Minimizing CAN response-time jitter by message manipulation

Thomas Nolte, Hans Hansson and Christer Norström
Mälardalen Real-Time Research Centre
Department of Computer Engineering
Mälardalen University, Västerås, SWEDEN
http://www.mrtc.mdh.se

Abstract

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

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

1. Introduction

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

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

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

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

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

In this paper we provide a method that will minimise the variations of frame lengths caused by bit-stuffing. The number of stuff-bits in a CAN frame can vary between 0 and 29, depending on the CAN format (standard or extended), the frame length (the number of data bytes in the frame) and the frame bit pattern. This variation of frame length is problematic for e.g. control applications based on event-triggered architectures. Problems and degradation of performance caused by jitter in control applications have been shown in [5][12][17].

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

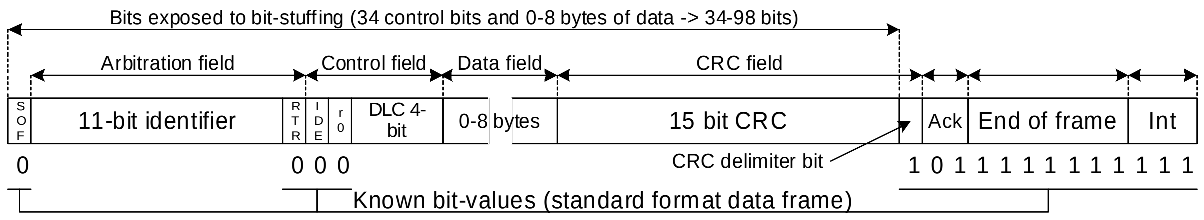

Figure 1. CAN frame layout (standard format data frame).

show how to combine this with our previous work, in order to have a method that minimizes the variations in frame length for the whole CAN frame.

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

Outline: Section 2 specifically discusses the scheduling of frame sets in Controller Area Networks under a general fault model, and describes the theory behind bit-stuffing. In Section 3 we show how we can eliminate the occurrence of stuff-bits in the header part of the CAN frame and in Section 4 we present our independent bit-stuffing model along with a method for data transformation which significantly reduces the number of stuff-bits in the data part of the CAN frame. In Section 5 we combine the techniques described in Section 3 and Section 4, and in Section 6 we show the result of using our methods and models in a case-study. Finally Section 7 presents our conclusions and outlines future work.

2. Traditional schedulability analysis of CAN frames

The Controller Area Network (CAN) [15] is a broadcast bus designed to operate at speeds of up to 1 Mbps. CAN is extensively used in automotive systems, as well as in other applications. CAN transmits data in frames containing between 0 and 8 bytes of data and 47 control bits, as shown in Figure 1. (There is also an extended format, which contains 20 more control bits. The main difference is that the extended format has 29 identifier bits instead of 11 bits. Please consult [4] for more details.)

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

CAN is a collision-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 required to be unique within the system, a station transmitting the last bit of the identifier without detecting a higher priority frame must be transmitting the highest priority queued frame, and hence can start transmitting the body of the frame.

2.1. Classical CAN bus analysis

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

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

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

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

$$q_i^{n+1} = B_i + \sum_{j \in hp(i)} \left\lceil \frac{q_i^n + J_j + \tau_{bit}}{T_j} \right\rceil C_j + E(q_i + C_i) \tag{2}$$

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-preemptive, i.e., after a bus arbitration has started the frame with the highest priority among competing frames will be transmitted until completion, even if a frame with higher priority gets queued before the transmission is completed. However, in case of errors a frame can be interrupted/preempted during transmission, requiring a complete retransmission of the entire frame. The extra cost for this is catered for in the error term E above.

Note that (2) is a recurrence relation, where the approximation to the value of q_i^{n+1} is found in terms of the *n*th approximation, with the first approximation set to zero. A solution is reached when $q_i^{n+1} = q_i^n$.

2.2. Effects of bit-stuffing, worst case

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

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

$$8s + g + 13 \tag{3}$$

where s is the number of bytes of payload data ($s = [0, 8]$) and g + 13 is the number of bits in the control part of the CAN frame. The frame layout is defined such that only g of these g + 13 bits are subject to bit-stuffing (see Figure 1). In the standard format $g = 34$ and in the extended format $g = 54$. Therefore the total number of bits after bit-stuffing can be no more than:

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

Intuitively the above formula captures the number of stuff-bits in the worst case scenario, shown in Figure 2.

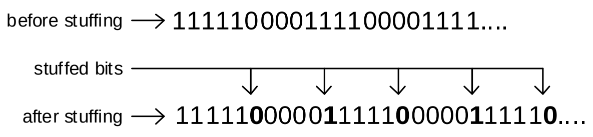
before stuffing ⟶ 11111000001111100001111....
stuffed bits
after stuffing ⟶ 1111100000111111000000111110....

Figure 2. The worst-case scenario when stuffing bits.

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

$$C_i = \left(8s_i + g + 13 + \left\lfloor \frac{g + 8s_i - 1}{4} \right\rfloor \right) \tau_{bit} \tag{5}$$

3. Careful priority usage

In this section we will investigate how it is possible to avoid/minimize stuff-bits in the header part of the CAN frame. For simplicity we will focus on the standard format, but the same reasoning holds for the extended format. The obtained data for the extended format is shown in the end of this section.

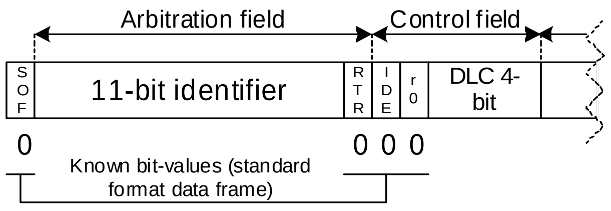

Figure 3. CAN frame header, the first 6 fields of the CAN frame (standard format).

The priority of the standard format CAN frame, which is also the arbitration field, consists of 11 bits (as can be seen in Figure 3), which are subject to bit-stuffing before the frame is actually transmitted.

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

Table 1. Amount of remaining priorities for various data lengths and their corresponding number of stuff-bits (standard format).

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

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

By carefully selecting priorities we can avoid the effect of stuff-bits in the frame header, i.e., by excluding the identifiers that lead to bit-stuffing we can *à priori* make sure that there will be no stuff-bits in any of the fields shown in Figure 3. The drawback of this is that we have forbidden the usage of some selected priorities, which obviously comes at a cost, since originally we could use all 11 bits to represent the priority and identity of the CAN frame, which gave us 2^{11} (2048) different priorities, and after the removal of selected priorities, it turns out that we have either of the following two scenarios: (1) we can eliminate the number of stuff-bits in the CAN header, or (2) we can minimize the number of stuff-bits in the CAN header to 1.

The actual numbers of stuff-bits, by forbidding priorities, are described in Table 1. Worth noticing is that the number of stuff-bits depends on the number of data bytes in the frame. This since the DLC field, see Figure 3, consists of 4 bits describing the number of bytes of data in the frame. Thus, this bit pattern will affect the number of stuff-bits generated in the frame header (all frame fields before the data part of the CAN frame, as shown in Figure 3).

What we can see in Table 1 is that we have 3 different groups of scenarios:

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

2. The second scenario is when we have 4-7 bytes of data. Here we can eliminate the number of stuff-bits in the CAN header by forbidding priorities, leaving 745 usable priorities. One can argue that forbidding priorities would be the same as to use redundant bits as "virtual stuff-bits" (since the number of usable priorities require less bits for representation compared to the number of bits that are allocated for describing the priority; some bits are left "unused"). Although there is some truth in this reasoning, the CAN header has a fixed number of bits. Hence, even if we are using fewer priorities, the number of bits in the CAN header stays the same.

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

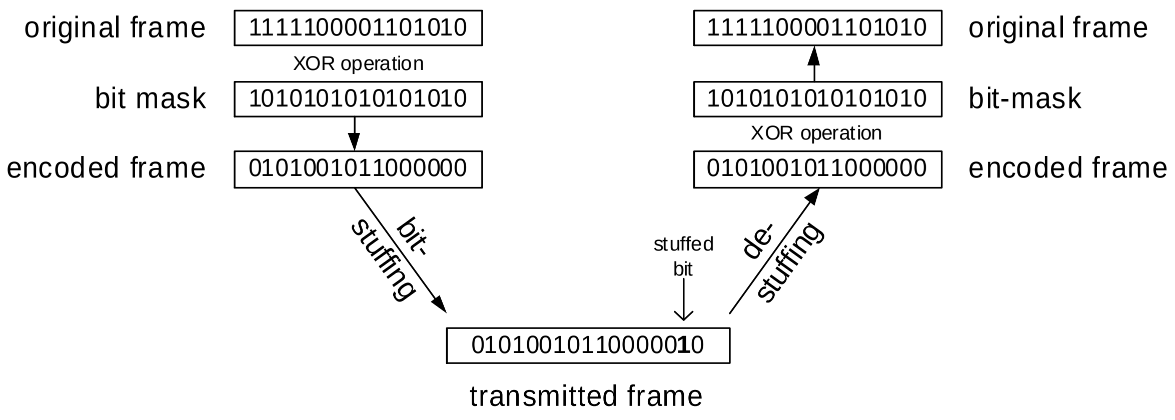

Figure 4. Encoding/decoding process for the proposed method.

forbidding priorities. The number of usable priorities is then 1131.

Conclusions of what is presented in Table 1 is that we can eliminate the occurrences of stuff-bits in the CAN header (when the message contains 4-8 bytes of data) by forbidding priorities, and the cost for this is a reduction of the number of available priorities. Therefore we believe that this method can be used, depending on the application's need of priorities, to eliminate the effect of bit-stuffing in the header part of the CAN message frame.

Corresponding values for the extended format are shown in Table 2.

4. Independent bit-stuffing model and a method for data transformation

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

The method, show in Figure 4, reduces the actual number of stuff-bits in the CAN data frame by transforming the message using an XOR operation on the data together with a bit-mask. By doing this, we showed with a case-study that the actual number of stuff-bits was significantly reduced, as can be seen in Figure 5. Here we can see (Real traffic) the number of stuff-bits in an industrial application (samples taken from one of our automotive partners). In relation to this, we also see the number of stuff-bits in artificial data generated by assuming independent and equal probability of a '1' and '0' in each bit position (50/50), and the number of stuff-bits in the same industrial data, but after using the method described above (Real traffic using XOR).

5. Combination of techniques

The methods described in Section 3 and Section 4 can be combined in order to significantly reduce the variation of CAN message frame lengths, i.e., reducing the jitter. We will in this section additionally integrate the last field in the CAN frame, the CRC field, in the jitter reduction.

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

Combining this with the second method we further reduce the number of stuff-bits. As can be seen in Figure 5 we have reduced the number of stuff-bits in an 8 byte data part of a frame to 3 from 13 (analytically 15).

Finally, the last part of the CAN frame to investigate is the CRC field at the end of the frame, shown in Figure 1. We believe, since CRC-generation essentially coincides with pseudo random binary sequence generation, that the 50/50 model described in [13] and in Section 4 is suitable for describing these bits, i.e., we assume that the CRC essentially is a sequence of bits with equal and independent probability for bit value 0 and 1, respectively. The model assumes independence among bits and equal probability for having bit-value 0 or 1. What we do then is that we use our model for both the data part and the CRC field of the CAN frame. According to the model, the number of stuff-bits and their corresponding probabilities for the data and the CRC part of the frame are described in Table 3.

By using our model we can see, when for example using 8 bytes of data, that the number of stuff-bits is reduced from, analytically 24 to 11 when the acceptable probability of exceeding the maximum frame size is in the order of 10^{-6}, since $\sum_{11 \leq i \leq 19} P_i \leq 10^{-6}$ where P_i = probability of having exactly i stuff-bits. Therefore, we have sig-

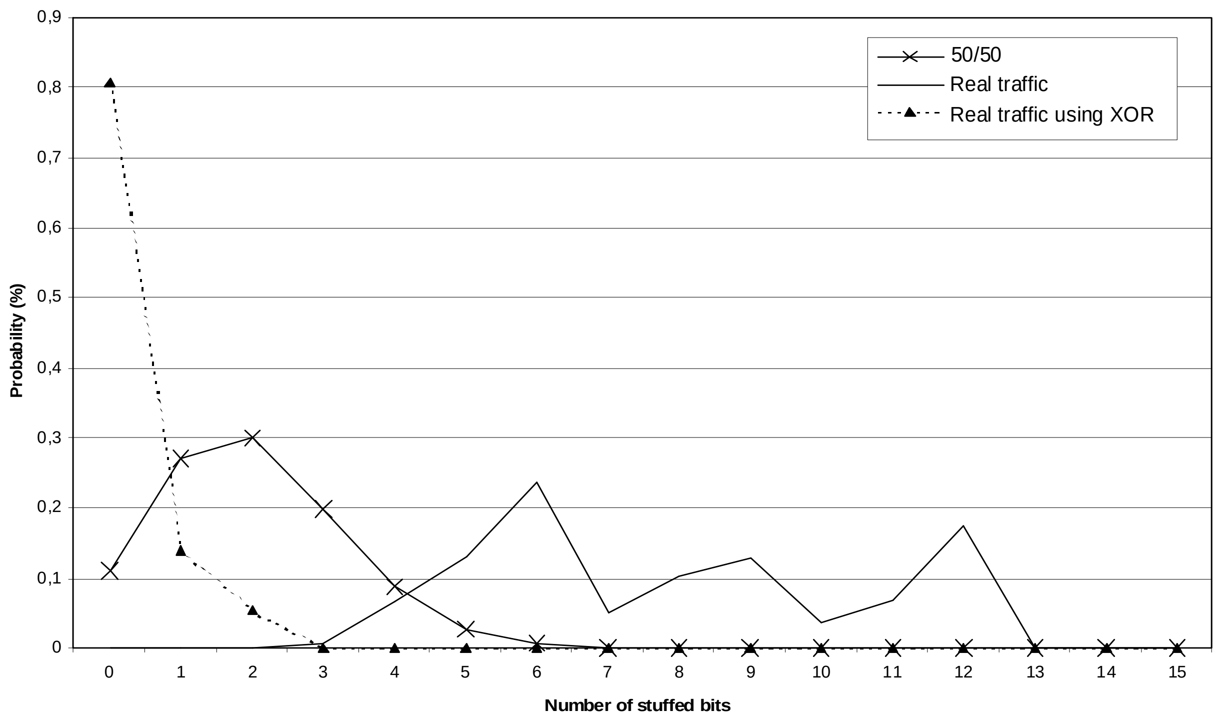

Figure 5. Probability density functions, PDF:s, showing the number of stuff-bits in a 64 bit frame. We show here our independent 50/50 model, the real CAN traffic and the manipulated real CAN traffic.

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

Table 3. Number of stuff-bits, with corresponding probability of occurrence (xEy equals $x \times 10^y$).

nificantly reduced the maximum number of stuff-bits and thus, the interval between maximum and minimum number of stuff-bits is smaller, i.e., we have reduced the considered jitter.

We must also remember that these values are based on our model. When using our method to decrease the number of stuff-bits in a real system the actual number of stuff-bits can be even smaller, as shown in Figure 5.

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

Table 4. Number of stuff-bits in the samples, with corresponding probability of occurrence.

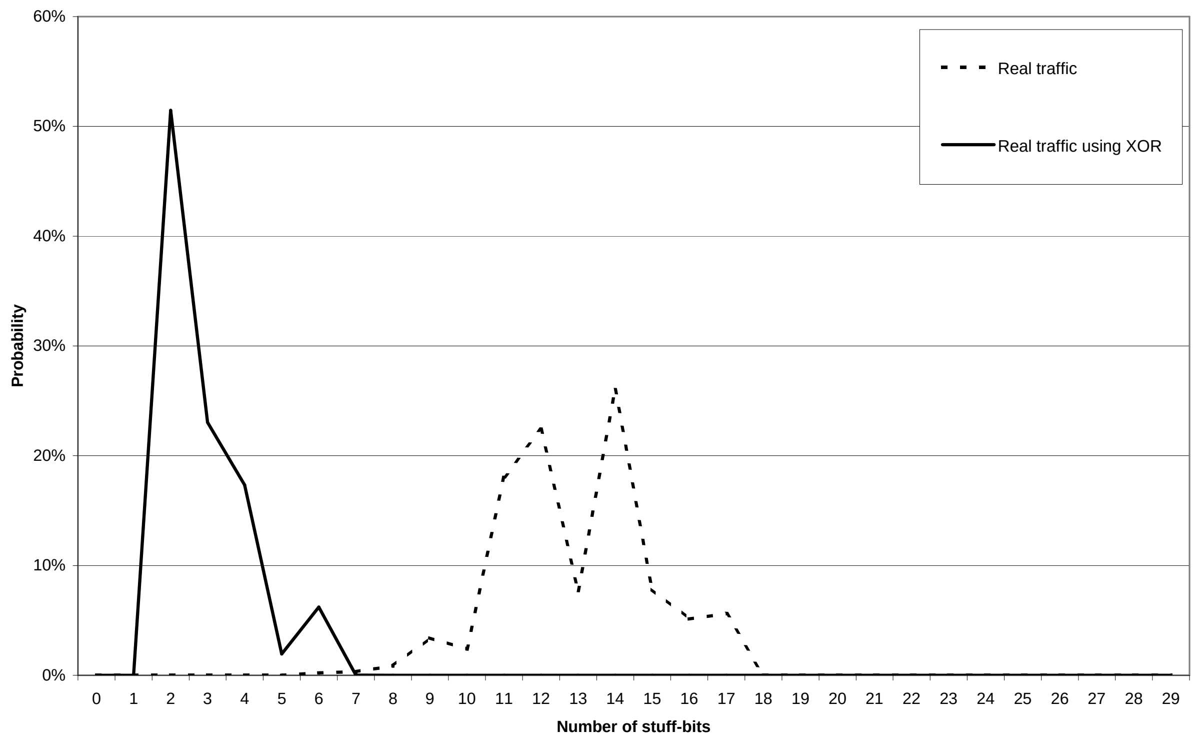

Figure 6. Probability density functions, PDF:s, showing the number of stuff-bits in a CAN frame (extended format). We show here real traffic along with the same traffic but manipulated with XOR.

6. Case-study

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

The number of stuff-bits in the CAN frame, both with the XOR manipulation as described in Section 4, and without manipulation, are shown in Figure 6. What we can read from the figure is that the actual worst-case number of stuff-

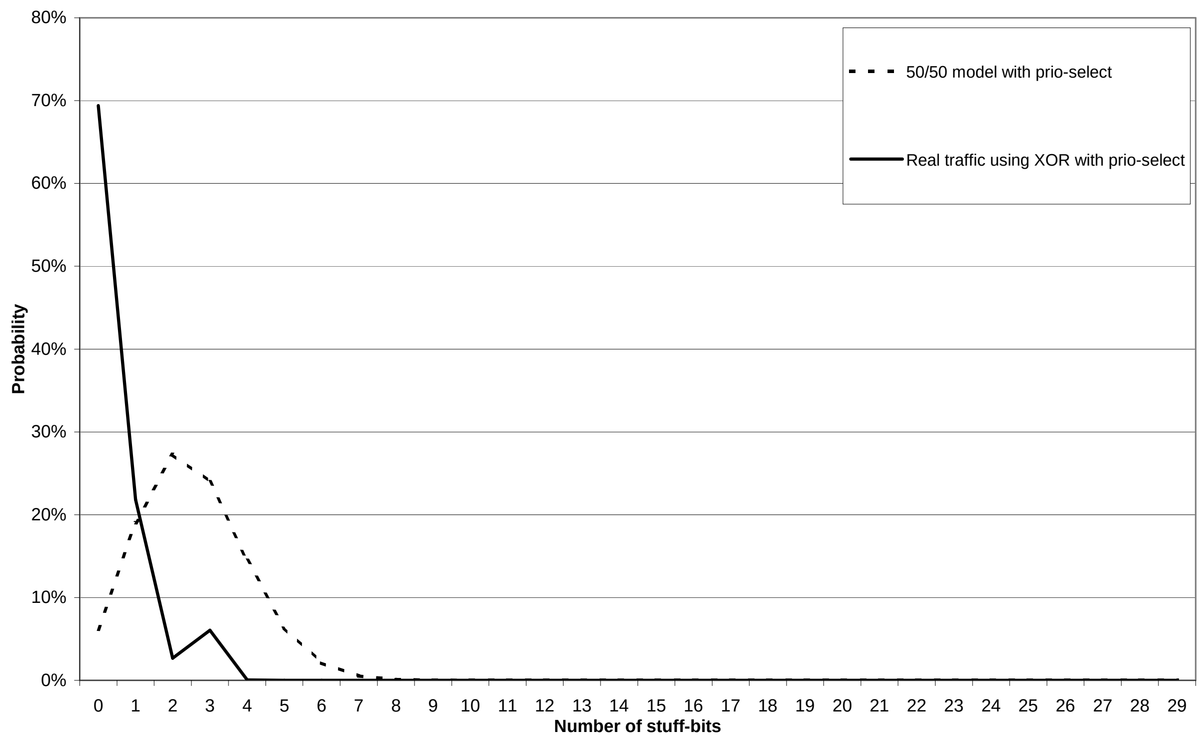

Figure 7. Probability density functions, PDF:s, showing the number of stuff-bits in a CAN frame (extended format). We show here real traffic manipulated with XOR and careful priority selecting. Our independent model is also shown with respect to the careful priority select.

bits has dropped from 17 to 7, this as a result of removing patterns of consecutive bits in the data part of the CAN frame. We used the same bit-pattern for the mask, as shown in Figure 4. Note that we have not used the method for selecting priorities yet.

In order to further reduce the number of stuff-bits in the CAN frame we also make use of the method based on forbidding some priorities, as described in Section 3 The result of this is shown in Figure 7 along with the independent model described in Section 4 (also shown as the right most column of Table 3). Note here that with the knowledge of elimination of stuff-bits in the CAN header, we use the 50/50 model only for the data part and the CRC part of the CAN frame. The result of carefully selecting priorities gives us even less stuff-bits. We have now reduced the actual worst-case number of stuff-bits from 17 to 4, as can be seen in Figure 7.

The results from all experiments within the case-study are shown in Table 4. Here we can see the number of stuff-bits in the header, data and CRC part of the original frame as well as the number of stuff-bits in the whole CAN frame. Furthermore, the number of stuff-bits in the data and CRC part of the frame after the XOR method are shown. Finally, the number of stuff-bits in the whole CAN frame, after applying both the XOR method and the priority selection, is shown.

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

7. Conclusions

In dimensioning safety critical systems, a central activity is to validate that sufficient resources are allocated to provide required behavioural, timing, and reliability guarantees. Reducing utilisation is essential, since it may allow the use of cheaper solutions in applications. Since the validation of a system or a product typically is based on a model of a system, it is important to reduce the modelled utilisation, i.e., the utilisation given by the model. This can be achieved either by more accurate modelling, or by reducing the actual utilisation of the system. Focusing on bit-stuffing in CAN, we have in this paper presented a method that both increases the accuracy of the modeling, and reduces the actual bus utilisation. What we achieve by doing this is an improvement in terms of reducing jitter. By lowering the maximum number of stuff-bits that can occur in a frame, we have significantly reduced the jitter caused by the varying number of stuff-bits in a CAN frame.

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

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

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

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

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

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

Acknowledgements

The authors wish to express their gratitude to the anonymous reviewers for their helpful comments. The work presented in this paper was supported by the Swedish Foundation for Strategic Research (SSF) via the research programme ARTES, the Swedish Foundation for Knowledge and Competence Development (KK-stiftelsen), and Mälardalen University.

References

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

[2] J. Barreiros, E. Costa, J. A. Fonseca, and F. Coutinho. Jitter Reduction in a Real-Time Message Transmission System Using Genetic Algorithms. *Proceedings of CEC'2000 - IEEE Congress on Evolutionary Computation*, 2:1095–1102, July 2000.

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

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

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

[6] F. Coutinho, J. A. Fonseca, J. Barreiros, and E. Costa. Jitter Minimization with Genetic Algorithms. *Proceedings of WFCS'2000 - 3rd IEEE International Workshop on Factory Communication Systems*, pages 267–273, September 2000.

[7] F. Coutinho, J. A. Fonseca, J. Barreiros, and E. Costa. Using Genetic Algorithms to Reduce Jitter in Control Variables Transmitted over CAN. *Proceedings of ICC'2000 - 7th International CAN Conference*, October 2000.

[8] J. Decotignie. Some Future Directions in Fieldbus Research and Development. *Proceedings of FeT'99 - Fieldbus Systems and Applications Conference*, September 1999.

[9] H. Hansson, T. Nolte, C. Norström, and S. Punnekkat. Integrating Reliability and Timing Analysis of CAN-based Systems. *IEEE Transaction on Industrial Electronics*, 2002. To appear in a special issue on factory communication systems.

[10] H. Hansson, C. Norström, and S. Punnekkat. Integrating Reliability and Timing Analysis of CAN-based Systems. *Proceedings of WFCS'2000 - 3rd IEEE International Workshop on Factory Communication Systems*, pages 165–172, September 2000.

[11] 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.

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

[13] T. Nolte, H. Hansson, C. Norström, and S. Punnekkat. Using Bit-Stuffing Distributions in CAN Analysis. *IEEE/IEE Real-Time Embedded Systems Workshop (RTES'01)*, December 2001.

[14] S. Punnekkat, H. Hansson, and C. Norström. Response Time Analysis under Errors for CAN. *Proceedings of RTAS'2000 - 6th IEEE Real-Time Technology and Applications Symposium*, pages 258–265, June 2000.

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

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

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

[18] 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.

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

[20] K. W. Tindell, H. Hansson, and A. J. Wellings. Analysing Real-Time Communications: Controller Area Network (CAN). *Proceedings of RTSS'94 -* 15^{th} *IEEE Real-Time Systems Symposium*, pages 259–265, December 1994.

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