IEEE Real-Time Embedded Systems Workshop, Dec. 3, 2001

Using bit-stuffing distributions in CAN analysis

Thomas Nolte, *Student member*, *IEEE*, Hans Hansson, *Member*, *IEEE*, Christer Norström, *Member*, *IEEE*, Sasikumar Punnekkat, *Member*, *IEEE*

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

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

Index Terms—controller area network, CAN, bit-stuffing, reliability analysis, modelling

1 INTRODUCTION

During the last decade real-time researchers have extended schedulability analysis to a mature technique which for non-trivial systems can be used to determine whether a set of tasks executing on a single CPU or in a distributed system will meet their deadlines or not [1][2][9] [13]. The essence of this analysis is to investigate if deadlines are met in a worst-case scenario. Whether this worst case actually will occur during execution, or if it is likely to occur, is not normally considered.

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

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

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

In our previous work [8], we proposed a model for calculating worst-case latencies of Controller Area Network (CAN) [7] frames (messages) under error assumptions. This model is pessimistic, in the sense that there are systems that the analysis determines unschedulable, even though deadlines will only be missed in extremely rare situations with pathological combinations of errors.

In [5][6] we have reduced the level of pessimism by introducing a better fault model, and in [4] we also consider variable phasings between message queuings, in order to make the model more realistic.

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

1. Bit-stuffing distributions based on assuming independent bit-values of the data before encoding, i.e., equal probability of a bit having value 1 or 0. With this information we create a model for making assumptions about the number of stuffed bits in a packet of data.
2. Bit-stuffing distributions extracted from real CAN-bus traffic.

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

The outline of the article is as follows. Section 2 specifically discusses the scheduling of frame sets in Controller Area Networks under a general fault model. Further, it describes the theory behind bit-stuffing and we present the effects of bit-stuffing distributions along with our model. In section 3 we investigate some real CAN traffic and in Section 4 we give a proposal of how to align the real traffic to our model. Finally, Section 6 presents our conclusions and future work.

Thomas Nolte, Hans Hansson, and Christer Norström are with the Department of Computer Engineering, Mälardalen University, Västerås, Sweden. Email: {thomas.nolte, hans.hansson, christer.norstrom}@mdh.se. Sasikumar Punnekkat is from Vikram Sarabhai Space Centre, Trivandrum, India

2 Traditional Schedulability Analysis of CAN Frames

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

CAN is a collision-detect broadcast bus, which uses deterministic collision resolution to control access to the bus. The basis for the access mechanism is the electrical characteristics of a CAN bus: if multiple stations are transmitting concurrently and one station transmits a '0' then all stations monitoring the bus will see a '0'. Conversely, only if all stations transmit a '1' will all processors monitoring the bus see a '1'. During arbitration, competing stations are simultaneously putting their identifiers, one bit at the time, on the bus. By monitoring the resulting bus value, a station detects if there is a competing higher priority frame and stops transmission if this is the case. Because identifiers are unique within the system, a station transmitting the last bit of the identifier without detecting a higher priority frame must be transmitting the highest priority queued frame, and hence can start

2.1 *Classical CAN bus analysis*

Tindell et al. [10] [11] [12] present analysis to calculate the worst-case latencies of CAN frames. This analysis is based on the standard fixed priority response time analysis for CPU scheduling [1].

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

$$R_i = J_i + q_i + C_i \tag{1}$$

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

$$q_i = B_i + \sum_{j \in hp(i)} \left\lceil \frac{q_i + J_j + \tau_{bit}}{T_j} \right\rceil C_j + E(q_i + C_i) \tag{2}$$

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

2.2 *Effects of Bit-stuffing, worst case*

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

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

$$8s + 47 \tag{3}$$

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

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

Intuitively the above formula captures the number of stuffed bits in the worst case scenario, shown in Figure 1.

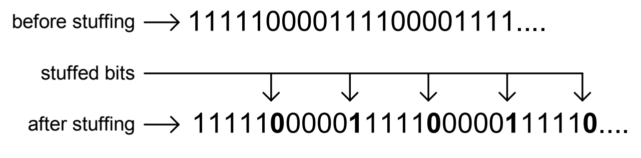

Figure 1: The worst case scenario when stuffing bits.

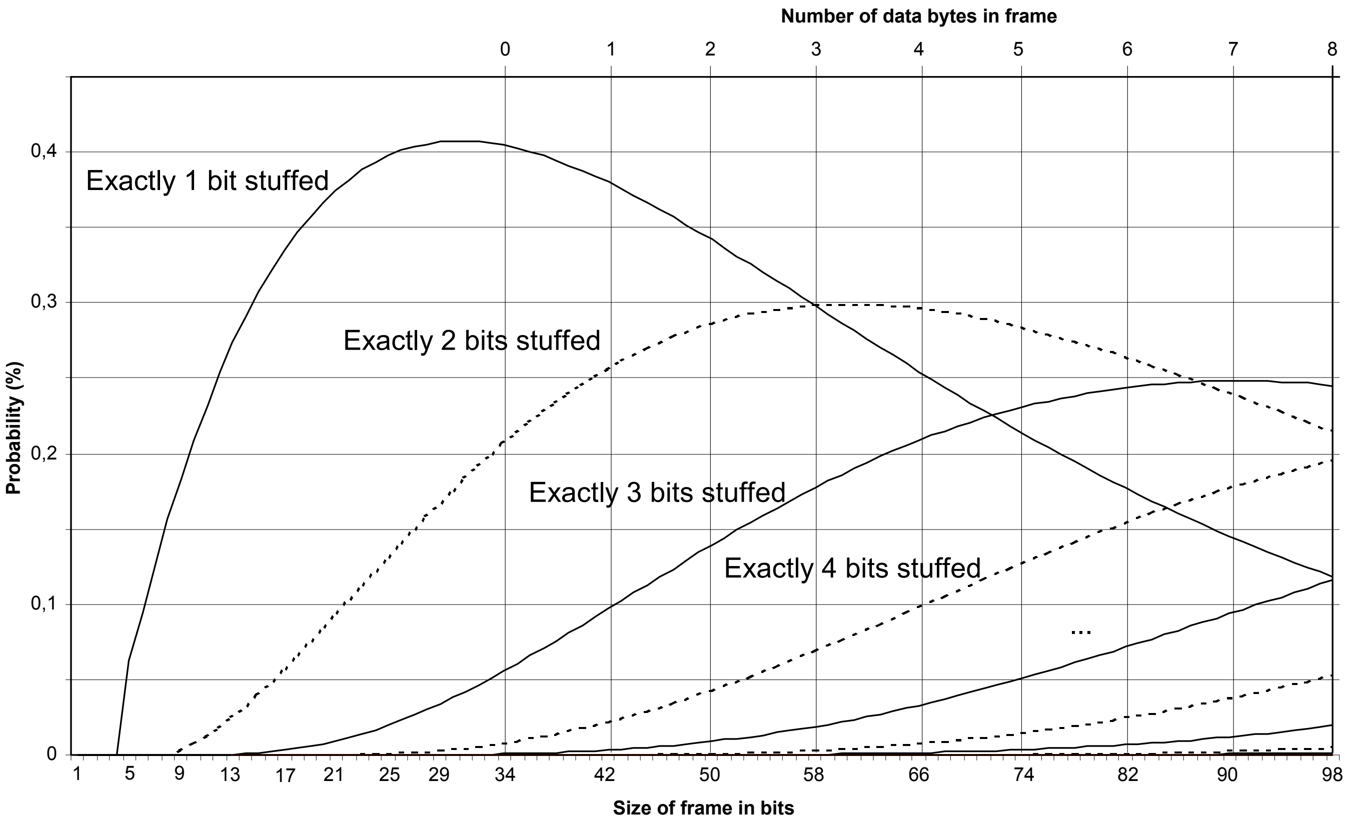

Figure 2: Probability of a specific number of stuffed bits in a frame, assuming our probabilistic frame model. The 9 lengths are marked as vertical lines.

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

$$C_i = \left(8s_i + 47 + \left\lfloor \frac{34 + 8s_i - 1}{4} \right\rfloor \right) \tau_{bit} \qquad (5)$$

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

2.3 *Independent bit-stuffing model*

If we look into how bit stuffing actually transforms the data instead of using the worst-case method as presented above, we will get a very different result. The length of a frame, before bit-stuffing, can be at most 111 bits (8 bytes data and 47 control bits), and among them 98 bits are exposed to bit stuffing. By assuming equal probability of bit-value 1 and 0 among the bits and no dependency among bits, we can calculate the actual probabilities of having a certain frame length after bit-stuffing. These probabilities are for different frame sizes (number of bits) shown in Figure 2. The graph is the result of an exhaustive analysis of all possible frame patterns. The nine different frame lengths (0-8 bytes of data) are visualised in the graph as vertical lines. Note that only the first 8 cases of stuffed bits (1-8 bits stuffed in the frame) are visible in the graph, since the probability of getting more than 8 bits stuffed is very low. For example, the probability of getting exactly 10, 15 and 20 bits stuffed never exceeds 10^{-4}, 10^{-9}, and 10^{-17}, respectively.

3 CASE STUDY: REAL CAN TRAFFIC

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

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

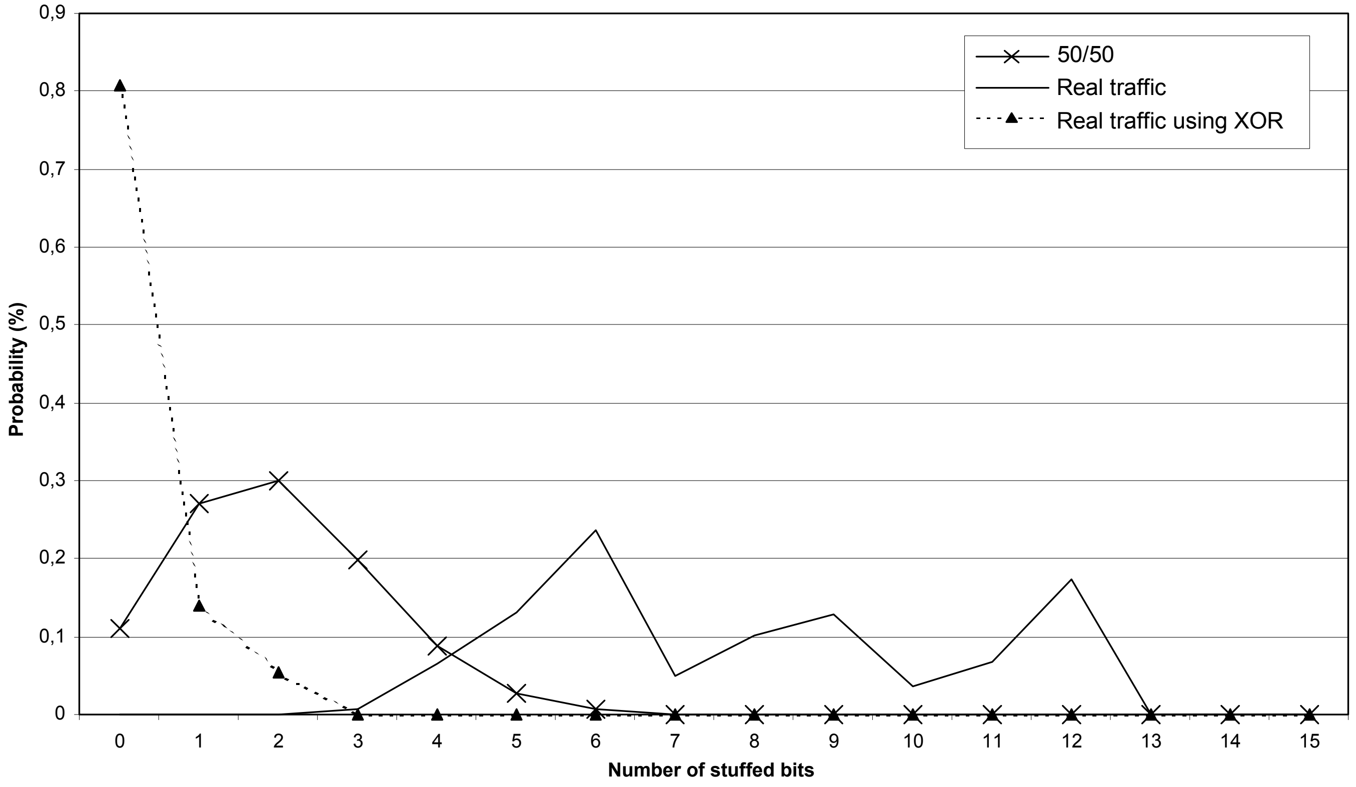

Figure 3: Probability density functions, PDF:s, showing the number of stuffed bits in a 64 bit frame. We show here our independent 50/50 model, the real CAN traffic and the manipulated real CAN traffic.

'1', we will send "0000000000000001", i.e., 15 consecutive 0:s.

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

In our investigation of real CAN traffic, we considered some 25 000 frames. Due to the format of the obtained data, we investigated only the data part of the frames, which in this case were 8 bytes for all frames. The rest of the CAN frames (control fields and so on) were not considered. The obtained distribution of stuff-bits is shown in Figure 3 ("Real traffic"). Worth noticing is that the actual worst case is here 13 bits, to be compared with the worst-case result of 15 stuffed bits when applying traditional analysis for a frame size of 64 bits. The figure also shows the distribution obtained with our 50/50 model ("50/50"), as well as the distribution obtained for the real-traffic when applying the coding ("Real traffic using XOR") that we will present next.

4 A SIMPLE CODING SCHEME TO REDUCE BIT-STUFFING

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

For example, we can use a simple coding scheme in which the original frame is XORed with a particular bit-pattern, the bit mask. XOR is a logical operation performed in a single operation by most CPUs. In our case we use the bit-pattern 101010101010... in order to kill sequences of 1:s and 0:s. On the receiving side, the same XOR operation is performed, with the same bit mask, to decode the data. Figure 4 illustrates the encoding/decoding process.

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

We have applied the simple XOR-coding to our 25 000 automotive CAN frames. The result is presented in Figure 3 ("Real traffic using XOR"). Here we compare the number of bits stuffed into a frame of size 64 bits, i.e., 8 bytes. Our 50/50 independent model give us quite good results, since we will seldom (probability in the order of 10^{-5}) have frames extended with more than 8 bits, i.e., 46% smaller than the traditional worst case figure. For the frames obtained after the XOR transformation we did not find any frame with more than 3 extra bits, i.e., 80% smaller than the

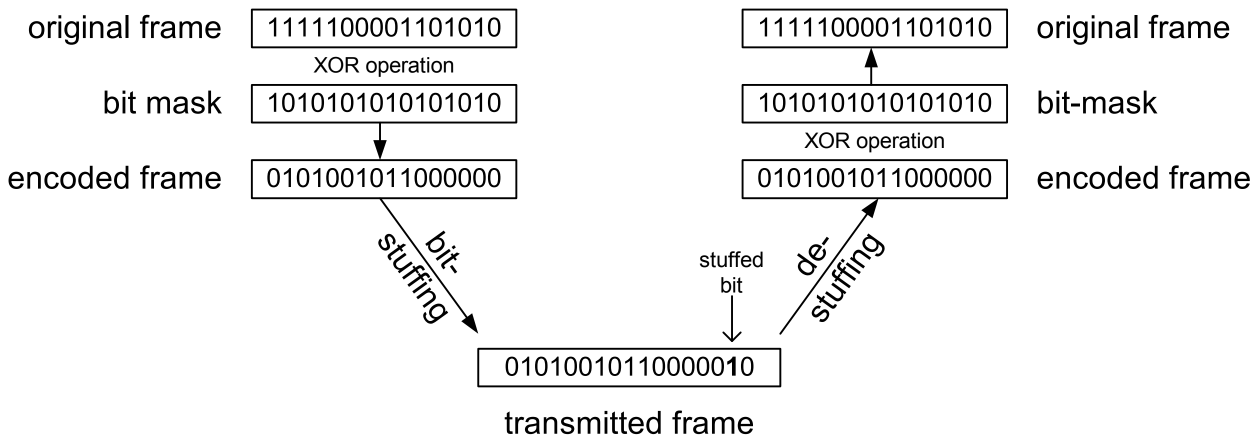

Figure 4: Encoding/decoding process for the proposed method.

worst case. Compared to the original real traffic, we will now transmit one byte less. (All of this should of course be compared with the worst-case analysis result of 15 bits.) It should be noted that with the XOR we now have even better performance than our previously suggested 50/50 model. The reason is that our real CAN data contains many long sequences of consecutive 1:s and 0:s, and by masking this data using our bit pattern, we will almost eliminate the occurrence of bit-stuffing. But in the general case, we will get a performance closer to the 50/50 model.

5 Conclusions

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

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

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

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

In our future work we plan to investigate the exact effects of this further, including the integration of bit-stuffing effects in our framework for analysing reliability and timing trade-offs [4]. On a more detailed level, we will investigate the effects of bit stuffing the control fields of CAN frames. This includes the effects of fixed fields in the CAN control frame, as well as the bit stuffing of the arbitration field.

6 Acknowledgements

The authors wish to express their gratitude to Lucia Lo Bello for useful discussions and to the anonymous reviewers for their helpful comments. The work presented in this paper was supported by the Swedish Science Foundation via ARTES, the Swedish Foundation for Knowledge and Competence Development (KK-stiftelsen), and Mälardalen University.

7 References

[1] N. C. Audsley, A. Burns, M. Richardson, K. Tindell, and A. Wellings. Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling. Software Engineering Journal, 8(5):284-292, September 1993.

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

[3] A. Burns, S. Punnekkat, L. Strigini, and D. Wright. Probabilistic scheduling guarantees for fault-tolerant real-time systems. Proceedings of DCCS-7, IFIP International Working Conference on Dependable Computing for Critical Applications, California, January 1999.

[4] H. Hansson, T. Nolte, C. Norström, and S. Punnekkat. Integrating reliability and timing analysis of can-based systems. IEEE Transaction on Industrial Electronics. To appear in a special issue on factory communication systems.

[5] H. Hansson, C. Norström, and S. Punnekkat. Integrating Reliability and Timing Analysis of CAN-based Systems. In Proc. 2000 IEEE International Workshop on Factory Communication Systems (WFCS'2000), Porto, Portugal, September 2000. IEEE Computer Society.

[6] H. Hansson, C. Norström, and S. Punnekkat. Reliability Modelling of Time-Critical Distributed Systems. In M. Joseph, editor, Formal Techniques in Real-Time and Fault-Tolerant Systems, volume 1926 of Lecture Notes in Computer Science (LNCS), 6th International Symposium, FTRTFT 2000, Pune, India, September 2000. Springer-Verlag.

[7] I. S. O. (ISO). Road Vehicles- Interchange of digital information -Controller Area Network (CAN) for high-speed communication. ISO Standard-11898, Nov 1993.

[8] S. Punnekkat, H. Hansson, and C. Norström. Response Time Analysis under Errors for CAN. Proceedings of IEEE Real-Time Technology and Applications Symposium (RTAS 2000), pages 258-265, June 2000.

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

[10] K. W. Tindell and A. Burns. Guaranteed message latencies for distributed safety-critical hard real-time control networks. Technical Report YCS229, Dept. of Computer Science, University of York, June 1994.

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

[12] K. W. Tindell, H. Hansson, and A. J. Wellings. Analysing Real-Time Communications: Controller Area Network (CAN). Proceedings 15th IEEE Real-Time Systems Symposium, pages 259-265, December 1994.

[13] J. Xu and D. L. Parnas. Priority scheduling versus pre-run-time scheduling. Real-Time Systems Journal, 18(1), January 2000.