have different constraints in terms of time, e.g., *execution times*, *periods*, and *deadlines*. Several tasks can be executing on the same processor, i.e., sharing the same processor. Important issues to determine is whether all tasks can execute as planned during peak-load. By calculating the *response-time* of the tasks in the worst-case scenarios (at peak-load) we can determine if they will fulfil this requirement.

A real-time system can be *pre-emptive* or *non pre-emptive*. In a pre-emptive system, tasks can pre-empt each other, always letting the task with the highest priority execute. In a non pre-emptive system a task that has been allowed to start will always execute until its completion. The difference between pre-emptive and non pre-emptive execution is shown in Figure 2.2. Here, two tasks, task A and task B, are executing on a node. Task A has higher priority than task B. Task B arrives before task A. Scenarios for both non pre-emptive execution (a) and pre-emptive execution (b) are shown in the figure.

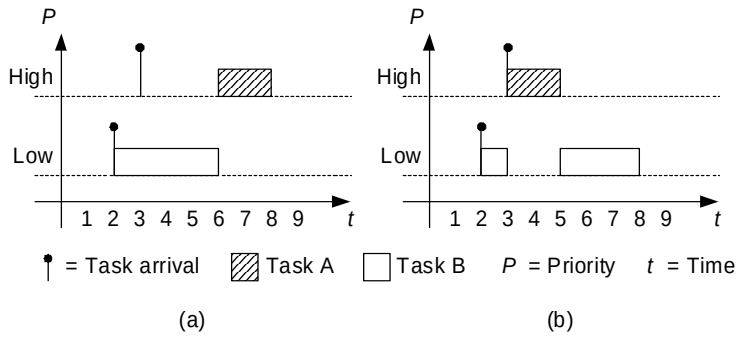

Figure 2.2: Task execution in a non pre-emptive (a) and a pre-emptive (b) system.

Tasks can be further categorised into either being *periodic*, *sporadic*, or *aperiodic*. Periodic tasks are executing periodically with a specified time (period) between task releases. Aperiodic tasks have no information saying when the task is to execute and usually they are triggered by interrupts, whereas sporadic tasks, although having no period, have a known minimum inter-arrival

time. The difference between periodic, sporadic, and aperiodic task arrivals is illustrated in Figure 2.3. The difference between sporadic and aperiodic tasks is that we have a known minimum inter-arrival time for the sporadic tasks, whereas the aperiodic tasks have no known inter-arrival time. In Figure 2.3 the periodic task has a period equal to 2, i.e., inter-arrival time is 2, the sporadic task has a minimum inter-arrival time of 1, and the aperiodic task has no known inter-arrival time.

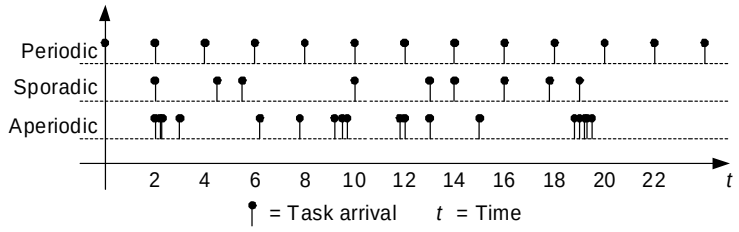

Figure 2.3: Periodic, sporadic, and aperiodic task arrival.

Classifying a task into periodic, sporadic, or aperiodic is done based on its purpose. Typically, measuring tasks are set to be periodic since they are collecting some value(s) every *n*th time unit. A sporadic task is typically reacting to an event that we know has a minimum inter-arrival time, e.g., an alarm or the emergency shut down of a production robot. The minimum inter-arrival time can be constrained by physical laws, or it can be enforced by some hardware mechanism. If we do not know the minimum time between two consecutive events, we must classify the event-handling task to be aperiodic.

Moreover, the task can be classified as a task with *hard*, *soft*, or *safety-critical* real-time requirements. Hard real-time tasks have high demands on their ability to fulfil their properties in terms of timeliness, and violation of these requirements may have severe consequences. If the violation is catastrophic, the task is classified as safety-critical. The failure of a safety-critical real-time task can cause loss of human life. However, many tasks have real-time demands although violation of these is not so severe, and in some cases violation can be tolerated. Examples of real-time systems including such tasks are robust control systems and systems that contain audio/video streaming.

A central problem when dealing with real-time systems is to determine how long time a real-time task will execute, in the worst case. The task is usually assumed to have a worst-case execution time (*WCET*). This is a field of

research in its own, which has not yet come to consensus, although there exists several techniques for estimation of the worst-case execution time [35, 12].

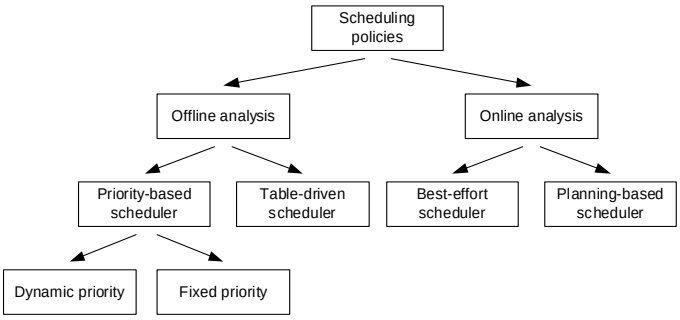

Figure 2.4: Different types of schedulability analysis.

A real-time *scheduler* schedules the real-time tasks sharing the same resources. The goal of the scheduler is to fulfil the requirements of these tasks in terms of timeliness. There are several different policies determining the behaviour of a real-time scheduler. The scheduler decides, based on the available task constraints, which task is to execute or to use the resource desired. *Schedulability analysis* is used to determine whether or not a set of tasks can share a resource without violating any task-constraints, e.g., deadlines. The theory of real-time schedulability analysis can be divided into two major groups; *online schedulability analysis* and *offline schedulability analysis*, as illustrated in Figure 2.4. Using online schedulability analysis, the schedulability of the task-set is determined at run-time, whereas when using offline-schedulability analysis, the schedulability of the task-set is done prior to run-time.

Scheduling policies that use online schedulability analysis can be classified as either *planning-based schedulers* or *best-effort schedulers*. Planning-based schedulers try to re-define the schedule upon arrival of a new task. If it finds the new task-set to be feasible, the new task can be accepted. Otherwise the new task will be rejected. Best-effort schedulers accept all new tasks and tries to perform as good as possible in terms of fulfilling task-constraints.

The schedulability analysis can be made offline for *table-driven schedulers* and *priority-based schedulers*. The table-driven schedulers create a schedule (the table) before the system is started, and are most often used in applications that have very high safety-critical demands. At runtime, a *dispatcher* follows

the schedule. Therefore the behaviour of table-driven schedulers is very deterministic. However, since the schedule is created offline, the *flexibility is very limited*, in the sense that as soon as the system will change (due to, e.g., adding of functionality or change of hardware), a new schedule has to be created and given to the dispatcher. Using priority-based schedulers, the *flexibility is increased*, since the schedule is created online, based on the current task-constraints, and it can cope with some changes, as long as the schedulability of the task-set is not violated. However, the exact behaviour of priority-based schedulers is harder to predict. Therefore, these schedulers are not used as often in the most safety-critical applications.

Two common priority-based scheduling policies are *Fixed Priority Scheduling (FPS)* and *Earliest Deadline First (EDF)*. The difference between these scheduling policies is whether the priorities of the real-time tasks are fixed or if they can change during execution.

In FPS, the task with the highest priority is scheduled for execution, and priorities can be arbitrary assigned to the tasks before execution. However, it can be proven that some priority assignments are better than others. For instance, for a simple task model with strictly periodic non-interfering tasks with deadlines equal to the period of the task, a Rate Monotonic (RM) priority assignment has been shown by Liu and Layland [22] to be optimal, in the sense that if this priority assignment leads to any missed deadline, then any other priority assignment will also lead to some missed deadline(s). In RM, the priority is assigned based on the period of the task. The shorter the period is, the higher priority will be assigned to the task.

In FPS, the priorities are fixed. Using EDF, the task with the nearest (earliest) deadline among all available tasks is selected for execution. Therefore the priority is not fixed, it changes with time.

In order for the priority-based schedulers to cope with aperiodic tasks, different service methods have been presented. The objective of these service methods are to give a good average response-time for aperiodic requests, while preserving the requirements demanded by hard real-time tasks. These services are implemented as *servers*. In the scheduling literature many types of servers are described. Using FPS, for instance, the Sporadic Server (SS) is presented by Sprunt *et al.* [39]. SS has a fixed priority chosen according to the RM policy. Using EDF, Spuri and Buttazzo [42, 43] 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 [42, 41]. 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.

The work in this thesis is strictly based on priority-based schedulers. Hence, schedulability analysis methods for these types of schedulers are presented in the following section.

2.2 Analysis for Priority-Based Schedulers

Basically, there are two different approaches for pre-run-time analysis of the offline scheduling approaches (schedulability tests). The first approach is based on the utilisation of the task-set under analysis (utilisation-based tests), and the second approach is based on calculating the worst-case response-time for each task in the task-set (response-time tests). However, utilisation-based tests can not be used for complicated task models, as shown by Tindell *et al.* [46, 48].

The task model notation used throughout this chapter is presented in Table 2.1.

Number of tasks in the task set	N
WCET	C
Period	T
Blocking-time	B
Response-time	R
Task under analysis	i
Task j has priority higher than task i	$j \in hp(i)$
Task k has priority lower than or equal to task i	$k \in lep(i)$

Table 2.1: Task model notation.

2.2.1 Utilisation-Based Tests

In [22], Liu and Layland present a utilisation-based test for the RM priority assignment. The task model they use consists of independent periodic tasks with deadline equal to their periods. If the test succeeds, the tasks will always meet their deadlines. The test is as follows:

$$\sum_{i=1}^{N} \frac{C_i}{T_i} \leq N \times \left(2^{1/N} - 1\right) \tag{2.1}$$

This test only guarantees that a task-set will not violate its deadlines if it passes this test. However, there are task-sets that may not pass the test, yet they will meet all their deadlines. Later on, Lehoczky developed an exact analysis [19]. However, the test developed by Lehoczky is a much more complex inequality compared to (2.1).

Inequality (2.1) has been extended in various ways, e.g., by Sha *et al.* [38] to also cover blocking-time, when higher priority tasks are blocked by lower priority ones.

Liu and Layland [22] also present a utilisation-based test for EDF (assuming the same type of task model as for RM):

$$\sum_{i=1}^{N} \frac{C_i}{T_i} \leq 1 \tag{2.2}$$

This inequality is a necessary and sufficient condition for the task-set to be schedulable. Baker [6] extends inequality (2.2) to also include blocking periods, and deadlines shorter than the period. Additional extensions include work by Zheng [52] to cover sporadic tasks.

2.2.2 Response-Time Tests

Joseph and Pandya presented the first response-time test for real-time systems [17]. They present a response-time test for pre-emptive fixed-priority systems. The worst-case response-time is calculated as follows:

$$R_i = I_i + C_i \tag{2.3}$$

where I_i is defined as:

$$I_i = \sum_{j \in hp(i)} \left(\left\lceil \frac{R_i}{T_j} \right\rceil \times C_j \right) \tag{2.4}$$

where $hp(i)$ is the set of tasks with higher priority than task i.

The worst-case response-time is found when all tasks are released synchronously at time 0, the so-called *critical instant*.

The processors level-*i* busy period is defined as the period preceding the completion of task *i*, i.e., the time in which task i and all other higher priority tasks still not yet have executed until completion. To solve Equation (2.4), it can be rewritten as:

$$R_i^{n+1} = \sum_{j \in hp(i)} \left(\left\lceil \frac{R_i^n}{T_j} \right\rceil \times C_j \right) + C_i \tag{2.5}$$

Note that equation (2.5) is a recurrence relation, where the approximation to the $(n+1)$th value is found in terms of the nth approximation. The first approximation is set to $R_i^0 = C_i$. A solution is reached when the $(n+1)$th value is equal to the nth, i.e., when $R_i^{n+1} = R_i^n$. The recurrence relation will terminate given that inequality (2.2) is fulfilled.

The work of Joseph and Pandya [17] has been extended by Audsley *et al.* [5] to also include a blocking factor and cover for the non pre-emptive fixed-priority context. The difference is that in a non pre-emptive system, the processors level-*i* busy period is not including the task i itself:

$$R_i^{n+1} = B_i + \sum_{j \in hp(i)} \left(\left\lceil \frac{R_i^n - C_i}{T_j} \right\rceil \times C_j \right) + C_i \tag{2.6}$$

where B_i is defined as follows:

$$B_i = \begin{cases} 0 & \text{if} \quad lep(i) = \emptyset, \\ \max\limits_{k \in lep(i)} \{C_k\} & \text{if} \quad lep(i) \neq \emptyset. \end{cases} \tag{2.7}$$

where $lep(i)$ is the set of tasks with priority less or equal than task i. The recurrence relation (2.6) is solved in the same way as equation (2.5). Note that we must re-define the critical instant. The maximum interference now occurs when task i and all other higher priority tasks are synchronously released just after the release of the longest lower priority task (other than task i).

In dynamic priority systems, the worst-case response-time for a task-set is not necessarily obtained considering the critical instant, i.e., releasing all tasks at time 0. Instead, Spuri [40] find the worst-case response-time in the processors deadline-*i* busy period. This is similar to the processors level-*i* busy period as presented in equation 2.5. However, in dynamic-priority systems the worst-case response-time can be found when all tasks, but *i*, are released at time 0. Then, we have to examine multiple scenarios when task *i* is released at time *a*.

For non pre-emptive EDF scheduling, George *et al.* [13] introduce an analysis where the deadline-*i* busy period is not including the task i.

2.2.3 Servers

In the second part of this thesis we use a simplified version of TBS, as well as CBS. The server assigns a deadline to an arriving task or message, which is then scheduled according to EDF. This deadline will be kept, if the total aperiodic utilization does not exceed what is allocated by the server. 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

$$d_n = \max(r_n, d_{n-1}) + \frac{C_n}{U_s} \tag{2.8}$$

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

2.3 Summary

In this chapter we presented basic concepts regarding real-time scheduling theory that will be used in the rest of the thesis.

Chapter 3

Related Work

In this chapter we present relevant related work that is the basis for the contributions of this thesis. A brief overview of fieldbuses is given and the Controller Area Network (CAN) is presented in some detail, followed by response-time analysis methods and higher-layer protocols running on top of CAN.

3.1 Fieldbuses

Fieldbuses are a family of factory communication networks that have evolved as a response to the demand to reduce cabling costs in factory automation systems. Moving from a situation in which every controller has its own cables connecting the sensors to the controller (parallel interface), to controllers sharing a bus (serial interface), costs could be cut and flexibility could be increased. Pushing for this evolution of technology was both the fact that the number of cables in the system increased as the number of sensors and actuators grew, together with controllers moving from being specialised with their own microchip, to sharing a microprocessor with other controllers. Fieldbuses were soon ready to handle the most demanding applications on the factory floor.

Several fieldbus technologies, usually very specialised, were developed by different companies to meet the demands of their applications. Not until the mid 90's the standardisation process stabilized. In 1993 CAN was standardised by the International Standardisation Organisation (ISO) [37].

3.2 The Controller Area Network

The Controller Area Network (CAN) [37, 10] 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 FPS type of analysis, Tindell *et al.* [47, 49, 50].

3.3 Classical Schedulability Analysis for CAN

CAN is a broadcast bus designed to operate at speeds of up to 1 Mbps. Data is transmitted in frames containing between 0 and 8 bytes of data and a number of control bits. Depending on the CAN format (standard or extended) the number of control bits are either 44 or 64. Between CAN frames sent on the bus, there is also a 3 bit inter-frame space. The standard CAN frame format (and the inter-frame space) is shown in Figure 3.1.

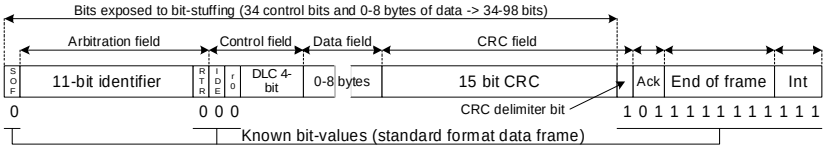

Figure 3.1: CAN frame layout (standard format data frame).

The difference between the standard and the extended format is that the extended format has 29 identifier bits instead of the 11 bits used in the standard format. The identifier is required to be unique, in the sense that two simultaneously active frames originating from different sources (i.e., nodes or CAN-controllers) must have distinct identifiers. The identifier serves two purposes: (1) assigning a priority to the frame, and (2) enabling receivers to filter frames. For a more detailed explanation of the different fields in the CAN frame, please consult [37, 10].

CAN is a collision-avoidance broadcast bus, which uses deterministic collision resolution to control access to the bus (so called CSMA/CA). The basis for the access mechanism is the electrical characteristics of a CAN bus: if multiple stations are transmitting concurrently and one station transmits a ‘0’ then all stations monitoring the bus will see a ‘0’. Conversely, only if all stations transmit a ‘1’ will all processors monitoring the bus see a ‘1’. During arbitra-

tion, competing stations are simultaneously putting their identifiers, one bit at the time, on the bus. By monitoring the resulting bus value, a station detects if there is a competing higher priority frame 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 queued frame, and hence can start transmitting the body of the frame.

3.3.1 Classical CAN Message Response-Time Analysis

Tindell *et al.* [47, 49, 50] present analysis to calculate the worst-case latencies of CAN frames. This analysis is based on the standard FPS response-time analysis for CPU scheduling presented by Audsley *et al.* [5] (Section 2.2.2).

The task model notation is used throughout this chapter is presented in Table 3.1.

Priority (defined by the message frame identifier)	P
Worst-case transmission time	C
Period	T
Blocking-time	B
Jitter	J
Response-time (latency)	R
Task under analysis	i
Task j has priority higher than task i	$j \in hp(i)$
Task k has priority lower than task i	$k \in lp(i)$

Table 3.1: Task model notation.

Calculating the response-times requires a bounded worst-case queuing pattern of frames. The standard way of expressing this is to assume a set of traffic streams, each generating frames with a fixed priority. The worst-case behaviour of each stream, in terms of network load, is to send as many frames as they are allowed, i.e., to periodically queue frames. In analogue with CPU scheduling, we obtain a model with a set $\mathcal{S}$ of streams (corresponding to CPU tasks). Each $S_i \in \mathcal{S}$ is a triple $< P_i, T_i, C_i >$. The worst-case latency R_i of a CAN frame sent on stream S_i is, if we assume the minimum variation in queuing time relative to T_i to be 0, defined by

$$R_i = J_i + q_i + C_i \tag{3.1}$$

where J_i is the queuing jitter of the frame, i.e., the maximum variation in queuing-time relative to the start of T_i, inherited from the sender task which queues the frame, and q_i represents the effective queuing-time, given by

$$q_i^n = B_i + \sum_{j \in hp(i)} \left\lceil \frac{q_i^{n-1} + J_j + \tau_{bit}}{T_j} \right\rceil (C_j + 3\tau_{bit}) \tag{3.2}$$

where

- $B_i = \max_{k \in lp(i)} (C_k) + 3\tau_{bit}$ is the worst-case blocking-time of frames sent on S_i. This definition of the blocking-time is the same as presented in Section 2.2.2. However, since identifiers are required to be unique, there are no message frames with priority equal to the message frame under analysis.
- τ_{bit} (the bit-time) caters for the difference in arbitration start-times at the different nodes due to propagation delays and protocol tolerances.
- C_j is the transmission time of message j. How to calculate C_j is presented in the Section 3.3.1.
- $3\tau_{bit}$ represents the inter-frame space (traditionally, Tindell *et al.* [47, 49, 50], the inter-frame space was considered a part of the data frame, but separating it removes a small source of pessimism in the equations, as pointed out by Broster *et al.* [7]).

Note that Equation 3.2 is a recurrence relation, where the approximation to the $(n+1)$th value is found in terms of the nth approximation, with the first approximation set to $q_i^0 = 0$. A solution is reached either when the $(n+1)$th value is equal to the nth, or when R_i exceeds its message deadline or period. The recurrence relation will terminate given that the total bus utilization is ≤ 1, i.e., $\sum_{S_i \in \mathcal{S}} \left(\frac{C_i + 3\tau_{bit}}{T_i} \right) \leq 1$.

Bit-Stuffing Mechanism

In CAN, six consecutive bits of the same polarity (111111 or 000000) are used for error and protocol control signalling. To avoid these special bit-patterns in transmitted frames, a bit of opposite polarity is inserted after five consecutive bits of the same polarity. By reversing the procedure, these bits are then removed at the receiver side. This technique, which is called *bit-stuffing*, implies

that the actual number of transmitted bits may be larger than the size of the original frame, corresponding to an additional transmission delay which needs to be considered in the analysis.

Let us first define the number of bits, beside the data part in the frame, that are exposed to the bit-stuffing mechanism as $g \in \{34, 54\}$. This since we have either 34 (CAN standard format) or 54 (CAN extended format) bits (besides the data part of the frame), which are exposed to the bit-stuffing mechanism. 10 bits in the CAN frame are not exposed to the bit-stuffing mechanism (see Figure 3.1). Now let us define the number of bytes of data in CAN message frame i as $L_i \in [0, 8]$. Recall, a CAN message frame can contain 0 to 8 bytes of data. Hence, the total number of bits in a CAN frame before bit-stuffing is

$$g + 10 + 8L_i \tag{3.3}$$

where 10 is the number of bits in the CAN frame not exposed to the bit-stuffing mechanism. Since only $g + 8L_i$ bits in the CAN frame are subject to bit-stuffing, the total number of bits after bit-stuffing can be no more than

$$g + 10 + 8L_i + \left\lfloor \frac{g + 8L_i - 1}{4} \right\rfloor \tag{3.4}$$

Intuitively the above formula captures the number of stuff-bits in the worst-case scenario, shown in Figure 3.2.

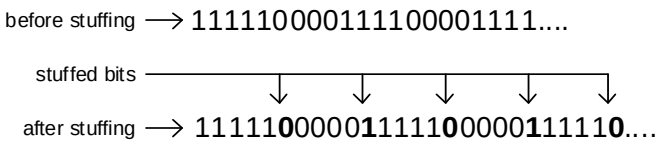

Figure 3.2: The worst-case scenario when stuffing bits.

Let τ_{bit} be the worst-case time taken to transmit a bit on the bus – the so-called *bit-time*. Hence, the worst-case time taken to transmit a given frame i is

$$C_i = \left(g + 10 + 8L_i + \left\lfloor \frac{g + 8L_i - 1}{4} \right\rfloor \right) \tau_{bit} \tag{3.5}$$

3.4 Extensions to the Classical Analysis

Several researchers have extended the classical response-time analysis for CAN, and in this section we give an overview of the most important work.

3.4.1 Reduction of Pessimism

Broster *et al.* [7] propose to reduce the pessimism in the original classical CAN analysis. The inter-frame space is separated from the CAN frame in order to remove a small source of pessimism by considering that the CAN frame is available as soon as the last bit of the previous frame is sent, rather than only after the inter-frame space following the completion of the sending of the previous frame.

3.4.2 Probabilistic Analysis

One of the contributions of this thesis is an extension of the classical analysis where we model the CAN frame length using distributions of stuff-bits. By using distributions of stuff-bits, instead of the worst-case value, pessimism is reduced, allowing for trade-offs regarding timeliness and reliability.

Reliability is defined as the probability that a system can perform its intended function, under given conditions, for a given time interval. A major issue is how to compose hardware reliability, software reliability, environmental model, and timely correctness to arrive at reasonable estimates of overall system reliability, as discussed in Section 1.1.

In [27] we present a model and a method that take the bit-stuffing mechanism into consideration by representing the number of stuff-bits as a distribution. We extended this work in [28] where whole CAN frames are covered. Finally, in [29] we put the result into the classical analysis presented in Section 3.3.1. The details of this work are presented in Chapter 6, Chapter 7, and Chapter 8.

Several probabilistic approaches for schedulability analysis of real-time systems have been presented, e.g., Atlas *et al.* [4] and Manolache [24]. However, none of these specifically address CAN.

3.4.3 Fault Models

In order to handle faults, the original analysis can be extended in order to handle the effect of errors in the channel. However, the fault model used by Tindell

et al. [49] is very simple and thus not really appropriate to describe real faults. Only strictly periodic interference is modelled. An extension to the original fault model is presented by Punnekkat *et al.* [34]. Here faults can be modelled as specific patterns. Basically, periodic bursts and multiple sources of interference can be modelled.

So far the fault models were based on an assumption of minimum inter-arrival time, i.e., bounded number of faults in a specified interval. However, several sources of interference, e.g., electromagnetic interference, are more accurately described as a random pulse train following a Poisson distribution [9]. Trying to represent this using minimum inter-arrival times is not easy. Rather, a probabilistic approach would be more suitable. Navet *et al.* [26] present a probabilistic fault model, where the faults are described as stochastic processes. These stochastic processes consider both the frequency of the faults and their gravity. Both single-bit faults and bursts of faults can be represented. However, the approach presented by Navet *et al.* [26] is very pessimistic. Broster *et al.* [8] present a more accurate probabilistic approach, where distributions of worst-case response-times can be obtained when there are probabilistic events in the system, e.g., faults.

Hansson *et al.* [14] present a completely different approach. Here the feasibility of the system is determined using simulation. Using simulation even more complex sources of interference can be used, achieving a more realistic result compared to the analytic approaches described above. However, the weakness of using simulation is to determine whether or not the coverage of the simulation is good enough for the application under evaluation.

3.5 Higher-Layer Protocols for CAN

Several higher-layer (3 and above) protocols have been developed for CAN. The reason for having a higher-layer protocol is most often to use another scheduling policy than the one offered by CAN. Original CAN is suitable to handle periodic real-time traffic according to the FPS approach. Limiting CAN to periodic traffic, the timing analysis can easily be applied and feasibility checked. However, due to the limitations of FPS scheduling, adaptations to allow other scheduling policies have been done. As an alternative to the fixed-priority mechanisms offered by CAN some higher-layer protocols have been developed to implement dynamic-priority schemes as well as time-triggered traffic.

3.5.1 EDF on CAN

As an alternative to the fixed-priority mechanisms offered by CAN, Zuberi *et al.* [53] developed an approach for EDF. They propose the usage of a Mixed Traffic Scheduler (MTS), which attempts to give a high utilisation (like EDF) while using CAN's 11-bit format for the identifier. 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.* [23] and Di Natale [25]. 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 [16].

Using FTT-CAN (described in next section), Pedreiras and Almeida [33] show how it is possible to send periodic messages according to EDF using the synchronous window of FTT-CAN. Also, they have developed a method for calculating the worst-case response-time of the messages using the asynchronous window [32]. 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 global scheduling of the CAN bus.

3.5.2 FTT-CAN

Almeida proposes Flexible Time Triggered communication on CAN (FTT-CAN) [2, 3] as a way to schedule CAN in a time-triggered fashion. In FTT-CAN, time is partitioned into Elementary Cycles (ECs) which are initiated by a special message, the Trigger Message (TM). This message triggers the start of the EC and it 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 into 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 3.3.

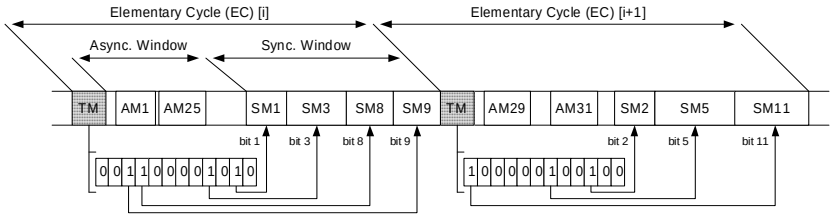

Figure 3.3: EC layout and TM data contents (FTT-CAN approach).

3.5.3 TT-CAN

Time-Triggered communication on CAN is specified by TT-CAN [36], a standardised session layer extension to original CAN. In TT-CAN, the exchange of messages is controlled by the temporal progression of time, and all nodes are following a pre-defined static schedule. One node, the master node, is periodically (or on the occurrence of a specific event) transmitting a specific message, the reference message, which acts as a reference in time. All nodes in the system are synchronising with this message, which gives a reference point in the temporal domain for the static schedule of the message transactions. The static schedule is based on a time division (TDA) scheme, where message exchanges may only occur during specific time slots or in time windows. Hence, the master's view of time is referred to as the network's global time.

TT-CAN appends a set of new features to the original CAN, and being standardised, several semiconductor vendors have started manufacturing TT-CAN compliant devices.

3.5.4 Server-Based Scheduling of CAN

Server-based scheduling [30] is one of the contributions of this thesis, presented in detail in Paper D (Chapter 9). The main motivation for server-based scheduling of CAN is that it provides fairness and bandwidth isolation among the users of the network. In server-based CAN, steams of messages are scheduled together on a master node. Time is partitioned into ECs, which are initiated by a TM (as with FTT-CAN), and the scheduling is performed on the master node based on EDF. Using server-based scheduling, we can schedule for unknown aperiodic and sporadic messages by assuming that they are arriving, and if me make an erroneous guess, we are not wasting any bandwidth.

3.6 Summary

In this section we presented an overview of relevant and related work. Since of the first part of the work presented in this thesis, the original response-time based schedulability analysis for CAN was presented. Moreover, we have presented extensions made to the classical analysis, such as reduction of pessimism, probabilistic analysis, and fault models. The second part of the work presented in this thesis proposes a higher-layer protocol to enable server-based scheduling of CAN. Hence, several other existing higher-layer protocols for CAN are presented.

Chapter 4

Technical Contributions

This chapter summarizes the main contributions of each paper in this thesis.

4.1 Part 1: Probabilistic Analysis

The bit-stuffing mechanism of CAN causes the message frame length to vary, depending on the original bit-pattern of the CAN frame (as presented in Section 3.3.1). This is not good, since firstly, it is a source of jitter that might have degrading affects on the performance of control systems, as discussed by Decotignie [11], and secondly, to include this behaviour into the analysis, a worst-case assumption is traditionally made [47, 49, 50]. This assumption is very pessimistic, as indicated by our experiments.

We have, in Paper A [27] and Paper B [28], addressed this pessimism and presented a new analysis in Paper C [29], using message frame length distributions that are taking the behaviour of the bit-stuffing mechanism into consideration. Also, in Paper A, we have presented a way to actually decrease the number of stuff-bits in a CAN frame.

4.1.1 Paper A (Chapter 6)

Thomas Nolte, Hans Hansson, Christer Norström, and Sasikumar Punnekkat[1], *Using Bit-Stuffing Distributions in CAN Analysis*, In Proceedings of IEEE/IEE

[1] Vikram Sarabhai Space Centre, Trivandrum, India.

Real-Time Embedded Systems Workshop (RTES'01), London, England, December 2001.

Summary This paper investigates the level of pessimism in the traditional schedulability analysis for CAN, caused by the bit-stuffing mechanism. We model the bit-stuffing mechanism and use bit-stuffing distributions instead of worst-case bit-stuffing. This allows us to obtain bus utilisation values more close to reality.

We introduce a model and a method that relaxes the pessimism of the worst-case bit-stuffing assumption, and we show the validity of our method by considering both an artificial traffic model and samples of real CAN traffic. Also, we propose a simple coding scheme that substantially reduces the number of stuff-bits in the considered real traffic. The coding scheme transforms the bit-pattern using the logical operation exclusive or (XOR). It should be noted that by using the XOR operation, it is not possible to completely eliminate bad bit-patterns in a general CAN message frame. However, it is possible to transform one type of bit-patterns to another type. Hence, if bad bit-patterns are common in an application, these can be transformed to more harmless ones using the XOR operation with a suiting bit-mask.

My contribution This work is based on joint ideas. I am the main author of this paper and have done essentially all work under supervision of, and in discussion with, my supervisors.

4.1.2 Paper B (Chapter 7)

Thomas Nolte, Hans Hansson, and Christer Norström, *Minimizing CAN Response-Time Jitter by Message Manipulation*, In Proceedings of the 8^{th} IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'02), San Jose, CA, USA, September 2002.

Summary Here the work presented in Paper A is extended to cover the whole CAN message frame. We show that by introducing some restrictions, such as a small reduction of available frame identifiers, the number of stuff-bits in a CAN frame can be reduced. To further reduce the number of stuff-bits in the worst case, we make use of the restriction of identifiers together with the work presented in Paper A. We show the actual penalty introduced by forbidding some frame identifiers, and we show the overall improvement by using these techniques in a small case-study.

My contribution The main ideas in this paper are mine, and I have done essentially all work under supervision of, and in discussion with, my supervisors.

4.1.3 Paper C (Chapter 8)

Thomas Nolte, Hans Hansson, and Christer Norström, *Probabilistic Worst-Case Response-Time Analysis for the Controller Area Network*, In Proceedings of the 9^{th} IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'03), Toronto, Canada, May 2003.

Summary The methods presented in Paper A and Paper B, to model the stuff-bits using distributions, is put into the framework of classical FPS response-time analysis presented by Tindell *et al.* [47, 49, 50]. By doing this, the modelled response-times will be less pessimistic compared to the traditional worst-case ones, and since we are lowering the bandwidth usage of the messages, we will also be able to schedule more messages in the same system. However, this comes at the cost of introducing some optimism, which can cause deadline failures. The actual probability of these failures to occur is selected to be satisfactory low for the desired system.

By using distributions of stuff-bits, instead of the worst-case number of stuff-bits, we obtain a distribution of response-times, allowing us to calculate less pessimistic (compared to traditional worst-case) response-times.

My contribution The ideas in this paper are mine, but originating from a remark by Alan Burns. I have done all work in this paper, aided by useful and constructive discussions with my supervisors.

4.2 Part 2: Scheduling

In the real-time scheduling research community there exists several different types of scheduling. We can divide the classical scheduling paradigms presented in Chapter 2 into the following 3 groups:

1. Priority-driven (e.g., FPS or EDF) [22].
2. Time-driven (table-driven) [18, 15].
3. Share-driven [31, 45].

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.* [47, 49, 50]. This analysis is based on the standard fixed-priority response-time analysis for CPU scheduling presented by Audsley *et al.* [5]. TT-CAN [36] 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. These scheduling methods are presented in more detail in Section 3.5. However, *share-driven* scheduling for CAN has not yet been investigated. The server-based scheduling presented in this thesis 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.

4.2.1 Paper D (Chapter 9)

Thomas Nolte, Mikael Sjödin, and Hans 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.

Summary One of the strongest properties of *server-based scheduling techniques* (based on EDF), such as Total Bandwidth Server (TBS) by Spuri *et al.* [42, 41], or Constant Bandwidth Server (CBS) by Abeni [1], is that fairness among message streams is guaranteed (i.e., "misbehaving" aperiodic processes cannot starve well-behaved processes).

Paper D presents a general approach to server-based scheduling for CAN. Using servers, the whole network will be scheduled as one resource, providing bandwidth isolation as well as fairness among the users of the communication network.

My contribution The paper is based on my ideas. I am the main author and have done essentially all work, but I have had a very constructive collaboration with Mikael Sjödin, in particular regarding the structuring of the paper. Hans Hansson has helped with constructive reviewing.

4.2.2 Paper E (Chapter 10)

Thomas Nolte and Kwei-Jay Lin[2], *Distributed Real-Time System Design using CBS-based End-to-end Scheduling*, In Proceedings of the 9^{th} IEEE International Conference on Parallel and Distributed Systems (ICPADS'02), Taipei, Taiwan, ROC, December 2002.

Summary This paper presents an implementation of the concepts of Paper D. Here an approach for end-to-end analysis is presented for a distributed real-time system design scheme using CBS. The scheduling scheme utilizes CBS to allocate both CPU shares and network bandwidth to a distributed real-time application when it arrives at the system. Hence, using the same scheduling paradigm for both resources we believe the system can have a more consistent scheduling objective and may achieve a tighter schedulability condition.

My contribution The paper is based on an idea of Kwei-Jay Lin. I am responsible for the CAN and server parts, and Kwei-Jay Lin for the rest.

[2]Department of Electrical and Computer Engineering, University of California, Irvine, CA 92697, USA.

Chapter 5

Conclusions

5.1 Summary

We have presented a new response-time analysis for the Controller Area Network (CAN), which take the CAN bit-stuffing mechanism into consideration. By representing the number of stuff-bits as a distribution, the level of pessimism is significantly lowered which allows for a greater degree of trade-offs between timing and reliability.

We have also presented a novel way of scheduling CAN. By providing server-based scheduling, bandwidth isolation between users of the communication network is guaranteed, which allows for efficient handling of sporadic and aperiodic message streams.

5.2 Directions for Future Work

The overall reliability of a system consists of several parts, as discussed in Section 1.1, and we have in this thesis addressed the communication part. A future direction would be to look into the branch covering software executing on the nodes, as well as the combination of software and communication. Preliminary work in that direction is presented in [14].

There are also more direct extensions to the work presented in the papers in this thesis. For instance, the work presented in Paper C (based on the work in Paper A and Paper B) only focuses on a single aspect, namely a probabilistic worst-case response-time, based on using bit-stuffing distributions. There are

other parameters, including execution times and phasings of message queuings, which have similar variations and effects on the response-time analysis. The analysis of Paper C could be extended to also cover these aspects.

Moreover, the analysis of Paper C has no fault model. Therefore, extending it with some of the work presented in Section 3.4.3 could be useful. This since, in order to use this type of timing analysis for safety-critical systems, faults must also be considered.

Regarding the server-based scheduling approach; when we have a system running the server-based scheduling of the CAN bus it would be interesting to investigate if it is possible to increase the performance in terms of meeting deadlines and bandwidth throughput by applying bandwidth sharing, as presented by Lipari *et al.* [21] and Lin *et al.* [20]. Since all state information kept to run the servers are located in one node in the distributed system (centralized approach), it is easy to exchange parameters between the servers. If one or more servers are using less than their reserved bandwidth, some of their bandwidth could be temporarily be given to some of the others.

Bibliography

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

[2] L. Almeida, J.A. 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.A. 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] A. Atlas and A. Bestavros. Statistical Rate Monotonic Scheduling. In *Proceedings of the* 19^{th} *IEEE Real-Time Systems Symposium (RTSS'98)*, pages 123–132, Madrid, Spain, December 1998. IEEE Computer Society.

[5] 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.

[6] T. Baker. Stack-Based Scheduling of Real-Time Processes. *Real-Time Systems*, 3(1):67–99, March 1991.

[7] I. Broster and A. Burns. Timely Use of the CAN Protocol in Critical Hard Real-Time Systems With Faults. In *Proceedings of the* 13^{th} *Euromicro Conference on Real-Time Systems (ECRTS'01)*, pages 95–102, Delft, The Netherlands, June 2001. IEEE Computer Society.

[8] I. Broster, A. Burns, and G. Rodrígues-Navas. Probabilistic Analysis of CAN with Faults. In *Proceedings of the* 23^{rd} *IEEE Real-Time Systems Symposium (RTSS'02)*, pages 269–278, Austin, Texas, USA, December 2002. IEEE Computer Society.

[9] M. J. Buckingham. *Noise in Electronic Devices and Systems*. Series in Electrical and Electronic Engineering, Ellil Horwood/Wiley, 1983.

[10] 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.

[11] J.D. Decotignie. Some Future Directions in Fieldbus Research and Development. In *Proceedings of the International Conference on Fieldbus Technology (FeT'99)*, Magdeburg, Germany, September 1999.

[12] J. Engblom. *Processor Pipelines and Static Worst-Case Execution Time Analysis*. PhD thesis, Uppsala University, Dept. of Information Technology, Box 337, Uppsala, Sweden, April 2002.

[13] L. George, N. Rivierre, and M. Spuri. Preemptive amd Non-Preemptive Real-Time Uni-Processor Scheduling. Rapport de Recherche RR-2966, INRIA, Le Chesnay Cedex, France, 1996.

[14] H. Hansson, T. Nolte, C. Norström, and S. Punnekkat. Integrating Reliability and Timing Analysis of CAN-based Systems. *IEEE Transaction on Industrial Electronics*, 49(6), December 2002.

[15] 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.

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

[17] M. Joseph and P. Pandya. Finding Response Times in a Real-Time System. *The Computer Journal - British Computer Society*, 29(5):390–395, October 1986.

[18] 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.

[19] J. P. Lehoczky. Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines. In *Proceedings of the* 11^{th} *IEEE Real-Time Systems Symposium (RTSS'90)*, pages 201–209, Lake Buena Vista, Florida, USA, December 1990. IEEE Computer Society.

[20] K.-J. Lin, S. Wang, T.-Y. Kuan, and S. Cho. The Implementation of Hierarchical CBS Schedulers in RED-Linux. In *Proceedings of the* 8^{th} *International Conference on Real-Time Computing Systems and Applications (RTCSA'02)*, pages 287–294, Tokyo, Japan, March 2002.

[21] G. Lipari and S. Baruah. A Hierarchical Extension to the Constant Bandwidth Server Framework. In *Proceedings of the* 7^{th} *IEEE Real-Time Technology and Applications Symposium (RTAS'01)*, pages 26–35, Taipei, Taiwan, ROC, May 2001. IEEE Computer Society.

[22] 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.

[23] 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.

[24] S. Manolache. Schedulability Analysis of Real-Time Systems with Stochastic Task Execution Times. Licentiate Thesis No. 985, Dept. of Computer and Information Science, IDA, Linköping University, Sweden, December 2002.

[25] M. Di 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.

[26] N. Navet, Y.-Q. Song, and F. Simonot. Worst-Case Deadline Failure Probability in Real-Time Applications Distributed over Controller Area Network. *Journal of Systems Architecture*, 7(46):607–617, September 2000.

[27] T. Nolte, H. Hansson, and C. Norström. Using Bit-Stuffing Distributions in CAN Analysis. 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.

[28] T. Nolte, H. Hansson, and C. Norström. Minimizing CAN Response-Time Analysis Jitter by Message Manipulation. In *Proceedings of the* 8^{th} *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'02)*, pages 197–206, San Jose, CA, USA, September 2002. IEEE Computer Society.

[29] T. Nolte, H. Hansson, and C. Norström. Probabilistic Worst-Case Response-Time Analysis for the Controller Area Network. In *Proceedings of the* 9^{th} *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'03)*, Toronto, Canada, May 2003. IEEE Computer Society.

[30] T. Nolte, M. Sjödin, and Hans 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.

[31] 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 Transactions on Networking*, 1(3):344–357, June 1993.

[32] 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.

[33] 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.

[34] S. Punnekkat, H. Hansson, and C. Norström. Response Time Analysis under Errors for CAN. In *Proceedings of the* 6^{th} *IEEE Real-Time Technology and Applications Symposium (RTAS'00)*, pages 258–265, Washington DC, USA, June 2000. IEEE Computer Society.

[35] P. Puschner and A. Burns. A Review of Worst-Case Execution-Time Analysis. *Real-Time Systems*, 18(2/3):115–128, May 2000.

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

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

[38] L. Sha, R. Rajkumar, and J.P. Lehoczky. Priority Inheritance Protocols: An Approach to Real-Time Synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, September 1990.

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

[40] M. Spuri. Analysis of Deadline Scheduled Real-Time Systems. Technical report, Rapport de Recherche RR-2772, INRIA, Le Chesnay Cedex, France, 1996.

[41] 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.

[42] M. Spuri and G.C. 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.

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

[44] J.A. Stankovic. Misconceptions About Real-Time Computing. *IEEE Computer*, 21(10):10–19, October 1988.

[45] 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.

[46] K. W. Tindell. An Extendible Approach for Analysing Fixed Priority Hard Real-Time Tasks. Technical Report YCS 189, Dept. of Computer Science, University of York, York, England, December 1992.

[47] 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.

[48] K. W. Tindell, A. Burns, and A. J. Wellings. An Extendible Approach for Analysing Fixed Priority Hard Real-Time Tasks. *Real-Time Systems*, 6(2):133–151, March 1994.

[49] 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.

[50] 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.

[51] E. Tovar. *Supporting Real-Time Communications with Standard Factory-Floor Networks*. PhD thesis, Faculdade de Engenharia da Universidade do Porto, July 1999.

[52] Q. Zheng. *Real-Time Fault-Tolerant Communication in Computer Networks*. PhD thesis, University of Michigan, Ann Arbor, MI, USA, 1993.

[53] K.M. Zuberi and K.G. 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.

II

Included Papers

Chapter 6

Paper A: Using Bit-Stuffing Distributions in CAN Analysis

Thomas Nolte, Hans Hansson, Christer Norström, and Sasikumar Punnekkat[1]
In Proceedings of IEEE/IEE Real-Time Embedded Systems Workshop (RTES'01), London, UK, December 2001.
Technical Report, Department of Computer Science, University of York.

[1]Vikram Sarabhai Space Centre, Trivandrum, India.

Abstract

This paper investigates the level of pessimism of traditional schedulability analysis for the Controller Area Network (CAN). Specifically, we investigate the effects of considering bit-stuffing distributions instead of worst-case bit-stuffing. This allows us to obtain bus utilisation values more close to reality. On the other hand, since our analysis is based on assumptions concerning distributions of stuff-bits, our response-times will only be met with some probability.

We introduce a model and a method, that relaxes the pessimism of the worst-case analysis, and we show the effect of our method by considering both an artificial traffic model and samples of real CAN traffic. Our conclusion from this investigation is that actual frame sizes, with a very high probability, is in the order of 10% smaller than the worst cases used in traditional analysis. Also, we propose a simple coding scheme that substantially reduces the number of stuff-bits in the considered real traffic.

6.1 Introduction

During the last decade real-time researchers have extended schedulability analysis to a mature technique which for non-trivial systems can be used to determine whether a set of tasks executing on a single CPU or in a distributed system will meet their deadlines or not [1, 2, 8, 12]. The essence of this analysis is to investigate if deadlines are met in a worst-case scenario. Whether this worst-case actually will occur during execution, or if it is likely to occur, is not normally considered.

In contrast with schedulability analysis, reliability modelling involves study of fault models, characterisation of distribution functions of faults and development of methods and tools for composing these distributions and models in estimating an overall reliability figure for the system.

This separation of deterministic (0/1) schedulability analysis and stochastic reliability analysis is a natural simplification of the total analysis. The schedulability analysis is, however, quite pessimistic, since it assumes that a missed deadline in the worst-case is equivalent to always missing the deadline for all instances of a task, whereas the stochastic analysis extend the knowledge of the system by providing information on how often a deadline is violated. Furthermore, the failure semantics could be extended allowing the system to miss some deadlines and still not classify it as a failure.

There are many other sources of pessimism in the analysis, including considering worst-case execution times and worst-case phasings of executions, as well as the usage of pessimistic fault models.

In our previous work [6], we proposed a model for calculating worst-case latencies of Controller Area Network (CAN) [7] frames (messages) under error assumptions. This model is pessimistic, in the sense that there are systems that the analysis determines unschedulable, even though deadlines will only be missed in extremely rare situations with pathological combinations of errors.

In [4, 5] we have reduced the level of pessimism by introducing a better fault model, and in [3] we also consider variable phasings between message queuing, in order to make the model more realistic.

In this paper we focus on another source of pessimism, namely bit-stuffing of CAN frames. We will use distributions of frame lengths after stuffing instead of the traditional worst-case stuffed frame lengths. We will look into two different scenarios:

1. Bit-stuffing distributions based on assuming independent bit-values of the data before encoding, i.e., equal probability of a bit having value 1

or 0. With this information we create a model for making assumptions about the number of stuff-bits in a packet of data.

2. Bit-stuffing distributions extracted from real CAN-bus traffic.

Since the number of stuff-bits in the real traffic is substantially larger than that of our model, we additionally propose a simple (and efficient) method to align the real traffic data with the model. The result is a substantial reduction of the number of stuff-bits in the real traffic.

The outline of this paper is as follows. Section 6.2 specifically discusses the scheduling of frame sets in CAN under a general fault model. Further, it describes the theory behind bit-stuffing and we present the effects of bit-stuffing distributions along with our model. In section 6.3 we investigate some real CAN traffic and in Section 6.4 we give a proposal of how to align the real traffic to our model. Finally Section 6.5 presents our conclusions and future work.

6.2 Traditional Schedulability Analysis of CAN-Frames

The Controller Area Network (CAN) [7] is a broadcast bus designed to operate at speeds of up to 1 Mbps. Data is transmitted in frames containing between 0 and 8 bytes of data and 47[2] control bits. Among those control bits there is an 11-bit identifier associated with each frame. The identifier is required to be unique, in the sense that two simultaneously active frames originating from different sources must have distinct identifiers. The identifier serves two purposes: (1) assigning a priority to the frame, and (2) enabling receivers to filter frames.

CAN is a collision-avoidance broadcast bus, which uses deterministic collision resolution to control access to the bus (so called CSMA/CA). The basis for the access mechanism is the electrical characteristics of a CAN bus: if multiple stations are transmitting concurrently and one station transmits a ‘0’ then all stations monitoring the bus will see a ‘0’. Conversely, only if all stations transmit a ‘1’ will all processors monitoring the bus see a ‘1’. During arbitration, competing stations are simultaneously putting their identifiers, one bit at the time, on the bus. By monitoring the resulting bus value, a station detects if there is a competing higher priority frame and stops transmission if this is the

[2]Standard format CAN frame. There is also an extended format.

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 queued frame, and hence can start transmitting the body of the frame.

6.2.1 Classical CAN Bus Analysis

Tindell *et al.* [9, 10, 11] present analysis to calculate the worst-case latencies of CAN frames. This analysis is based on the standard fixed-priority response-time analysis for CPU scheduling [1].

Calculating the response-times requires a bounded worst-case queuing pattern of frames. The standard way of expressing this is to assume a set of traffic streams, each generating frames with a fixed priority. The worst-case behaviour of each stream, in the sense of network load, is to assume that each frame is periodically queued. In analogue with CPU scheduling, we obtain a model with a set $\mathcal{S}$ of streams (corresponding to CPU tasks). Each $S_i \in \mathcal{S}$ is a triple $< P_i, T_i, C_i >$, where P_i is the priority (defined by the frame identifier), T_i is the period and C_i the worst-case transmission time of frames sent on stream S_i. The worst-case latency R_i of a CAN frame sent on stream S_i is defined by

$$R_i = J_i + q_i + C_i \tag{6.1}$$

where J_i is the queuing jitter of the frame, i.e., the maximum variation in queuing-time relative T_i, inherited from the sender task which queues the frame, and q_i represents the effective queuing-time, given by

$$q_i^{n+1} = B_i + \sum_{j \in hp(i)} \left\lceil \frac{q_i^n + J_j + \tau_{bit}}{T_j} \right\rceil C_j + E(q_i + C_i) \tag{6.2}$$

where the term B_i is the worst-case blocking-time of frames sent on S_i, $hp(i)$ is the set of streams with priority higher than S_i, τ_{bit} (the bit-time) caters for the difference in arbitration start-times at the different nodes due to propagation delays and protocol tolerances, and $E(q_i + C_i)$ is an error term denoting the time required for error signalling and recovery. The reason for the blocking factor is that transmissions are non pre-emptive, i.e., after a bus arbitration has started, the frame with the highest priority among competing frames will be transmitted till completion, even if a frame with higher priority gets queued before the transmission is completed. However, in case of errors a frame can

be interrupted/pre-empted during transmission, requiring a complete retransmission of the entire frame. The extra cost for this is catered for in the error term E above.

Note that (6.2) is a recurrence relation, where the approximation to the value of q_i^{n+1} is found in terms of the *n*th approximation, with the first approximation set to zero. A solution is reached when $q_i^{n+1} = q_i^n$.

6.2.2 Effects of Bit-Stuffing, Worst-Case

In CAN, six consecutive bits of the same polarity (111111 or 000000) is used for error signalling. To avoid these special bit-patterns in transmitted frames, a bit of opposite polarity is inserted after five consecutive bits of the same polarity. By reversing the procedure, these bits are then removed at the receiver side. This technique, which is called *bit-stuffing*, implies that the actual number of transmitted bits may be larger than the size of the original frame, corresponding to an additional transmission delay which need to be considered in the analysis.

According to the CAN standard [7], the total number of bits in a CAN frame before bit-stuffing is

$$8s + 47 \tag{6.3}$$

where s is the number of bytes of payload data ($s = [0, 8]$) and 47 is the number of control bits in a CAN frame. The frame layout is defined such that only 34 of these 47 bits are subject to bit-stuffing. Therefore the total number of bits after bit-stuffing can be no more than

$$8s + 47 + \left\lfloor \frac{34 + 8s - 1}{4} \right\rfloor \tag{6.4}$$

Intuitively the above formula captures the number of stuff-bits in the worst-case scenario, as shown in Figure 6.1.

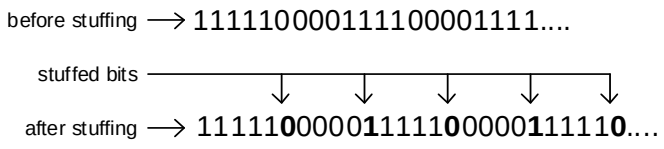

Figure 6.1: The worst-case scenario when stuffing bits.

Let τ_{bit} be the worst-case time taken to transmit a bit on the bus – the so-called *bit-time*. The time taken to transmit a given frame i is therefore

$$C_i = \left(8s_i + 47 + \left\lfloor \frac{34 + 8s_i - 1}{4} \right\rfloor\right) \tau_{bit} \tag{6.5}$$

If we put $s_i = 8$ into the equation, and assume a bus speed of 1Mbit/sec (τ_{bit} = 1 μs), we get $C_i = 135\mu s$. This is a good figure to remember: the largest frame takes 135 bit-times to send.

6.2.3 Independent Bit-Stuffing Model

If we look into how bit-stuffing actually transforms the data instead of using the worst-case method as presented above, we will get a very different result. The length of a frame, before bit-stuffing, can be at most 111 bits (8 bytes data and 47 control bits), and among them 98 bits are exposed to bit-stuffing. By assuming equal probability of bit-value 1 and 0 among the bits and no dependency among bits, we can calculate the actual probabilities of having a certain frame length after bit-stuffing. These probabilities are for different frame sizes (number of bits) shown in Figure 6.2. The graph is a result of an exhaustive analysis of all possible frame patterns. The nine different frame lengths (0-8 bytes of data) are visualised in the graph as vertical lines. Note that only the first 8 cases of stuff-bits (1-8 stuff-bits in the frame) are visible in the graph, since the probability of getting more than 8 stuff-bits is very low. For example, the probability of getting exactly 10, 15 and 20 stuff-bits never exceeds 10^{-4}, 10^{-9}, and 10^{-17}, respectively.

6.3 Case-Study: Real CAN Traffic

In real industrial situations the 50/50 ratio does not apply, since we can not always assume independence among bits. In order to make the above reasoning more realistic we have gathered some traffic from a real automotive system developed by one of our industrial partners.

What we know by experience is that the probability of having consecutive 0:s or 1:s in real frames is quite high, since the data sent often are low integer numbers or frames used for control, e.g., coded as 0 or -1, thus leaving a large number of consecutive bits with the same polarity. For example if we use a 16-bit integer representation and send a “1”, we will send “0000000000000001”, i.e., 15 consecutive 0:s.

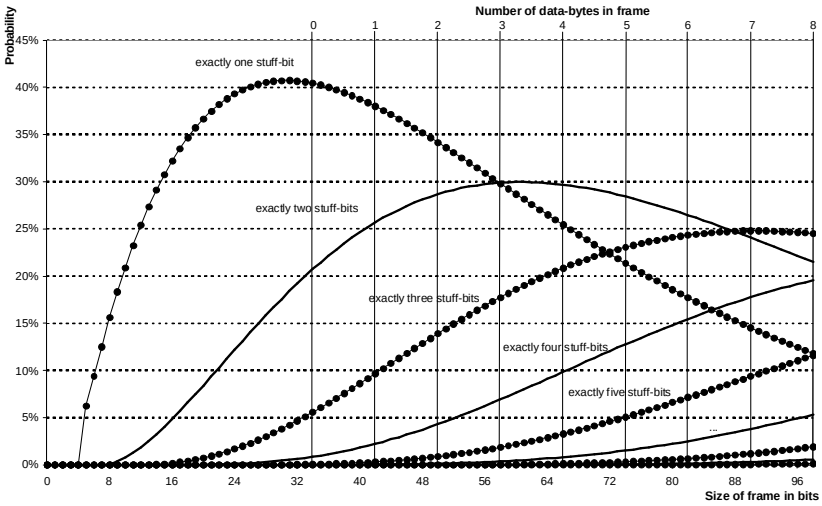

Figure 6.2: Probability of a specific number of stuff-bits in a frame, assuming our probabilistic frame model. The 9 frame lengths are marked as vertical lines.

The conclusion of this is that the actual number of stuff-bits in our real traffic is higher compared to the previous section where we assumed a 50/50 ratio between 1:s and 0:s.

In our investigation of real CAN traffic, we considered some 25 000 frames. Due to the format of the obtained data, we investigated only the data part of the frames, which in this case were 8 bytes for all frames. The rest of the CAN frames (control fields and so on) was not considered. The obtained distribution of stuff-bits is shown in Figure 6.3 (”Real traffic”). Worth noticing is that the actual worst-case here is 13 bits, to be compared with the worst-case result of 15 stuff-bits when applying traditional analysis for a frame size of 64 bits. The figure also shows the distribution obtained with our 50/50 model (”50/50”), as well as the distribution obtained for the real-traffic when applying the coding (”Real traffic using XOR”) that we will present next.

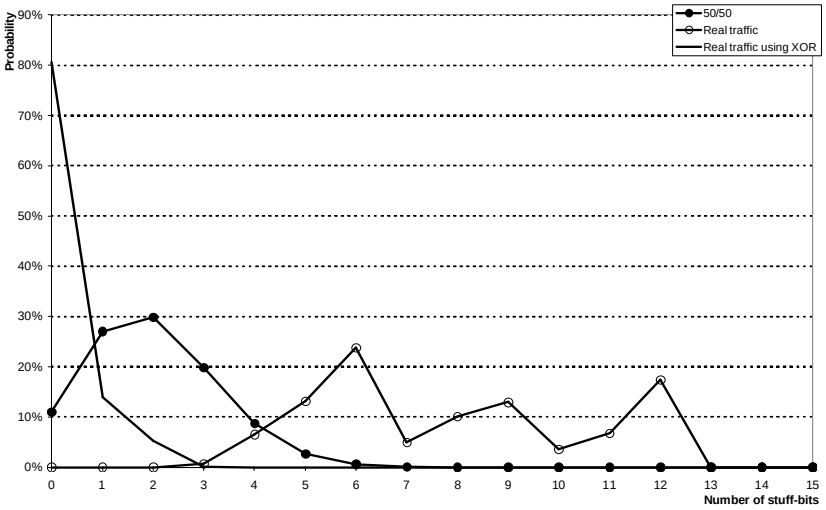

Figure 6.3: Probability density functions, PDF:s, showing the number of stuff-bits in a 64 bit frame. We show here our independent 50/50 model, the real CAN traffic and the manipulated real CAN traffic.

6.4 A Simple Coding Scheme to Reduce Bit-Stuffing

In order to reduce the number of stuff-bits in the real CAN traffic, we can use some kind of bit-operation on the original data to remove consecutive 1:s and 0:s. The general idea of this transformation is to align the real-traffic distribution with that of our 50/50 model.

A	B	eXclusive OR
0	0	0
0	1	1
1	0	1
1	1	0

Table 6.1: XOR operation.

For example, we can use a simple coding scheme in which the original frame is XORed with a particular bit-pattern, the bit-mask. XOR (shown in Table 6.1), is a logical operation performed in a single operation by most CPUs. In our case we use the bit-pattern 101010101010... in order to kill sequences of 1:s and 0:s. On the receiving side, the same XOR operation is performed, with the same bit-mask, to decode the data. Figure 6.4 illustrates the encoding/decoding process.

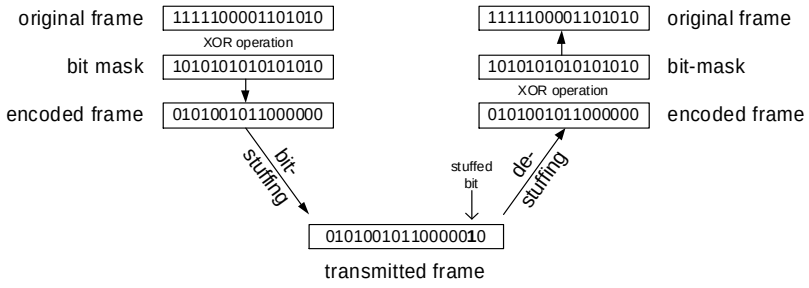

Figure 6.4: Encoding/decoding process for the proposed method.

Our choice of bit-pattern is just an example. The actual bit-pattern needed to get the maximum reduction in the number of stuff-bits is dependent on the characteristics of the transferred data. In fact, it may even be desirable to use different bit-patterns for different frames. The details of how this can be realised is, however, outside the scope of this paper.

We have applied the simple XOR-coding to our 25 000 automotive CAN frames. The result is presented in Figure 6.3 ("Real traffic using XOR"). Here we compare the number of stuff-bits in a frame of size 64 bits, i.e., 8 bytes. Our 50/50 independent model give us quite good result, since we will seldom (probability in the order of 10^{-5}) have frames extended with more than 8 bits, i.e., 46% smaller than the traditional worst-case figure. For the frames obtained after the XOR transformation we did not find any frame with more than 3 extra bits, i.e., 80% smaller than the worst case. Compared to the original real traffic, we will now transmit one byte less. (All of this should of course be compared with the worst-case analysis result of 15 bits.) It should be noted that with the XOR we now have even better performance than our previously suggested 50/50 model. The reason is that our real CAN data contains many long sequences of consecutive 1:s and 0:s, and by masking this data using our

bit-pattern, we will almost eliminate the occurrence of bit-stuffing. But in the general case, we will get a performance closer to the 50/50 model.

6.5 Conclusions

In dimensioning safety critical-systems, a central activity is to validate that sufficient resources are allocated to provide required behavioural, timing, and reliability guarantees. The method that we present here provides information on distributions of stuff-bits in transmitted Controller Area Network (CAN) frames. This information can be used to obtain more accurate reliability analysis, which by allowing occasional deadline misses may substantially reduce the resource demands, without violating the system requirements. Reducing utilisation is essential, since it may allow the use of cheaper solutions.

Since the validation of a system or a product typically is based on a model of a system, it is important to reduce the modelled utilisation, i.e., the utilisation given by the model. This can be achieved either by more accurate modelling, or by reducing the actual utilisation of the system. Focusing on bit-stuffing in CAN, we have in this paper presented both a method to increase the accuracy in the modelling, and a coding method which reduces the actual bus utilisation.

We achieved increased accuracy in the modelling by taking bit-stuffing distributions into consideration. This allowed us to reduce the frame size used when performing timing analysis of the CAN bus. This may have dramatic effects on the calculated response-times, e.g., a system, which with traditional worst-case analysis is deemed unschedulable, may be shown to, with a very high probability, meet its deadlines.

We have shown with a case-study, including 25 000 messages from a real automotive system, that the observed worst-case number of stuff-bits is 13 compared to the worst-case of 15 bits derived by traditional worst-case analysis. Furthermore, our model indicated that it is relatively safe to assume at most 8 stuff-bits because the probability for more stuff-bits is very low. Additionally, by using our XOR coding scheme we can reduce the number of stuff-bits to 3.

In our future work we plan to investigate the exact effects of this further, including the integration bit-stuffing effects in our framework for analysing reliability and timing trade-offs [3]. On a more detailed level, we will investigate the effects of bit-stuffing the control fields of CAN frames. This includes the effects of fixed fields in the CAN control frame, as well as the bit-stuffing of the arbitration field.

Acknowledgements

The authors wish to express their gratitude to Lucia Lo Bello for useful discussions and to the anonymous reviewers for their helpful comments. 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.

Bibliography

[1] 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.

[2] A. Burns. Preemptive Priority Based Scheduling: An Appropriate Engineering Approach. Technical Report YCS 214, University of York, York, England, 1993.

[3] H. Hansson, T. Nolte, C. Norström, and S. Punnekkat. Integrating Reliability and Timing Analysis of CAN-based Systems. *IEEE Transaction on Industrial Electronics*, 49(6), December 2002.

[4] H. Hansson, C. Norström, and S. Punnekkat. Integrating Reliability and Timing Analysis of CAN-based Systems. In *Proceedings of the* 3^{rd} *IEEE International Workshop on Factory Communication Systems (WFCS'00)*, pages 165–172, Porto, Portugal, September 2000. IEEE Industrial Electronics Society.

[5] H. Hansson, C. Norström, and S. Punnekkat. Reliability Modelling of Time-Critical Distributed Systems. volume 1926 of *Lecture Notes in Computer Science (LNCS)*. Springer-Verlag, 6^{th} International Symposium, FTRTFT 2000, Pune, India, September 2000.

[6] S. Punnekkat, H. Hansson, and C. Norström. Response Time Analysis under Errors for CAN. In *Proceedings of the* 6^{th} *IEEE Real-Time Technology and Applications Symposium (RTAS'00)*, pages 258–265, Washington DC, USA, June 2000. IEEE Computer Society.

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

[8] L. Sha, R. Rajkumar, and J.P. Lehoczky. Priority Inheritance Protocols: An Approach to Real-Time Synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, September 1990.

[9] 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.

[10] 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.

[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.

[12] J. Xu and D. L. Parnas. Priority Scheduling Versus Pre-Run-Time Scheduling. *Real-Time Systems*, 18(1):7–23, January 2000.

Chapter 7

Paper B: Minimizing CAN Response-Time Jitter by Message Manipulation

Thomas Nolte, Hans Hansson, and Christer Norström
In Proceedings of the 8^{th} IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'02), San Jose, CA, USA, September 2002.

Abstract

Delay variations (jitter) in computations and communications cause degradation of performance in, e.g., control applications. There are many sources of jitter, including variations in execution time and bus contention.

This paper presents methods to reduce the variation of frame (message) transmission time caused by the bit-stuffing mechanism in the Controller Area Network (CAN). By introducing some restrictions, such as a small reduction of available frame priorities, we are able to reduce the number of stuff-bits in the worst case. We also combine this with some of our previous work that reduces the number of stuff-bits in the data part of the frame. We show the actual penalty introduced by forbidding priorities, and we show the overall improvement by using these techniques together in a small case-study.

7.1 Introduction

During the last decade real-time researchers have extended schedulability analysis to a mature technique which for non-trivial systems can be used to determine whether a set of tasks executing on a single CPU or in a distributed system will meet their deadlines or not [1, 3, 16, 21]. The essence of this analysis is to investigate if deadlines are met in a worst-case scenario. Whether this worst-case actually will occur during execution, or if it is likely to occur, is not normally considered.

In contrast with schedulability analysis, reliability modelling involves study of fault models, characterisation of distribution functions of faults and development of methods and tools for composing these distributions and models in estimating an overall reliability figure for the system.

This separation of deterministic (0/1) schedulability analysis and stochastic reliability analysis is a natural simplification of the total analysis. This is because the deterministic schedulability analysis unfortunately is quite pessimistic, since it only considers the worst-case, i.e., it does not distinguish the case when the deadline is only missed in the (possibly very rare) worst-case from the case when the deadline is always missed.

There are many other sources of pessimism in the analysis, including considering worst-case execution times and worst-case phasings of executions, as well as the usage of pessimistic fault models.

In our previous work [14], we have proposed a model for calculating worst-case latencies of Controller Area Network (CAN) [15] frames under error assumptions. This model is pessimistic, in the sense that there are systems that the analysis determines unschedulable, even though deadlines will only be missed in extremely rare situations with pathological combinations of errors. In [10, 11] we have reduced the level of pessimism by introducing a better fault model, and in [9] we also consider variable phasings between message queuings, in order to make the model more realistic. In [13] we reduced the pessimism introduced by the worst-case analysis of CAN message response-times, by using bit-stuffing distributions instead of the traditional worst-case frame sizes.

In this paper we provide a method that will minimize the variations of frame lengths caused by bit-stuffing. The number of stuff-bits in a CAN frame can vary between 0 and 29, depending on the CAN format (standard or extended), the frame length (the number of data bytes in the frame), and the frame bit-pattern. This variation of frame length is problematic for, e.g., control applications based on event-triggered architectures. Problems and degradation

of performance caused by jitter in control applications have been shown in [5, 12, 17].

Hence, it is desirable to minimize this variation of frame lengths, as shown in [8]. To do this, we make use of our previous work [13] where we presented a method to reduce the number of stuff-bits in the data part of the CAN frame. We will here extend this work by also considering the control part of the CAN frame. We show how bit-stuffing can be eliminated in the header part of the CAN frame and we show how to combine this with our previous work, in order to have a method that minimizes the variations in frame length for the whole CAN frame.

There has been work done to reduce jitter caused by variations in queuing times for CAN frames using genetic algorithms [2, 6, 7]. This is done by giving periodic messages initial phasings, found by using genetic algorithms. These phasings can be set both offline and online, although the technique requires a relatively high computational overhead. Our method, on the other hand, focuses on the jitter caused by variations of frame lengths. Our approach is done mostly offline, and the online part requires very little CPU-time.

Outline: Section 7.2 specifically discusses the scheduling of frame sets in CAN under a general fault model, and describes the theory behind bit-stuffing. In Section 7.3 we show how we can eliminate the occurrence of stuff-bits in the header part of the CAN frame and in Section 7.4 we present our independent bit-stuffing model along with a method for data transformation which significantly reduces the number of stuff-bits in the data part of the CAN frame. In Section 7.5 we combine the techniques described in Section 7.3 and Section 7.4, and in Section 7.6 we show the result of using our methods and models in a case-study. Finally Section 7.7 presents our conclusions and outlines future work.

7.2 Traditional Schedulability Analysis of CAN-Frames

The Controller Area Network (CAN) [15] 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 other applications. CAN transmits data in frames containing between 0 and 8 bytes of data and 47 control bits, as shown in Figure 7.1. (There is also an extended format, which contains 20 more control bits. The main difference is that the extended format has 29 identifier bits instead of 11 bits. Please consult [4] for more details.)

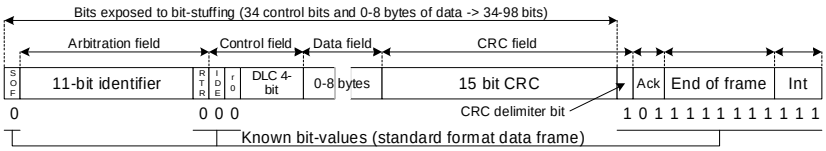

Figure 7.1: CAN frame layout (standard format data frame).

Among the control bits there is an 11-bit identifier associated with each frame (plus another 18 when using the extended format). The identifier is required to be unique, in the sense that two simultaneously active frames originating from different sources must have distinct identifiers. The identifier serves two purposes: (1) assigning a priority to the frame, and (2) enabling receivers to filter frames. For a more detailed explanation of the different fields in the CAN frame, please consult [15] or [4].

CAN is a collision-avoidance broadcast bus, which uses deterministic collision resolution to control access to the bus (so called CSMA/CA). The basis for the access mechanism is the electrical characteristics of a CAN bus: if multiple stations are transmitting concurrently and one station transmits a ‘0’ then all stations monitoring the bus will see a ‘0’. Conversely, only if all stations transmit a ‘1’ will all processors monitoring the bus see a ‘1’. During arbitration, competing stations are simultaneously putting their identifiers, one bit at the time, on the bus. By monitoring the resulting bus value, a station detects if there is a competing higher priority frame and stops transmission if this is the case. Because identifiers are required to be 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 queued frame, and hence can start transmitting the body of the frame.

7.2.1 Classical CAN Bus Analysis

Tindell *et al.* [18, 19, 20] present analysis to calculate the worst-case latencies of CAN frames. This analysis is based on the standard fixed-priority response-time analysis for CPU scheduling [1].

Calculating the response-times requires a bounded worst-case queuing pattern of frames. The standard way of expressing this is to assume a set of traffic streams, each generating frames with a fixed priority. The worst-case behaviour of each stream, in terms of network load, is to send as many frames as they are

allowed, i.e., to periodically queue frames. In analogue with CPU scheduling, we obtain a model with a set $\mathcal{S}$ of streams (corresponding to CPU tasks). Each $S_i \in \mathcal{S}$ is a triple $< P_i, T_i, C_i >$, where P_i is the priority (defined by the frame identifier), T_i is the period and C_i the worst-case transmission time of frames sent on stream S_i. The worst-case latency R_i of a CAN frame sent on stream S_i is, if we assume the minimum variation in queuing time relative T_i to be 0, defined by

$$R_i = J_i + q_i + C_i \tag{7.1}$$

where J_i is the queuing jitter of the frame, i.e., the maximum variation in queuing-time relative T_i, inherited from the sender task which queues the frame, and q_i represents the effective queuing-time, given by

$$q_i^{n+1} = B_i + \sum_{j \in hp(i)} \left\lceil \frac{q_i^n + J_j + \tau_{bit}}{T_j} \right\rceil C_j + E(q_i + C_i) \tag{7.2}$$

where the term B_i is the worst-case blocking-time of frames sent on S_i, $hp(i)$ is the set of streams with priority higher than S_i, τ_{bit} (the bit-time) caters for the difference in arbitration start-times at the different nodes due to propagation delays and protocol tolerances, and $E(q_i + C_i)$ is an error term denoting the time required for error signalling and recovery. The reason for the blocking factor is that transmissions are non pre-emptive, i.e., after a bus arbitration has started the frame with the highest priority among competing frames will be transmitted until completion, even if a frame with higher priority gets queued before the transmission is completed. However, in case of errors a frame can be interrupted/pre-empted during transmission, requiring a complete retransmission of the entire frame. The extra cost for this is catered for in the error term E above.

Note that (7.2) is a recurrence relation, where the approximation to the value of q_i^{n+1} is found in terms of the *n*th approximation, with the first approximation set to zero. A solution is reached when $q_i^{n+1} = q_i^n$.

7.2.2 Effects of Bit-Stuffing, Worst-Case

In CAN, six consecutive bits of the same polarity (111111 or 000000) are used for error and protocol control signalling. To avoid these special bit-patterns in transmitted frames, a bit of opposite polarity is inserted after five consecutive bits of the same polarity. By reversing the procedure, these bits are then removed at the receiver side. This technique, which is called *bit-stuffing*, implies

that the actual number of transmitted bits *may* be larger than the size of the original frame, corresponding to an additional transmission delay which needs to be considered in the analysis.

According to the CAN standard [15], the total number of bits in a CAN frame before bit-stuffing is

$$8s + g + 13 \tag{7.3}$$

where s is the number of bytes of payload data ($s = [0, 8]$) and $g + 13$ is the number of bits in the control part of the CAN frame. The frame layout is defined such that only g of these $g + 13$ bits are subject to bit-stuffing (see Figure 7.1). In the standard format $g = 34$ and in the extended format $g = 54$. Therefore the total number of bits after bit-stuffing can be no more than

$$8s + g + 13 + \left\lfloor \frac{g + 8s - 1}{4} \right\rfloor \tag{7.4}$$

Intuitively the above formula captures the number of stuff-bits in the worst-case scenario, shown in Figure 7.2.

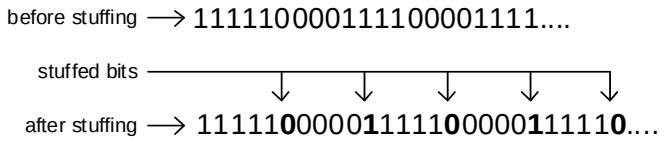

Figure 7.2: The worst-case scenario when stuffing bits.

Let τ_{bit} be the worst-case time taken to transmit a bit on the bus – the so-called *bit-time* (including the inter-frame space). The worst-case time taken to transmit a given frame i is therefore

$$C_i = \left(8s_i + g + 13 + \left\lfloor \frac{g + 8s_i - 1}{4} \right\rfloor\right) \tau_{bit} \tag{7.5}$$

7.3 Careful Priority Usage

In this section we will investigate how it is possible to avoid/minimize stuff-bits in the header part of the CAN frame. For simplicity we will focus on the

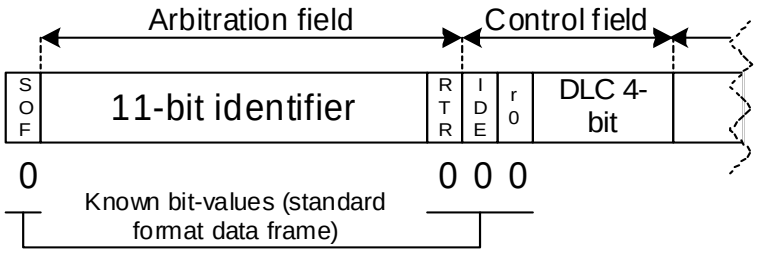

Figure 7.3: CAN frame header, the first 6 fields of the CAN frame (standard format).

Number of stuff-bits	Number of bytes of data in the CAN message frame				
	0	1	$2-3$	$4-7$	8
0	0	0	0	745	1131
1	1332	1436	1490	1005	765
2	634	560	520	279	145
3	81	51	38	19	7
4	1	1	0	0	0

Table 7.1: Amount of remaining priorities for various data lengths and their corresponding number of stuff-bits (standard format).

standard format, but the same reasoning holds for the extended format. The obtained data for the extended format is shown in the end of this section.

The priority of the standard format CAN frame, which is also the arbitration field, consists of 11 bits (as can be seen in Figure 7.3), which are subject to bit-stuffing before the frame is actually transmitted.

By carefully selecting priorities we can avoid the effect of stuff-bits in the frame header, i.e., by excluding the identifiers that lead to bit-stuffing we can *a priori* make sure that there will be no stuff-bits in any of the fields shown in Figure 7.3. The drawback of this is that we have forbidden the usage of some selected priorities, which obviously comes at a cost, since originally we could use all 11 bits to represent the priority and identity of the CAN frame, which gave us 2^{11} (2048) different priorities, and after the removal of selected priorities, it turns out that we have either of the following two scenarios: (1) we can eliminate the number of stuff-bits in the CAN header, or (2) we can

Number of stuff-bits	Number of bytes of data in the CAN message frame				
	0	1	$2-3$	$4-7$	8
0	0	0	0	15.09	22.91
1	26.98	29.98	30.18	35.60	38.67
2	40.09	40.74	41.03	31.49	26.59
3	24.01	22.65	21.94	13.93	9.62
4	7.49	6.44	5.93	3.39	1.97
5	1.31	1.00	0.86	0.46	0.23
6	0.13	< 0.01	0.06	0.03	0.01
7	0.01	< 0.01	< 0.01	< 0.01	< 0.01
8	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
9	< 0.01	0	0	0	0

Table 7.2: Amount of remaining priorities for various data lengths and their corresponding number of stuff-bits (extended format). Due to large numbers, only percentages are shown (percentages of 2^{11+18}).

minimize the number of stuff-bits in the CAN header to 1.

The actual numbers of stuff-bits, by forbidding priorities, are described in Table 7.1. Worth noticing is that the number of stuff-bits depends on the number of data bytes in the frame. This since the DLC field, see Figure 7.3, consists of 4 bits describing the number of bytes of data in the frame. Thus, this bit pattern will affect the number of stuff-bits generated in the frame header (all frame fields before the data part of the CAN frame, as shown in Figure 7.3).

What we can see in Table 7.1 is that we have 3 different groups of scenarios:

1. The first group is when we have 0-3 bytes of data. Here it is impossible to eliminate the occurrence of stuff-bits in the CAN header, but we can make sure that we will only have at most one stuff-bit. However, by forbidding priorities, the number of priorities that we can use decrease to 1332 (0 bytes of data), 1436 (1 byte of data) or 1490 (for 2-3 bytes of data).

2. The second scenario is when we have 4-7 bytes of data. Here we can eliminate the number of stuff-bits in the CAN header by forbidding priorities, leaving 745 usable priorities. One can argue that forbidding priorities would be the same as to use redundant bits as “virtual stuff-bits” (since the number of usable priorities require less bits for representation

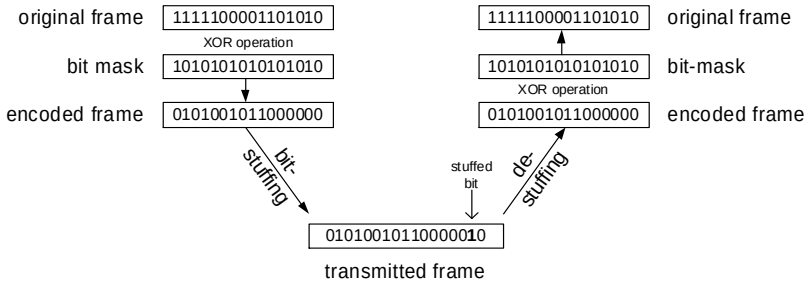

Figure 7.4: Encoding/decoding process for the proposed method.

compared to the number of bits that are allocated for describing the priority; some bits are left "unused"). Although there is some truth in this reasoning, the CAN header has a fixed number of bits. Hence, even if we are using fewer priorities, the number of bits in the CAN header stays the same.

3. The third and final scenario is when we have 8 bytes of data. Also here we can eliminate the stuff-bits by forbidding priorities. The number of usable priorities is then 1131.

Conclusions of what is presented in Table 7.1 is that we can eliminate the occurrences of stuff-bits in the CAN header (when the message contains 4-8 bytes of data) by forbidding priorities, and the cost for this is a reduction of the number of available priorities. Therefore we believe that this method can be used, depending on the application's need of priorities, to eliminate the effect of bit-stuffing in the header part of the CAN message frame.

Corresponding values for the extended format are shown in Table 7.2.

7.4 Independent Bit-Stuffing Model and a Method for Data Transformation

In our previous paper [13] we propose a method to reduce the effect of bit-stuffing in the data part of the CAN frame. The motivation is to investigate the level of pessimism of traditional schedulability analysis for CAN.

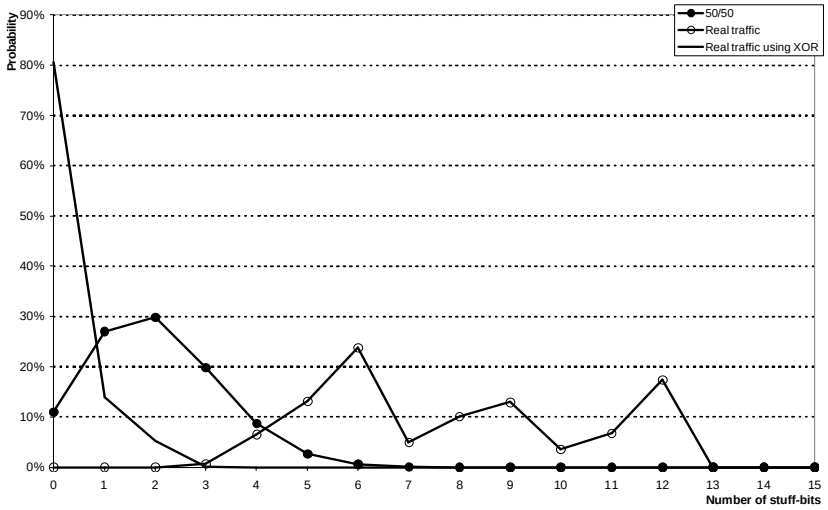

Figure 7.5: Probability density functions, PDF:s, showing the number of stuff-bits in a 64 bit frame. We show here our independent 50/50 model, the real CAN traffic and the manipulated real CAN traffic.

The method, show in Figure 7.4, reduces the actual number of stuff-bits in the CAN data frame by transforming the message using an XOR operation on the data together with a bit-mask. By doing this, we showed with a case-study that the actual number of stuff-bits was significantly reduced, as can be seen in Figure 7.5. Here we can see (Real traffic) the number of stuff-bits in an industrial application (samples taken from one of our automotive partners). In relation to this, we also see the number of stuff-bits in artificial data generated by assuming independent and equal probability of a ’1’ and ’0’ in each bit position (50/50), and the number of stuff-bits in the same industrial data, but after using the method described above (Real traffic using XOR).

7.5 Combination of Techniques

The methods described in Section 7.3 and Section 7.4 can be combined in order to significantly reduce the variation of CAN message frame lengths, i.e., reducing the jitter. We will in this section additionally integrate the last field in

Nof data bytes	0	1	2	3	4	5	6	7	8
Nof bits	0	8	16	24	32	40	48	56	64
Total (CRC + data)	15	23	31	39	47	55	63	71	79
0	6.76E-01	4.85E-01	3.61E-01	2.69E-01	2.00E-01	1.49E-01	1.11E-01	8.25E-02	6.14E-02
1	2.29E-01	3.88E-01	4.07E-01	3.91E-01	3.57E-01	3.15E-01	2.71E-01	2.29E-01	1.90E-01
2	3.23E-02	1.12E-01	1.84E-01	2.41E-01	2.78E-01	2.96E-01	2.99E-01	2.90E-01	2.73E-01
3	6.10E-04	1.41E-02	4.23E-02	8.10E-02	1.24E-01	1.64E-01	1.98E-01	2.23E-01	2.40E-01
4	0	6.93E-04	5.18E-03	1.62E-02	3.46E-02	5.90E-02	8.73E-02	1.17E-01	1.45E-01
5	0	0	3.20E-04	1.96E-03	6.31E-03	1.45E-02	2.70E-02	4.37E-02	6.35E-02
6	0	0	8.27E-06	1.38E-04	7.54E-04	2.48E-03	6.04E-03	1.21E-02	2.09E-02
7	0	0	4.94E-08	5.11E-06	5.76E-05	2.94E-04	9.82E-04	2.50E-03	5.29E-03
8	0	0	0	8.01E-08	2.65E-06	2.38E-05	1.16E-04	3.91E-04	1.03E-03
9	0	0	0	2.27E-10	6.54E-08	1.27E-06	9.80E-06	4.60E-05	1.57E-04
10	0	0	0	0	6.76E-10	4.11E-08	5.77E-07	4.02E-06	1.84E-05
11	0	0	0	0	1.46E-12	7.16E-10	2.26E-08	2.56E-07	1.65E-06
12	0	0	0	0	0	5.17E-12	5.43E-10	1.15E-08	1.12E-07
13	0	0	0	0	0	7.44E-15	7.00E-12	3.45E-10	5.56E-09
14	0	0	0	0	0	0	3.68E-14	6.36E-12	1.96E-10
15	0	0	0	0	0	0	3.66E-17	6.25E-14	4.64E-12
16	0	0	0	0	0	0	0	2.46E-16	6.75E-14
17	0	0	0	0	0	0	0	1.76E-19	5.19E-16
18	0	0	0	0	0	0	0	0	1.57E-18
19	0	0	0	0	0	0	0	0	8.30E-22

Table 7.3: Number of stuff-bits, with corresponding probability of occurrence (xEy equals $x \times 10^y$).

the CAN frame, the CRC field, in the jitter reduction.

With the first method, we reduced the worst-case number of stuff-bits in the frame header to 0 or 1 (depending on the number of data bytes in the CAN frame) from 4, which is the theoretical value that we have to use in a safe worst-case analysis.

Combining this with the second method we further reduce the number of stuff-bits. As can be seen in Figure 7.5 we have reduced the number of stuff-bits in an 8 byte data part of a frame to 3 from 13 (analytically 15).

Finally, the last part of the CAN frame to investigate is the CRC field at the end of the frame, shown in Figure 7.1. We believe, since CRC-generation essentially coincides with pseudo random binary sequence generation, that the 50/50 model described in [13] and in Section 7.4 is suitable for describing these bits, i.e., we assume that the CRC essentially is a sequence of bits with equal and independent probability for bit value 0 and 1, respectively. The model assumes independence among bits and equal probability for having bit-value 0 or 1. What we do then is that we use our model for both the data part and the CRC field of the CAN frame. According to the model, the number of stuff-bits and their corresponding probabilities for the data and the CRC part of the frame are described in Table 7.3.

By using our model we can see, when for example using 8 bytes of data, that the number of stuff-bits is reduced from, analytically 24 to 11 when the acceptable probability of exceeding the maximum frame size is in the order of

Nof bits	Head	Data	CRC	Entire frame	Entire w prio.	Data XOR	New CRC	Entire XOR	Entire w XOR+prio
0	0	0	0.36618	0	0	0.78605	0.87834	0	0.69409
1	0	0	0.41301	0	0	0.14786	0.11973	0	0.21820
2	0.59550	0	0.22081	0	0	0.01449	0.00193	0.51457	0.02668
3	0.38962	0.00020	0	0	0	0.05160	0	0.23032	0.06047
4	0.00469	0.00341	0	0	0.00225	0	0	0.17338	0.00056
5	0.01019	0.01505	0	0	0.00678	0	0	0.01942	0
6	0	0.01613	0	0.00225	0.02291	0	0	0.06211	0
7	0	0.04057	0	0.00325	0.01677	0	0	0.00020	0
8	0	0.22984	0	0.00863	0.09020	0	0	0	0
9	0	0.22972	0	0.03419	0.11608	0	0	0	0
10	0	0.18682	0	0.02387	0.30644	0	0	0	0
11	0	0.00389	0	0.18076	0.16419	0	0	0	0
12	0	0.21551	0	0.22410	0.11556	0	0	0	0
13	0	0.05886	0	0.07700	0.07696	0	0	0	0
14	0	0	0	0.26021	0.05622	0	0	0	0
15	0	0	0	0.07824	0.02564	0	0	0	0
16	0	0	0	0.05132	0	0	0	0	0
17	0	0	0	0.05618	0	0	0	0	0

Table 7.4: Number of stuff-bits in the samples, with corresponding probability of occurrence.

10^{-6}, since $\sum_{11 \leq i \leq 19} P_i \leq 10^{-6}$ where $P_i =$ probability of having exactly i stuff-bits. Therefore, we have significantly reduced the maximum number of stuff-bits and thus, the interval between maximum and minimum number of stuff-bits is smaller, i.e., we have reduced the considered jitter.

We must also remember that these values are based on our model. When using our method to decrease the number of stuff-bits in a real system the actual number of stuff-bits can be even smaller, as shown in Figure 7.5.

7.6 Case-Study

In order to validate our method and model, we make use of samples taken from one of our industrial partners. Firstly, we investigate the actual number of stuff-bits in some 25 000 CAN frames (extended format). This result is then compared with the same CAN frames, both with and without the usage of the methods described in this paper.

The number of stuff-bits in the CAN frame, both with the XOR manipulation as described in Section 7.4, and without manipulation, are shown in Figure 7.6. What we can read from the figure is that the actual worst-case number of stuff-bits has dropped from 17 to 7, this as a result of removing patterns of consecutive bits in the data part of the CAN frame. We used the same bit-pattern for the mask, as shown in Figure 7.4. Note that we have not used the method for selecting priorities yet.

In order to further reduce the number of stuff-bits in the CAN frame we

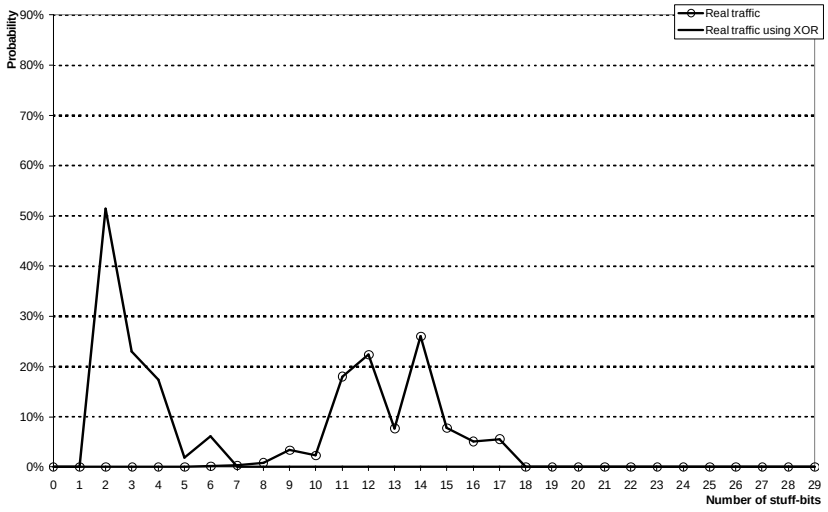

Figure 7.6: Probability density functions, PDF:s, showing the number of stuff-bits in a CAN frame (extended format). We show here real traffic along with the same traffic but manipulated with XOR.

also make use of the method based on forbidding some priorities, as described in Section 7.3. The result of this is shown in Figure 7.7 along with the independent model described in Section 7.4 (also shown as the right most column of Table 7.3). Note here that with the knowledge of elimination of stuff-bits in the CAN header, we use the 50/50 model only for the data part and the CRC part of the CAN frame. The result of carefully selecting priorities gives us even less stuff-bits. We have now reduced the actual worst-case number of stuff-bits from 17 to 4, as can be seen in Figure 7.7.

The results from all experiments within the case-study are shown in Table 7.4. Here we can see the number of stuff-bits in the header, data and CRC part of the original frame as well as the number of stuff-bits in the whole CAN frame. Furthermore, the number of stuff-bits in the data and CRC part of the frame after the XOR method are shown. Finally, the number of stuff-bits in the whole CAN frame, after applying both the XOR method and the priority selection, is shown.

This case-study shows that we can, by using the methods described in this

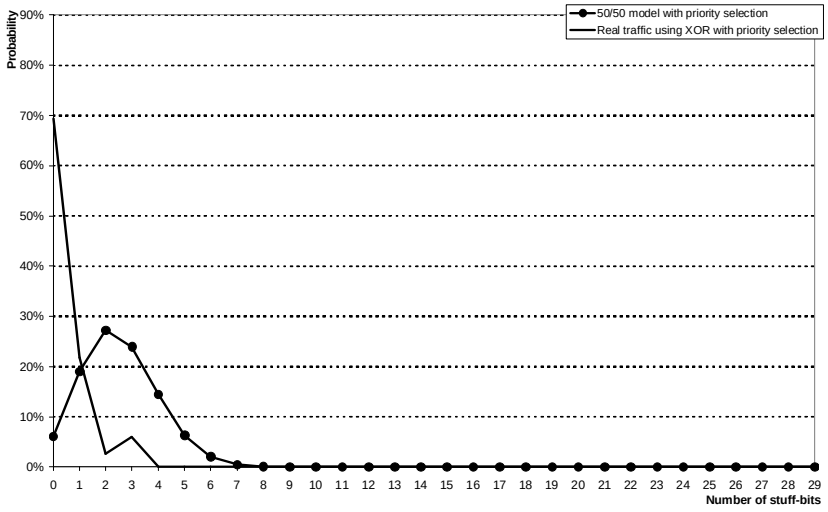

Figure 7.7: Probability density functions, PDF:s, showing the number of stuff-bits in a CAN frame (extended format). We show here real traffic manipulated with XOR and careful priority selecting. Our independent model is also shown with respect to the careful priority select.

paper, substantially reduce the worst-case number of stuff-bits in a message; in our case from 17 to 4. This should be compared to the analytical value of 29, which is the theoretical value that we must use in a worst-case analysis. Also worth noticing is that the variation of frame length has decreased a lot, i.e., the jitter is substantially reduced.

7.7 Conclusions

In dimensioning safety-critical systems, a central activity is to validate that sufficient resources are allocated to provide required behavioural, timing, and reliability guarantees. Reducing utilization is essential, since it may allow the use of cheaper solutions in applications. Since the validation of a system or a product typically is based on a model of a system, it is important to reduce the modelled utilisation, i.e., the utilisation given by the model. This can be achieved either by more accurate modelling, or by reducing the actual utilisa-

tion of the system. Focusing on bit-stuffing in the Controller Area Network (CAN), we have in this paper presented a method that both increases the accuracy of the modelling, and reduces the actual bus utilisation. What we achieve by doing this is an improvement in terms of reducing jitter. By lowering the maximum number of stuff-bits that can occur in a frame, we have significantly reduced the jitter caused by the varying number of stuff-bits in a CAN frame.

We achieved increased accuracy in the modelling by taking bit-stuffing distributions into consideration. This allowed us to reduce the frame size used when performing timing analysis of the CAN bus. This may have dramatic effects on the calculated response-time, e.g., a system that with traditional worst-case analysis is deemed unschedulable may be shown to with a very high probability meet its deadlines.

We have also carefully selected a number of valid priorities, among all possible priorities, in order to eliminate the number of stuff-bits in the frame header. The combination of these two methods gives us a method to decrease the number of stuff-bits in the whole CAN frame. The true effects of our methods have been shown in a case-study.

From a strict hard real-time perspective, our contribution is that we illustrate the level of inherent pessimism in such analysis. From a more pragmatic industrial perspective, our results indicate the feasibility of sufficiently safe analysis methods, which at the penalty of just a slight and controllable optimism has a potential to substantially reduce the system resource requirements, compared to the resource requirements suggested by the hard real-time analysis.

In our future work we plan to investigate this further, by examining if it is possible to completely eliminate the occurrence of stuff-bits in the data part of the frame. Furthermore, it would be interesting to see the result by combining this method with the work done in [2, 6, 7] in order to reduce the jitter caused by the blocking of other messages.

We also want to set up a real system to test the methods with respect to latency.

Our ultimate goal is to combine all of this into a complete engineering method for making well founded trade-offs between levels of timing guarantees and reliability.

Acknowledgements

The authors wish to express their gratitude to the anonymous reviewers for their helpful comments. 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.

Bibliography

[1] 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.

[2] J. Barreiros, E. Costa, J. A. Fonseca, and F. Coutinho. Jitter Reduction in a Real-Time Message Transmission System Using Genetic Algorithms. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'00)*, volume 2, pages 1095–1102, La Jolla, CA, USA, July 2000. IEEE Computer Society.

[3] A. Burns. Preemptive Priority Based Scheduling: An Appropriate Engineering Approach. Technical Report YCS 214, University of York, York, England, 1993.

[4] 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.

[5] CAN Specification Version 2.0. Robert Bosch GmbH, Postfach 50, D-7000 Stuttgart 1, Germany. 1991.

[6] F. Coutinho, J. A. Fonseca, J. Barreiros, and E. Costa. Jitter Minimization with Genetic Algorithms. In *Proceedings of the* 3^{rd} *IEEE International Workshop on Factory Communication Systems (WFCS'00)*, pages 267–273, Porto, Portugal, September 2000. IEEE Industrial Electronics Society.

[7] F. Coutinho, J. A. Fonseca, J. Barreiros, and E. Costa. Using Genetic Algorithms to Reduce Jitter in Control Variables Transmitted over CAN. In *Proceedings of the* 7^{th} *International CAN Conference (ICC'00)*, Amsterdam, The Netherlands, October 2000.

[8] J.D. Decotignie. Some Future Directions in Fieldbus Research and Development. In *Proceedings of the International Conference on Fieldbus Technology (FeT'99)*, Magdeburg, Germany, September 1999.

[9] H. Hansson, T. Nolte, C. Norström, and S. Punnekkat. Integrating Reliability and Timing Analysis of CAN-based Systems. *IEEE Transaction on Industrial Electronics*, 49(6), December 2002.

[10] H. Hansson, C. Norström, and S. Punnekkat. Integrating Reliability and Timing Analysis of CAN-based Systems. In *Proceedings of the* 3^{rd} *IEEE International Workshop on Factory Communication Systems (WFCS'00)*, pages 165–172, Porto, Portugal, September 2000. IEEE Industrial Electronics Society.

[11] H. Hansson, C. Norström, and S. Punnekkat. Reliability Modelling of Time-Critical Distributed Systems. volume 1926 of *Lecture Notes in Computer Science (LNCS)*. Springer-Verlag, 6^{th} International Symposium, FTRTFT 2000, Pune, India, September 2000.

[12] S. H. Hong. Scheduling Algorithm of Data Sampling Times in the Integrated Communication and Control Systems. *IEEE Transactions on Control Systems Technology*, 3(2):225–230, June 1995.

[13] T. Nolte, H. Hansson, and C. Norström. Using Bit-Stuffing Distributions in CAN Analysis. 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.

[14] S. Punnekkat, H. Hansson, and C. Norström. Response Time Analysis under Errors for CAN. In *Proceedings of the* 6^{th} *IEEE Real-Time Technology and Applications Symposium (RTAS'00)*, pages 258–265, Washington DC, USA, June 2000. IEEE Computer Society.

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

[16] L. Sha, R. Rajkumar, and J.P. Lehoczky. Priority Inheritance Protocols: An Approach to Real-Time Synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, September 1990.

[17] A. Stothert and I.M. MacLeod. Effect of Timing Jitter on Distributed Computer Control System Performance. In *Proceedings of the* 15^{th} *IFAC Workshop on Distributed Computer Control Systems (DCCS'98)*, September 1998.

[18] 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.

[19] 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.

[20] 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.

[21] J. Xu and D. L. Parnas. Priority Scheduling Versus Pre-Run-Time Scheduling. *Real-Time Systems*, 18(1):7–23, January 2000.

Chapter 8

Paper C: Probabilistic Worst-Case Response-Time Analysis for the Controller Area Network

Thomas Nolte, Hans Hansson, and Christer Norström
In Proceedings of the 9^{th} IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'03), Toronto, Canada, May 2003.

Abstract

This paper presents a novel approach for calculating a probabilistic worst-case response-time for messages in the Controller Area Network (CAN). CAN uses a bit-stuffing mechanism to exclude forbidden bit-patterns within a message frame. The added bits eliminate the forbidden patterns but cause an increase in frame length. How much the length is increased depends on the bit-pattern of the original message frame.

Traditional response-time analysis methods assume that all frames have a worst-case number of stuff-bits. This introduces pessimism in the analysis.

In this paper we introduce an analysis approach based on using probability distributions to model the number of stuff-bits. The new analysis additionally opens up for making trade-offs between reliability and timeliness, in the sense that the analysis will provide a certain probability for missing deadlines, which in the reliability analysis can be treated as a probability of failure. We evaluate the performance of our method using a subset of the SAE[1] benchmark.

[1] See [11] for details.

8.1 Introduction

During the last decade real-time researchers have extended schedulability analysis to a mature technique which for non-trivial systems can be used to determine whether a set of tasks executing on a single CPU or in a distributed system will meet their deadlines or not [2, 4, 12, 16]. The essence of this analysis is to investigate if deadlines are met in a worst-case scenario. Whether this worst-case actually will occur during execution, or if it is likely to occur, is not normally considered.

In contrast with schedulability analysis, reliability modelling involves study of fault models, characterization of distribution functions of faults and development of methods and tools for composing these distributions and models in estimating an overall reliability figure for the system.

This separation of deterministic (0/1) schedulability analysis and stochastic reliability analysis is a natural simplification of the total analysis. However the deterministic schedulability analysis is unfortunately quite pessimistic, since it assumes that a missed deadline in the worst-case is equivalent to always missing the deadline. There are also other sources of pessimism in the analysis, including considering worst-case execution times and the usage of pessimistic fault models.

Reliability is defined as the probability that a system can perform its intended function, under given conditions, for a given time interval. A major issue is how to compose hardware reliability, software reliability, environmental model, and timely correctness to arrive at reasonable estimates of overall system reliability, as depicted in Figure 8.1.

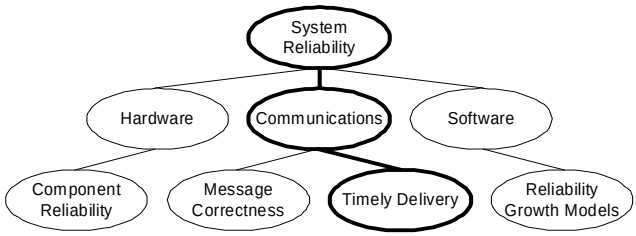

Figure 8.1: System reliability: a top-down view.

The Controller Area Network (CAN) is extensively used in small scale distributed systems, such as automotive, medical and industrial applications. In

this paper we provide a probabilistic response-time analysis method for messages in CAN. Several probabilistic approaches for schedulability analysis of real-time systems have been presented, e.g., [1, 7]. However, none of these specifically address CAN.

We have in our previous work presented a method to model the number of *stuff-bits* in a CAN message frame [8, 9]. Stuff-bits are extra bits added by the CAN protocol. There is a built in mechanism in the CAN protocol, which removes forbidden bit-patterns (e.g., patterns used for error signalling and the communication protocol) within the message frame by "inserting" stuff-bits at specific positions. This mechanism causes a variation in the CAN message frame length.

When performing worst-case response-time analysis, the worst-case number of stuff-bits is traditionally used. In this paper we will introduce a worst-case response-time analysis method, which uses distributions of stuff-bits instead of the worst-case values. This makes the analysis less pessimistic in the sense that we obtain a distribution of worst-case response-times corresponding to all possible combinations of stuff-bits of all message frames involved in the response-time analysis. Using a distribution rather than a fixed value makes it possible to select a worst-case response time based on a desired probability p of violation, i.e., the selected worst-case response-time is such that the probability of a response-time exceeding it is $\leq p$. Our main motivation for calculating such probabilistic response-times is that they allow us to reason about trade-offs between reliability and timeliness.

However, it should be noted that this paper focuses on a single aspect, namely a probabilistic *worst-case response-time*, based on using bit-stuffing distributions. There are other parameters, including execution times and phasings of message queuings, that have similar variations and effects on the response-time analysis. However, our calculations are based on the "critical instant" worst-case scenario.

Outline: Section 8.2 presents the traditional schedulability analysis for CAN. In Section 8.3 we present the new probabilistic response-time analysis, and in Section 8.4 the analysis is evaluated using the SAE [11] benchmark. Finally Section 8.5 concludes the paper and presents some future work.

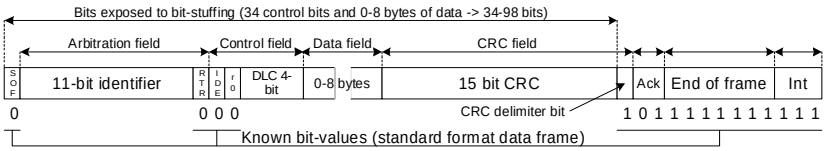

Figure 8.2: CAN frame layout (standard format data frame).

8.2 Traditional Schedulability Analysis of CAN-Frames

The Controller Area Network (CAN) [10] is a broadcast bus designed to operate at speeds of up to 1 Mbps. Data is transmitted in frames containing between 0 and 8 bytes of data and a number of control bits. Depending on the CAN format (standard or extended) the number of control bits are either 44 or 64. Between CAN frames sent on the bus, there is also a 3 bit inter-frame space. The standard format CAN frame (and the inter-frame space) is shown in Figure 8.2.

The difference between the standard and the extended format is that the extended format has 29 identifier bits instead of the 11 bits used in the standard format. The identifier is required to be unique, in the sense that two simultaneously active frames originating from different sources (i.e., nodes or CAN-controllers) must have distinct identifiers. The identifier serves two purposes: (1) assigning a priority to the frame, and (2) enabling receivers to filter frames. For a more detailed explanation of the different fields in the CAN frame, please consult [10, 5].

CAN is a collision-avoidance broadcast bus, which uses deterministic collision resolution to control access to the bus (so called CSMA/CA). The basis for the access mechanism is the electrical characteristics of a CAN bus: if multiple stations are transmitting concurrently and one station transmits a ‘0’ then all stations monitoring the bus will see a ‘0’. Conversely, only if all stations transmit a ‘1’ will all processors monitoring the bus see a ‘1’. During arbitration, competing stations are simultaneously putting their identifiers, one bit at the time, on the bus. By monitoring the resulting bus value, a station detects if there is a competing higher priority frame 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 queued frame, and hence can start transmitting the body of the frame.

8.2.1 Classical CAN Bus Analysis

Tindell *et al.* [13, 14, 15] present analysis to calculate the worst-case latencies of CAN frames. This analysis is based on the standard fixed-priority response-time analysis for CPU scheduling [2].

Calculating the response-times requires a bounded worst-case queuing pattern of frames. The standard way of expressing this is to assume a set of traffic streams, each generating frames with a fixed priority. The worst-case behaviour of each stream, in terms of network load, is to send as many frames as they are allowed, i.e., to periodically queue frames. In analogue with CPU scheduling, we obtain a model with a set $\mathcal{S}$ of streams (corresponding to CPU tasks). Each $S_i \in \mathcal{S}$ is a triple $< P_i, T_i, C_i >$, where P_i is the priority (defined by the message frame identifier), T_i is the period and C_i the worst-case transmission time of frames sent on stream S_i. The worst-case latency R_i of a CAN frame sent on stream S_i is, if we assume the minimum variation in queuing time relative to T_i to be 0, defined by

$$R_i = J_i + q_i + C_i \tag{8.1}$$

where J_i is the queuing jitter of the frame, i.e., the maximum variation in queuing-time relative start of T_i, inherited from the sender task which queues the frame, and q_i represents the effective queuing-time, given by

$$q_i^n = B_i + \sum_{j \in hp(i)} \left\lceil \frac{q_i^{n-1} + J_j + \tau_{bit}}{T_j} \right\rceil (C_j + 3\tau_{bit}) \tag{8.2}$$

where

- $B_i = \max_{k \in lp(i)} (C_k) + 3\tau_{bit}$ is the worst-case blocking-time of frames sent on S_i, where $lp(i)$ is the set of streams with priority lower than S_i. The reason for the blocking factor is that transmissions are non pre-emptive, i.e., after bus arbitration has started the frame with the highest priority among competing frames will be transmitted until completion, even if a frame with higher priority gets queued before the transmission is completed.
- $hp(i)$ is the set of streams with priority higher than S_i.

- τ_{bit} (the bit-time) caters for the difference in arbitration start-times at the different nodes due to propagation delays and protocol tolerances.
- C_j is the transmission time of message j. How to calculate C_j is presented in the next section.
- $3\tau_{bit}$ represents the inter-frame space (traditionally [13, 14, 15], the inter-frame space was considered a part of the data frame, but separating it [3] removes a small source of pessimism in the equations).

Note that Equation 8.2 is a recurrence relation, where the approximation to the $(n+1)$th value is found in terms of the nth approximation, with the first approximation set to $q_i^0 = 0$. A solution is reached either when the $(n+1)$th value is equal to the nth, or when R_i exceeds its message deadline or period. The recurrence relation will terminate given that the total bus utilization is ≤ 1, i.e., $\sum_{S_i \in \mathcal{S}} \left(\frac{C_i + 3\tau_{bit}}{T_i} \right) \leq 1$.

We rewrite Equation 8.1 and Equation 8.2 into a single expression since our probabilistic equations, in the following section, will be based on having such an expression. Having a single expression we will be able to separate the "fixed size" part of the calculations from the "varying size part" based on distributions. The new expression is

$$R_i^n = J_i + B_i + C_i + \sum_{j \in hp(i)} I_j \left(R_i^{n-1} - J_i - C_i \right) (C_j + 3\tau_{bit}) \qquad (8.3)$$

where $I_j(t)$ is defined as the worst-case number of periodic message releases, for a message j, in a time interval of t

$$I_j(t) = \left\lceil \frac{t + J_j + \tau_{bit}}{T_j} \right\rceil \qquad (8.4)$$

where J_j is the worst-case release jitter, and T_j is the period of the message.

As Equation 8.2, Equation 8.3 is a recurrence relation. The only difference is that the first approximation is in this case set to $R_i^0 = J_i + C_i$.

8.2.2 The Bit-Stuffing Mechanism

In CAN, six consecutive bits of the same polarity (111111 or 000000) is used for error and protocol control signalling. To avoid these special bit-patterns in transmitted frames, a bit of opposite polarity is inserted after five consecutive

bits of the same polarity. By reversing the procedure, these bits are then removed at the receiver side. This technique, which is called *bit-stuffing*, implies that the actual number of transmitted bits may be larger than the size of the original frame, corresponding to an additional transmission delay which needs to be considered in the analysis.

Let us first define the number of bits, beside the data part in the frame, which are exposed to the bit-stuffing mechanism as $g \in \{34, 54\}$. This since we have either 34 (CAN standard format) or 54 (CAN extended format) bits (beside the data part in the frame) which are exposed to the bit-stuffing mechanism. 10 bits in the CAN frame are not exposed to the bit-stuffing mechanism (see Figure 8.2). Now let us define the number of bytes of data in CAN message frame i as $L_i \in [0, 8]$. Recall, a CAN message frame can contain 0 to 8 bytes of data. According to the CAN standard [10], the total number of bits in a CAN frame before bit-stuffing is therefore

$$8L_i + g + 10 \tag{8.5}$$

where 10 is the number of bits in the CAN frame not exposed to the bit-stuffing mechanism. Since only $g + 8L_i$ bits in the CAN frame are subject to bit-stuffing, the total number of bits after bit-stuffing can be no more than

$$8L_i + g + 10 + \left\lfloor \frac{g + 8L_i - 1}{4} \right\rfloor \tag{8.6}$$

Intuitively the above formula captures the number of stuff-bits in the worst-case scenario, shown in Figure 8.3.

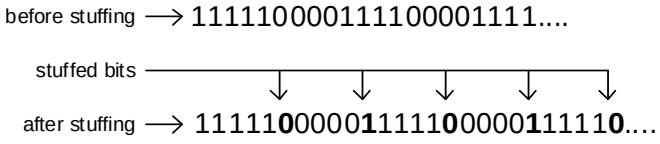

Figure 8.3: The worst-case scenario when stuffing bits.

Let τ_{bit} be the worst-case time taken to transmit a bit on the bus – the so-called *bit-time*. The worst-case time taken to transmit a given frame i is therefore

$$C_i = \left(8L_i + g + 10 + \left\lfloor \frac{g + 8L_i - 1}{4} \right\rfloor\right) \tau_{bit} \tag{8.7}$$

8.3 New Approach

The expression (8.6) describes the length of a CAN frame in the worst case. However, in our previous work [8, 9] we represent the number of stuff-bits as a distribution. By using a distribution of stuff-bits instead of the worst-case number of stuff-bits, we obtain a distribution of response-times allowing us to calculate less pessimistic (compared to traditional worst-case) response-times based on probability.

Firstly, let us define Υ as the distribution of stuff-bits in a CAN message frame. Υ is a set of pairs containing the number of stuff-bits with corresponding probability of occurrence. Each pair is defined as $(x, P(x)) \in \Upsilon$, where $P(x)$ is the probability of exactly x stuff-bits in the CAN frame. Note that $\sum_{x=0}^{\infty} P(x) = 1$.

From [9] we can extract 9 different distributions of stuff-bits depending on the number of bytes of data in the CAN message frame. We define Υ_{L_i} as the distribution representing a CAN frame containing L_i bytes of data. Recall that L_i is the number of bytes of data (0 to 8) in a message frame i.

We define $n = \Upsilon(p)$ as the worst-case number of stuff-bits, n, to expect with a probability p based on the stuff-bit distribution Υ, i.e., $\sum_{x=n+1}^{\infty} P(x) \leq p$, or to express it in another way, the probability of finding more than n stuff-bits, based on the stuff-bit distribution Υ, is $\leq p$.

Note that the selection of a probability p should be done based on the requirements of the application. With a proper value for p, the worst-case mean time to failure should sufficiently exceed what is required.

Finally, by assuming (as in [9]) that CAN message frames are independent in the sense of number of stuff-bits, we can define $\prod_{n} \Upsilon$ as the joint distribution corresponding to the combination of n distributions of stuff-bits, i.e., the number of stuff-bits caused by a sequence of n messages sent on the bus is described by $\prod_{n} \Upsilon = \underbrace{\Upsilon \times \Upsilon \times \cdots \times \Upsilon}_{n}$, where $\times$ denotes multiplicative combination of discrete distributions, as illustrated in the example below. If the distributions happens to be equal, $\overline{\prod_{n}} \Upsilon$ is defined as the joint distribution of n *equal* distributions of stuff-bits, i.e., the number of data bytes are the same for all messages considered by the expression.

8.3.1 Example

As an illustration, let us use an example where we assume $\Upsilon = \{(0, 0.1), (1, 0.8), (2, 0.1)\}$. Calculating $\bar{\prod_2}\Upsilon$ is done by multiplying the probabilities for all combinations of stuff-bits, i.e., $(a, P(a)) * (b, P(b)) = (a+b, P(a) * P(b))$ where $a, b \in \{0, 1, 2\}$. The result of a multiplication is a new number of stuff-bits with a corresponding probability. In our example the multiplication yields

$$\begin{aligned}\bar{\prod_2}\Upsilon = &\{(0, 0.01), (1, 0.08), (2, 0.01), (1, 0.08), (2, 0.64),\\ &(3, 0.08), (2, 0.01), (3, 0.08), (4, 0.01)\}\end{aligned} \tag{8.8}$$

However, all probabilities in $\bar{\prod_2}\Upsilon$ of equal number of stuff-bits are added together leaving

$$\bar{\prod_2}\Upsilon = \{(0, 0.01), (1, 0.16), (2, 0.66), (3, 0.16), (4, 0.01)\} \tag{8.9}$$

In our example, with $p = 10^{-1}$, $\Upsilon(p) = 1$ and $\left(\bar{\prod_2}\Upsilon\right)(p) = 3$.

8.3.2 Probabilistic Worst-Case Response-Time

In order to include the bit-stuffing distributions in Equation 8.3 we need to redefine C_i and B_i to $C_i(p)$ and $B_i(p)$ where

- $C_i(p)$ is the transmission time of message i

$$C_i(p) = c_i + \Upsilon_{L_i}(p)\tau_{bit} \tag{8.10}$$

where Υ_{L_i} is the distribution of stuff-bits in the message, and c_i is the transmission time of message i excluding stuff-bits

$$c_i = (8L_i + g + 10)\,\tau_{bit} \tag{8.11}$$

where 10 is the number of bits in the CAN frame not exposed to the bit-stuffing mechanism.

- $B_i(p)$ is the blocking-time caused by message i having to wait for a lower priority message sent on the bus. Since the bus is non pre-emptive, the worst-case scenario is that the biggest (in size) lower priority message just started its transmission when message i becomes ready to transmit. Thus we can define the blocking-time of a message i as

$$B_i(p) = b_i + \Upsilon_{\max\limits_{k \in lp(i)}(L_k)}(p)\tau_{bit} \tag{8.12}$$

  where $\Upsilon_{\max\limits_{k \in lp(i)}(L_k)}$ is the distribution of stuff-bits of the blocking message k (the biggest lower priority message), and b_i is the blocking-time not considering the bit-stuffing mechanism

$$b_i = \max_{k \in lp(i)}(c_k) + 3\tau_{bit} \tag{8.13}$$

  where $3\tau_{bit}$ is the inter-frame space. Note that (8.12) is pessimistic in the sense that we always assume that we will be blocked by a message. Taking probability of blocking actually occurring into consideration as well as not always assuming biggest blocking message would give a less pessimistic result. However, since we are basing the analysis on a "critical instant", we create a worst-case scenario but we use distributions of values instead of worst-case ones when calculating the response-time.

Taking the probabilistic definitions of Equation 8.10 and Equation 8.12 into consideration we can reformulate Equation 8.3 as

$${R_i}^n(p) = J_i + b_i + c_i + \sum_{j \in hp(i)} I_j({R_i}^{n-1}(p) - J_i - c_i)\,(c_j + 3\tau_{bit}) + \Psi_i(p)\tau_{bit} \tag{8.14}$$

where Ψ_i is defined as the distribution of the total number of stuff-bits of all messages involved in the response-time analysis for message i

$$\Psi_i = \Upsilon_{\max\limits_{k \in lp(i)}(L_k)} \times \Upsilon_{L_i} \times \prod_{j \in hp(i)} \bar{\prod_{I_j(R_i(p) - J_i - c_i)}} \Upsilon_{L_j} \tag{8.15}$$

where $\Upsilon_{\max\limits_{k \in lp(i)}(L_k)}$ is the distribution of stuff-bits caused by the longest lower priority blocking message, Υ_{L_i} is the distribution of stuff-bits in the message

under analysis, and $\prod_{j \in hp(i)} \bar{\prod}_{I_j(R_i(p)-J_i-c_i)} \Upsilon_{L_j}$ is the distribution of stuff-bits in all interfering messages of higher priority sent before message i will be sent, i.e., the higher priority messages sent causing message i to be queued.

Having the distribution Ψ_i, a proper total number of stuff-bits is selected depending on the desired probability of response-time violation p, i.e., for every step in the recurrence relation (8.14), a value $\Psi_i(p)$ must be extracted from Equation 8.15.

8.3.3 Complexity

Regarding the complexity of the analysis, the dominating component is the calculation in Equation 8.15. Since all parameters in Equation 8.15 are distributions, and distributions are multiplied together causing multiplications of all combinations of stuff-bits, the complexity of solving the expression is as follows

$$\mathcal{O}(k^l) \tag{8.16}$$

where l is the number of messages involved in Equation 8.15, and k is the number of stuff-bits in the biggest size message, having largest number of stuff-bits in its distribution. However, due to the iterative nature of the equations, solving Equation 8.15 can be done with a much lower complexity. In fact, the complexity of calculating the joint distribution can be reduced to

$$\mathcal{O}(l * k^2) \tag{8.17}$$

since we in each iteration can reduce the number of considered values to $k * l$ by adding all values with equal number of stuff-bits together, as illustrated in Section 8.3.1.

8.3.4 Example

To illustrate our method we use a small example with 3 messages, message 1-3, where message 1 has the highest priority, and message 3 the lowest priority. We assume that we have no jitter, i.e., $J = 0$ for all messages, and that all messages have the same size. Note that the assumption regarding the message length to be equal is just for simplicity for the reader. This is not a requirement. We assume $c = 10$, $\tau_{bit} = 1$, and Υ is as in Section 8.3.1. Finally, again for

simplicity, all message periods are so big causing Equation 8.4 never to exceed 1, i.e., $I(t) = 1$.

Based on our assumptions, the worst-case scenario for message 2 would be as illustrated in Figure 8.4, i.e., message 2 is blocked by message 3 (the lowest priority message) and delayed by message 1 (the highest priority message).

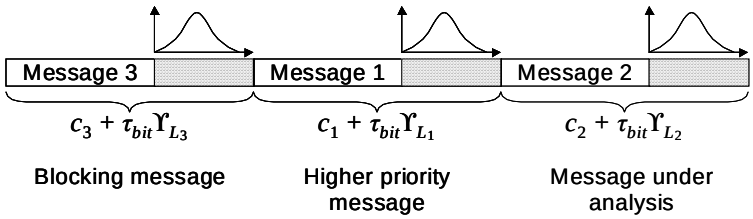

Figure 8.4: Worst-case message sequence for message 2.

Using Equation 8.14 we can calculate the response-time $R_2(p)$ as

$$R_2(p) = b_2 + c_2 + (c_1 + 3\tau_{bit}) + \Psi_2(p)\tau_{bit} \tag{8.18}$$

where $b_2 = c_3 + 3\tau_{bit}$ and $\Psi_2 = \Upsilon_{L_3} \times \Upsilon_{L_2} \times \Upsilon_{L_1} = \overline{\prod_{3}}\Upsilon$ (since $L_1 = L_2 = L_3$) where $\overline{\prod_{3}}\Upsilon$ is calculated to be

$$\begin{aligned}\overline{\prod_{3}}\Upsilon = \{&(0, 0.001), (1, 0.024), (2, 0.195), (3, 0.56),\\ &(4, 0.195), (5, 0.024), (6, 0.001)\}\end{aligned} \tag{8.19}$$

We select an acceptable probability of worst-case response-time violation p to be 10^{-1}. Based on p, $\Psi_2(p) = 4$, causing $R_2(p) = (10+3)+10+(10+3)+4 = 40$.

8.4 Evaluation

In order to demonstrate the performance of our new approach for calculating a probabilistic worst-case response-time we make use of the widely published

Priority (ID)	Bytes	C_i (ms)	T_i (ms)	D_i (ms)	R_i (ms)	$p = 10^{-24}$ $R_i(p)$ (ms)	gain (%)	$p = 10^{-12}$ $R_i(p)$ (ms)	gain (%)	R_i^{sim} (ms)
17	1	0.480	1000	5	1.416	1.384	2.26	1.328	6.21	0.680
16	2	0.560	5	5	2.016	1.936	3.97	1.864	7.54	1.240
15	1	0.480	5	5	2.536	2.448	3.47	2.360	6.94	1.720
14	2	0.560	5	5	3.136	3.032	3.32	2.920	6.89	2.280
13	1	0.480	5	5	3.656	3.536	3.28	3.424	6.35	2.760
12	2	0.560	5	5	4.256	4.120	3.20	4.000	6.02	3.320
11	6	0.864	10	10	5.016	4.840	3.51	4.720	5.90	4.184
10	1	0.480	10	10	8.376	5.368	35.91	5.248	37.34	4.664
9	2	0.560	10	10	8.976	8.480	5.53	8.336	7.13	5.224
8	2	0.560	10	10	9.576	9.144	4.51	9.000	6.02	8.424
7	1	0.480	100	100	10.096	9.728	3.65	9.592	4.99	8.904
6	4	0.712	100	100	19.096	15.256	20.11	10.304	46.04	9.616
5	1	0.480	100	100	19.616	18.472	5.83	18.176	7.34	10.096
4	1	0.480	100	100	20.136	19.224	4.53	18.968	5.80	18.320
3	3	0.632	1000	1000	28.976	19.928	31.23	19.704	32.00	18.952
2	1	0.480	1000	1000	29.496	27.920	5.34	20.400	30.84	19.432
1	1	0.480	1000	1000	29.520	28.352	3.96	27.944	5.34	19.912

Table 8.1: SAE CAN messages.

simplification [14] of the Society of Automotive Engineers (SAE) benchmark [11].

We use a bus speed of 125kbit/s, and we select the acceptable probability of violating the calculated worst-case response-time, p, to be 10^{-24} and 10^{-12} respectively. Then, we calculate the worst-case response-time both according to the traditional approach (8.1) and the probabilistic approach (8.14). The response-times of all messages of the subset of SAE messages are shown in Table 8.1, where R_i denotes the results of traditional analysis and $R_i(p)$ the results of our new probabilistic analysis. To have some "real" response-times to compare the analytic ones with, we simulated the SAE message set using the worst-case transmission times. The system was simulated for 2000000 ms. The worst-case measured response-time is presented as R_i^{sim} in the rightmost column of Table 8.1. Note that the difference between the simulated value and the analytic worst-case is due to that in the simulation all messages are released at time "0". Hence, the worst-case blocking as defined by Equation 8.12 might not occur due to the phasings of messages.

What we see in Table 8.1 is that the probabilistic response-times $R_i(p)$ are significantly lower than the traditional worst-case response-times R_i. An interesting observation is that the gain is substantially higher for some mes-

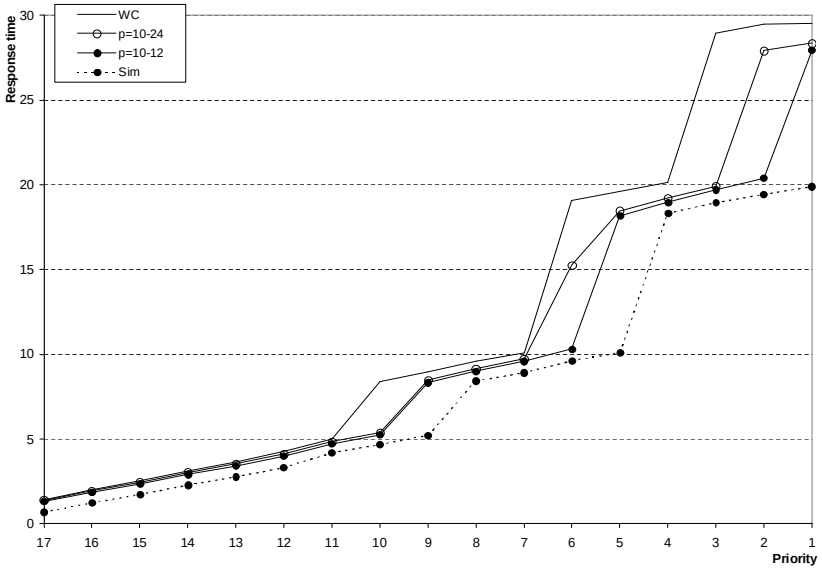

Figure 8.5: Message response-times (priority is the message ID as in Table 8.1).

sages. The reason for this is that a slight additional interference, e.g., caused by an additional stuff-bit, will in these cases extend the response-time such that transmission will be delayed by one or more additional higher priority message transmissions. Note that all calculated probabilistic response-times are never optimistic in comparison with the simulation result (as seen in Figure 8.5). This even though we are using worst-case transmission times. Using bit-stuffing distributions in the simulation would give even shorter response-times.

8.5 Conclusions

In this paper we have presented a new probabilistic approach to calculate response-times for messages in the Controller Area Network (CAN). The key element to this approach is that we use bit-stuffing distributions instead of worst-case values. The performance of our method is evaluated using a subset of the SAE benchmark.

Our main motivation for calculating probabilistic response-times is that they allow us to reason about trade-offs between reliability and timeliness. We have in [6] presented a method for such analysis of CAN subject to external interference. An obvious next step would be to integrate the bit-stuffing distribution-based analysis presented here with that analysis.

Acknowledgements

The authors wish to express their gratitude to the anonymous reviewers for their helpful comments. 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.

Bibliography

[1] A. Atlas and A. Bestavros. Statistical Rate Monotonic Scheduling. In *Proceedings of the* 19^{th} *IEEE Real-Time Systems Symposium (RTSS'98)*, pages 123–132, Madrid, Spain, December 1998. IEEE Computer Society.

[2] 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.

[3] I. Broster and A. Burns. Timely Use of the CAN Protocol in Critical Hard Real-Time Systems With Faults. In *Proceedings of the* 13^{th} *Euromicro Conference on Real-Time Systems (ECRTS'01)*, pages 95–102, Delft, The Netherlands, June 2001. IEEE Computer Society.

[4] A. Burns. Preemptive Priority Based Scheduling: An Appropriate Engineering Approach. Technical Report YCS 214, University of York, York, England, 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] H. Hansson, T. Nolte, C. Norström, and S. Punnekkat. Integrating Reliability and Timing Analysis of CAN-based Systems. *IEEE Transaction on Industrial Electronics*, 49(6), December 2002.

[7] S. Manolache. Schedulability Analysis of Real-Time Systems with Stochastic Task Execution Times. Licentiate Thesis No. 985, Dept. of Computer and Information Science, IDA, Linköping University, Sweden, December 2002.

[8] T. Nolte, H. Hansson, and C. Norström. Using Bit-Stuffing Distributions in CAN Analysis. 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.

[9] T. Nolte, H. Hansson, and C. Norström. Minimizing CAN Response-Time Analysis Jitter by Message Manipulation. In *Proceedings of the* 8^{th} *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'02)*, pages 197–206, San Jose, CA, USA, September 2002. IEEE Computer Society.

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

[11] SAE. Class C Application Requirement Considerations-SAE J2056/1. *SAE Handbook*, pages 23.366–23.371, June 1993.

[12] L. Sha, R. Rajkumar, and J.P. Lehoczky. Priority Inheritance Protocols: An Approach to Real-Time Synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, September 1990.

[13] 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.

[14] 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.

[15] 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.

[16] J. Xu and D. L. Parnas. Priority Scheduling Versus Pre-Run-Time Scheduling. *Real-Time Systems*, 18(1):7–23, January 2000.

Chapter 9

Paper D: Server-Based Scheduling of the CAN Bus

Thomas Nolte, Mikael Sjödin, and Hans Hansson,
Technical Report ISSN 1404-3041 ISRN MDH-MRTC-99/2003-1-SE, Mälardalen Real-Time Research Centre, Mälardalen University, Sweden, April 2003.

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.

9.1 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) [21, 20], 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 9.2 related work is presented. In Section 9.3 we present the server-based CAN network. Section 9.4 presents an approach to analysis, and in Section 9.5 the proposed method is evaluated using simulation. Finally, in Section 9.6 we conclude and present future work.

9.2 Background and Related Work

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

9.2.1 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 outputting 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.

9.2.2 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.

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 9.1.

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.

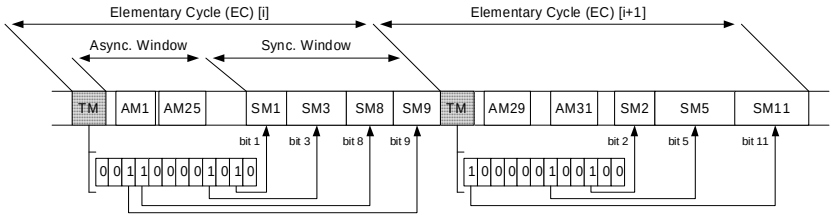

Figure 9.1: EC layout and TM data contents (FTT-CAN approach).

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.

9.3 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.

9.3.1 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 [21, 22] 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 [21, 20]. 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

$$d_n = \max(r_n, d_{n-1}) + \frac{C_n}{U_s} \tag{9.1}$$

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

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.)

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

$$d_s^n = \max(r_n + D_s, d_s^{n-1}) \tag{9.2}$$

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

$$d_s^n = d_s^{n-1} + D_s \tag{9.3}$$

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

9.3.2 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 9.2.2. 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 9.2 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.

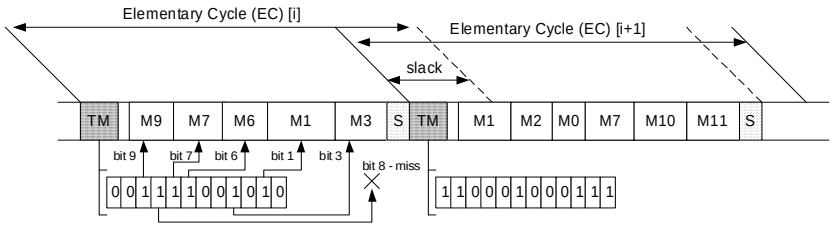

Figure 9.2: EC layout and TM data contents (server approach).

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.

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] 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. 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 9.3.1.

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 9.3.1.

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.

9.4 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 9.4. We can not allocate N-Servers with a total utilisation higher than inequality 9.4, 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

$$\sum_{\forall s} \left(\frac{S \times M}{T_s} \right) \leq \left(\frac{S \times T_{EC} - (TM + STOP)}{S \times T_{EC}} \right) \tag{9.4}$$

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.

9.4.1 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 9.4 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 $2 \times T_{EC}$.

9.5 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 9.4)	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 9.1: 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 9.1.

9.5.1 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 9.3, 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 9.4, 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.

9.5.2 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 9.5, 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 9.6, 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 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 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 9.4. Note that only 1 message from a total of 12964 messages was delivered as late as $T_s + 2 \times T_{EC}$.

9.5.3 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.

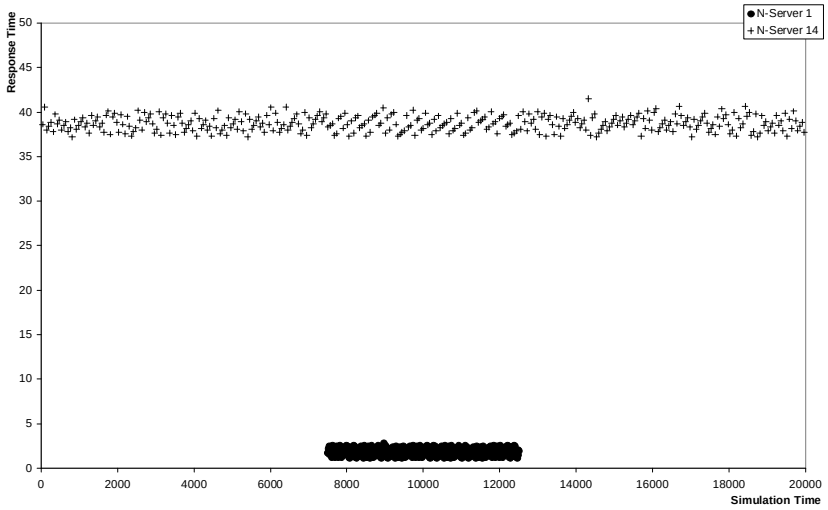

Figure 9.3: Experiment 1 (medium bandwidth usage) – scenario 1.

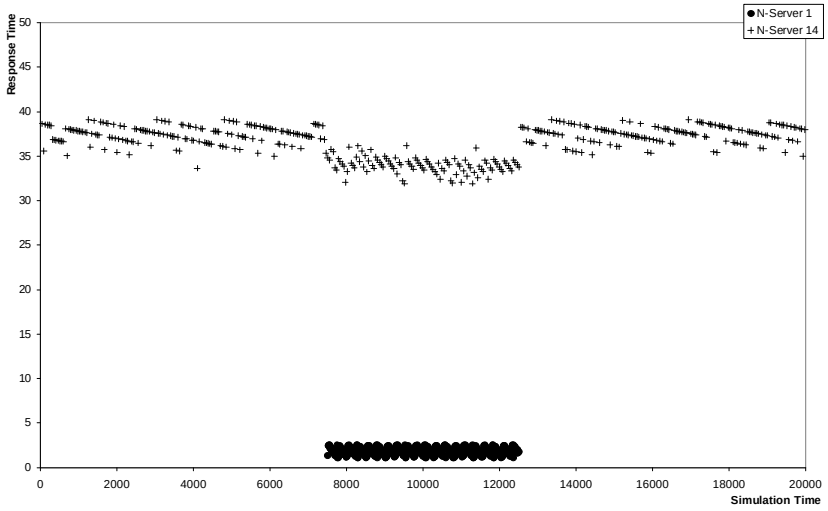

Figure 9.4: Experiment 2 (high bandwidth usage) – scenario 1.

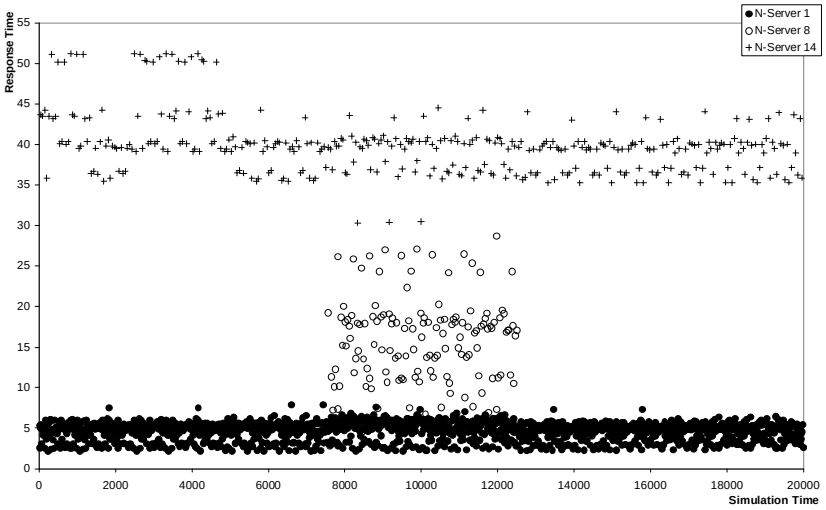

Figure 9.5: Experiment 1 (medium bandwidth usage) – scenario 2.

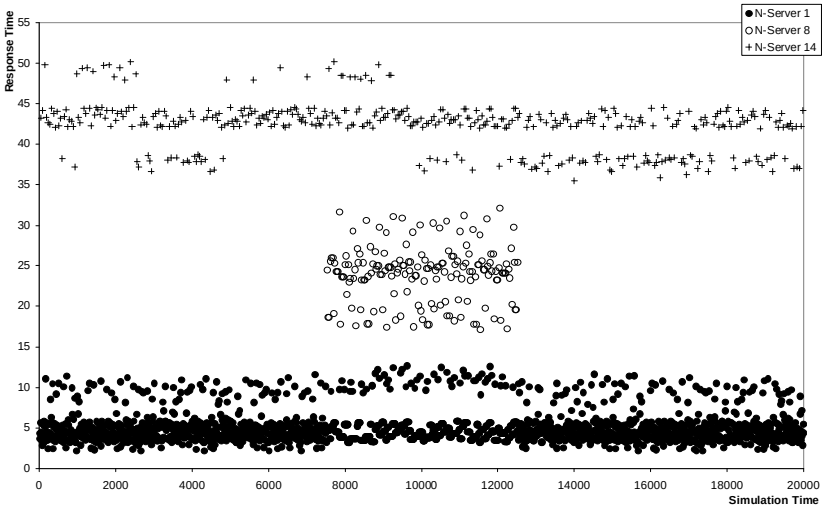

Figure 9.6: Experiment 2 (high bandwidth usage) – scenario 2.

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 $T_s + 2 \times T_{EC}$.

9.5.4 Discussion

The data obtained from all 3 scenarios under both experiments is presented in Table 9.2, 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 9.4.1). 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 9.4.1). 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 9.4.

9.6 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)

		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 9.2: Summary of simulation results.

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 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.

Bibliography

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

[2] L. Almeida, J.A. 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.A. 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] SAE 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. Di 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.K Parekh and R.G. 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.P. Lehoczky. Aperiodic Task Scheduling for Hard Real-Time Systems. *Real-Time Systems*, 1(1):27–60, 1989.

[20] 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.

[21] M. Spuri and G.C. 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.

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

[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.M. Zuberi and K.G. 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.

Chapter 10

Paper E: Distributed Real-Time System Design using CBS-based End-to-end Scheduling

Thomas Nolte, and Kwei-Jay Lin[1]
In Proceedings of the 9^{th} IEEE International Conference on Parallel and Distributed Systems (ICPADS'02), Taipei, Taiwan, ROC, December 2002.

[1]Department of Electrical and Computer Engineering, University of California, Irvine, CA 92697, USA.

Abstract

Distributed real-time applications share a group of processors connected by some local area network. A rigorous and sound methodology to design real-time systems from independently designed distributed real-time applications is needed. In this paper, we study a distributed real-time system design scheme using CBS-based end-to-end scheduling. The scheduling scheme utilizes CBS to allocate both CPU shares and network bandwidth to a distributed real-time application when it arrives at the system. Our proposed solution uses the same scheduling paradigm for both resources. In this way, we believe the system can have a more consistent scheduling objective and may achieve a tighter schedulability condition.

10.1 Introduction

There has been an increasing demand on distributed systems with real-time requirements. In distributed systems, many real-time applications may be developed independently and then run on a distributed computing system by sharing a group of processors connected by some local area network. For example, an automated factory control system may have several hard real-time sensor monitoring applications and several robot control tasks running concurrently in a distributed computing environment. Although each hard real-time application may have been verified to meet its deadlines when running by itself, its performance may not be as predictable when it is running concurrently with a dynamic set of real-time applications. We need a rigorous and sound methodology to compose complex real-time systems from independently designed distributed real-time applications.

The Constant Bandwidth Server (CBS) [1] is a scheduling algorithm based on reserving a fraction of the processor bandwidth to serve aperiodic jobs. CBS uses a deadline postponing mechanism to efficiently provide bandwidth isolation, and is able to achieve per-server performance guarantees.

In this paper, we study a distributed real-time system design scheme using CBS-based end-to-end scheduling. The scheduling scheme utilizes CBS to allocate both CPU shares and network bandwidth to a distributed real-time application when it arrives at the system. An admission server will be used to monitor the system workload on all processors and the network. If the system has enough capacity, and the total worst-case delay from all computing components (CPU's and network) can meet the application requirement, the application will be admitted. In this way, hard real-time constraints can always be met.

Our proposed solution differs from earlier work in the following ways. First, most previous work concentrates on either CPU or network scheduling. To our knowledge, very few work study integrated CPU and network scheduling. Second, we use the same scheduling paradigm for both resources. In this way, we believe the system can have a more consistent scheduling objective. Third, by using CBS, we provide the capability to serve both periodic and aperiodic applications at the same time. This is very desirable for most real-time systems that require interactive control while monitoring periodic events.

The paper is organized as follows. In Section 10.2 we review the background on CBS and real-time open environment. We also review the CBS-based CAN protocol in Section 10.3. Section 10.4 presents our proposed CBS-based real-time computing architecture. Some simulation results are presented

in Section 10.5 and the paper is concluded in Section 10.6.

10.2 CBS and Real-Time Open Environment

In this paper, we propose a CBS-based distributed real-time system architecture. Before we present our idea, let us review previous work on CBS scheduling and also real-time open architecture.

10.2.1 Constant Bandwidth Server (CBS)

A CBS server [1] is defined by (Q_i, T_i), where T_i is the period of the server and Q_i is the maximum budget in each period. The ratio $U_i = \frac{Q_i}{T_i}$ is the fraction of CPU bandwidth reserved for the server. At run time, each server S_i maintains a pair of parameters: (c_i, d_i), where c_i is the current budget available, and d_i is current scheduling deadline. Each application is associated with a CBS server. During run time when the application has a job waiting to be executed, it is executed only if the server still has any budget left. If so, the current server deadline is assigned to be the current job deadline.

The system is assumed to have n CBS servers and a kernel scheduler based on EDF. Each server has a job list which includes the jobs to be served on this server. In CBS, all tasks in this system are assumed to be independent from one another; in other words, they do not share any resources.

The most important property of CBS is the bandwidth isolation property which ensures that each server S_i is guaranteed at least U_i of the total system utilization regardless of other servers' loads. The misbehavior of some task will not jeopardize other tasks' bandwidth allocations. Another property of CBS is the hard schedulability property in that the schedulability test can be independently performed for each real-time task.

10.2.2 The Real-Time Open Environment Architecture

Rate-driven scheduling has been an active research topic in the past decade. Researchers have proposed various rate-driven scheduling algorithms for periodic and sporadic processes based on the notion of General Processor Sharing (GPS) [1, 5, 10]. GPS is a scheduling algorithm based on the concept of the reservation ratio of processor computation time. Suppose a GPS server executes at a fixed rate r (which is usually one), and each process τ_i has a *reservation ratio* θ_i which is a positive real number. Each process τ_i is guaranteed

to be served at a rate of

$$g_i = \frac{\theta_i}{\sum_j \theta_j} r$$

independent of the actual workloads of other processes.

Using the guaranteed CPU sharing scheme similar to GPS, Liu *et al.* [3] propose the *open real-time environment* architecture to integrate independently developed hard real-time applications on EDF kernel schedulers. In an open real-time environment, real-time and non-real-time applications are allowed to join and leave the system dynamically. However, the schedulability of each real-time application must be guaranteed independent of any other applications in the system. A two-level hierarchical scheduling scheme following the idea of the *GPS scheduling* scheme is proposed to provide a fair sharing of computing budget among applications running on the same processor. Applications are scheduled for execution by a Constant Utilization Server (CUS) [3]. Each CUS server has its reserved CPU budget and is scheduled by an Earliest Deadline First (EDF) scheduler [6] in the kernel. Using the reserved computing budget for the servers, the schedulability for real-time applications with or without shared global resources can be guaranteed [3].

Our proposed system architecture is similar to the work by [3] except we use CBS instead of CUS. Moreover, in our study, we extend the scheduling from CPU only to include both CPU and network. By using CBS on Controller Area Network (CAN) [9, 2], we are able to support real-time network scheduling with an efficient bandwidth control.

10.3 CBS-based CAN Network

We propose a system with a central admission controller which, at new application arrival, verifies that there exits enough computing power in the system as well as network bandwidth demanded by the application. Each application is assumed to have more than one job running on different nodes. If admitted, some CBS servers will be created, at the nodes decided by the admission control, to schedule jobs in the application. Also, a Network-CBS (described in the following section) will be created if the application also needs some guaranteed network bandwidth. The global scheduler will decide which nodes to execute the application, and, in turn, the servers needed.

For distributed systems with loosely-coupled processors, the process migration cost is significant. We therefore assume that each job will be executed only on one processor. We also assume that any job considered in this paper

will require no more than one CPU at a time. Since each job executes on only one computing node, the system scheduler must assign the CBS for each job to a computing node if the application passes the admission control. After that, the scheduling issue for the CBS will be the same as in the uniprocessor case.

In order to provide guaranteed network bandwidth for real-time messages between jobs, we make use of the Controller Area Network (CAN) [9, 2]. By using the method described in [7] we can assign all nodes in a distributed system with a Network CBS (N-CBS) which is used for non-real-time traffic as well as messages used to implement the centralized admission control. Therefore, the whole network can be scheduled as one resource, providing bandwidth isolation as well as fairness among the users of the network. In the following sections we will describe CAN and N-CBS proposed in [7].

10.3.1 The Controller Area Network

CAN 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 between 0 and 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. The identifier serves two purposes: (1) assigning a priority to the frame, and (2) enabling receivers to filter frames. For a more detailed explanation of the different fields in the CAN frame, please consult [9, 2].

The basis for the access mechanism is the electrical characteristics of a CAN bus. During arbitration, competing stations are simultaneously putting their identifiers, one bit at the time, on the bus. By monitoring the resulting bus value, a station detects if there is a competing higher priority frame 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 queued frame, and hence can start transmitting the body of the frame. Thus, CAN behaves as a priority based queue since at all nodes, the message chosen for arbitration is always the highest priority message.

10.3.2 CBS on CAN

In order to support fair network scheduling with bandwidth isolation we make use of an approach [7] to implement the CBS on the CAN. The method supports fairness and potentially hard deadlines among the messages sent on the network. In [7] periodic and aperiodic messages are treated separately but scheduled together on a *Master Node*. Aperiodic messages are scheduled using CBS servers and periodic messages get exclusive service. However, in this paper we only consider scheduling CBS servers. Both periodic and aperiodic messages are sent using these servers.

All messages are scheduled together on a central node called *Master Node*. In order to handle periodic and aperiodic messages, all nodes that are sending messages are assigned with one or more Network CBS servers (N-CBS).

Using N-CBS servers, the messages are scheduled based on their deadlines according to EDF. In order to implement EDF scheduling on the network, we need to have a feedback mechanism to the centralized network scheduler (located on the Master Node), so that it can deduce the internal deadline for each N-CBS. This feedback is not based on message passing, since this would further reduce the already low bandwidth offered by CAN. In our design, the Master Node is maintaining all network information (e.g., N-CBS state variables) and performing the message scheduling. The slaves in the system, are just "dumb" nodes following the schedule provided by the Master Node. In the following section we will describe how it works.

Scheduling

The Master Node maintains an internal state for each N-CBS in the system. This state is never explicitly sent from slaves to the master. Instead it is the Master's job to try to predict and track this state. Thus, N-CBS servers assigned to different nodes in the system can be thought of as *virtual* N-CBS servers.

When scheduling messages, time is divided into intervals called *Elementary Cycles* (ECs). The length of an EC is denoted by T_{EC}. The states of N-CBS servers initially are based on the *assumption* that messages always arrive at each N-CBS. The EC-schedule is then constructed by selecting messages (in deadline order) from all "arrived" messages to fit in the next EC. Then, the schedule is encoded in a Trigger Message (TM) and sent to slaves. After reception of this TM message eligible nodes will start to send their messages on the CAN bus.

Maintaining N-CBS Server States

Since we assume that every CBS always has one message arrival, there may be some messages allocated in an EC which actually are not there. We now describe a scheme to reclaim this potential slack. We use the following mechanism to "prematurely" terminate the EC when the bus becomes idle.

After a TM has been sent, the Master sends a stop message (STOP) in the EC.[2] The STOP message is defined to have the lowest priority possible. After sending the STOP message to the CAN controller, the Master Node reads all messages sent on the bus. When it reads the STOP message it knows that all eligible messages in the EC have been sent (since the CAN bus behaves like a priority-based queue). The remaining time before the end of the EC will be the slack.

There are basically two approaches to do with the slack: either to just consider the EC completed and start the next EC immediately, or to maintain the original EC periodicity and add the slack to the next EC, thus making the length of the upcoming EC longer. Both approaches have their advantages and disadvantages. By keeping the original periodicity and by expressing all message periods as multiples of T_{EC}, we can eliminate the possibility of a message having a deadline inside an EC. This is undesirable since messages scheduled within an EC do not follow the deadline order. A message with an earlier deadline than the end of EC will get a penalty in terms of message response time (to be described later). The other approach, by keeping fixed-size EC, will loose the periodicity and thus making it impossible to avoid having deadlines within an EC. In this paper we keep the periodicity. After taking care of the slack, the Master immediately initiates the next EC.

With the help of the STOP message, we have eliminated the slack in the EC (note that this approach takes care of slack both due to slaves not sending messages and due to messages smaller than the maximum allowed size). Using this scheme, the Master needs to look at all the messages that have been sent within the EC and identify them. In this way the Master will know which N-CBS servers that have utilized their scheduled bandwidth within the EC and which have not. Thus the state of the servers can easily be updated by using the CBS scheduling rules for message arrival. The message identification can be made by either having a lookup table mapping messages to N-CBS servers, or by assigning ranges of priorities to the different N-CBS servers.

[2] We need to make sure that this message is not sent before the other nodes in the system have processed the TM in order to find out whether they are allowed to send or not.

Response-Time

An important issue is to decide the length of the EC. In fact, the length of the EC is the temporal resolution for the message delivery since the CAN arbitration mechanism decides when, within an EC, a message will be delivered. When we deliver the schedule for the upcoming EC we tell which nodes that are allowed to send messages. All nodes will submit their messages immediately. However, when exactly each message is delivered within the EC, depends on the competing messages within the EC.

By assigning an application A_i with some bandwidth, characterised by a N-CBS with period T_i and capacity C_i, we have some scenarios. If messages arriving at an N-CBS do not exceed the reserved bandwidth, all messages may be delivered in the deadline containing (deadline = period) EC. However, since messages may arrive to the network in the middle of an EC, they may have to wait until the next EC before they are treated. In other words, messages will be delivered by time

$$R_i = T_i + T_{EC} \tag{10.1}$$

On the other hand, the early termination of EC may cause a higher bandwidth consumption by the protocol (due to the more frequent trigger messages and stop messages). If the system has a high utilisation, this behaviour can cause overload and cause messages to be scheduled in a later EC.

If the message traffic is not periodic we can make use of leaky buckets in order to shape the traffic to the N-CBS servers. Then, the queuing time q_i of the leaky bucket must also be considered, giving us a response-time of

$$R_i = q_i + T_i + T_{EC} \tag{10.2}$$

The queuing time q_i is determined based on the bucket size n and the N-CBS. Hence, $q_i = \left\lceil \frac{(n-1)*c}{C_i} \right\rceil * T_i$, where c is the worst-case message size. The bucket size n is typically decided based on the bursty characteristics of A_i,

10.4 CBS-based Real-Time Architecture

At system startup, all nodes are equipped with one N-CBS to handle non-real-time traffic as well as control messages used to implement the admission control (described below). We let the Master Node be the node performing both the admission control and the network scheduling. This way we can easily exchange parameters from the admission control to the network scheduler, e.g.,

adding N-CBS servers of new applications in the system and removing N-CBS servers of terminated applications.

10.4.1 End-to-end Response-Time

By using the concept of CBS in the whole system the end-to-end response-time is easily calculated. We will examine the scenario of one producer application A_p and one consumer application A_c. Depending of their characteristics we have a number of scenarios. Note that since we use CBS servers, we have bandwidth isolation and thus we do not need to consider interference from other users of the distributed system (as long as we do not use shared resources).

Firstly, lets assume that A_p is a periodic application. In order to make the worst-case response-time as small as possible, we should make the periods of the servers, T_p, T_{N-CBS} and T_c as short as possible. The end-to-end response-time for the distributed system consists of 3 parts; the time it takes to produce the message at the producer, the time it takes to transfer that message and the time it takes to read that message and perform execution. Naturally, the first part is as follows

$$R_p = \left\lceil \frac{C_p}{C_i} \right\rceil * T_i \tag{10.3}$$

where C_i and T_i are the CBS parameters assigned for A_p.

The second part is described in Section 10.3.2. By selecting proper parameters for the N-CBS based on the behaviour of A_p the network is characterized by (10.1).

The third part is derived the same way as (10.3) where j is the CBS assigned to the consumer application A_c.

$$R_c = \left\lceil \frac{C_c}{C_j} \right\rceil * T_j \tag{10.4}$$

However, (10.4) is valid if the consumer is event-triggered. If the consumer is time-triggered, then we need to add T_j to (10.4). Assuming the consumer is event-triggered we get the following end-to-end response time:

$$R = \left\lceil \frac{C_p}{C_i} \right\rceil * T_i + T_{N-CBS} + T_{EC} + \left\lceil \frac{C_c}{C_j} \right\rceil * T_j \tag{10.5}$$

If the producer application is not periodic we can use (10.5) to calculate the worst-case end-to-end response-time based on the assumption of known

burst size, i.e., we assume that n requests for the producer occur at the same time. We simply exchange C_p with $n * C_p$ to get the response-time of the *n*th message together with having proportional bandwidth allocation on both nodes as well as the network, i.e., at any time t, the network can service at least as many end-to-end requests as the producer and the consumer can service at least as many end-to-end requests as the network.

10.4.2 Implementation

The admission process is as follows:

- When a new application A_i arrives at a distributed system on node j, the *Controller Task* of node j sends an *Application Arrival* message to the Master Node, which performs the admission control. Note that if the node of arrival, j, is the Master Node, the request for admission is simply passed to the admission control directly without involving the network.
- The Master Node performs the admission as described in Section 10.4.3 and sends an *Admission Result* message back to node j. The result of the admission is either approval or denial. If the application is approved, information on which node(s) n the application A_i shall execute on is extracted from the Admission Result message.
  - If the node of approval, n, is not the same as arrival, j, ($n \neq j$) the application is migrated to the node of approval.
  - If the admission is denied, the Controller Task of node j rejects the application request.
- If requested, the N-CBS is created at the Master Node and the network scheduler is notified which message identifiers application A_I are using.
- The Controller Task of node n creates the CBS used to schedule application A_i and the application is started.
- When A_i is finished, the CBS and the N-CBS used by A_i are removed from the system.

10.4.3 Admission Control

To guarantee that all real-time applications can meet their deadlines, an open environment must conduct admission control before any real-time application

is deployed in the distributed system. Since no CBS, i.e., application, may migrate across computing nodes, and each CBS executes on one processor, the admission control problem of new applications is, in fact, a bin packing problem [4, 8].

Let a new real-time (or non-real-time) application A_i with processor utilization factor equal to U_i and network bandwidth requirement B_i arrive at a distributed system with M computing nodes. The application A_i typically consists of a set of j jobs. Each of these jobs has some processor utilization requirement u_j and bandwidth requirement b_j. Note that $\sum u_j = U_i$ and $\sum b_j = B_i$. Furthermore, each job may be run on a subset of nodes, S_j, in the distributed system that have the specific capability or devices for the job's execution. The application A_i is admitted if the following two conditions are true:

- For all j jobs in application A_i, there must exist a node $k \in S_j$ where $\sum U_k + u_j \leq 100\%$.
- Based on all l existing N-CBS servers, there must be network bandwidth available so $\sum B_l + B_i \leq 100\%$.

In this paper, we only provide a general admission control mechanism. We refer interested readers to previous research results on the bin packing problem, e.g., [4, 8]. In the worst-case, the processor capacity may be only 50% utilized if all applications need a utilization slightly more than 50% of any single-CPU utilization. However, the network may be fully utilized.

10.5 Simulation Results

In order to verify the performance of a CBS end-to-end system we performed some simulation. The most interesting part is the CBS performance of the network, since CBS CPU scheduling has been studied and verified before. The theoretical upper bound for the network utilisation is derived from the length of the EC in seconds, T_{EC}, and the network speed in bits/second, S, by the following expression (note that TM and STOP are the sizes of the TM and the STOP messages in bits, typically 135 and 55 bits):

$$\frac{S * T_{EC} - (TM + STOP)}{S * T_{EC}} \tag{10.6}$$

We simulated a 125 Kbps network. The size of the EC is chosen to contain 10 messages which gives us a theoretical system load upper bound of 0.876623.

In the simulation we randomised (with rectangular distribution) a set of 32 periodic messages. These message periods are selected within specific ranges. Messages 1-4 have a period of 1 to 2 T_{EC}, messages 5-16 have a period of 2 to 4 T_{EC}, and messages 17-32 have a period of 4 to 6 T_{EC}. A valid set of messages was specified to have a total utilisation less than the maximum theoretical utilisation + 5%.

The valid set of messages was simulated for 20 seconds, typically generating some 500 to 1500 instances of messages and the message response-times were recorded. When the system load is less than 80% all messages are delivered within their period. But when the system load approaches the theoretical upper bound typically 1 to 3 messages have a worst measured response-time of $T + T_{EC}$. The reason is that, we believe, pessimistic assumption regarding message readiness at all system nodes causes increased bandwidth usage by the protocol. This leads to shorter periods, and system loads greater than the theoretical upper bound, thus causing some messages to be scheduled in the following EC. However, in most cases messages are never delivered at a time later than the message period.

Knowing that the network is performing as we expected, together with the CBS CPU scheduling performance, we believe that the CBS end-to-end scheduling presented in this paper is a feasible approach.

10.6 Conclusions

In this paper we study distributed real-time system design. In this work, we concentrate on automated systems where all nodes are connected using the Controller Area Network (CAN). The N-CBS algorithm [7] is used since it provides bandwidth isolation as well as real-time performance on CAN. However, other networks with similar properties may also be used. Our design uses the same scheduling scheme, i.e., Constant Bandwidth Server (CBS), for both CPU and network scheduling. In this way, it is easy for us to derive the end-to-end delay for application execution. We plan to continue the performance study on a practical CAN bus system in the near future.

Acknowledgements

Thomas Nolte 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), LM Ericsson's

Research Foundation, and Mälardalen University. Kwei-Jay Lin was supported in part by NSF CCR-9901697.

Bibliography

[1] L. Abeni and G. Buttazzo. Integrating Multimedia Applications in Hard Real-Time Systems. pages 4–13, Madrid, Spain, December 1998. IEEE Computer Society.

[2] 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.

[3] Z. Deng and J.W.-S. Liu. Scheduling Real-Time Applications in an Open Environment. In *Proceedings of the* 18^{th} *IEEE Real-Time Systems Symposium (RTSS'97)*, pages 308–319, San Francisco, CA, USA, December 1997. IEEE Computer Society.

[4] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman & Company, Publishers, San Francisco, CA, USA, 1979.

[5] T.-W. Kuo, W.-R. Yang, and K.-J. Lin. EGPS: A Class of Real-Time Scheduling Algorithms Based on Processor Sharing. In *Proceedings of the* 10^{th} *Euromicro Conference on Real-Time Systems (ECRTS'98*, pages 27–34, Berlin, Germany, June 1998. IEEE Computer Society.

[6] 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.

[7] 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.

[8] C. Papadimitriou and K. Steiglitz. Combinatorial Optimization: Algorithms and Complexity. *Prentice-Hall, Inc*, 1982.

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

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