Robust Priority Assignment for Messages on Controller Area Network (CAN)

Robert Davis and Alan Burns
Real-Time Systems Research Group, Department of Computer Science,
University of York, YO10 5DD, York (UK)
rob.davis@cs.york.ac.uk, alan.burns@cs.york.ac.uk

Abstract

This paper addresses the problem of determining the most robust priority assignment for CAN messages that are subject to transmission errors due to Electromagnetic Interference. In the presence of errors on the bus, CAN messages have a non-zero probability of missing their deadlines. An appropriate choice of priority ordering can minimise the overall worst-case deadline failure probability resulting in a more robust system.

This paper shows that "Deadline minus jitter" monotonic priority assignment, commonly used for priority assignment in commercial CAN systems, does not always result in the most robust priority ordering.

A Robust Priority Assignment algorithm is presented that computes the most robust priority ordering for CAN messages subject to bit errors on the bus. This algorithm is optimal in the sense that it can be used to (i) maximise the number of errors tolerated, (ii) maximise the delay tolerated by any message, or (iii) minimise the probability of any message failing to meet its deadline. This algorithm is efficient and appropriate for use in an engineering context.

1. Introduction

1.1. Background

Controller Area Network (CAN) is a serial communications bus designed to provide simple, efficient and reliable communications for in-vehicle networks (Bosch, 1991). Deployment of CAN in production vehicles began with Mercedes in 1991, with the majority of the European automotive industry adopting CAN by the end of the 1990s. An indication of the scale of adoption of CAN by the automotive industry can be gained from the sales of microcontrollers with on-chip CAN peripherals. Over 1 billion such devices have been deployed in automotive applications during the last three years (CiA, 2006).

In automotive applications, CAN is typically used to provide high speed networks (500Kbits/s) connecting chassis and power-train Electronic Control Units (ECUs) and low speed networks (100 or 125Kbits/s) connecting body electronics. Messages sent on CAN are used to communicate state information, referred to as signals, between different ECUs. Examples of signals include: wheel speeds, engine rpm, gear selection, switch positions, climate control settings, fault codes, and so on. The majority of these signals have real-time constraints associated with them, in the range of 5ms to 1 second (Society of Automotive Engineers, 1993).

The CAN protocol uses bit stuffing and a 15-bit CRC to enable nodes on the network to detect errors in messages sent on the bus. When an error is detected in a transmitted message, this leads to a cascade of error flag transmissions from other nodes; effectively aborting the errant message and causing it to be re-queued for subsequent re-transmission. As a result of this error recovery mechanism, a single bit error leads to an additional error recovery overhead, as well as re-transmission of the affected message.

Errors on CAN can be caused by Electromagnetic Interference (EMI). Predicting the effects of EMI is both difficult and uncertain (Ladkin, 1997), due to its diverse causes such as sparks, lightning, radar, mobile phones, high voltage switching etc. Whilst shielding and "twisted-pair" wiring can help to reduce the incidence of errors, bit errors can and do occur in commercial systems. Typically error rates of 10^{-11} to 10^{-7} errors/bit are observed depending heavily upon environmental conditions (Ferreira et al., 2004).

Annex G, (Road Transport) of the IET fact file on Electromagnetic Compatibility and Functional Safety (IEE, 2000) states that: *"The automotive EMC environment is one of the most severe and the most unpredictable. Road vehicles are inherently mobile and thus able to drive near to any fixed transmitter. Vehicle owners and operators believe it to be their right to attach any sort of transmitter (even very high-powered ones) to the vehicle while expecting it to function correctly. Owners also expect to be able to fit electronic equipment into the vehicle and power it off the vehicle's power supply".* The conclusion this leads to is that errors on CAN caused by EMI cannot be predicted with any great accuracy.

1.2. Related research on schedulability analysis of CAN

Tindell and Burns (1994), and Tindell et al. (1994)

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 the messages on a network.

Tindell et al. (1995) further developed the analysis of CAN, dealing with messages subject to single bit errors on the bus. These errors were modelled as a sporadic stream of faults with some minimum inter-arrival time between them. This deterministic fault model was later generalised by Punnekkat et al. (2000), to deal with interference caused by several sporadic sources. Leading on from this early work, Rufino et al. (1998), and Rufino (2002), developed an alternative fault model based on bounded omission and bounded inaccessibility assumptions.

Navet et al. (2000) proposed an alternative fault model based on random arrivals, where faults are assumed to occur according to a Poisson distribution. Navet et al. (2000) introduced the idea of a "tolerable error threshold", corresponding to the maximum number of errors that a message can tolerate before it becomes un-schedulable. Navet et al. (2000) used the tolerable error threshold in a calculation of worst-case deadline failure probability (WCDFP). Subsequently, Broster et al. (2002, 2005), and Broster (2003) extended the work of Navet et al. (2000), correcting and improving upon the WCDFP analysis.

Nolte et al. (2002, 2003) considered probabilistic rather than worst-case bit-stuffing and its impact on message worst-case response times.

Recently, Davis et al. (2007) highlighted and corrected significant flaws in the original schedulability analysis of CAN developed by Tindell and Burns (1994), and Tindell et al. (1994, 1995) and utilised in much of the subsequent research (Punnekkat et al., 2000; Rufino et al., 1998; Rufino, 2002; Navet et al., 2000; Broster et al., 2002, 2005; Broster, 2003; Nolte et al., 2002, 2003).

1.3. Motivation

In their work on the schedulability analysis of CAN Tindell and Burns (1994), and Tindell et al. (1994, 1995) claimed that "deadline minus jitter" monotonic priority order (D-JMPO)[1] was the optimal priority assignment policy to use. This conjecture has since been proven incorrect (Davis et al., 2007). Davis et al. (2007) showed that an optimal priority assignment for CAN messages can be found using Audsley's optimal priority assignment algorithm (Audsley, 1991). This algorithm is optimal in the sense that it guarantees to find a schedulable priority ordering if one exists. However, in the case of systems which are schedulable with a number of different priority orderings, the priority ordering found by Audsley's algorithm is heavily dependent on the initial message ordering and can result in systems that are close to being un-schedulable. Such a priority ordering may be fragile rather than robust, as a small amount of additional interference in the form of errors on the bus could cause deadline failures.

The work of Navet et al. (2000), Broster et al. (2002, 2005), and Broster (2003) showed that the worst-case response time of a CAN message and its tolerance to errors on the bus are heavily dependent on its relative priority. In commercial applications of CAN, what is required is a message priority ordering that is as robust as possible to additional interference (Davis and Burns, 2007). Determining such a robust priority ordering is the focus of this paper.

1.4. Related research on priority assignment

Research into priority assignment for fixed priority scheduling on single-processor systems has mainly focussed on finding the *optimal* priority assignment policy or algorithm. A priority assignment policy or algorithm is referred to as *optimal* if it provides a feasible priority ordering (resulting in a schedulable system) whenever such an ordering exists.

For fixed priority pre-emptive systems, Serlin (1972), and Liu and Layland (1973), showed that for synchronous tasks (that share a common release time), that comply with a restrictive system model, and that have deadlines equal to their periods $(D = T)$, then r*ate monotonic* priority ordering (RMPO) is optimal.

Leung and Whitehead (1982) showed that for synchronous tasks with deadlines less than or equal to their periods $(D \leq T)$, but otherwise compliant with Liu and Layland's system model, then d*eadline monotonic* priority ordering (DMPO) is optimal. They noted that for *asynchronous* tasks (that do not share a common release time), DMPO is not optimal.

More recently, Zuhily and Burns (2007) confirmed that "deadline minus jitter" monotonic priority ordering (D-JMPO) is optimal for synchronous task sets with $D \leq T$ and arbitrary release jitter. Both DMPO and RMPO are effectively special cases of D-JMPO.

Audsley (1991, 2001) solved the problem of priority assignment for asynchronous task sets. Audsley's priority assignment algorithm is optimal in the sense that it finds a schedulable priority ordering if one exists.

George et al. (1996) provided schedulability analysis for non-pre-emptive fixed priority scheduling. They showed that in the non-pre-emptive case, DMPO is no longer optimal for synchronous tasks with deadlines less than or equal to their periods $(D_i \leq T_i)$.

[1] D-JMPO assigns priorities in order of "deadline minus jitter", such that the message with the smallest value of deadline minus jitter is assigned the highest priority.

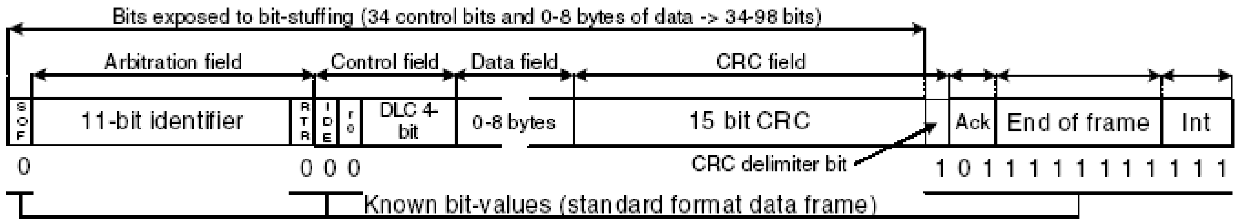

Figure 1: Standard Format Data Frame

George et al. (1996) showed that Audsley's optimal priority assignment algorithm is however applicable in this case. George et al. (1996) claimed that DMPO is optimal for the restricted case of non-pre-emptive scheduling where $D_i \le T_i$ and $D_i \ge D_j \Rightarrow C_i \ge C_j$. The appendix to this paper provides a counter example, showing that DMPO is not optimal in this case.

Bletsas and Audsley (2006) showed that both Audsley's algorithm and DMPO remain optimal, for pre-emptive systems, in the presence of blocking when resources are accessed according to the Stack Resource Policy (SRP) (Baker, 1991) developed from the Priority Ceiling Protocol (PCP) (Sha et al., 1990).

Related research by Lehoczky et al. (1989), Katcher et al. (1993), Punnekkat et al. (1997), and Regehr (2002) used the *critical scaling factor*[2] as a metric for examining schedulability. Regehr (2002) explored the idea of a robust-optimal class of scheduling algorithms that maximise the critical scaling factor. Regehr showed that for tasksets where DMPO is optimal, it is also robust-optimal with respect to the critical scaling factor.

Davis and Burns (2007) introduced the concept of a r*obust priority ordering*, defined as the priority ordering that is schedulable, and also tolerates the maximum amount of additional interference of any feasible priority ordering. Davis and Burns gave a Robust Priority Assignment algorithm that finds the robust priority ordering for a wide range of systems scheduled according to fixed priorities. Davis and Burns (2007) also showed that for systems where D-JMPO is the optimal priority assignment policy, it is also a robust priority assignment policy, effectively independent of the additional interference function.

The research presented in this paper extends the idea of robust priority assignment to cover both deterministic and probabilistic analysis of CAN. In doing so, it builds upon previous work on: probabilistic analysis of CAN (Navet et al., 2000; Broster et al., 2002, 2005; Broster, 2003), optimal priority assignment (Audsley, 1991, 2001), and robust priority assignment (Davis and Burns, 2007).

[2] The critical scaling factor is the largest factor by which the execution time of every task can be increased and the system remain schedulable.

1.5. Organisation

Section 2 describes the CAN protocol and terminology before outlining the scheduling model and notation used in subsequent sections. Section 3 outlines basic schedulability analysis for CAN. This is then extended to cover both deterministic and probabilistic fault models. Section 4 shows that the Robust Priority Assignment algorithm introduced by Davis and Burns (2007) is applicable to CAN and then adapts it to determine robust priority orderings that maximise three robustness metrics. Section 5 concludes with a summary of the main contributions of the paper and recommendations for future research.

2. Controller Area Network

This section gives an outline of elements of the CAN protocol and the characteristics of a system model that are needed to formulate response time analysis for CAN. For a complete description of the CAN protocol, the reader is referred to the CAN specification version 2.0 (Bosch, 1991).

2.1. CAN protocol and terminology

CAN is an asynchronous, multi-master, broadcast, serial data bus that uses Carrier Sense Multiple Access/ Collision Resolution (CSMA/CR) to determine access. Message transfer over CAN is controlled by 4 different types of *frame*: Data frames, Remote Transmit Request (RTR) frames, Overload frames and Error frames. In this paper we are interested in data frames and error frames. This work can however easily be extended to include RTR frames using the approach given by Tindell et al. (1995).

The layout of a standard format data frame is shown in Figure 1. This figure highlights the various control fields, with the 0-8 byte data field shown at a reduced scale. Each CAN data frame is required to have a unique identifier. Identifiers may be 11-bit (standard format) or 29-bit (extended format). The CAN protocol uses the message identifier as a priority to determine which message, among those contending for the bus, will be transmitted next.

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.

2.1.1 Priority based arbitration

The CAN protocol calls for nodes to wait until a *bus idle period*[3] 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 the numerically lowest 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 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.

2.1.2 Error detection

CAN was designed as a robust and reliable form of communication for short messages. It provides a number of mechanisms to detect errors on the bus.

The transmitting node effectively reads back each bit it sends on the bus. This means that it is potentially able to immediately detect bit errors, with the following two exceptions (Rufino and Verissimo, 1995):

1. Corrupt bits in the arbitration field, where all transmitters send a '1' (recessive) but receive back a '0' (dominant) are not immediately detected by the transmitter(s). They are instead interpreted as another node transmitting the identifier for a higher priority frame.
2. Corrupt bits in the acknowledge slot, where the transmitter sends a '1' and receives back a '0'. This is interpreted as at least one node having received the message correctly.

As well as the transmitter checking the bits sent, the receiving nodes check the message for bit-stuffing errors (see Section 2.1.3), form-errors in the fixed parts of the message, CRC-errors, and acknowledgement errors.

A 15-bit Cyclic Redundancy Check (CRC) is used by receiving nodes to check for bit errors in the variable part of the transmitted message. The CRC is calculated over all of the fields in the message up to and including the CRC (see Figure 1).

If a node detects an error in the transmitted message, 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. 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[4] of error recovery overhead plus re-transmission of the message itself.

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 Figure 1. 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.

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:

$$C_m = (55 + 10 s_m)\tau_{bit} \tag{1}$$

for standard (11-bit) identifiers and

$$C_m = (80 + 10 s_m)\tau_{bit} \tag{2}$$

for extended (29-bit) identifiers, where τ_{bit} is the transmission time for a single bit.

2.2. Scheduling model

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

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.

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 unique identifier and hence a unique priority. For convenience, we will refer to the messages priorities as being from 1 to *n* (where *n* is the total number of messages). 1 therefore represents the highest priority and *n* the lowest. Each message has a maximum number of data bytes s_m, and a maximum transmission time C_m, given by Equation (1) or (2) as

[3] 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 Figure 1.

[4] The maximum error recovery overhead is specified as 29 bits on page 8 of part A of the CAN specification 2.0 (Bosch 1991) for standard identifiers only, and as 31 bits on page 40 of the CAN specification 2.0 Part B (Bosch 1991) for both standard and extended identifiers.

appropriate.

The event that triggers queuing of each 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 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 that triggers queuing of the message, 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.

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. We assume that all messages have deadlines less than or equal to their periods $(D_m \le T_m)$.

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 correctly 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 $(R_m \le D_m)$. The system is schedulable if and only if all of the messages in the system are schedulable.

2.3. Error model

In this section we describe a basic model of the effects of errors caused by Electromagnetic Interference. This model is used in the remainder of the paper.

We assume that EMI results in single bit-errors, effectively corrupting the value of a bit on the CAN bus from recessive to dominant or vice-versa. We assume that all bit errors are detected; either immediately by the transmitter (with the exceptions noted in Section 2.1.2) or subsequently by receiving nodes.

For immediate detection to occur the transmitting node must be amongst the group of nodes that the error manifests itself to; in other words, the transmitter must read back a '1' when it transmitted a '0' or vice-versa. This is not always the case with errors caused by EMI. It is arguable that nodes further from the transmitter and/or nearer the source of EMI are much more likely to see a corrupt bit than the transmitter, which is actively driving the voltage differential on the bus. In this case, error detection will typically not be immediate; with a group of receivers reading the corrupt value, but only signalling an error once the CRC field has been transmitted.

For this reason, we make worst-case assumptions with respect to the effects of single bit-errors caused by EMI, We assume that:

(i) Each bit-error affects a separate message transmission.

(ii) The message affected is the longest one that could delay successful transmission of message *m*.

(iii) The error is detected on the last bit of the message.

(iv) The error recovery overhead *F*, is the maximum possible.

The maximum interference, on message *m*, resulting directly from a single bit error is therefore:

$$F + \max_{k \in hep(m)} (C_k) \tag{3}$$

where *hep(m)* is the set of messages with the same priority as message *m* or higher, and *F* is the maximum error recovery overhead.

2.4. Summary of notation

Table 1 summarises the notation used in this paper.

Table 1: Notation

Symbol	Description
τ_{bit}	Transmission time for a single bit.
s_m	Number of data bytes in message *m*.
C_m	Maximum transmission time for message *m*
T_m	Minimum inter-arrival time of message *m*.
D_m	Relative deadline of message *m*.
R_m	Worst-case response time of message *m*.
w_m	Worst-case queuing delay before message *m* commences successful transmission.
F	Maximum error recovery overhead for a single bit fault.
$R_{m\|K}$	The worst-case response time for message *m*, assuming *K* single bit faults prior to successful transmission.
K_m	The maximum number of single bit faults that message *m* can tolerate without missing its deadline.
λ	Average number of bit errors per second.
$p(K,t)$	Probability of *K* faults occurring in a time interval of length *t*.
$p(R_{m\|K})$	Upper bound on the probability of response time $R_{m\|K}$ occurring.
$p_m(fail)$	Upper bound on the probability that message *m* will fail to meet its deadline. Referred to as the worst-case deadline failure probability (WCDFP).

Symbol	Description
$p(fail)$	Upper bound on the worst-case deadline failure probability for any message.
$p^Q(fail)$	Upper bound on worst-case deadline failure probability for any message, given priority ordering *Q*.
$E(\alpha, w, m)$	Additional interference function, with α as a scaling factor.
α_m^P	Value of α characterising the maximum amount of additional interference tolerated by message *m* under priority ordering *P*.
α^P	Value of α characterising the maximum amount of additional interference tolerated by any message under priority ordering *P*.
$hp(m)$	The set of messages with priority strictly higher than *m*.
$hep(m)$	The set of messages with priority higher than or equal to *m*.
$hp(k,P)$	The set of messages with priority strictly higher than *k* in priority ordering *P*.
$hep(k,P)$	The set of messages with priority higher than or equal to *k* in priority ordering *P*.
$lp(k,P)$	The set of messages with priority strictly lower than *k* in priority ordering *P*.

3. Schedulability analysis for CAN

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. Response time analysis for CAN was first provided by Tindell and Burns (1994), and Tindell et al. (1994, 1995), however, flaws in this original analysis have recently been discovered and corrected (Davis et al., 2007).

3.1. Basic schedulability analysis

In this paper, we make use of the simple sufficient but not necessary schedulability test given by Davis et al. (2007). We note that this analysis is exact for many commercial CAN systems that have 8 data byte (soft) real-time messages present at lower priorities. The interested reader is referred to (Davis et al., 2007) for details of other schedulability tests for CAN and the conditions under which the various tests provide sufficient or exact analysis.

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 for transmission.

(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.

(iii) The transmission time C_m, corresponding to the longest time that the message can take to be transmitted.

The queuing delay w_m can be determined using the following recurrence relation:

$$w_m^{n+1} = \max(B_m, C_m) + \sum_{\forall k \in hp(m)} \left\lceil \frac{w_m^n + J_k + \tau_{bit}}{T_k} \right\rceil C_k \tag{4}$$

where *hp(m)* is the set of messages with priority higher than *m*, and $\max(B_m, C_m)$ corresponds to longest possible time for which an invocation of message *m* can be blocked either by lower priority messages or via push through blocking due to the previous invocation of the same message. A suitable starting value is $w_m^0 = \max(B_m, C_m)$. 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 message is given by:

$$R_m = J_m + w_m + C_m \tag{5}$$

3.2. Deterministic fault model

Navet et al. (2000) showed that the worst-case response time $R_{m|K}$, for message *m*, assuming *K* single bit errors prior to successful transmission, can be calculated via the following recurrence relation:

$$w_m^{n+1} = \max(B_m, C_m) + K\left(F + \max_{k \in hep(m)}(C_k)\right) + \sum_{\forall k \in hp(m)} \left\lceil \frac{w_m^n + J_k + \tau_{bit}}{T_k} \right\rceil C_k \tag{6}$$

Using Equation (6), a set of response times $R_{m|K}$, can be determined for increasing values of *K* from 0 to K_m, the maximum number of single bit errors that can be tolerated by message *m* without missing its deadline.

3.3. Probabilistic fault model

Broster et al. (2005) suggested that a Poisson distribution is the most appropriate way of modelling the occurrence of bit errors on CAN due to EMI. For a Poisson arrival process, the probability of *K* errors occurring in some time interval *t* is given by:

$$p(K,t) = \frac{e^{-\lambda t}(\lambda t)^K}{K!} \tag{7}$$

Broster et al. (2005) derived the following equation giving an upper bound on the probability $p(R_{m|K})$, that response time $R_{m|K}$ occurs.

$$p(R_{m|K}) = p(K, R_{m|K}) - \sum_{j=0}^{K-1} p(R_{m|j})\, p(K-j, R_{m|K} - R_{m|j}) \tag{8}$$

Essentially, Equation (8) states that the probability $p(R_{m|K})$ of response time $R_{m|K}$ occurring is given by the probability $p(K, R_{m|K})$, of exactly *K* faults occurring in a time interval of length $R_{m|K}$, less the probabilities of all such scenarios where K faults occur in the interval, but their distribution within the interval is such that the message successfully completes transmission before $R_{m|K}$ and thus has some smaller response time $R_{m|j}$ with the remaining $K-j$ faults occurring between $R_{m|j}$ and $R_{m|K}$, after the message has been successfully transmitted.

Noting that $p(R_{m|0}) = p(0, R_{m|0})$ and therefore that $p(R_{m|0})$ can be obtained directly from Equation (7). Equation (8) can be used to obtain an upper bound on the probability that each response time $R_{m|K}$ occurs, for numbers of faults from 0 to K_m. These values can then be used to compute the worst-case deadline failure probability (WCDFP).

Navet et al. (2000), and Broster et al. (2005) showed that an upper bound on the WCDFP for each message is given by:

$$p_m(fail) = 1 - \sum_{K=0..K_m} p(R_{m|K}) \tag{9}$$

where $p(R_{m|K})$ is the upper bound on the probability that response time $R_{m|K}$ occurs, given by Equation (8). Effectively Equation (9) states that the worst-case deadline failure probability is given by one minus the probabilities of all possible schedulable worst-case response times.

The WCDFP calculated according to the analysis of Navet et al. (2000), Broster et al. (2002, 2005), and Broster (2003) provides an upper bound on the probability that an invocation of a message will miss its deadline. However, a number of worst-case assumptions mean that this WCDFP may be significantly larger than the actual probability of deadline failure averaged over a large number of invocations of the message:

(i) The message is assumed to be queued at a critical instant, simultaneously with all higher priority messages.

(ii) The message is assumed to be subject to the maximum possible delay due to blocking.

(iii) All messages are assumed to have their maximum possible transmission time due to worst-case bit stuffing. Nolte et al. (2002, 2003) showed that the probability of this happening in practice is very small.

We note that the practical application of Equations (6) to (9), with typical time constraints on CAN messages and realistic error rates (for example $\lambda = 10$ bit errors/s), can easily lead to values of $K_m > 10$, and some very small probabilities ($< 10^{-20}$). As a result of the way the probabilities are composed, it is essential to use an arbitrary precision floating point arithmetic package when solving the equations, otherwise the values obtained may be incorrect by orders of magnitude due to rounding errors[5].

4. Priority assignment for CAN

We now consider the problem of priority assignment for CAN. For a commercial CAN system, it is important that messages are both schedulable and can tolerate delays on the network due to errors. The priority assignment chosen has a significant impact on the number of errors that can be tolerated by each message on the bus.

In this section, we show that the Robust Priority Assignment (RPA) algorithm introduced by Davis and Burns (2007) is applicable to CAN and then adapt it to determine robust priority orderings that are not only schedulable, but also:

(i) maximize the number of faults tolerated,

(ii) maximize the delay tolerated, or

(iii) minimise the maximum worst-case deadline failure probability for any message.

4.1. Applicability of the RPA algorithm to CAN

Davis and Burns (2007) gave four conditions that must be met for the RPA algorithm to be applicable to a fixed priority system. These four conditions are stated below as they apply to CAN messages:

Condition 1: The worst-case response time of a message is dependent on the set of higher priority messages, but not on the relative priority ordering of those messages.

Condition 2: The worst-case response time of a message may be dependent on the set of lower priority messages, but not on the relative priority ordering of those messages.

Condition 3: When the priorities of any two messages are swapped, the worst-case response time of the message being assigned a higher priority cannot increase with respect to its previous value.

Condition 4: When the priorities of any two messages are swapped, the worst-case response time of the message being assigned a lower priority cannot decrease with respect to its previous value.

It is evident from the schedulability analysis equations for CAN (Equations (4), (5) and (6) in Section 3.1) that all four conditions hold and that the RPA algorithm can therefore be used to determine robust priority orderings for CAN messages.

The RPA algorithm (Davis and Burns, 2007) also assumes additional interference in the form of a function $E(\alpha, w, m)$, where α is a scaling factor used to model variability in the amount of additional interference (for

[5] 64-bit floating point arithmetic does not provide sufficient resolution for these calculations.

example due to faults), w is the length of the time interval over which the interference occurs and m is a priority level affected by the interference. The RPA algorithm requires only that $E(\alpha, w, m)$ is a monotonic non-decreasing function of its parameters. Hence for any fixed values of α and w, $E(\alpha, w, j) \ge E(\alpha, w, k)$ if and only if priority level j has a higher numeric value (i.e. a lower priority) than k. Similarly, if time interval $w' > w''$, then $E(\alpha, w', m) \ge E(\alpha, w'', m)$ for any fixed values of α and m and finally, if the scaling factor $\alpha' > \alpha''$, then $E(\alpha', w, m) \ge E(\alpha'', w, m)$ for any fixed values of w and m.

These *monotonicity requirements* on $E(\alpha, w, m)$ are met for CAN: α is a scaling factor and so by definition, $E(\alpha, w, m)$ can be formulated to be monotonically non-decreasing in α. Interference due to errors on the bus is never less in a longer time interval than it is in a shorter one, and finally, errors affecting a high priority message also cause interference on messages at lower priority levels and so the additional interference function, $E(\alpha, w, m)$ is monotonically non-decreasing with respect to priority level.

Accounting for the additional interference $E(\alpha, w, m)$, the queuing delay and hence worst-case response time of message m can be calculated using the following recurrence relation:

$$w_m^{n+1} = \max(B_m, C_m) + E(\alpha, w, m) + \sum_{\forall k \in hp(m)} \left\lceil \frac{w_m^n + J_k + \tau_{bit}}{T_k} \right\rceil C_k \qquad (10)$$

Details of additional interference functions appropriate for CAN are given in subsequent sections; see Equation (12) and Equation (13).

4.2. Robust Priority Assignment algorithm for CAN

The RPA algorithm (Davis and Burns, 2007) is given below. The algorithm proceeds by considering each priority level in turn, lowest first. At each priority level, the algorithm determines which of the currently unassigned messages are both schedulable and can tolerate the maximum amount of additional interference (largest value of α) at that priority level, and assigns that message to the priority level.

Note, in the RPA algorithm, when each unassigned message is tried at a given priority level, it is assumed that the other unassigned messages all have higher priorities, although exactly what these priorities are is as yet unknown.

The RPA algorithm determines a schedulable priority ordering P, for any system where such an ordering exists. Further, the algorithm computes the maximum additional interference represented by α_m^P that can be tolerated by each message m under priority ordering P. The maximum additional interference that can be tolerated by the system as a whole is given by:

$$\alpha^P = \min_{\forall m}(\alpha_m^P) \qquad (11)$$

```
Robust Priority Assignment Algorithm

for each priority level m, lowest first
{
   for each unassigned message M
   {
       binary search for the largest value
       of α for which message M is
       schedulable at priority m
   }
   if no messages are schedulable at
    priority m
       return unschedulable
   else
       assign the schedulable message that
       tolerates the max α at priority m to
       priority m
}
return schedulable
```

Definition: The priority ordering P, determined by the RPA algorithm is a *robust priority ordering* in the sense that there are no systems, compliant with the system model, that are both schedulable and can tolerate additional interference characterized by a scaling factor α^Q using another priority ordering Q that are not also both schedulable and can tolerate additional interference characterized by the same or larger scaling factor $\alpha^P \ge \alpha^Q$, using the priority ordering P, generated by the RPA algorithm.

4.3. Maximising the number of faults tolerated

We now show how the RPA algorithm can be used to determine a robust priority ordering, such that messages on the network are able to tolerate the maximum number (α) of errors in the worst-case without missing their deadlines. In this case, by reference to Equation (3), the additional interference function is given by:

$$E(\alpha, w, m) = \alpha(F + \max_{j \in hep(m)}(C_j)) \qquad (12)$$

Note that here α corresponds to the number of single bit errors tolerated by the message; equivalent to the parameter K used in Equation (6), Section 3.2, and the "tolerable error threshold" described by Navet et al. (2000). F is the maximum error recovery overhead, which is 29 τ_{bit} assuming a network using only standard 11-bit identifiers.

The algorithm requires $n(n+1)/2$ binary searches to determine the robust priority ordering. Suitable starting values for each binary search are: lower limit: $\alpha = 0$, upper limit: $\alpha = D_M / F$ where D_M is the deadline of the message, and F is the maximum error recovery overhead. (Note this upper limit is guaranteed to be unschedulable as the additional interference exceeds the message deadline).

Theorem 1: The priority assignment produced by the RPA algorithm using the additional interference function given by Equation (12) maximises the number of faults that the system can tolerate without any message missing its deadline.

Proof of Theorem 1 follows the same logic employed in the proof of Theorem 2 in (Davis and Burns, 2007).

Proof: We assume (for contradiction) that there is an alternative priority ordering Q, which tolerates a larger number of faults and therefore greater additional interference than the priority ordering P found by the RPA algorithm; so $\alpha^{Q} > \alpha^{P}$. For the purposes of the proof, we will refer to this alternative priority ordering as Q_n. We will iteratively transform Q_n into $Q_{n-1} .. Q_1$, where Q_1 is the same ordering as P. The transformation will be such that $\alpha^{Q_{k-1}} \geq \alpha^{Q_k}$, thus proving the theorem via the contradiction: $\alpha^{P} \geq \alpha^{Q}$.

We use k as an iteration count and also the priority level that we will transform. Thus k counts down from an initial value of n to 1. We note that as a result of the transformations, the messages at priority levels lower than k become the same in both Q_k and P, hence Q_1 and P represent the same priority ordering.

On iteration k, we transform priority ordering Q_k as follows: First we find the priority level i in Q_k of the message assigned to priority level k in P. We refer to this message as m_k, as we intend to assign it to priority level k. Note that as the messages of lower priority than k are the same in both Q_k and P, priority level i must be either higher than or equal to k.

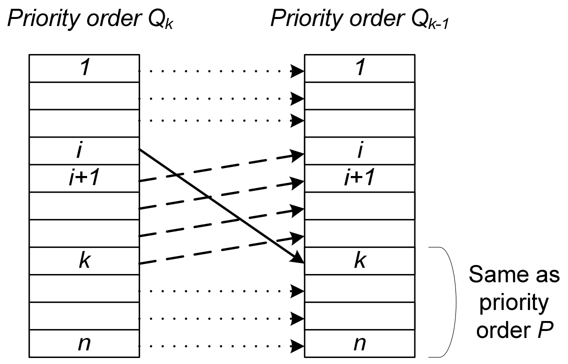

Figure 2: Transformation of priority order

There are two cases to consider:

1. Message m_k is at priority k in both P and Q_k, in which case no transformation is required on this iteration, and so Q_{k-1} is identical to Q_k.
2. Message m_k is at a higher priority i in Q_k. In this case, we form priority ordering Q_{k-1} by modifying Q_k as follows: Message m_k is moved down in priority from priority level i to priority level k, and the messages at priority levels i+1 to k are all moved up one priority level (see Figure 2).

We now introduce a concise notation to aid in the discussion of groups of messages within a priority ordering:

hep(k,P) is the set of messages with priority higher than or equal to k in priority ordering P.

hp(k,P) is the set of messages with priority strictly higher than k in priority ordering P.

lp(k,P) is the set of messages with priority strictly lower than k in priority ordering P.

Comparing the messages in priority order Q_{k-1} with their counterparts in Q_k. There are effectively four groups of messages to consider:

1. $hp(i, Q_{k-1})$: These messages are assigned the same priorities in both Q_k and Q_{k-1} and so can tolerate the same additional interference.
2. $hp(k, Q_{k-1}) \cap lep(i, Q_{k-1})$: These messages retain the same partial order but are shifted up one priority level in Q_{k-1} and so can tolerate at least as much additional interference as they can in Q_k.
3. Message m_k, which is at priority level i in Q_k and at the lower priority level k in Q_{k-1}: We know, from the RPA algorithm, that m_k can tolerate at least as much additional interference when at priority k as any of the messages in $hep(k, P)$, when they are assigned priority k. Now $lp(k, Q_k) = lp(k, P)$ implies that $hep(k, Q_k) = hep(k, P)$, and so m_k can tolerate at least as much additional interference at priority k as the message at priority k in Q_k.
4. $lp(k, Q_{k-1})$: These messages are assigned the same priorities in both Q_k and Q_{k-1}, and as $hep(k, Q_{k-1}) = hep(k, Q_k)$, they are subject to interference from the same set of higher priority messages, and so can tolerate the same additional interference in each case.

For every message in Q_{k-1}, the above analysis identifies a message in Q_k which does not have a greater tolerance to additional interference. Thus Q_{k-1} can tolerate at least as much additional interference as Q_k and so $\alpha^{Q_{k-1}} \geq \alpha^{Q_k}$.

A total of n iterations of the above procedure (for values of k from n down to 1) are sufficient to transform any arbitrary priority ordering Q into the priority ordering P, generated by the RPA algorithm. Further, this transformation is achieved without any reduction in the maximum amount of additional interference (α) that the system can tolerate. As α corresponds to the number of single-bit errors □

Example: To illustrate the operation of the RPA algorithm using the additional interference function from Equation (12), we use a simple example with the message parameters given in Table 2 below.

Table 2: Message parameters

Message	Period (ms)	Deadline (ms)	Number of bits	TX time (ms)
A	5.75	5.75	135	1.08
B	6.75	6.75	135	1.08
C	7.25	7.25	65	0.52
D	15.0	15.0	135	1.08
E	17.3	17.3	65	0.52

Recall that for each priority level, lowest first, the RPA algorithm selects the unassigned message that tolerates the maximum value of α at that priority level. For each priority level, the values of α computed by the RPA algorithm are given in Table 3.

The maximum value of α at each priority is highlighted in bold, indicating the message that is subsequently assigned to that priority level. Entries in the table marked '-' indicate that no value was computed by the algorithm, as the message had already been assigned a lower priority.

Table 3: Computed values of α (errors)

	Message				
Priority	**A**	**B**	**C**	**D**	**E**
5	0	1	0	4	**4**
4	0	1	1	**4**	-
3	1	**2**	1	-	-
2	2	-	**2**	-	-
1	**2**	-	-	-	-

The robust priority ordering for this example is (A, C, B, D, E). With this robust priority ordering, the overall tolerance to additional interference is characterised by $\alpha = 2$. In other words, all of the messages can tolerate at least two errors in the worst-case. By comparison, using "deadline minus jitter" monotonic priority ordering (D-JMPO), (A, B, C, D, E), yields values of α of (2, 2, 1, 4, 4), with message C able to tolerate only a single error in the worst case.

We note that in the above example, when considering priority levels 2 and 5, two maximum values of α are obtained. Such ties may be broken arbitrarily without affecting the overall tolerance of the system to additional interference.

At priority level 2, there is a tie between messages A and C. In this case, the alternative priority ordering of message C at priority 1 and message A at priority 2 results in message C tolerating up to 7 errors. The disparity between the two cases is caused by the difference in the length of the two messages; with the longer message magnifying the impact of transmission errors.

4.4. Maximising the delay tolerated

Whilst the previous metric can undoubtedly improve upon priority assignments achieved considering message schedulability alone, it is arguable that the susceptibility of messages to errors is dependent on the maximum delay that they can tolerate. For example, a message that can tolerate a delay of $490\,\tau_{bit}$ will be more robust to errors than a message that can tolerate a delay of just $330\,\tau_{bit}$; even though in the worst-case, both messages can only tolerate 2 errors.

We now use the RPA algorithm to determine a robust priority order, such that messages on the network are able to tolerate the maximum additional interference in terms of a number (α) of bit times delay without missing their deadlines. In this case, the appropriate additional interference function is:

$$E(\alpha, w, i) = \alpha \tau_{bit} \qquad (13)$$

In this case, suitable starting values for the binary searches are, lower limit: $\alpha = 0$, upper limit: $\alpha = D_M / \tau_{bit}$ where D_M is the deadline of the message.

Theorem 2: The priority assignment produced by the RPA algorithm using the additional interference function given by Equation (13) maximises the delay that the system can tolerate without any message missing its deadline.

Proof: Follows directly from the proof of Theorem 1, noting that in this case, α corresponds to the delay □

Example: Table 4 gives the values of α computed by the RPA algorithm for the example set of messages, assuming the additional interference function given by Equation (13).

Table 4: Computed values of α (delay)

	Message				
Priority	**A**	**B**	**C**	**D**	**E**
5	51	176	112	681	**690**
4	186	311	247	**960**	-
3	251	**376**	312	-	-
2	386	-	**447**	-	-
1	**451**	-	-	-	-

The robust priority ordering with this additional interference function is (A, C, B, D, E). With this robust priority ordering, the overall tolerance to additional interference is characterised by $\alpha = 376$. In other words, all of the messages can tolerate a delay of at least $376\,\tau_{bit}$. By comparison, using D-JMPO, (A, B, C, D, E), yields values of α of (451, 441, 312, 746, 690), with message C able to tolerate a delay of at most $312\,\tau_{bit}$.

Although this metric could be viewed as an improvement over considering the number of errors tolerated, it has a number of drawbacks: Firstly, the delay tolerated by a message does not necessarily reflect the number of errors which it can tolerate. For example, if message C (with 1 byte of data) was assigned the highest priority, then it could tolerate 5 errors, whereas message A (with 8 bytes of data) can only tolerate 2 errors when it is at the highest priority. Secondly, the probability of a message missing its deadline depends upon a number of other factors besides the delay that it can tolerate. These factors include: the delay an error can cause for a particular message and the time interval over which errors may impact the message. These factors are addressed in the next section.

4.5. Minimising the probability of deadline failure

We now modify the RPA algorithm for the purpose of determining a robust priority ordering such that message transmission has the smallest probability of deadline failure.

The Probabilistic RPA algorithm, given below, computes the worst-case deadline failure probability (WCDFP) for each schedulable but unassigned message using the analysis of Broster et al (2002, 2005), and Broster (2003) given by Equations (8) and (9). It then assigns the schedulable message with the lowest WCDFP to that priority level.

```
Probabilistic Robust Priority Assignment
Algorithm

for each priority level m, lowest first
{
   for each unassigned message M
   {
       Compute the WCDFP of message M at
       priority m
   }
   if no messages are schedulable at
   priority m
       return unschedulable
   else
       assign the schedulable message with
       the smallest WCDFP at priority m to
       priority m
}
return schedulable
```

Recall that in Equation (9), we use the notation $p_m(fail)$ to mean the WCDFP for the message at priority *m*. Similarly, let $p(fail)$ be the WCDFP for *any* message. Hence:

$$p(fail) = \max_{\forall m}(p_m(fail)) \quad (14)$$

Theorem 3: If a schedulable priority ordering exists, then the Probabilistic RPA algorithm generates a priority ordering that has the minimum WCDFP for *any* message ($p(fail)$) of any feasible priority ordering.

Proof of Theorem 3 follows the logic employed in the proof of Theorem 1. We note the following points: Shifting a message up in priority cannot:

(i) decrease the maximum delay that the message can tolerate,

(ii) increase the worst-case delay to the message caused by an error, nor

(iii) increase the worst-case response time of the message for a given delay that was previously tolerated.

Hence shifting a message up in priority cannot increase the WCDFP of the message.

Proof: We assume (for contradiction) that there is an alternative priority ordering *Q*, which has a smaller WCDFP for *any* message ($p^Q(fail)$) than the priority ordering *P* found by the Probabilistic RPA algorithm; so $p^Q(fail) > p^P(fail)$. For the purposes of the proof, we will refer to this alternative priority ordering as Q_n. We will iteratively transform Q_n into $Q_{n-1} .. Q_1$, where Q_1 is the same ordering as *P*. The transformation will be such that $p^{Q_{k-1}}(fail) \geq p^{Q_k}(fail)$, thus proving the theorem via the contradiction: $p^P(fail) \geq p^Q(fail)$.

We use *k* as an iteration count and also the priority level that we will transform. Thus *k* counts down from an initial value of *n* to 1. We note that as a result of the transformations, the messages at priority levels lower than *k* become the same in both Q_k and *P*, hence Q_1 and *P* represent the same priority ordering.

On iteration *k*, we transform priority ordering Q_k as follows: First we find the priority level *i* in Q_k of the message assigned to priority level *k* in *P*. We refer to this message as m_k, as we intend to assign it to priority level *k*. Note that as the messages of lower priority than *k* are the same in both Q_k and *P,* priority level *i* must be either higher than or equal to *k*.

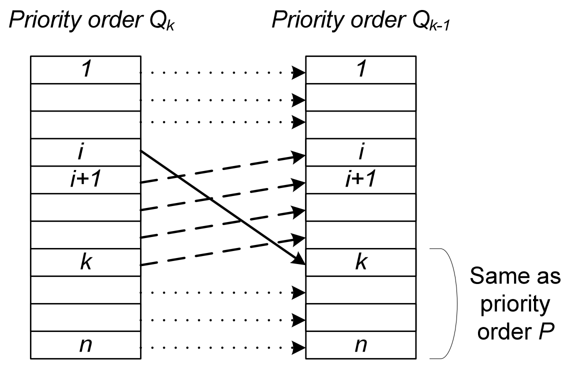

Figure 3: Transformation of priority order

There are two cases to consider:

1. Message m_k is at priority *k* in both *P* and Q_k, in which case no transformation is required on this iteration, and so Q_{k-1} is identical to Q_k.
2. Message m_k is at a higher priority *i* in Q_k. In this case, we form priority ordering Q_{k-1} by modifying Q_k as follows: Message m_k is moved down in priority from priority level *i* to priority level *k*, and the messages at priority levels $i+1$ to *k* are all moved up one priority level (see Figure 3).

Comparing the messages in priority order Q_{k-1} with their counterparts in Q_k. There are effectively four groups of messages to consider:

1. $hp(i,Q_{k-1})$: These messages are assigned the same priorities in both Q_k and Q_{k-1} and so have the same WCDFPs.
2. $hp(k,Q_{k-1}) \cap lep(i,Q_{k-1})$: These messages retain the same partial order but are shifted up one priority level in Q_{k-1} and so have WCDFPs that are no greater than they are in Q_k.
3. Message m_k, which is at priority level *i* in Q_k and at the lower priority level *k* in Q_{k-1}: We know, from the Probabilistic RPA algorithm, that m_k at priority *k*, has a WCDFP no greater than the WCDFP of any of the messages in $hep(k,P)$, when they are assigned priority *k*. Now $lp(k,Q_k) = lp(k,P)$ implies that $hep(k,Q_k) = hep(k,P)$, and so m_k at priority *k*, has a WCDFP no greater than the WCDFP of the message at priority *k* in Q_k.
4. $lp(k,Q_{k-1})$: These messages are assigned the same priorities in both Q_k and Q_{k-1}, and as $hep(k,Q_{k-1}) = hep(k,Q_k)$, they are subject to interference from the same set of higher priority messages, and so have the same WCDFPs in each case.

For every message in Q_{k-1}, the above analysis identifies a message in Q_k which has a WCDFP that is the same or greater. Thus Q_{k-1} has an overall WCDFP for *any* message that is no greater than that of Q_k, i.e. $p^{Q_{k-1}}(fail) \geq p^{Q_k}(fail)$.

A total of *n* iterations of the above procedure (for values of *k* from *n* down to 1) are sufficient to transform any arbitrary priority ordering *Q* into the priority ordering *P*, generated by the Probabilistic RPA algorithm. Further, this transformation is achieved without any increase in the minimum WCDFP for *any* message ($p(fail)$), given by Equation (14) □

Example: We now apply the Probabilistic RPA algorithm to the example message set given in Table 2. Here, we assume that faults occur according to a Poisson arrival process with $\lambda = 10$ faults/s approximately equivalent to an error rate of 10^{-4} errors/bit for a 125Kbit/s bus.

Table 5 gives the maximum number of faults tolerated, the response time (in ms) given that number of faults and the WCDFP for each message, as computed by the Probabilistic RPA algorithm. For example, at priority 5, message A can tolerate no errors, has a worst-case response time of 5.366ms, and a WCDFP of 5.20x10^{-2} (given, in this case, by the probability of no faults occurring in 5.366ms). Entries in the table marked '-' indicate that no value was computed by the algorithm, as the message had already been assigned a lower priority.

Table 5: Computed values of α, $R_{m|\alpha}$, WCDFP

	Message				
Pri	**A**	**B**	**C**	**D**	**E**
5	0 5.336 5.20x10^{-2}	1 6.648 2.03x10^{-3}	0 5.336 5.20x10^{-2}	**4** **14.344** **2.88x10^{-7}**	4 17.024 4.90x10^{-7}
4	1 5.568 1.41x10^{-3}	1 5.568 1.41x10^{-3}	1 5.568 1.41x10^{-3}	-	**5** **16.176** **9.83x10^{-9}**
3	1 5.048 1.15x10^{-3}	**2** **6.360** **3.50x10^{-5}**	1 5.048 1.15x10^{-3}	-	-
2	2 5.280 1.85x10^{-5}	-	**2** **5.280** **1.85x10^{-5}**	-	-
1	**2** **4.760** **1.27x10^{-5}**	-	-	-	-

We note that in the above example, when considering priority level 5, although both message E and message D could tolerate 4 errors, the fact that the errors would need to occur during a shorter response time of 14.344ms for message D compared to 17.024ms for message E, mean that message D has a lower probability of failure and is therefore assigned priority 5.

The robust priority ordering found for this example, was (A, C, B, E, D). With this priority ordering, the maximum WCDFP of any message is 3.5x10^{-5}. By comparison, D-JMPO (A, B, C, D, E) yields a maximum WCDFP of 1.15x10^{-3}. Stated otherwise, there is an upper bound on the probability of failure of any given message transmission of approximately 1 in 28,500 when priorities are assigned according to the Probabilistic RPA algorithm, and 1 in 870 with D-JMPO - a factor of over 30 worse.

Figure 4 plots the maximum WCDFP for the example set of messages, against the 120 different permutations of priority assignment, in lexicographical (dictionary) order: (A,B,C,D,E), (A,B,C,E,D), (A,B,D,C,E), (A,B,D,E,C), (A,B,E,C,D), (A,B,E,D,C), (A,C,B,D,E), (A,C,B,E,D) … Of these orderings, D-JMPO is the 1st one generated and the robust priority ordering determined by the Probabilistic RPA algorithm is the 8th generated. The 120 priority orderings can be classified as follows:

- 54 priorities orderings result in a high maximum WCDFP of approximately 0.05, corresponding to a failure rate of 1in 20.
- 62 priority orderings, including D-JMPO, result in a maximum WCDFP in the range 0.002 to 0.001, corresponding to failure rates between 1 in 500 and 1 in 1000.

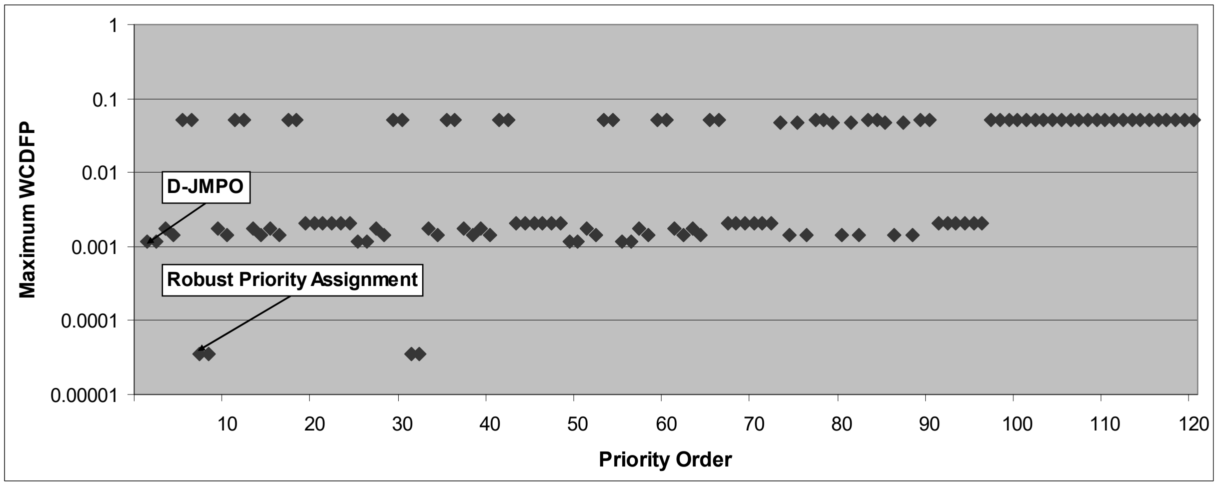

Figure 4: Maximum WCDFP for different priority orderings

- Just 4 priority orderings are robust. The maximum WCDFP for these priority orderings is 3.5×10^{-5}, corresponding to a failure rate of 1 in 28,500.

This example shows how robust priority ordering of CAN messages can significantly reduce the worst-case deadline failure probability. Further it serves to illustrate that "deadline minus jitter" monotonic priority assignment, commonly used to assign priorities to CAN messages, is not necessarily the most robust priority ordering to use.

We note that "deadline minus jitter" monotonic priority ordering (D-JMPO) is a robust priority ordering for systems where D-JMPO is the optimal priority ordering when additional interference is not considered (Davis and Burns, 2007). However, D-JMPO is not the optimal priority ordering for CAN (as shown by Davis et al. (2007) and the Appendix) due to the non-pre-emptive nature of message transmission.

4.6. Experimental investigation

In this section, we investigate how often the Probabilistic RPA algorithm improves upon D-JMPO in terms of the maximum WCDFP for a set of CAN messages.

To explore this question, we generated random sets of messages. Each message set contained 8 messages, each with 1 to 8 bytes of data. The number of data bytes was chosen from a uniform random distribution. Message periods were chosen from the range 2.5ms to 20ms, in steps of 0.25ms, again according to a uniform random distribution. Message deadlines were set equal to their periods and message jitter set to zero. In each case, a 125 Kbit/s network, standard 11-bit identifiers, and 8 data-byte background messages were assumed.

In calculating WCDFPs, faults were assumed to occur according to a Poisson arrival process with $\lambda = 10$ faults/s. We determined the schedulability and maximum WCDFP for the message sets using D-JMPO and compared the results to those obtained using the Probabilistic RPA algorithm.

In total, 10,000 message sets were generated; 1,000 for each of ten 5% utilisation bands, starting at 50-55% and finishing at 95-100%.

Figure 5 plots, for each utilisation level, the percentage of message sets that were schedulable according to each priority assignment policy, along with the percentage of message sets where the maximum WCDFP differed depending on the priority assignment policy used.

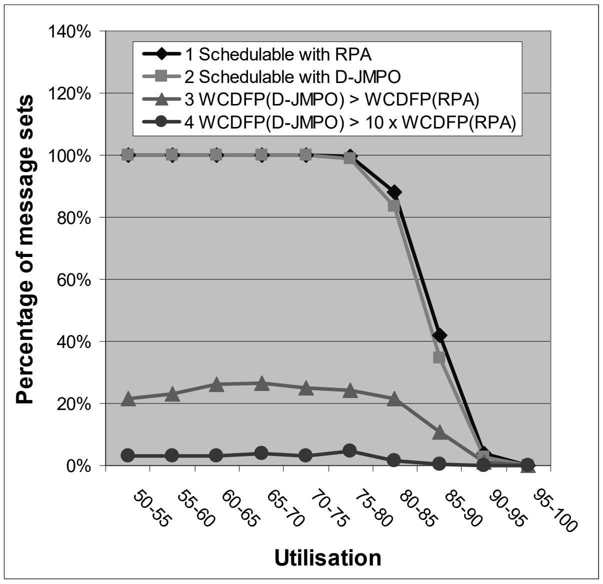

Figure 5:

Line 3 in Figure 5 shows that typically 20-30% of schedulable message sets at each utilisation level have a maximum WCDFP that is strictly smaller with priorities assigned according to the Probabilistic RPA algorithm, than with D-JMPO.

Similarly, line 4 shows that typically 2-5% of schedulable message sets at each utilisation level have a maximum WCDFP that is more than an order of magnitude smaller with priorities assigned according to the Probabilistic RPA algorithm, than with D-JMPO.

The overall results of this experimental investigation can be summarised as follows:

- All of the message sets with utilisation below 70% were schedulable with both priority assignment policies.
- None of the 1,000 message sets with utilisation exceeding 95% were schedulable with either of the priority assignment policies. In total, 27% (2666 out of 10,000) of the message sets generated were not schedulable.
- 6% (121 out of 2,000) of the messages sets with utilisation in the range 80-90% were schedulable with priorities assigned according to the Probabilistic RPA algorithm, but unschedulable with D-JMPO.
- 25% (1802 out of 7,334) of the schedulable message sets had a lower maximum WCDFP with priorities assigned according to the Probabilistic RPA algorithm, than with D-JMPO.
- 3% (223 out of 7,334) of the schedulable message sets had a maximum WCDFP more than an order of magnitude smaller with priorities assigned according to the Probabilistic RPA algorithm, than with D-JMPO.

Overall, this investigation showed that although D-JMPO cannot be guaranteed to provide an optimal solution, it is a reasonable heuristic, which gives the most robust solution approximately 70-80% of the time.

For a significant number of cases, 25% of the *schedulable* message sets in our randomised test; D-JMPO is not the most robust priority ordering. In around 3% of schedulable cases, D-JMPO gave a maximum WCDFP at least an order of magnitude higher than necessary, and for approximately 6% of schedulable message sets with utilisations between 80-90%, D-JMPO did not result in a schedulable system.

Finally, recall that both deadline monotonic priority ordering (DMPO) and rate monotonic priority ordering (RMPO) are special cases of D-JMPO. As the CAN messages in this experimental investigation had zero jitter, and deadlines equal to their periods, then the above results apply equally to rate monotonic and deadline monotonic priority orderings, neither of which are guaranteed to be optimal or robust priority orderings for CAN.

5. Summary and conclusions

In this paper, we showed how the Robust Priority Assignment algorithm could be adapted to the problem of providing a robust priority ordering for CAN messages that are subject to errors on the bus caused by Electromagnetic Interference.

5.1. Contribution

The major contributions of this work are as follows:

- Adapting the Robust Priority Assignment algorithm to the problem of assigning priorities to CAN messages. We showed how the RPA algorithm could be used to provide priority orderings that:
  - (i) maximize the number of faults tolerated,
  - (ii) maximize the delay tolerated,
  - (iii) minimise the probability of message deadline failure (Equation (14)).

  Of the three, we recommend (iii) as providing the most suitable approach to the problem of assigning priorities to CAN messages.
- Proving that the Probabilistic RPA algorithm provides the most robust priority ordering possible in the sense that of all feasible priority orderings, it is the one with the minimum probability of message deadline failure (Equation (14)).
- Showing that "deadline minus jitter" monotonic priority assignment, considered for many years to be optimal for CAN (Tindell and Burns, 1994; Tindell et al., 1994; Tindell et al., 1995), and widely used to assign priorities to CAN messages in commercial systems, does not necessarily result in the most robust priority ordering, nor does it always result in a schedulable priority ordering when one exists.
- Showing that assigning priorities according to the Probabilistic RPA algorithm reduced the maximum probability of deadline failure in around 25% of cases, as compared to "deadline minus jitter" monotonic priority assignment.

5.2. Conclusion

The Probabilistic RPA algorithm ensures that the configuration of message identifiers chosen is robust to errors on the bus in a way that previous approaches using "deadline minus jitter" monotonic priority assignment suggested by Tindell and Burns (1994), and Tindell et al. (1994, 1995), or Audsley's optimal priority assignment algorithm suggested by Davis et al. (2007), do not. Thus the Probabilistic Robust Priority Assignment algorithm is useful to engineers wishing to configure commercial CAN systems in a way that makes them as robust as possible to errors on the bus, without the need to switch to a higher baud rate, or to compromise functionality by omitting some messages or reducing their transmission rates.

5.3. Extensions and Future work

In many systems, it is possible to classify CAN messages according to their importance as well as their deadlines, for example, in an automotive application, messages containing data to switch the brakes lights on/off, raise/lower the windows might be considered more important than messages containing data that adjusts the volume/tuning of the radio. In this case, messages could be grouped according to importance and the Robust Priority Assignment algorithm modified so that at each priority level it selects, from the lowest importance group containing a schedulable message, the message with the minimum WCDFP. This would have the effect of minimising the overall WCDFP for the most important group of messages.

The problem of bi-criteria or multi-criteria optimisation, for example, minimising the maximum WCDFP for the most important group of messages, then minimising the maximum WCDFP of the next most important group of messages and so on, is beyond the scope of this paper, however it is an interesting area for future research.

5.4. Acknowledgements

This work was funded in part by the EU Frescor project.

Appendix

George et al. (1996) state the following theorem for non-pre-emptive fixed priority scheduling (assuming no jitter):

*"**Theorem 16**:*

Deadline monotonic priority order is an optimal priority assignment for periodic or sporadic task sets with $D_i \le T_i$ and $D_i \ge D_j \Rightarrow C_i \ge C_j$".

In many commercial CAN systems, all messages carry the maximum 8 bytes of data as a way of ameliorating the large overheads of the control fields and 15-bit CRC. If correct, the above theorem would indicate that DMPO (or more likely D-JMPO[6]) was the optimal priority assignment policy to use in such systems.

The proof given by George et al. (1996), assumes that *"as $\forall i$, $D_i \le T_i$ the worst-case response time of any task is found in its first instance"*; however this assumption was shown to be false by Davis et al. (2007). They noted that *"the proof is undermined; however the theorem may or may not still be true"*.

The following counter-example resolves this question, showing that Theorem 16 in (George et al., 1996) is in fact false. Hence DMPO is not optimal for fixed priority non-pre-emptive systems with $D_i \le T_i$ and $D_i \ge D_j \Rightarrow C_i \ge C_j$. In fact, the counter-example also shows that it is also not optimal for the interesting case where $\forall i, j \quad C_i = C_j$.

[6] If correct, the proof given in (George et al., 1996) could be extended to the case where messages have non-zero jitter.

The parameters of the counter-example message set are given in Table 6 below. The total bus utilisation is 93.6%.

Table 6: Message parameters

Message	Period (ms)	Deadline (ms)	Number of bits	TX time (ms)
A	2.5	2.5	125	1.0
B	4.0	3.0	125	1.0
C	3.5	3.25	125	1.0

Assuming DMPO, the worst-case response times of the three messages (computed via the exact analysis given by Davis et al. (2007)) are R_A=2.0ms, R_B=3.0ms, and R_C=3.5ms. The worst-case response time of message C occurs for the second invocation of the message within the busy period, as shown in Figure 6. The response time for this invocation is 3.5ms, exceeding its deadline of 3.25ms.

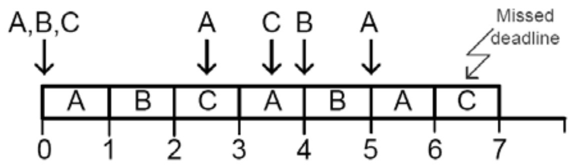

Figure 6: DMPO unschedulable

With the alternative priority ordering; A, C, B, all three messages can meet their deadlines in the worst-case. The busy period for the lowest priority message (now B) is shown in Figure 7.

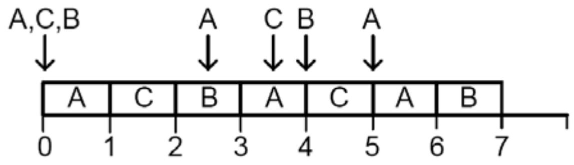

Figure 7: Alternative priority ordering: schedulable

The worst-case response times of the three messages, in this case are: R_A=2.0ms, R_B=3.0ms, and R_C=3.0ms.

This set of messages is unschedulable with DMPO, but schedulable with an alternative priority assignment, thus proving that Theorem 16 from (George et al., 1996) is *false*.

Summary

The simple counter-example given in this appendix serves to illustrate that DMPO/D-JMPO is not optimal for even the simplest of CAN systems (three messages, with deadlines less than or equal to their periods, zero jitter, and the same transmission times). When configuring commercial CAN systems, it is important to use an appropriate method of assigning priorities to CAN messages. We recommend the use of the Robust Priority Assignment algorithm described in the main body of this paper.

6. References

N.C. Audsley, "Optimal Priority Assignment and Feasibility of Static Priority Tasks with Arbitrary Start Times", *Technical Report YCS 164*, Dept. Computer Science, University of York, UK, December 1991.

N.C. Audsley, "Optimal Priority Assignment in Fixed Priority Scheduling". *Information Processing Letters* Vol. 79, No. 1, pp39-44, 2001.

T.P. Baker. "Stack-based Scheduling of Real-Time Processes." *Real-Time Systems Journal* (3)1, pp. 67-100, 1991.

K. Bletsas, and N.C. Audsley, "Optimal Priority Assignment in the Presence of Blocking". *Information Processing Letters* Vol. 99, No. 3, pp83-86, August. 2006.

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

I. Broster, A. Burns , G. Rodríguez-Navas, "Probabilistic Analysis of CAN with Faults", *In Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS'02)*, pp. 269-278, December, 2002

I. Broster. "Flexibility in Dependable Communication". *PhD Thesis*, Department of Computer Science, University of York, UK, August 2003.

I. Broster, A. Burns and G. Rodriguez-Navas, "Timing Analysis of Real-time Communication under Electromagnetic Interference", *Real-Time Systems*, 30(1-2) pp. 55-81, May 2005.

CiA "Special Edition Automotive" *CiA CAN newsletter,* 2006.

R.I. Davis, A. Burns, R.J. Bril, and J.J. Lukkien. "Controller Area Network (CAN) Schedulability Analysis: Refuted, Revisited and Revised". *Real-Time Systems*, Volume 35, Number 3, pp 239-272. April 2007.

R.I. Davis, A. Burns,. "Robust Priority Assignment in Fixed Priority Real-time Systems". *IEEE Real-Time Systems Symposium,* 2007 (to appear).

J. Ferreira, A. Oliveira, P. Fonseca, J. A. Fonseca. "An Experiment to Assess Bit Error Rate in CAN" In *proceedings 3rd International workshop on real-time Networks RTN 2004*. June 2004.

L. George, N. Rivierre, and M. Spuri. "Pre-emptive and Non-pre-emptive Real-time Uniprocessor Scheduling. *Technical Report 2966*, Institut National de Recherche et Informatique et en Automatique (INRIA), France, September 1996

D.I. Katcher, H. Arakawa, J.K. Strosnider, "Engineering and Analysis of Fixed Priority Schedulers". *IEEE Transactions on Software Engineering*, 19(9):920–934, September 1993.

IEE 2000. "EMC and Functional Safety" IEE Guidance Document, IEE. Available from http://www.theiet.org/publicaffairs/electro/index.cfm

P.B. Ladkin. "Electromagnetic Interference with Aircraft Systems: Why Worry?" Technical Report RVS-J-97-03, University of Bielfield, Faculty of Technology. October 1997.

J.P. Lehoczky, L. Sha, Y. Ding, "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behaviour". In *Proceedings of the 10th IEEE Real-Time Systems Symposium*, pp. 166–171, Santa Monica, CA, December 1989.

C. L. Liu and J. W. Layland. "Scheduling Algorithms for Multiprogramming in a Hard Real-time Environment", *Journal of the ACM*, 20(1): 46-61, January 1973.

J. Y.-T. Leung and J. Whitehead, "On the Complexity of Fixed-priority Scheduling of Periodic Real-time Tasks," *Performance Evaluation*, 2(4): 237-250, December 1982.

N. Navet, Y-Q. Song, and F. Simonot. "Worst-case Deadline Failure Probability in Real-time Applications distributed over controller area network". *Journal of Systems Architecture* Volume 46 Number 1. pp. 607–617. 2000.

T. Nolte, H. Hansson, and C. Norstrom. "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, September 2002.

T. Nolte, H. Hansson, and C. Norstrom, "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, May 2003.

S. Punnekkat, R. Davis, A. Burns, "Sensitivity Analysis of Real-time Task Sets". In *Proceedings of the Asian Computing Science Conference*, pp72–82, Nepal, December 1997.

S. Punnekkat, H. Hansson, C. Norstrom. "Response Time Analysis under Errors for CAN". In *Proceedings 6th Real-Time Technology and Applications Symposium*, pp. 258-265, IEEE Computer Society Press, May/June 2000.

J. Regehr, "Scheduling Tasks with Mixed Pre-emption Relations for Robustness to Timing Faults". In *proceedings 23th IEEE Real-Time Systems Symposium*, pp. 315–326, IEEE Computer Society Press, December 2002.

J. Rufino and P. Verissimo. "A Study on the Inaccessibility Characteristics of Contoller Area Network" In *Proceedings of 2nd International CAN Conference*, October 1995.

J. Rufino, P. Verissimo, G. Arroz, C. Almeida, and L. Rodrigues. "Fault-tolerant Broadcasts in CAN". In *Digest of Papers, The 28th IEEE International Symposium on Fault-Tolerant Computing (FTCS'98)*. pp. 150-159, June 1998.

J. Rufino "Computational System for Real-Time Distributed Control". *PhD-Thesis*, Technical University of Lisbon, Instituto Superior, July 2002.

O. Serlin, "Scheduling of Time Critical Processes". In *proceedings AFIPS Spring Computing Conference*, pp 925-932, 1972.

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

Society of Automotive Engineers. "Class C Application Requirement Considerations, Recommended Practice", *SAE Technical Report J2056/1*. June 1993.

K.W. Tindell and A. Burns. "Guaranteeing Message Latencies on Controller Area Network (CAN)", In *Proceedings of 1st International CAN Conference*, pp. 1-11, September 1994.

K.W. Tindell, H. Hansson, and A.J. Wellings. "Analysing Real-time Communications: Controller Area Network (CAN)". In *Proceedings 15th Real-Time Systems Symposium (RTSS'94)*, pp. 259-263. IEEE Computer Society Press, December 1994.

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

A. Zuhily and A. Burns "Optimality of (D-J)-Monotonic Priority Assignment". *Information Processing Letters.* Volume 103, Number 6, pp. 247-250, April 2007.

Biographies

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 Senior Research Fellow in 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.

Professor **Alan Burns** co-leads 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 400 papers and 15 books, with a large proportion of them concentrating on real-time systems and programming languages. 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 high integrity real-time programs and their timing properties.