Real-Time Syst (2007) 35:239–272
DOI 10.1007/s11241-007-9012-7

Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised

Robert I. Davis · Alan Burns · Reinder J. Bril · Johan J. Lukkien

Published online: 30 January 2007

Abstract Controller Area Network (CAN) is used extensively in automotive applications, with in excess of 400 million CAN enabled microcontrollers manufactured each year. In 1994 schedulability analysis was developed for CAN, showing how worst-case response times of CAN messages could be calculated and hence guarantees provided that message response times would not exceed their deadlines. This seminal research has been cited in over 200 subsequent papers and transferred to industry in the form of commercial CAN schedulability analysis tools. These tools have been used by a large number of major automotive manufacturers in the design of in-vehicle networks for a wide range of cars, millions of which have been manufactured during the last decade.

This paper shows that the original schedulability analysis given for CAN messages is flawed. It may provide guarantees for messages that will in fact miss their deadlines in the worst-case. This paper provides revised analysis resolving the problems with the original approach. Further, it highlights that the priority assignment policy, previously claimed to be optimal for CAN, is not in fact optimal and cites a method of obtaining an optimal priority ordering that is applicable to CAN. The paper discusses the possible impact on commercial CAN systems designed and developed using flawed schedulability analysis and makes recommendations for the revision of CAN schedulability analysis tools.

R. I. Davis (✉) · A. Burns
Real-Time Systems Research Group, Department of Computer Science, University of York,
YO10 5DD, York, UK
e-mail: rob.davis@cs.york.ac.uk

A. Burns
e-mail: alan.burns@cs.york.ac.uk

R. J. Bril · J. J. Lukkien
Technische Universiteit Eindhoven (TU/e), Den Dolech 2, 5600 AZ Eindhoven, The Netherlands
e-mail: r.j.bril@tue.nl

J. J. Lukkien
e-mail: j.j.lukkien@tue.nl

Keywords Controller Area Network (CAN) · Fixed priority scheduling · Non-pre-emptive scheduling · Schedulability analysis · Response time analysis · Priority assignment policies

1 Introduction

1.1 Background

Controller Area Network (CAN) is a serial communications bus designed to provide simple, efficient and robust communications for in-vehicle networks. CAN was developed by Robert Bosch GmbH, beginning in 1983, and presented to a wider audience at the Society of Automotive Engineers (SAE) Congress in 1986—effectively the "birth of CAN". In 1987, the first CAN controller chips were released by Intel (82526) and Philips (82C200). In the early 1990s, Bosch submitted the CAN specification (Bosch, 1991) for standardisation, leading to publication of the first ISO standard for CAN (11898) in 1993 (ISO, 1993).

Mercedes was the first automotive manufacturer to deploy CAN in a production car, the 1991 S-class. By the mid 1990s, the complexity of automotive electronics was increasing rapidly. The number of networked Electronic Control Units (ECUs) in Mercedes, BMW, Audi and VW cars went from 5 or less at the beginning of the 1990s to around 40 at the turn of the millennium. With this explosion in complexity traditional point-to-point wiring became increasingly expensive to manufacture, install, and maintain due to the hundreds of separate connections and tens of kilograms of copper wire required. As a result CAN was rapidly adopted by the cost-conscious automotive industry, providing an effective solution to the problems posed by increasing vehicle electronics content. Following on from Mercedes, other manufacturers including Volvo, Saab, BMW, Volkswagen, Ford, Renault, PSA, Fiat and others all adopted CAN technology.

As a result of the wholesale adoption of CAN by the automotive industry, sales of CAN nodes (8, 16 and 32-bit microcontrollers with on-chip CAN peripherals) grew from just under 50 million in 1999 to over 340 million in 2003[1]—see Fig. 1.

By 2004, there were at least 15 silicon vendors manufacturing, in total, over 50 different microprocessor families with on-chip CAN capability.

Today almost all of the cars manufactured in Europe are equipped with at least one CAN bus. In the United States, the Environmental Protection Agency has mandated the use of CAN, for On Board Diagnostics, in all cars and light trucks sold in the US from model year 2008 onwards.

1.2 Automotive applications

In automotive applications, CAN is typically used to provide high speed networks (500 Kbits/s) connecting chassis and power-train ECUs, for example engine management and transmission control. It is also used for low speed networks (100 or 125 Kbits/s)

[1] Figures from the CAN in Automation (CiA) website www.can-cia.org

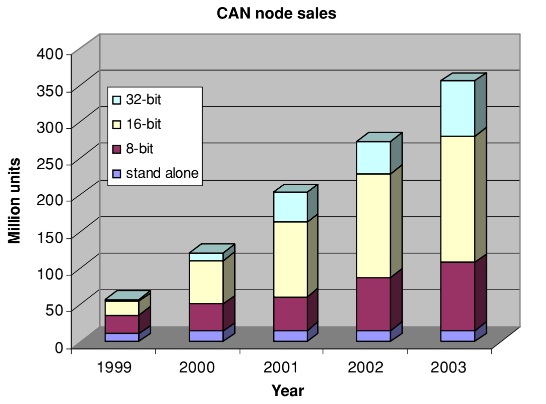


Fig. 1 Sales of microcontrollers with onchip CAN peripherals

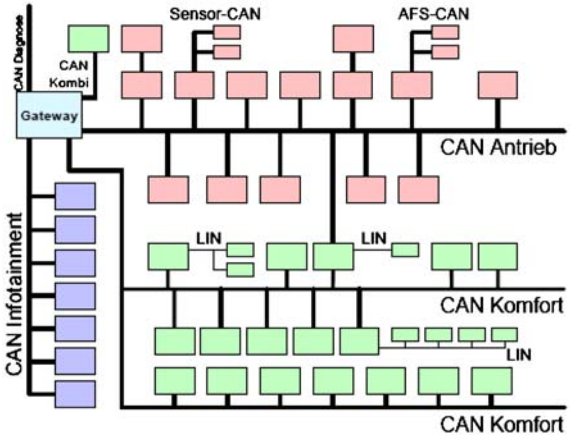


Fig. 2 VW Passat network architecture

connecting body and comfort electronics, for example door modules, seat modules and climate control. Data required by ECUs on different networks is typically gatewayed between the different CAN buses by a powerful ECU connected to both.

The network architecture of the VW Passat (Leohold, 2005) shown in Fig. 2, reproduced with permission from Nolte (2006), illustrates how a number of CAN buses are used to connect around 45 ECUs in that vehicle. Also shown in Fig. 2 are three Local Interconnect Networks (LIN). LIN is a complementary technology to CAN, and is used to provide inexpensive, low speed (20 Kbits/s) connectivity (LIN Consortium, 2003).

Table 1 summarises the requirements placed on in-vehicle networks for the BMW 7 Series. This is typical of automotive applications, where individual CAN buses are used to connect between 2 and 32 ECUs at bandwidths ranging from 100 to 500 Kbits/s (Frischkorn, 2005).

In automotive applications the *messages* sent on CAN are used to communicate state information, referred to as *signals*, between different ECUs. Examples of signals include: wheel speeds, oil and water temperature, engine rpm, gear selection, accelerator position, dashboard switch positions, climate control settings, window switch positions, fault codes, diagnostic information and so on. In a high-end vehicle, such as

Table 1 BMW 7 Series network requirements

	No. of ECUs	Bandwidth	No. of Messages	Cycle times
Body	14–30	100 Kbits/s	300	50 ms-2 s
Chassis	6–10	500 Kbits/s	180	10 ms-1 s
Power-train	3–6	500 Kbits/s	36	10 ms-10 s

the VW Phaeton, there can be more than 2500 distinct signals (Leohold, 2004), each effectively replacing what would, in a traditional point-to-point wiring loom, have been a separate wire.

Many of these signals have real-time constraints associated with them. For example, an ECU reads the position of a switch attached to the brake pedal. This ECU must send a message over the CAN network, carrying information (a signal) that the brakes have been applied. The ECU responsible for the rear light clusters receives the message, recognises the change in the value of the signal, and switches the brake lights on. All within a few tens of milliseconds of the brake pedal being pressed. Engine, transmission, and stability control systems typically place even tighter time constraints on signals, which may need to be sent as frequently as once every 5 milliseconds to meet their time constraints (Society of Automotive Engineers, 1993).

1.3 Research and real-time analysis

CAN is a serial data bus that supports priority based message arbitration and non-preemptive message transmission. In the early 1990s, a common misconception about CAN was that although the protocol was very good at transmitting the highest priority message with low latency, it was not possible to guarantee that less urgent signals, carried in lower priority messages, would meet their deadlines.

Tindell and Burns (1994) and Tindell et al. (1994b, 1995) showed how research into fixed priority pre-emptive scheduling for single processor systems could be adapted and applied to the scheduling of messages on CAN. This analysis provided a method of calculating the worst-case response times of all CAN messages. Using this analysis it became possible to engineer CAN based systems for timing correctness, providing guarantees that all messages, and the signals that they carry, would meet their deadlines.

Tindell's seminal research heavily influenced the design of on-chip CAN peripherals such as Motorola msCAN (Motorola, 1998) and has lead to a large body of work into schedulability theory and error models for CAN (Punnekkat et al., 2000; Nolte et al., 2002, 2003; Broster et al., 2002, 2005; Hansson et al., 2002; Broster and Burns, 2003), including at least two PhD theses (Broster, 2003; Nolte, 2006). Overall, this research into CAN scheduling has been cited in over 200[2] subsequent papers.

In 1995, Tindell's research was recognised by Volvo Car Corporation and successfully used in the configuration and analysis of the CAN buses for the forthcoming

[2] As of August 2006, reference (Tindell and Burns, 1994) has 78 citations, reference (Tindell et al., 1995) 199 citations and reference (Tindell et al., 1994b) 110 citations (Google Scholar).

Table 2 CAN messages highlighting flawed analysis

Message	Priority	Period	Deadline	TX time
A	1	2.5 ms	2.5 ms	1 ms
B	2	3.5 ms	3.25 ms	1 ms
C	3	3.5 ms	3.25 ms	1 ms

Volvo S80 (P23) (Casparsson et al., 1998). Following the success of this project, Volcano Communications Technologies AB[3] used Tindell's analysis as the basis of a commercial CAN schedulability analysis tool, called Volcano Network Architect.

Since 1998, these tools and the Volcano concept (Casparsson et al., 1998) have been used in the design and development of CAN networks and electronics systems for the Volvo XC90, S80, S/V/XC70, S60, V50 and S40 as well as many other cars from different manufacturers.

Prior to Tindell's work, low levels of bus utilization, up to 30 or 40%, were typical in automotive applications, with extensive testing required to obtain confidence that CAN messages would meet their deadlines. With the advent of a systematic approach based on schedulability analysis, CAN bus utilization could be increased to around 80% (DeMeis, 2005) whilst still guaranteeing that deadlines would be met.

1.4 Motivation

The design and development of many in vehicle Controller Area Networks relies on the schedulability analysis of CAN given by Tindell and Burns (1994) and Tindell et al. (1994b, 1995). In this section we show that this analysis is flawed. It may result in computed worst-case response times for messages that are optimistic, i.e. less than the response times that may actually occur. The set of CAN messages listed in Table 2 serve to highlight the problem with the existing schedulability analysis of CAN. As a simple example, we have assumed a 125 Kbit/s network with 3 messages, each of which carries 7 bytes of signal data. Assuming 11-bit identifiers and worst-case bit-stuffing, the maximum length of each message is 125 bits. The maximum transmission time of each message is therefore 1 ms.

The analysis method given by Tindell and Burns (1994) and Tindell et al. (1994b, 1995) calculates the worst-case response times of messages A, B and C as 2 ms, 3 ms and 3 ms respectively. Hence the system is deemed to be schedulable—the analysis supposedly guarantees that all of the messages will meet their deadlines in the worst case, despite the high bus utilisation of 97%.

Figure 3 illustrates the worst-case scenario for transmission of message C. We note that the first instance of this message is delayed by higher priority messages A and

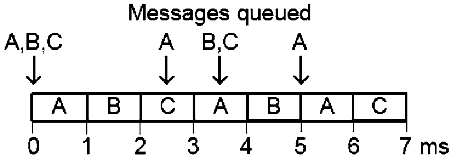


Fig. 3 Worst-case scenario for message C

[3] Volcano Communications Technologies AB was acquired by Mentor Graphics in May 2005.

B, leading to a response time of 3 ms—this is the "worst-case response time" calculated using existing CAN schedulability analysis methods. However, as message transmission is non-pre-emptive, the first transmission of message C has a knock on effect, delaying subsequent transmissions of higher priority messages A and B. Some of this higher priority interference is *pushed through* into the next period of message C leading to a longer response time for the second instance of message C.

At time $t = 7$ ms, the second instance of message C completes transmission with a response time of 3.5 ms. (Note at time $t = 7$ ms, there are no higher priority messages awaiting transmission and so there is no further push through interference that could delay subsequent instances of message C).

The actual worst-case response time for message C is 3.5 ms, which is greater than its deadline of 3.25 ms. The system is therefore unschedulable; contrary to the guarantees given by Tindell and Burns (1994) and Tindell et al. (1994b, 1995).

In fact, if the periods of messages B and C are shortened from 3.5 ms to 3.25 ms then the existing analysis results in unchanged worst-case response times, implying that the messages will still meet their deadlines. However, with these shorter periods the overall bus utilisation exceeds 100% and so the system cannot possibly be schedulable!

1.5 Related work

The schedulability analysis for CAN builds on previous research into fixed priority scheduling of tasks on single processor systems.

Lehoczky (1990) introduced the concept of a busy period and showed that if tasks have deadlines greater than their periods, referred to as *arbitrary deadlines*, then it is necessary to examine the response times of all invocations of a task falling within a busy period in order to determine the worst-case response time. Harbour et al. (1991) showed that if deadlines are less than or equal to periods, but priorities vary during execution, then again multiple invocations must be inspected to determine the worst-case response time. We note that non-pre-emptive scheduling is effectively a special case of pre-emptive scheduling with varying execution priority—as soon as a task starts to execute its priority is raised to the highest level. Tindell et al. (1994a) improved upon the work of Lehoczky (1990) providing a formulation for arbitrary deadline analysis based on a recurrence relation.

Building upon these earlier results, George et al. (1996) provided comprehensive schedulability analysis of non-pre-emptive fixed priority scheduling of single processor systems.

Bril (2006) refuted the analysis of fixed priority systems with deferred pre-emption given by Burns (1994), showing that this analysis may result in computed worst-case response times that are optimistic. The schedulability analysis for CAN given by Tindell and Burns (1994) and Tindell et al. (1994b, 1995) builds upon Burns (1994) and suffers from essentially the same flaw. A similar issue with work on pre-emption thresholds (Wang and Saksena, 1999) was first identified and corrected by Regehr (2002).

The revised schedulability analysis presented in this paper aims to provide an evolutionary improvement upon the analysis of CAN given by Tindell and Burns (1994) and Tindell et al. (1994b, 1995). To do so, it draws upon the analysis of Tindell et al. (1994a) for fixed priority pre-emptive scheduling of systems with arbitrary deadlines and the analysis of George et al. (1996) for fixed priority non-pre-emptive systems.

A technical report (Bril et al., 2006a) and a workshop paper (Bril et al., 2006b) highlighted the problem for CAN but did not provide a specific in-depth solution. That is the purpose of this paper. A further technical report (Bril et al., 2006c) complements this paper, providing formal proofs of the worst-case response time of tasks under fixed priority scheduling with deferred pre-emption.

1.6 Organisation

The remainder of this paper is organised as follows: Section 2 describes the CAN protocol and terminology before outlining a suitable scheduling model and notation on which to base revised schedulability analysis. Section 3 provides new schedulability analysis for CAN, correcting the flaws in the existing approach. Section 4 discusses the system and message parameters needed for the flaws in the existing analysis to result in incorrect worst-case response times and hence misleading guarantees. Section 5 discusses the issue of optimal priority assignment for CAN messages. Section 6 summarises the implications of flaws in the existing analysis for commercial CAN applications. Finally, Section 7 concludes with a summary of the main contributions of this paper and recommendations for further research.

2 Controller area network

This section describes elements of the CAN protocol and characteristics of a system model that are needed to formulate a schedulability test. For a complete description of the CAN protocol, see the CAN specification version 2.0 (Bosch, 1991)

2.1 CAN protocol and terminology

CAN is an asynchronous multi-master serial data bus that uses Carrier Sense Multiple Access/Collision Resolution (CSMA/CR) to determine access.

CAN was designed as a simple and robust broadcast bus capable of operating at speeds of up to 1 Mbit/s. Message transfer over CAN is controlled by 4 different types of *frame*: Data frames, Remote Transmit Request (RTR) frames, Overload frames and Error frames.

The layout of a standard format data frame is shown in Fig. 4. Each CAN data frame is required to have a unique identifier. Identifiers may be 11-bit (standard format) or

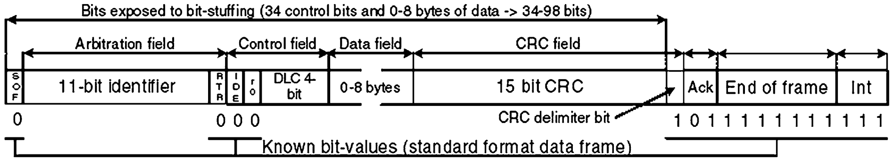


Fig. 4 Standard format data frame

29-bit (extended format). The identifier serves two purposes beyond simply identifying the message. First, the identifier is used as a priority to determine which message, among those contending for the bus, will be transmitted next. Second, the identifier may be used by receivers to filter out messages they are not interested in, and so reduce the load on the receiver's host microprocessor.

In this paper we are interested in the schedulability of data frames, with error frames also considered in Section 3.5. The schedulability analysis can however easily be extended to include RTR frames using the approach given by Tindell et al. (1995).

2.1.1 Priority based arbitration

The CAN physical layer supports two states termed *dominant* ('0') and *recessive* ('1'). If two or more CAN controllers are transmitting at the same time and at least one of them transmits a '0' then the value on the bus will be a '0'. This mechanism is used to control access to the bus and also to signal errors.

The CAN protocol calls for nodes to wait until a *bus idle period*[4] is detected before attempting to transmit. If two or more nodes start to transmit at the same time, then by monitoring each bit on the bus, each node can determine if it is transmitting the highest priority message (with a numerically lower identifier) and should continue or if it should stop transmitting and wait for the next bus idle period before trying again. As the message identifiers are unique, a node transmitting the last bit of the identifier field, without detecting a '0' bit that it did not transmit, must be transmitting the highest priority message that was ready for transmission at the start of arbitration. This node then continues to transmit the remainder of its message, all other nodes having backed off.

The requirement for a node to be able to overwrite a recessive bit, and the transmitting node detect this change, limits the combination of physical length and speed of a CAN bus. The duration of each bit must be sufficient for the signal to propagate the length of the network. This limits the maximum data rate to 1 Mbit/s for a network up to 40 m in length or to 125 Kbit/s for a 500 m long network.

The arbitration mechanism employed by CAN means that messages are sent as if all the nodes on the network shared a single global priority based queue. In effect messages are sent on the bus according to fixed priority non-pre-emptive scheduling.

The above high level description is a somewhat simplified view of the timing behaviour of CAN. CAN does not have a global concept of time, rather each CAN controller typically has its own clock which, within a tolerance specified by the protocol, may drift with respect to the clocks of other nodes. The CAN protocol therefore requires that nodes re-synchronise on each message transmission. Specifically, every node must synchronise to the leading edge of the *start of frame* bit caused by whichever node starts to transmit first.

Normally, CAN nodes are only allowed to start transmitting when the bus is idle. Thus, when the bus is idle beyond the 3-bit *inter-frame space* and a node starts to transmit a message beginning with the dominant start of frame bit ('0'), then all the other nodes synchronise on the leading edge of this bit and become *receivers*—i.e.

[4] A *bus idle period* is an interval of arbitrary length comprising only recessive bits and beginning with the last bit of the *inter-frame space*—the final 3-bit field shown in Fig. 4.

they are not permitted to transmit until the bus next becomes idle. In this case any message that becomes ready for transmission after the leading edge of the start of frame bit has to wait for the next bus idle period before it can enter into arbitration.

However, to avoid problems due to clock drift, the CAN protocol also specifies that, if a CAN node has a message ready for transmission and detects a dominant bit at the 3rd bit of the inter-frame space, it will interpret this as a start of frame bit, and with the next bit, start transmitting its own message with the first bit of the identifier without first transmitting a start of frame bit and without becoming a receiver.[5] Again the leading edge of the start of frame bit causes a synchronisation. This behaviour ensures that any messages that become ready for transmission, whilst another message is being sent on the bus, are entered into the next round of arbitration, irrespective of any within tolerance clock drift.

2.1.2 Error detection

CAN was designed as a robust and reliable form of communication for short messages. Each data frame carries between 0 and 8 bytes of payload data and has a 15-bit Cyclic Redundancy Check (CRC). The CRC is used by receiving nodes to check for errors in the transmitted message. If a node detects an error in the transmitted message, which may be a bit-stuffing error (see Section 2.1.3), a CRC error, a form error in the fixed part of the message, or an acknowledgement error, then it transmits an error flag. The error flag consists of 6 bits of the same polarity: ‘000000’ if the node is in the error active state and ‘111111’ if it is error passive. Transmission of an error flag typically causes other nodes to also detect an error, leading to transmission of further error flags.

Figure 5 illustrates CAN error frames, for further details see (Bosch, 1991; Punnekkat et al., 2000). The length of an error frame is between 17 and 31 bits. Hence each message transmission that is signalled as an error can lead to a maximum of 31 additional bits[6] of error recovery overhead plus re-transmission of the message itself.

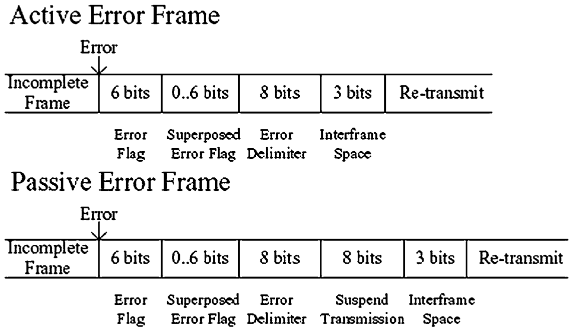
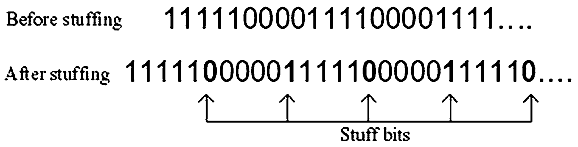


Fig. 5 CAN error frames

[5] See page 54 of the CAN Specification version 2.0 (Bosch, 1991).

[6] The analysis given by Tindell and Burns (1994) and Tindell et al. (1994b, 1995) uses 29 bits as the error recovery overhead as specified on page 8 of part A of the CAN specification 2.0 (Bosch, 1991) for standard identifiers only. We use 31 bits as specified on page 40 of the CAN specification 2.0 Part B (Bosch, 1991) for both standard and extended identifiers.

Fig. 6 Worst-case bit stuffing



2.1.3 Bit stuffing

As the bit patterns ‘000000’ and ‘111111’ are used to signal errors, it is essential that these bit patterns are avoided in the variable part of a transmitted message—see Fig. 4. The CAN protocol therefore requires that a bit of the opposite polarity is inserted by the transmitter whenever 5 bits of the same polarity are transmitted. This process is referred to as *bit-stuffing*, and is reversed by the receiver.

The worst-case scenario for bit-stuffing is shown in Fig. 6. Note that each stuff bit begins a sequence of 5 bits that is itself subject to bit stuffing.

Stuff bits increase the maximum transmission time of CAN messages. Including stuff bits and the inter-frame space, the maximum transmission time C_m, of a CAN message m containing s_m data bytes is given by:[7]

$$C_m = \left(g + 8s_m + 13 + \left\lfloor \frac{g + 8s_m - 1}{4} \right\rfloor \right) \tau_{\text{bit}} \tag{1}$$

where g is 34 for standard format (11-bit identifiers) or 54 for extended format (29-bit identifiers), $\lfloor a/b \rfloor$is notation for the *floor* function, which returns the largest integer less than or equal to a/b, and τ_{bit} is the transmission time for a single bit.

The formula given in Eq. (1) simplifies to:

$$C_m = (55 + 10s_m)\tau_{\text{bit}} \tag{2}$$

for 11-bit identifiers and

$$C_m = (80 + 10s_m)\tau_{\text{bit}} \tag{3}$$

for 29-bit identifiers.

2.2 Scheduling model

In this section we describe an appropriate system model and notation that can be used to analyse worst-case response times of messages on CAN and hence determine system schedulability.

The system is assumed to comprise a number of nodes (microprocessors) connected via CAN. Each node is assumed to be capable of ensuring that, at any given time when arbitration starts, the highest priority message queued at that node is entered into arbitration.

[7] This formula corrects a similar one by Tindell and Burns (1994) and Tindell et al. (1994b, 1995) which does not account for the fact that stuff bits are themselves also subject to bit stuffing.

The system is assumed to contain a static set of hard real-time messages, each statically assigned to a node on the network. Each message m has a fixed identifier and hence a unique priority. As priority uniquely identifies each message, in the remainder of this paper we will overload m to mean either message m or priority m as appropriate. Each message has a maximum number of data bytes s_m, and a maximum transmission time C_m, given by Eq. (1).

Each message is assumed to be queued by a software task, process or interrupt handler executing on the host microprocessor. This task is either invoked by, or polls for, the event and takes a bounded amount of time, between 0 and J_m, to queue the message ready for transmission. J_m is referred to as the *queuing jitter* of the message and is inherited from the overall response time of the task, including any polling delay.

The event that triggers queuing of the message is assumed to occur with a minimum inter-arrival time of T_m, referred to as the message *period*. This model supports events that occur strictly periodically with a period of T_m, events that occur sporadically with a minimum separation of T_m, and events that occur only once before the system is reset, in which case T_m is infinite.

Each message has a hard deadline D_m, corresponding to the maximum permitted time from occurrence of the initiating event to the end of successful transmission of the message, at which time the message data is assumed to be available on the receiving nodes that require it. Tasks on the receiving nodes may place different timing requirements on the data, however in such cases we assume that D_m is the tightest such time constraint.

The *worst-case response time* R_m, of a message is defined as the longest time from the initiating event occurring to the message being received by the nodes that require it.

A message is said to be *schedulable* if and only if its worst-case response time is less than or equal to its deadline ($R_m \leq D_m$). The system is schedulable if and only if all of the messages in the system are schedulable.

2.3 Practical implications of the model

Engineers wanting to use the analysis given in Section 3 to analyse CAN based systems must be careful to ensure that all of the assumptions of the above model hold for their system.

In particular, it is important that each CAN controller and device driver is capable of ensuring that, at any given time when arbitration starts, the highest priority message queued at that node is entered into arbitration. This behaviour is essential if message transmission is to take place as if there were a single global priority queue and for the analysis given in Section 3 to be applicable. As noted by Tindell and Burns (1994), the Philips 82C500 CAN controller cannot in general support this behaviour. Also the Intel 82527 CAN controller has a feature where messages are entered into arbitration in slot order rather than identifier order. In this case it is important that messages are allocated to slots in identifier order to preserve the correct priority based behaviour.

Many on-chip CAN controllers have multiple slots that can be allocated to either transmit or receive a specific message. For example some Motorola, National Semiconductor, Fujitsu and Hitachi on-chip CAN peripherals have 14, 15 or 16 such slots.

These slots typically have only a single buffer and therefore it is necessary to ensure that the previous instance of a message has been transmitted before any new data is written into the buffer, otherwise the previous message will be overwritten and lost. This behaviour provides an additional constraint on message transmission: the deadline of each message must be less than or equal to its period ($D_m \leq T_m$).

Recall that the worst-case response time of a message is from the occurrence of the initiating event to the end of successful message reception at the receiving nodes. As noted by Broster (2003), receiving nodes can access the message following the end of frame marker and before the 3-bit inter-frame space—see Fig. 4. The analysis given in the remainder of this paper is slightly pessimistic in that it includes the 3-bit inter-frame space in the computed worst-case response times. To remove this small degree of pessimism, it is valid to simply subtract $3\tau_{\text{bit}}$ from the computed response time values.

The time constraint of interest to engineers is the overall end-to-end response time from an initiating event occurring, such as the brake pedal switch closing, to the corresponding output response, for example the brake lights illuminating. The worst-case response time of a message represents only part of this overall end-to-end response time. There is an additional delay to consider, corresponding to the worst-case time between the message becoming available at the receiving node and the output being made. Processing of the signal information contained in a message is typically done either by a task that polls for the message or by an interrupt handler that is triggered by message reception. The worst-case response time of the receiving task or interrupt handler, including any polling delay, needs to be added to the worst-case response time of the message to determine the overall end-to-end response time.

The scheduling model assumed in this paper uses only one time domain, whilst CAN typically has a separate clock source for each node on the network. To ensure that the schedulability analysis for a real network does not produce optimistic results, it is necessary to take clock tolerances into account. This can be achieved by converting to real-time as follows: for message jitters and bit times on the bus the conversion to real-time should assume that the node clocks run as slowly as their tolerance allows. Similarly, message periods and deadlines derived from node clocks should be converted to real-time assuming that the node clocks run as quickly as their tolerance allows.

3 Response time analysis

Response time analysis for CAN aims to provide a method of calculating the worst-case response time of each message. These values can then be compared to the message deadlines to determine if the system is schedulable. Initially we provide analysis assuming no errors on the CAN bus. This analysis is then extended, in Section 3.5, to account for errors on the bus.

For systems complying with the scheduling model given in Section 2.2, CAN effectively implements fixed priority non-pre-emptive scheduling of messages. Following the analysis given by Tindell and Burns (1994) and Tindell et al. (1994b, 1995) the worst-case response time of a message can be viewed as being made up of three elements:

(i) The queuing jitter J_m, corresponding to the longest time between the initiating event and the message being queued, ready to be transmitted on the bus.
(ii) The queuing delay w_m, corresponding to the longest time that the message can remain in the CAN controller slot or device driver queue, before commencing successful transmission on the bus.
(iii) The transmission time C_m, corresponding to the longest time that the message can take to be transmitted.

The worst-case response time of message m is given by:

$$R_m = J_m + w_m + C_m \tag{4}$$

The queuing delay w_m, comprises two elements:

(i) *blocking* B_m, due to lower priority messages which may be in the process of being transmitted when message m is queued, and
(ii) *interference* due to higher priority messages which may win arbitration and be transmitted in preference to message m.

Given the behaviour of CAN described in the final two paragraphs of Section 2.1.1, the maximum amount of blocking occurs when a lower priority message starts transmission immediately before message m is queued and hence ready to be transmitted on the bus. Message m must wait until the bus is idle before it can be entered into arbitration. The maximum blocking time B_m, is given by:

$$B_m = \max_{k \in lp(m)} (C_k) \tag{5}$$

where *lp(m)* is the set of messages with lower priority than m.

The concept of a *busy period*, introduced by Lehoczky (1990), is fundamental in analysing worst-case response times. Modifying the definition of a busy period given by Harbour et al. (1991) to apply to CAN messages, a priority level-m busy period is defined as follows:

(i) It starts at some time t^s when a message of priority m or higher is queued ready for transmission and there are no messages of priority m or higher waiting to be transmitted that were queued strictly before time t^s.
(ii) It is a contiguous interval of time during which any message of priority lower than m is unable to start transmission and win arbitration.
(iii) It ends at the earliest time $t^e$when the bus becomes idle, ready for the next round of transmission and arbitration, yet there are no messages of priority m or higher waiting to be transmitted that were queued strictly before time t^e.

The key characteristic of a busy period is that all messages of priority m or higher, queued strictly before the end of the busy period, are transmitted during the busy period. These messages cannot therefore cause any interference on a subsequent instance of message m queued at or after the end of the busy period.

In mathematical terminology, busy periods can be viewed as right half-open intervals: $[t^s, t^e)$ where t^s is the start of the busy period and t^e the end of the busy period. Thus the end of one busy period may correspond to the start of another separate busy period. This is in contrast to the simpler definition given by Lehoczky (1990), which unifies two adjacent busy periods, as we have defined them, and therefore sometimes results in analysis of more message instances than is strictly necessary.

The worst-case queuing delay for message m occurs for some instance of message m queued within a priority level-m busy period that starts immediately after the longest lower priority message begins transmission. This *maximal* busy period begins with a so-called *critical instant* (Liu and Layland, 1973) where message m is queued simultaneously with all higher priority messages and then each of these higher priority messages is subsequently queued again after the shortest possible time interval. In the remainder of this paper whenever we refer to a busy period we mean this maximum length busy period.

If more than one instance of message m is transmitted during a priority level-m busy period, then it is necessary to determine the response time of each instance, in order to find the overall worst-case response time of the message.

3.1 Basic analysis and stopping condition

Tindell and Burns (1994) and Tindell et al. (1994b, 1995) give the following equation for the worst-case queuing delay:

$$w_m = B_m + \sum_{\forall k \in hp(m)} \left\lceil \frac{w_m + J_k + \tau_{\text{bit}}}{T_k} \right\rceil C_k \tag{6}$$

where *hp(m)* is the set of messages with priorities higher than m, and $\lceil a/b \rceil$ is notation for the *ceiling* function which returns the smallest integer greater than or equal to *a*/*b*.

Although w_m appears on both sides of Eq. (6), as the right hand side is a monotonic non-decreasing function of w_m, the equation may be solved using the recurrence relation given below.

$$w_m^{n+1} = B_m + \sum_{\forall k \in hp(m)} \left\lceil \frac{w_m^n + J_k + \tau_{\text{bit}}}{T_k} \right\rceil C_k \tag{7}$$

A suitable starting value is $w_m^0 = B_m$. The recurrence relation iterates until, either $J_m + w_m^{n+1} + C_m > D_m$ in which case the message is not schedulable, or $w_m^{n+1} = w_m^n$ in which case the worst-case response time of the *first instance of the message in the busy period* is given by: $J_m + w_m^{n+1} + C_m$.

The flaw in the above analysis is that, given the constraint $D_m \leq T_m$, it implicitly assumes that if message m is schedulable, then the priority level-m busy period will end at or before T_m. We observe that with fixed priority pre-emptive scheduling this would always be the case, as on completion of transmission of message m no higher priority message could be awaiting transmission. However, with fixed priority non-pre-emptive scheduling, a higher priority message can be awaiting transmission when message m completes transmission, and thus the busy period can extend beyond T_m, as shown by the example in Section 1.4.

The length t_m, of the priority level-m busy period is given by the following recurrence relation, starting with an initial value of $t_m^0 = C_m$, and finishing when $t_m^{n+1} = t_m^n$:

$$t_m^{n+1} = B_m + \sum_{\forall k \in hep(m)} \left\lceil \frac{t_m^n + J_k}{T_k} \right\rceil C_k \tag{8}$$

where *hep* is the set of messages with priority higher than or equal to m. As the right hand side is a monotonic non-decreasing function of t_m, then the recurrence relation is guaranteed to converge provided that the bus utilisation U_m, for messages of priority m and higher, is less than 1:

$$U_m = \sum_{\forall k \in hep(m)} \frac{C_k}{T_k} \tag{9}$$

If $t_m \le T_m - J_m$, then the busy period ends at or before the second instance of message m is queued. This means that only the first instance of the message is transmitted during the busy period. The existing analysis calculates the worst-case queuing time for this instance, via Eq. (7), and hence provides the correct worst-case response time in this case.

If $t_m > T_m - J_m$, then the existing analysis may give an optimistic worst-case response time, depending on whether the first, or a subsequent instance of message m has the longest response time.

We observe that the analysis presented in appendix A.2 of George et al. (1996) suggests that t_m is the smallest value that is a solution to Eq. (8), however this is not strictly correct. For the lowest priority message $B_m = 0$, and so $t_m = 0$ is trivially the smallest solution when all of the messages have zero jitter. We avoid this problem by using an initial value of $t_m^0 = C_m$.

3.2 Checking multiple instances

The number of instances Q_m, of message m that become ready for transmission before the end of the busy period is given by:

$$Q_m = \left\lceil \frac{t_m + J_m}{T_m} \right\rceil \tag{10}$$

To determine the worst-case response time of message m, it is necessary to calculate the response time of each of the Q_m instances. The maximum of these values then gives the worst-case response time.

In the following analysis, we use the index variable q to represent an instance of message m. The first instance in the busy period corresponds to $q = 0$, and the final instance to $q = Q_m - 1$. The longest time from the start of the busy period to instance q beginning successful transmission is given by:

$$w_m^{n+1}(q) = B_m + qC_m + \sum_{\forall k \in hp(m)} \left\lceil \frac{w_m^n + J_k + \tau_{\text{bit}}}{T_k} \right\rceil C_k \tag{11}$$

Table 3 CAN messages

Message	Priority	Period	Deadline	TX time
A	1	2.5 ms	2.5 ms	1 ms
B	2	3.5 ms	3.25 ms	1 ms
C	3	3.5 ms	3.25 ms	1 ms

The recurrence relation starts with a value of $w_m^0(q) = B_m + qC_m$, and ends when $w_m^{n+1}(q) = w_m^n(q)$, or when $J_m + w_m^{n+1}(q) - qT_m + C_m > D_m$ in which case the message is unschedulable. For values of $q > 0$ an efficient starting value is given by $w_m^0(q) = w_m(q-1) + C_m$.

The event initiating instance q of the message occurs at time $qT_m - J_m$ relative to the start of the busy period, so the response time of instance q is given by:

$$R_m(q) = J_m + w_m(q) - qT_m + C_m \tag{12}$$

The worst-case response time of message m is therefore:

$$R_m = \max_{q=0..Q_m-1}(R_m(q)) \tag{13}$$

We note that the analysis presented above is also applicable when messages have deadlines that are greater than their periods, so called arbitrary deadlines. However, if such timing characteristics are specified then the software device drivers or CAN controller hardware may need to be capable of buffering more than one instance of a message. N_m, the number of instances of each message that need to be buffered is bounded by:

$$N_m = \left\lceil \frac{R_m}{T_m} \right\rceil \tag{14}$$

We observe that the analysis presented by George et al. (1996) effectively uses $Q_m = \lfloor t_m/T_m \rfloor + 1$ rather than $Q_m = \lceil t_m/T_m \rceil$. This yields a value which is one too large when the length of the busy period plus jitter is an integer multiple of the message period. Although this does not give rise to problems, we prefer the more efficient formulation given by Eq. (10).

3.3 Example

In Section 1.4 we showed, with the aid of a simple example, how the existing analysis can provide optimistic worst-case response times and hence flawed guarantees that messages will meet their deadlines. We return to this example to illustrate how the analysis presented in this paper computes the correct worst-case response times. For ease of reference, the table of message parameters is repeated below.

Using the new analysis, the worst-case response time of message C ($m = 3$) is calculated as follows: As there are no lower priority messages, $B_3 = 0$. Starting with a value of $t_3^0 = C_3 = 1$, the recurrence relation given by Eq. (8) iterates as follows: $t_3^1 = 3$, $t_3^2 = 4$, $t_3^3 = 6$, $t_3^4 = 7$, converging as $t_3^5 = t_3^4 = 7$. The length of the busy period is therefore 7.0 ms, and the number of instances of message C that need to be examined is given by Eq. (10):

$$Q_3 = \left\lceil \frac{7.0}{3.5} \right\rceil = 2$$

This tells us that there is the possibility that the existing analysis will calculate an optimistic worst-case response time. The value could, however, still be correct if the first instance of the message has the longest response time.

Calculation of the response time of the first instance proceeds using Eq. (11): $w_3^0(0) = 0$, $w_3^1(0) = 2$, converging as $w_3^2(0) = w_3^1(0) = 2$. Using Eq. (12), we have $R_3(0) = 3$, the same response time calculated by the existing analysis.

Moving on to the second instance, $w_3^0(1) = w_3(0) + C_m = 3$, $w_3^1(1) = 4$, $w_3^2(1) = 5$, $w_3^3(1) = 6$. At this point computation would normally stop as the response time, given by $J_3 + w_3(q) - qT_3 + C_3$, has reached 3.5 ms which is greater than the message deadline. However, if we continue iterating, assuming a longer deadline, then the recurrence relation converges on $w_3^4(1) = w_3^3(1) = 6$ and hence $R_3(1) = 3.5$ ms. The worst-case response time of message C is in fact 3.5 ms, as previously illustrated by Fig. 3 in Section 1.4.

3.4 Sufficient schedulability tests

The analysis given in Sections 3.1 and 3.2 corrects a significant flaw in the existing schedulability analysis for CAN. However, the schedulability test presented is more complex, potentially requiring the computation of multiple response times.

In this section we present two simpler but more pessimistic schedulability tests. These tests are "sufficient but not necessary". By "sufficient" we mean that all systems deemed to be schedulable by the tests are in fact schedulable, and by "not necessary" we mean that not all systems deemed to be unschedulable by the tests are in fact unschedulable.

The schedulability tests given in this section are only applicable given the constraint that message deadlines do not exceed their periods.

The response time of the first instance of a message in the busy period is given by Eq. (7). Assuming that this first instance completes transmission before its deadline and hence before the end of its period, then we have two possibilities to consider.

(i) If the busy period ends before the next instance of message m is queued, then Eq. (7) gives the correct worst-case response time.

(ii) Alternatively, if the busy period continues beyond the time at which the next instance of message m is queued, then we must also consider the response time of the second and any subsequent instances of message m, queued before the end of the busy period.

First, we derive an upper bound on the maximum length of the interval $[s_{m,q}, s_{m,q+1})$, between the times, $s_{m,q}$ and $s_{m,q+1}$, at which two arbitrary but consecutive instances,

q and $q+1$, of message m start transmission. We then show that this upper bound is also an upper bound on the queuing delay for instance $q+1$, and can therefore be used as the basis for a sufficient schedulability test.

We assume that:

(i) all $q+1$ instances fall within the same busy period,
(ii) the first q instances are schedulable—we will return to this point later.

We observe that at time $s_{m,q}$, when instance q starts transmission, there can be no other messages currently being transmitted, and no messages of higher priority than m awaiting transmission. Thus, an upper bound on the length of the time interval $[s_{m,q}, s_{m,q+1})$ can be found by making the potentially pessimistic assumption that all higher priority messages are queued just as instance q starts transmission. The smallest solution to Eq. (15) provides an upper bound on the length of this interval.

$$w_m^{n+1} = C_m + \sum_{\forall k \in hp(m)} \left\lceil \frac{w_m^n + J_k + \tau_{\text{bit}}}{T_k} \right\rceil C_k \tag{15}$$

Here, C_m is the transmission time of the qth instance of message m, which is transmitted first in the interval. The summation term represents the interference from higher priority messages, released during the interval, and sent before the $(q+1)$th instance of message m can start to be transmitted.

Given the assumption that the first q instances in the busy period are schedulable, and the constraint that $D_m \le T_m$, then the start (and end) of transmission of the qth instance must happen before the end of its period, and hence before the $(q+1)$th instance is queued. This means that the queuing delay for the $(q+1)$th instance, as measured from the time at which it is queued to the start of its transmission, is less than the length of the interval $[s_{m,q}, s_{m,q+1})$. The queuing delay for the $(q+1)$th instance is therefore also bounded[8] by the solution to Eq. (15).

We now return to the assumption that the first q instances are schedulable. Schedulability of the $q=0$ instance can be determined using Eq. (7); whilst the schedulability of the second and all subsequent instances within the busy period can be determined, by induction, using Eq. (15).

To determine an upper bound on the queuing delay, a suitable starting value, for use in using Eq. (15), is $w_m^0 = C_m$. The recurrence relation iterates until, either $J_m + w_m^{n+1} + C_m > D_m$ in which case message m is deemed unschedulable, or $w_m^{n+1} = w_m^n$ in which case the second and subsequent instances of message m are schedulable, with an upper bound on their response times of $J_m + w_m^{n+1} + C_m$.

Intuitively, we might say that the second and subsequence instances of message m in the busy period are subject to blocking, of at most C_m, due to the previous instance of the same message.

[8] We observe that the queuing delay of the $(q+1)$th instance is in fact at least C_m less than the bound given by Eq. (15). This is because, to be schedulable, instance q must start transmission at least C_m before the end of its period. As accurate analysis is available, presented in Section 3.2, we do not pursue this potential improvement in the sufficient analysis further.

This result suggests a simple sufficient but not necessary schedulability test, formed by combining Eqs. (7) and (15) into a single equation—Eq. (16).

$$w_m^{n+1} = \max(B_m, C_m) + \sum_{\forall k \in hp(m)} \left\lceil \frac{w_m^n + J_k + \tau_{\text{bit}}}{T_k} \right\rceil C_k \tag{16}$$

We observe that the schedulability analysis embodied in Eq. (16) equates to assuming that an instance of message m can be subject to blocking; either of B_m, due to non-pre-emptive transmission of lower priority messages; or of C_m, due to the non-pre-emptive transmission of the previous instance of message m itself.

A further simplification is to assume that the blocking factor always takes its maximum possible value. This leads to a further sufficient but not necessary schedulability test:

$$w_m^{n+1} = B^{\text{MAX}} + \sum_{\forall k \in hp(m)} \left\lceil \frac{w_m^n + J_k + \tau_{\text{bit}}}{T_k} \right\rceil C_k \tag{17}$$

where B^{MAX} corresponds to the transmission time of the longest possible CAN message (8 data bytes), irrespective of the characteristics and priorities of the messages in the system.[9]

3.5 Error model

So far we have assumed that no errors occur on the CAN bus; however, as originally shown by Tindell and Burns (1994) and Tindell et al. (1994b, 1995), schedulability analysis of CAN may be extended to include an appropriate error model.

In this paper we consider only a very simple and general error model. We assume that the maximum number of errors present on the bus in some time interval t is given by the function $F(t)$. We assume no specific details about this function; save that it is a monotonic non-decreasing function of t. For a more detailed discussion of appropriate error models for CAN, see Punnekkat et al. (2000) and Broster et al. (2002, 2005).

We now modify the schedulability equations to account for the error recovery overhead. The worst-case impact of a single bit error is to cause transmission of an additional 31 bits of error recovery overhead plus re-transmission of the affected message. Only errors affecting message m or higher priority messages can delay message m from being successfully transmitted. The maximum additional delay caused by the error recovery mechanism is therefore given by:

$$E_m(t) = \left(31\tau_{\text{bit}} + \max_{k \in hep(m)}(C_k)\right) F(t) \tag{18}$$

[9] Tindell et al. (1995) state that the *blocking time on CAN is defined as the longest time that a message can take to be physically transmitted on "the bus"*. This simplified view provides a sufficient but not necessary schedulability test that corresponds to Eq. (17). However, later in Tindell et al. (1995), the blocking term is described as "*the longest time that any lower priority message can occupy the bus*". This description, also in Tindell and Burns (1994) and Tindell et al. (1994b), results in a flawed schedulability test.

Revising Eq. (8) to compute the length of the busy period we have:

$$t_m^{n+1} = E_m\left(t_m^n\right) + B_m + \sum_{\forall k \in hep(m)} \left\lceil \frac{t_m^n + J_k}{T_k} \right\rceil C_k \tag{19}$$

Again an appropriate initial value is $t_m^0 = C_m$. Eq. (19) is guaranteed to converge on a solution provided that the utilisation U_m, including error recovery overhead, is less than 1.

As before, Eq. (10) can be used to compute the number of message instances that need to be examined to find the worst-case response time.

$$w_m^{n+1}(q) = E_m\left(w_m^n + C_m\right) + B_m + qC_m + \sum_{\forall k \in hp(m)} \left\lceil \frac{w_m^n + J_k + \tau_{\text{bit}}}{T_k} \right\rceil C_k \tag{20}$$

Eq. (20) extends Eq. (11) to account for the error recovery overhead. Note that as errors can impact the transmission of message m itself, the time interval considered in calculating the error recovery overhead includes the transmission time of message m as well as its queuing delay. Eqs. (20), (12) and (13) can be used together to compute the response time of each message instance q, and hence find the worst-case response time of each message m in the presence of errors at the maximum rate specified by the error model.

The sufficient schedulability tests given in Section 3.4 can be similarly modified via the addition of the term $E_m(w_m^n + C_m)$ to account for the error recovery overhead.

4 Discussion

In this section we consider various characteristics of CAN systems and discuss whether flaws in the existing analysis can result in erroneous guarantees under specific circumstances that are relevant to real-world systems.

We seek to answer the following questions.

1. Can the existing analysis give faulty guarantees to messages of any priority?
2. If the bus utilization is low, can the existing analysis still result in optimistic response times?
3. Do error models give sufficient engineering margin for error to account for the flaw in the analysis?
4. Does the omission of diagnostic messages during normal operation reduce interference/blocking enough to ensure that the deadlines of the remaining messages will be met?
5. Which message guarantees can we be sure are not at risk?

4.1 Priorities of messages at risk

We have found that the existing analysis gives the correct worst-case response times for the highest priority and the 2nd highest priority message. However; it can compute

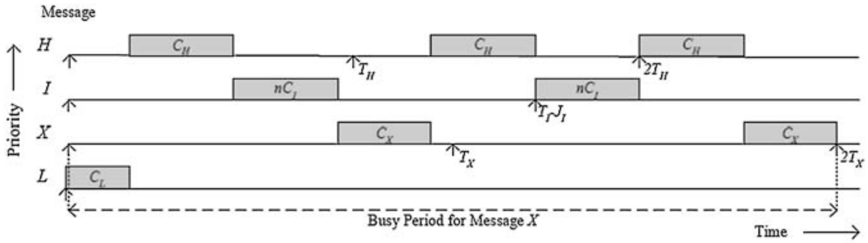


Fig. 7 Busy period for message X

incorrect worst-case response times for messages from the 3rd highest priority to the lowest priority.

This is illustrated by the example message set constructed below and depicted in Fig. 7. The example message set consists of;

(i) a high priority message H;
(ii) a group of n (where $n \geq 1$) intermediate priority messages, represented by I, which all have the same periods and transmission times;
(iii) a message X with a priority below those messages in group I,which highlights the flaw in the analysis and
(iv) a group of k (where $k \geq 0$) low priority messages represented by L, which all have the same transmission times.

The transmission times of the messages are C_H, C_I, C_X and C_L respectively, with the constraint that $C_X > C_L$.

The low priority messages L, are assumed to have very large periods and no jitter. These messages contribute only blocking to the response time of message X. (Note that if there are no lower priority messages, i.e. $k = 0$, then the example still holds with $C_L = 0$).

The period of message H is:

$$T_H = (C_L + 2C_H + 2nC_I + C_X)/2$$

The period of message X is:

$$T_X = (C_L + 3C_H + 2nC_I + 2C_X)/2$$

The period of the intermediate messages I, is assumed to be large ($T_I \gg 2T_X$); however the period less jitter for each intermediate message is:

$$T_I - J_I = C_L + 2C_H + nC_I + C_X$$

By contrast, messages H and X are assumed to have no jitter.

The busy period for message X is shown in Fig. 7. For simplicity, there is only one intermediate priority message shown in the diagram; however, the transmission time of this message is given as nC_I, representing the arbitrary number of intermediate messages that are considered.

We now show that under certain conditions message X exhibits the problem with the existing analysis. The length of the busy period for message X, given by Eq. (8), is:

$$t_X = C_L + 3C_H + 2nC_I + 2C_X = 2T_X$$

Hence, according to Eq. (10), there are two instances of message X in the busy period. We now compute the response times of these two instances. According to Eq. (11), and as $C_X > C_L$, the queuing delay of the first instance of message X is:

$$w_X(0) = C_L + C_H + nC_I$$

Similarly for the second instance:

$$w_X(1) = C_L + 3C_H + 2nC_I + C_X$$

According to Eq. (12), the response times of the two instances are:

$$R_X(0) = C_L + C_H + nC_I + C_X$$

and

$$R_X(1) = (C_L + 3C_H + 2nC_I + 2C_X)/2$$

Comparing the formulas for $R_X(0)$ and $R_X(1)$, then, provided that $C_H > C_L$, the response time of the second instance is greater than that of the first. Meaning that message X exposes the flaw in the existing analysis. (In fact, assuming that $D_X = T_X$, the second instance of message X is only just schedulable with $R_X = T_X$).

As we can choose an arbitrary number ($n \geq 1$) of intermediate priority messages, and similarly an arbitrary number ($k \geq 0$) of lower priority messages, message X may lie anywhere from the 3rd highest to the lowest priority in a set of messages with cardinality greater than or equal to 3. We conclude that any message from the lowest priority to the 3rd highest priority in a set of 3 or more messages can be given an optimistic response time and therefore a faulty guarantee by the existing analysis.

4.2 Breakdown utilisation

The example in Section 1.4 has a bus utilisation of 97%. It is interesting to ask if the existing analysis can yield optimistic worst-case response times for systems with much lower utilisation.

Returning to the example message set, constructed in Section 4.1, we now consider how low the utilisation of that message set can be.

To achieve the lowest possible utilisation, we need only consider the contribution from messages H and X; as the utilisation of both the intermediate messages I, and the low priority messages L, tends to zero when their periods are increased to an

Table 4 Utilisation of message sets breaking the existing analysis

Number of messages	Utilisation
3	45.5%
5	21.4%
10	9.2%
25	3.4%
100	0.82%

arbitrarily large value. We therefore have:

$$U = \frac{2C_X}{C_L + 3C_H + 2nC_I + 2C_X} + \frac{2C_H}{C_L + 2C_H + 2nC_I + C_X}$$

with the constraints that $C_H > C_L$and $C_X > C_L$.

The overall utilisation is minimised by choosing values of C_H and C_X as small as possible, and a value of C_I as large as possible. Given the constraints on CAN message sizes, the minimum occurs when we choose messages H and X to have zero data bytes, so $C_H = C_X = 55\tau_{\text{bit}}$, the intermediate messages to have 8 data bytes, so $C_I = 135\tau_{\text{bit}}$, and no lower priority messages, so $C_L = 0$.

We note that this message set is somewhat pathological, as all the intermediate priority messages have arbitrarily large periods/deadlines and correspondingly large queuing jitter. However, it does illustrate that in general the existing analysis breaks down at very low utilisation levels.

Table 4 provides an upper bound on this breakdown utilisation: the existing analysis is known to breakdown at these levels of utilisation, it may breakdown at still lower levels.

Whilst it is unlikely that real-world applications will have message configurations that replicate the pathological case discussed above, such systems may include messages with large amounts of queuing jitter. Typically these are ‘gatewayed’ messages that have inherited a large jitter from variability in the response time of a source message sent on another network. We conclude that, for applications characterised by non-zero queuing jitter, it is prudent to assume that there could be problems with the existing analysis, irrespective of overall bus utilisation.

In fact, for real-world CAN systems, characterised by messages with non-zero queuing jitter and consequently deadlines less than periods, overall bus utilisation is a poor indicator of system schedulability.

4.3 Margin for error

In Section 3.5 we saw how a generalised error model could be included in the revised schedulability analysis. Bit error rates on CAN are typically very low: 10^{-11} up to 10^{-6} depending on environmental conditions (Ferreira et al., 2004). However, errors do occur and it is therefore appropriate that any commercial application of CAN schedulability analysis should include at least a simple error model to account for sporadic errors on the bus. These errors are typically caused by external sources of Electromagnetic Interference (EMI) such as mobile phones, radar, radio transmitters, and lightning as well as other possible causes such as switch contacts, and shielding or wiring faults. As such errors are typically completely uncorrelated with message transmission, it is reasonable to assume that any useful error model allows for the possibility of an error occurring at any given time, and hence the error function $F(t) \geq 1$ for a time interval of any length t.

Let us now consider the situation where the schedulability analysis given by Tindell and Burns (1994) and Tindell et al. (1994b, 1995) has been used along with an error model with $F(t) = 1$ to determine the schedulability of a system. The recurrence relation used by the existing analysis is given below:

$$w_m^{n+1} = B_m + E_m\big(w_m^n + C_m\big) + \sum_{\forall k \in hp(m)} \left\lceil \frac{w_m^n + J_k + \tau_{\text{bit}}}{T_k} \right\rceil C_k \tag{21}$$

Given that $F(t) \geq 1$, then from Eq. (18), the maximum additional delay to message m due to the error recovery mechanism is always longer than the transmission time of message m, i.e. $E_m(t) > C_m$. Substituting C_m for $E_m(t)$ in Eq. (21) gives:

$$w_m^{n+1} = B_m + C_m + \sum_{\forall k \in hp(m)} \left\lceil \frac{w_m^n + J_k + \tau_{\text{bit}}}{T_k} \right\rceil C_k \tag{22}$$

We note that as $E_m(t) > C_m$, the solution to Eq. (22) cannot be larger than the solution to Eq. (21).

Recall that Eq. (16) provides a correct sufficient but not necessary schedulability test for the case where there are no errors on the CAN bus. Comparing Eqs. (22) and (16), we observe that, as $\max(B_m, C_m) \leq B_m + C_m$, the solution to Eq. (16) cannot be larger than the solution to Eq. (22) and hence cannot be larger than the solution to Eq. (21). This means that if message m is deemed to be schedulable given the queuing delay computed by Eq. (21) for the case where there are errors on the bus, then it must also be schedulable given the queuing delay computed via Eq. (16) for the case where there are no errors on the bus.

This is an important result. It means that if the existing analysis showed that every message was schedulable in the presence of any reasonable error model, with $F(t) \geq 1$, then, despite the flaw in the existing analysis, every message is actually guaranteed to be schedulable when no errors are present. Put another way, the engineering margin for error provided by the error model is sufficient to account for the error in the analysis.

We observe however, that the robustness of systems analysed using the schedulability analysis given by Tindell and Burns (1994), and Tindell et al. (1994b, 1995) may not be all that was expected. Flaws in the existing analysis could lead to message configurations that will miss their deadlines in the presence of errors at a rate within the parameters of the specified error model, even though we can be sure that they will not miss their deadlines when no errors are present on the bus.

4.4 Message omission

Many CAN applications allow for 8 data byte diagnostic messages that are not transmitted during the normal mode of operation. These messages are transmitted only when the system is in diagnostic mode[10] and linked to service equipment. In this section we consider whether the omission of diagnostic messages provides sufficient

[10] Typically, all normal mode messages continue to be transmitted during diagnostic mode.

reduction in interference/blocking to ensure that messages do not miss their deadlines during normal operation, despite being given potentially optimistic worst-case response times by the existing analysis.

To answer this question, we consider a system that is deemed to be schedulable by the existing analysis. We assume that this system includes an 8 data byte diagnostics message Y, which is only transmitted when the system is in diagnostic mode. We note that as message Y has the maximum number of data bytes, its transmission time is equivalent to the largest possible blocking factor, so $C_Y = B^{\text{MAX}}$. The blocking factor for each message m of higher priority than Y, is therefore given by $B_m = B^{\text{MAX}}$, which means that the existing analysis based on Eq. (7) computes exactly the same worst-case response time for each higher priority message m, as the correct sufficient but not necessary schedulability analysis test based on Eq. (17). The existing analysis cannot therefore result in optimistic worst-case response times for messages of higher priority than Y.

For each message of lower priority than Y, the interference due to message Y is at least B^{MAX}. Comparing Eq. (7) and (17), we observe that the solution to Eq. (7), with diagnostic message Y included in the set of higher priority messages, is at least as large as the solution to Eq. (17) when message Y is excluded. This means that if a lower priority message m is deemed to be schedulable by the existing analysis when message Y is present, then it must also be schedulable according to the correct sufficient but not necessary schedulability analysis when message Y is omitted.

We conclude that the omission of a single maximum length message of arbitrary priority provides sufficient reduction in interference/blocking to ensure that the flaw in the existing analysis cannot lead to any of the remaining messages missing their deadlines.

4.5 Message guarantees not at risk

In this section we consider the circumstances under which the first instance of a message in the busy period is guaranteed to have the longest response time. Under these circumstances, despite its flaws, the existing analysis gives correct results.

Assuming that message deadlines do not exceed their periods, then Eq. (16) in Section 3.4 provides an upper bound on the queuing delay for the second and subsequent instances of message m in the busy period. Comparing Eqs. (7) and (16), we observe that if $B_m \geq C_m$, then the first instance of message m is guaranteed to have a response time at least as long as subsequent ones. From the definition of B_m given in Eq. (5), we conclude the following important result: the existing analysis gives the correct response time for any message where there exists at least one lower priority message with equal or longer transmission time/message length.

5 Priority assignment policies

The analysis presented in Section 3 is applicable irrespective of the priority ordering of CAN messages. However, choosing an appropriate priority ordering is important in obtaining a schedulable system, and in maximising robustness to errors.

Table 5 CAN messages highlighting non-optimal priority assignment

Message	Period	Deadline	Number of bits	TX time
A	3.0 ms	3.0 ms	135	1.08 ms
B	4.0 ms	4.0 ms	135	1.08 ms
C	4.5 ms	4.5 ms	65	0.52 ms

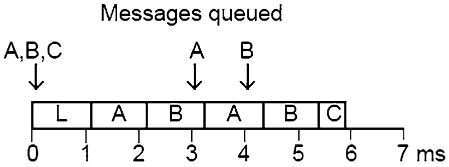


Fig. 8 Message response times with "optimal" priority assignment

Priority ordering is determined by a priority assignment policy. A priority assignment policy P is referred to as *optimal* if there are no systems that are schedulable using any other priority assignment policy that are not also schedulable using policy P.

Tindell and Burns (1994) and Tindell et al. (1995) claimed that *deadline monotonic* priority assignment (Leung and Whitehead, 1982) and "deadline minus jitter" or *(D-J)-monotonic* priority assignment (Zuhily, 2006) was optimal for CAN; however, whilst these policies are optimal for fixed priority pre-emptive scheduling, assuming deadlines no greater than periods, they are not optimal for fixed priority non-pre-emptive scheduling (George et al., 1996), and are therefore not optimal for CAN. This is illustrated by the following example using the set of messages given in Table 5.

This example assumes a 125 Kbit/s network and 11-bit identifiers. Messages A and B contain 8 data bytes and message C contains 1 data byte, giving transmission times of 1.08, 1.08 and 0.52 ms respectively, assuming worst-case bit stuffing. In addition, there are a number of lower priority messages, each containing 8 data bytes, which are also sent on the network; their transmission times are also 1.08 ms.

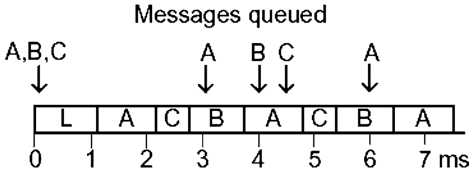
Setting message priorities in the order A—highest, then B, then C results in an unschedulable system. The worst-case response times of messages A and B are 2.16 ms and 3.24 ms respectively; however, in the worst case, message C does not even begin transmission before its deadline.

Figure 8 illustrates the long delays that message C is subject to before transmission. Messages A, B and C are assumed to be queued just too late to enter into arbitration at time $t = 0$ and hence the low priority message L is transmitted first.

The priority ordering A, B, C corresponds to both deadline monotonic and also (D-J)-monotonic priority ordering—as all the messages have zero queuing jitter. If these priority assignment policies are optimal then we should not be able to find another priority ordering which results in all of the deadlines being met; however, if we use the priority ordering A, C, B then the worst-case response times of the messages are: $R_A = 2.16$ ms, $R_C = 2.68$ ms and $R_B = 3.76$ ms, as illustrated in Fig. 9. With this priority ordering, all of the messages meet their deadlines.

The reason that the revised priority ordering results in a schedulable system is that giving the shortest message a higher priority enables all three messages to start transmission within 3 ms of being queued; hence none of them are subject to interference from a second instance of message A and subsequently a second instance of message

Fig. 9 Message response times with an alternative priority assignment



B. This example shows that the priority assignment policies assumed by Tindell and Burns (1994) and Tindell et al. (1995) to be optimal are not.

George et al. (1996) claimed that deadline monotonic priority assignment is optimal for non-pre-emptive systems with no jitter, provided that deadlines and execution times are in the same order i.e. $D_i < D_j$ implies $C_i \leq C_j$. The proof, given by George et al. (1996), assumes that *"as* $\forall i$, $D_i \leq T_i$ *the worst-case response time of any task is found in its first instance"*; however, this assumption is false, as we have seen with the simple example in Section 1.4, and so the proof is undermined. The theorem may or may not still be true.

George et al. (1996) also showed that the optimal priority assignment algorithm devised by Audsley (1991) is applicable to non-pre-emptive systems. In general, Audsley's algorithm is applicable provided that the worst-case response time of a message:

(i) does not depend upon the specific priority ordering of higher priority messages and,
(ii) does not get longer if the message is given a higher priority.

Inspection of the various equations presented in this paper shows that both of the above conditions hold. Neither the length of the queuing delay, nor the length of the busy period depends upon the specific priority order of higher priority messages. Similarly, although the blocking term can get larger with increased priority this is always counteracted by a decrease in interference that is at least as large; hence the length of the busy period and the length of the queuing delay cannot increase with increasing message priority. The optimal priority ordering of CAN messages can therefore be determined using Audsley's priority assignment algorithm, given below.

Optimal Priority Assignment Algorithm

```
for each priority level, lowest first
{
    for each unassigned message m
    {
        if m is schedulable at this priority
        {
            assign m this priority
            break (continue outer loop)
        }
    }
    return unschedulable
}
return schedulable
```

For n messages, Audsley's algorithm performs at most $n(n-1)/2$ schedulability tests and is guaranteed to find a schedulable priority assignment if one exists. It does not however specify an order in which messages should be tried at each priority level. This order heavily influences the priority assignment chosen if there is more than one ordering that is schedulable. In fact, a poor choice of initial ordering can result in a priority assignment that leaves the system only just schedulable. We therefore suggest that, as a useful heuristic, messages are tried at each priority level in (D-J) order, largest value of (D-J) first, with ties broken according to message length, longest first.

6 Implications and recommendations

In this section we discuss the implications of flaws in existing CAN schedulability analysis on commercial CAN schedulability analysis tools and deployed CAN applications.

6.1 CAN schedulability analysis tools

CAN schedulability analysis tools need to take account of the findings presented in this paper. This will involve checking, and if necessary updating, the analysis they employ; ensuring that it cannot provide optimistic worst-case response times and false guarantees.

The sufficient but not necessary schedulability tests given in Section 3.4 provide a "quick-fix" solution with minimal changes required to the existing analysis. These tests are however pessimistic and implementing the revised analysis given in Section 3 would potentially lead to a better technical solution.

Whilst "deadline minus jitter" or (D-J)-monotonic priority ordering is still a good heuristic to use, it is not necessarily the optimal priority assignment policy for CAN. Implementing priority ordering based upon Audsley's optimal priority assignment algorithm would ensure that a schedulable priority ordering is found whenever one exists.

6.2 Commercial CAN applications

System Designers configuring commercial CAN applications often take the engineering approach that all messages in the system should remain schedulable given the addition of any number of low priority messages that can be used for development and test purposes. Such analysis based on Tindell and Burns (1994) and Tindell et al. (1994b, 1995) would assume that every message is subject to the maximum blocking factor, as per the sufficient schedulability test given by Eq. (17). This schedulability test computes a correct upper bound on the actual response time of each message, and so provides a correct guarantee that the configured messages will meet their deadlines.

Given the flaws in the existing schedulability analysis, it would however be prudent for System Designers to check the precise details of the analysis used to compute worst-case response times for their systems. If the analysis used has the potential to compute erroneous worst-case response times, then the feasibility of all the CAN configurations designed, developed and deployed using that analysis should be checked to ensure that they are in fact schedulable, and robust to errors at the rate specified by the prescribed error model.

6.3 Faults in deployed systems

Many deployed CAN systems, for example those in automotive applications, will have been analysed using the pragmatic engineering approach described in the previous section. The flaws in the existing analysis cannot lead to a problem with a deployed system in this case.

Many CAN applications allow for maximum length (8 data byte) diagnostic messages that are not transmitted during normal operation. Assuming that the existing analysis deemed the deployed system to be schedulable with these diagnostic messages present, then Section 4.4 showed that the omission of a single diagnostic message provides sufficient reduction in interference/blocking to ensure that the flaws in the existing analysis cannot lead to other messages missing their deadlines during normal operation.

In Section 4.5 we saw that the existing analysis gives the correct response time for any message where there is at least one lower priority message with equal or longer transmission time/message length. Many CAN applications use exclusively 8 data byte messages as a means of addressing the high ratio of overhead to useful data on CAN. In this case, the existing analysis is guaranteed to compute correct response times for all but the lowest priority message.

Even if a message has the potential to be given an erroneous worst-case response time by the existing analysis, then, unless that message is close to being unschedulable, the computed worst-case response time is still likely to be the true value. Even if an optimistic value is computed, then the true value may still be less than the message deadline. Finally, for a deadline miss to actually happen in a deployed system requires that the worst-case message phasing occurs, and at that point a number of messages take close to their maximum transmission times. This requires worst-case or near worst-case bit stuffing to occur which is, in itself, highly unlikely (Nolte et al., 2002).

Normal practice with commercial CAN configurations is to ensure that the schedulability analysis used includes provision for a plausible error model. In this case, Section 4.3 showed that such systems are guaranteed to be schedulable when no errors are present on the CAN bus provided that they were deemed to be schedulable in the presence of errors by the existing analysis.

We conclude that deadline misses in deployed CAN systems due to flaws in the existing analysis are extremely unlikely. Any such deadline failures are more likely to occur due to errors occurring on the bus at a higher rate than that accounted for by the error model.

We note that embedded CAN-based systems are built to be resilient to some messages missing their deadlines, and to much simpler forms of error such as wiring faults. CAN is not used, in its basic form, for safety critical systems due to known issues such as the "double receive" and "babbling idiot" problems (Rufino et al., 1998; Broster and Burns, 2003; Rufino, 2002).

7 Summary and conclusions

In this paper we highlighted a significant flaw in long-standing, highly cited, and widely used schedulability analysis of CAN. We showed how this flaw could lead to the computation of optimistic worst-case response times for CAN messages, broken guarantees, and deadline misses. This paper provides revised analysis that can be used to calculate correct worst-case response times for CAN.

In addition, we showed that:

1. The existing analysis can provide optimistic worst-case response times for messages from the 3rd highest priority to the lowest priority.
2. The existing analysis can lead to broken guarantees and hence deadline misses in systems with low bus utilisation.
3. Where an error model has been considered, the flaw in the existing analysis is not sufficient to lead to CAN configurations that will result in missed deadlines when no errors are present on the bus. The desired robustness to errors may not however be achieved.
4. The omission of a single maximum length diagnostic message, accounted for by the existing analysis, reduces interference/blocking enough to ensure that the deadlines of all the remaining messages are met during normal operation.
5. Despite its flaws, the existing analysis gives the correct response time for any message where there is at least one lower priority message with the same or longer transmission time/message length.

We discussed the implications of these results for commercial CAN systems developed using flawed analysis and provided two simple, sufficient schedulability tests enabling a “quick-fix” to be made to commercial CAN schedulability analysis tools.

Finally, we showed that neither deadline monotonic nor (D-J)-monotonic priority assignment is optimal for CAN. Audsley’s priority assignment algorithm is however optimal for fixed priority non-pre-emptive systems and can be used to obtain a schedulable priority ordering for CAN whenever one exists.

7.1 Future work

A considerable body of academic work has grown up from Tindell’s seminal analysis of CAN. The flaws in that original work may have partly undermined some of the subsequent research built upon it. Authors that have cited the original CAN analysis in their work are therefore encouraged to check the implications. In particular the academic work most likely to be affected is that which extends the original analysis and pushes system schedulability to its limits, for example work on error models.

7.2 Postscript

Volcano Network Architect (VNA) is a commercial CAN design and analysis tool originally developed by Volcano Communications Technologies AB, and now owned by Mentor Graphics Corporation. By 2004, over 20 million cars, with an average of 20 ECUs per car, had been programmed using this technology (Oswald, 2004).

The engineering team responsible for Volcano Network Architect was given early visibility of this paper, enabling them to check the validity of the schedulability analysis used in their commercial products.

The analysis provided by Volcano Network Architect was found to be sufficient: it assumes the longest possible blocking time irrespective of message priority (Horvath, 2006) and therefore computes an upper bound on response times, similar to that given by the sufficient schedulability tests described in Section 3.4.

Acknowledgments This work was partially funded by the UK EPSRC funded DIRC project, the EU funded FRESCOR project and the IST-004527 funded ARTIST 2 network of excellence on Embedded Systems Design.

References

Audsley NC (1991) Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical Report YCS 164, Dept. Computer Science, University of York, UK

Bosch (1991) CAN Specification version 2.0. Robert Bosch GmbH, Postfach 30 02 40, D-70442 Stuttgart

Bril RJ (2006) Existing worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred pre-emption is too optimistic. CS-Report 06-05, Technische Universiteit Eindhoven (TU/e), The Netherlands

Bril RJ, Lukkien JJ, Davis RI, and Burns A (2006a) Message response time analysis for ideal Controller Area Network (CAN) refuted. CS-Report 06-19, Technische Universiteit Eindhoven (TU/e), The Netherlands.

Bril RJ, Lukkien JJ, Davis RI, Burns A (2006b) Message response time analysis for ideal Controller Area Network (CAN) refuted. In: Proceedings of the 5th International Workshop on Real-Time Networks (RTN'06)

Bril RJ, Lukkien JJ, Verhaegh WFJ (2006c) Worst-case response time analysis of real-time tasks under fixed priority scheduling with deferred preemption revisited. CS Report 06-34, Technische Universiteit Eindhoven (TU/e), The Netherlands

Broster I (2003) Flexibility in dependable communication. PhD Thesis, Department of Computer Science, University of York, UK

Broster I, Burns A, Rodríguez-Navas G (2002) Probabilistic analysis of CAN with Faults. In: Proceedings of the 23rd IEEE real-time systems symposium (RTSS'02), pp 269–278

Broster I, Burns A (2003) An analysable bus-guardian for event-triggered communication. In: Proceedings of the 24th real-time systems symposium. IEEE Computer Society Press, pp 410–419

Broster I, Burns A, Rodriguez-Navas G (2005) Timing analysis of real-time communication under electromagnetic interference. Real-Time Systems 30(1–2):55–81

Burns A (1994) Pre-emptive priority based scheduling: An appropriate engineering approach. In: S. Son (ed), Advances in Real-Time Systems, Prentice-Hall, pp 225–248

Casparsson L, Rajnak A, Tindell K, Malmberg P (1998/1) Volcano—a revolution in on-board communications. Volvo Technology Report

DeMeis R (2005) Cars sag under weighty wiring. Electronic Times, 10/24/2005

Ferreira J, Oliveira A, Fonseca P, Fonseca JA (2004) An experiment to assess bit error rate in CAN. In: Proceedings of 3rd international workshop of real-time networks (RTN2004). Cantania, Italy, pp 15–18

Frischkorn H-G (2005) Automotive architecture requirements. In: Proceedings of the summer school on architectural paradigms for dependable embedded systems. Vienna, Austria,. Vienna University of Technology, pp 45–74

George L, Rivierre N, Spuri M (1996) Pre-emptive and non-pre-emptive real-time uni-processor scheduling. Technical Report 2966, Institut National de Recherche et Informatique et en Automatique (INRIA), France

Hansson H, Nolte T, Norstrom C, Punnekkat S (2002) Integrating reliability and timing analysis of CAN-based Systems. IEEE Transactions on Industrial Electronics 49(6):1240–1250

Harbour MG, Klein MH, Lehoczky JP (1991) Fixed priority scheduling of periodic tasks with varying execution priority. In: Proceedings 12th IEEE real-time systems symposium. IEEE Computer Society Press, pp 116–128

Horvath I (2006) Private communication with the authors

ISO 11898-1 (1993) Road Vehicles—interchange of digital information—Controller Area Network (CAN) for high-speed communication. ISO Standard-11898, International Standards Organisation (ISO)

Lehoczky J (1990) Fixed priority scheduling of periodic task sets with arbitrary deadlines. In: Proceedings 11th IEEE real-time systems symposium. IEEE Computer Society Press, pp 201–209

Leohold J (2004) Communication requirements for automotive systems. Keynote speech 5th IEEE international workshop on factory communication systems, Vienna, Austria, Vienna University of Technology

Leohold J (2005) Automotive system architecture. In: Proceedings of the summer school on architectural paradigms for dependable embedded systems. Vienna, Austria, Vienna University of Technology, pp 545–591

Leung JY-T, Whitehead J (1982) On the complexity of fixed-priority scheduling of periodic real-time tasks. Performance Evaluation 2(4):237–250

LIN Consortium (2003) LIN Protocol Specification, Revision 2.0. www.lin-subbus.org

Liu CL, Layland JW (1973) Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM 20(1):46–61

Motorola Inc (1998) MSCAN Block Guide V03.01 Document No. SV12MSCANV3/D. FreeScale Semiconductor Inc. (Revised July 2004)

Nolte T, Hansson H, Norstrom C (2002) Minimizing CAN response-time analysis jitter by message manipulation. In: Proceedings 8th IEEE real-time and embedded technology and applications symposium (RTAS'02), pp 197–206

Nolte T, Hansson H, Norstrom C (2003) Probabilistic worst-case response-time analysis for the Controller Area Network. In: Proceedings of the 9th IEEE real-time and embedded technology and applications symposium (RTAS'03), pp 200–207

Nolte T (2006) Share-driven scheduling of embedded networks. PhD Thesis, Malardalen University Press

Oswald M (2004) Efficient automotive electronics. Automotive Engineer

Punnekkat S, Hansson H, Norstrom C (2000) Response time analysis under errors for CAN. In: Proceedings 6th real-time technology and applications symposium. IEEE Computer Society Press, pp 258–265

Regehr J (2002) Scheduling tasks with mixed pre-emption relations for robustness to timing faults. In: Proceedings 23rd real-time systems symposium. IEEE Computer Society Press, pp 315–326

Rufino J, Verissimo P, Arroz G, Almeida C Rodrigues L (1998) Fault-tolerant broadcasts in CAN. In Digest of Papers, The 28th IEEE international symposium on fault-tolerant computing (FTCS'98), pp 150–159

Rufino J (2002) Computational system for real-time distributed control. PhD-Thesis, Technical University of Lisbon, Instituto Superior

Society of Automotive Engineers (1993) Class C application requirement considerations, recommended practice, SAE Technical Report J2056/1

Tindell KW, Burns A (1994) Guaranteeing message latencies on Controller Area Network (CAN). In: Proceedings of 1st international CAN conference, pp 1–11

Tindell KW, Burns A, Wellings AJ (1994a) An extendible approach for analysing fixed priority hard real-time systems. Journal of Real-Time Systems 6(2):133–152

Tindell KW, Hansson H, Wellings AJ (1994b) Analysing real-time communications: Controller Area Network (CAN). In: Proceedings 15th real-time systems symposium (RTSS'94). IEEE Computer Society Press, pp 259–263

Tindell KW, Burns A, Wellings AJ (1995) Calculating Controller area network (CAN) message response times. Control Engineering Practice 3(8):1163–1169

Wang Y, Saksena M (1999) Scheduling fixed priority tasks with pre-emption threshold. In: Proceedings of the 6th international workshop on real-time computing systems and applications (RTCSA'99), pp 328–335

Zuhily A (2006) Optimality of (D-J)-monotonic priority assignment. Technical Report YCS404. Dept. of Computer Science, University of York, UK

Robert I. Davis received a DPhil in Computer Science from the University of York in 1995. Since then he has founded three start-up companies, all of which have succeeded in transferring real-time systems research into commercial product. At Northern Real-Time Technologies Ltd. (1995–1997) he was responsible for development of the Volcano CAN software library. At LiveDevices Ltd. (1997–2001) he was responsible for development of the Real-Time Architect suite of products, including an OSEK RTOS and schedulability analysis tools. In 2002, Robert returned to the University of York, and in 2004 he was involved in setting up a spin out company, Rapita Systems Ltd., aimed at transferring worst-case execution time analysis technology into industry. Robert is a member of the Real-Time Systems Research Group at the University of York, and a director of Rapita Systems Ltd. His research interests include scheduling algorithms and schedulability analysis for real-time systems.

Alan Burns is head of the Real-Time Systems Research Group at the University of York. His research interests cover a number of aspects of real-time systems including the assessment of languages for use in the real-time domain, distributed operating systems, the formal specification of scheduling algorithms and implementation strategies, and the design of dependable user interfaces to real-time applications. He has authored/co-authored over 370 papers and 10 books, with a large proportion of them concentrating on real-time systems and the Ada programming language. Professor Burns has been actively involved in the creation of the Ravenscar Profile, a subset of Ada's tasking model, designed to enable the analysis of real-time programs and their timing properties.

Reinder J. Bril received a B.Sc. and an M.Sc. (both with honours) from the University of Twente, and a Ph.D. from the Technische Universiteit Eindhoven, the Netherlands. He started his professional career in January 1984 at the Delft University of Technology. From May 1985 until August 2004, he was with Philips, and worked in both Philips Research as well as Philips' Business Units. He worked on various topics, including fault tolerance, formal specifications, software architecture analysis, and dynamic resource management, and in different application domains, e.g. high-volume electronics consumer products and (low volume) professional systems. In September 2004, he made a transfer back to the academic world, to the System Architecture and Networking (SAN) group of the Mathematics and Computer Science department of the Technische Universiteit Eindhoven. His main research interests are currently in the area of reservation-based resource management for networked embedded systems with real-time constraints.

Johan J. Lukkien has been head of the System Architecture and Networking Research group at Eindhoven University of Technology since 2002. He received an M.Sc. and a Ph.D. from Groningen University in the Netherlands. In 1991, he joined Eindhoven University, after two years leave at the California Institute of Technology. His research interests include the design and performance analysis of parallel and distributed systems. Until 2000 he was involved in large-scale simulations in physics and chemistry. Since 2000, his research focus has shifted to the application domain of networked resource-constrained embedded systems. Contributions of the SAN group are in the area of component-based middleware for resource-constrained devices, distributed co-ordination, Quality of Service in networked systems and schedulability analysis in real-time systems.