Reducing pessimism in CAN response time analysis

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

Abstract

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

We introduce a model and some methods, that relax the pessimism of the worst case analysis, and we show the effect of our methods by considering both an artificial traffic model and samples of real CAN traffic. Also, we propose a simple coding scheme that substantially reduces the number of stuff-bits in the considered real traffic.

Delay variations (jitter) in computations and communications cause degradation of performance in control applications. There are many sources of jitter, including variations in execution time and bus contention. By introducing some restrictions when using CAN, 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 the methods mentioned above 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] [22]. 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 deterministic schedulability analysis is unfortunately quite pessimistic, since it assumes that a missed deadline in the worst case is equivalent to always missing the deadline whereas the stochastic analysis extends the knowledge of the system by telling

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 [21].

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.

Related work have been presented in [14], where the authors proposed a model for calculating worst-case latencies of Controller Area Network (CAN) [15] 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 [10][11] the level of pessimism was reduced by the introduction of a better fault model, and in [9] they also consider variable phasings between message queuings, in order to make the model more realistic.

The work presented in this paper is based on the work presented in [13], where further reduction of the pessimism introduced by the worst-case analysis of CAN message response times has been achieved. This by using distributions of stuff-bits instead of the traditional worst-case frame sizes. We will use distributions of frame lengths after stuffing instead of the traditional worst-case stuffed frames. We will look into two different scenarios:

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

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

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

Also, by using this method, we provide a method that will minimise the variations of frame length caused by bit-stuffing. The number of extra stuff-bits in a CAN frame can vary between 0 and 24, depending on the frame length (the number of data bytes in the frame) and the frame bit pattern. This variation of frame length is problematic for 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 previous work done in [13] where the authors presented a method to reduce the number of stuff-bits in the data part of the CAN frame. This work is now extended by also considering the rest of the CAN frame, not only the data part. We show how bit-stuffing can be eliminated in the header part of the CAN frame and we show how to combine this with our previous work, in order to have a method that minimizes the variations in frame length for the whole CAN frame.

There has been work done to reduce jitter caused by variations in queuing times for CAN frames [2][6][7] using genetic algorithms. This is done by giving periodic messages initial phasings, found by using genetic algorithms, to reduce/eliminate queuing jitter. 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 can be done mostly offline, and the online part requires very little CPU-time.

The outline of this work is as follows: 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. Section 2.3 introduces a model to describe the number of stuff-bits in a CAN frame, based on some assumptions. In Section 3 we investigate the occurrence of bit-stuffing in the data part of some real

CAN traffic and in Section 4 we give a proposal of how to align the real traffic to our model. In Section 5 we show how we can eliminate the occurrence of stuff-bits in the header part of the CAN frame and in Section 6 we combine the techniques described in Section 5 and Section 4, and in Section 7 we show the result of using all our methods and models in a case-study. Finally Section 8 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 transmit 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.)

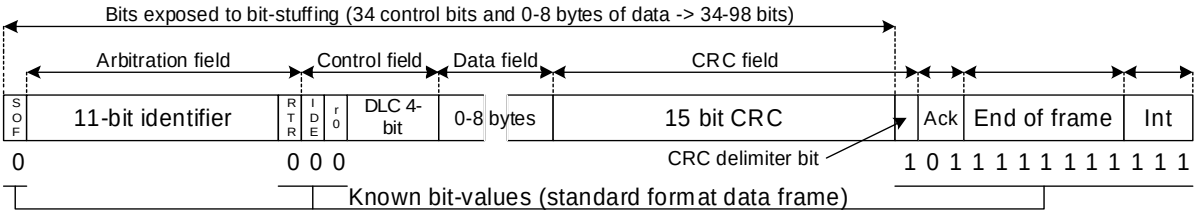

Figure 1. CAN frame layout (standard format data frame).

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

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

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.

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:

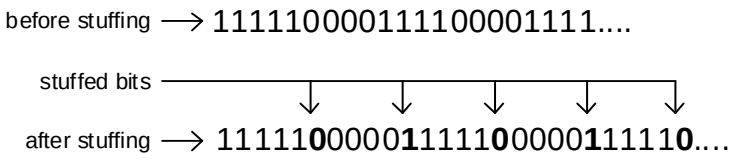

Figure 2. The worst case scenario when stuffing bits.

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

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

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 (standard format), before bit-stuffing, can be at most 111 bits (8 bytes data and 47 control bits), and among them 98 bits are exposed

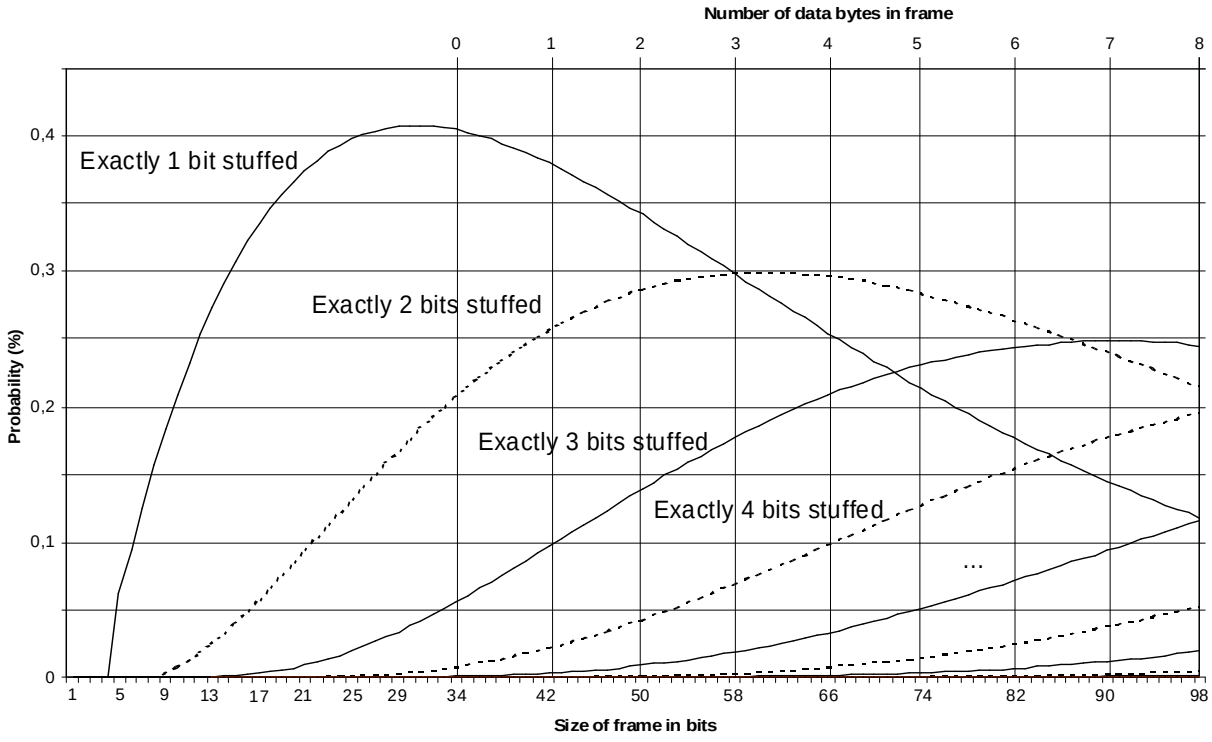

Figure 3. Probability of a specific number of stuff-bits in a frame, assuming our probabilistic frame model. The 9 lengths are marked as vertical lines.

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 3. The nine different frame lengths (0-8 bytes of data) are visualised in the graph as vertical lines. Note that only the first 8 cases of stuff-bits (1-8 bits stuffed in the frame) are visible in the graph, since the probability of getting more than 8 stuff-bits is very low. For example, the probability of getting exactly 10, 15 and 20 stuff-bits never exceeds 10^{-4}, 10^{-9}, and 10^{-17}, respectively. Figure 3 is the result of an exhaustive analysis of all possible frame patterns. The result of this analysis can be found in Appendix A and Appendix B where Appendix A contains the probability of having exactly n stuff-bits in m bits of data. Appendix B contains the number of combinations with exactly n stuff-bits in m bits of data (where the total number of combinations for a m bit data is 2^m).

3 Case study: Real CAN traffic

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

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

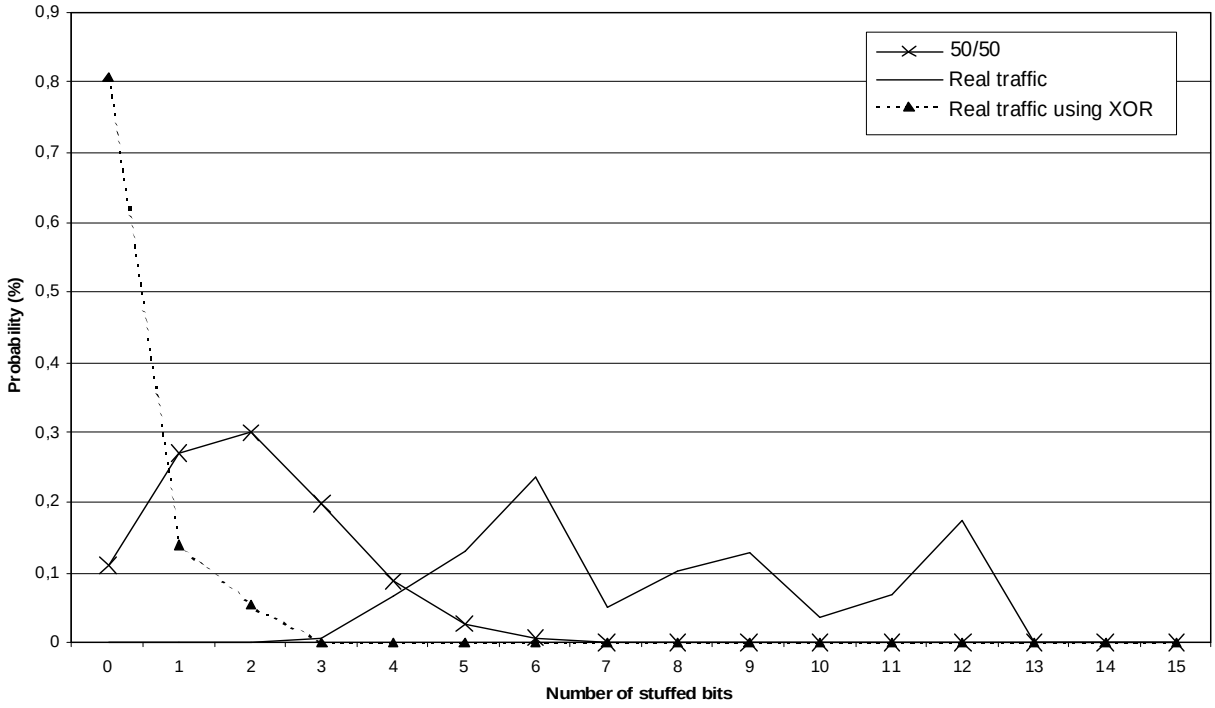

Figure 4. Probability density functions, PDF:s, showing the number of stuff-bits in a 64 bit frame. We show here our independent 50/50 model, the real CAN traffic and the manipulated real CAN traffic.

The 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 other parts of the CAN frames (control fields and so on) were not considered. The obtained distribution of stuff-bits is shown in Figure 4 (”Real traffic”). Worth noticing is that the actual worst case is here 13 bits, to be compared with the analytic worst-case result of 15 stuff-bits when applying traditional analysis for a frame size of 64 bits. The figure also shows the distribution obtained with our 50/50 model (”50/50”), as well as the distribution obtained for the real-traffic when applying the coding (”Real traffic using XOR”) that we will present next.

4 A simple coding scheme to reduce bit-stuffing

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

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 5 illustrates the encoding/decoding process.

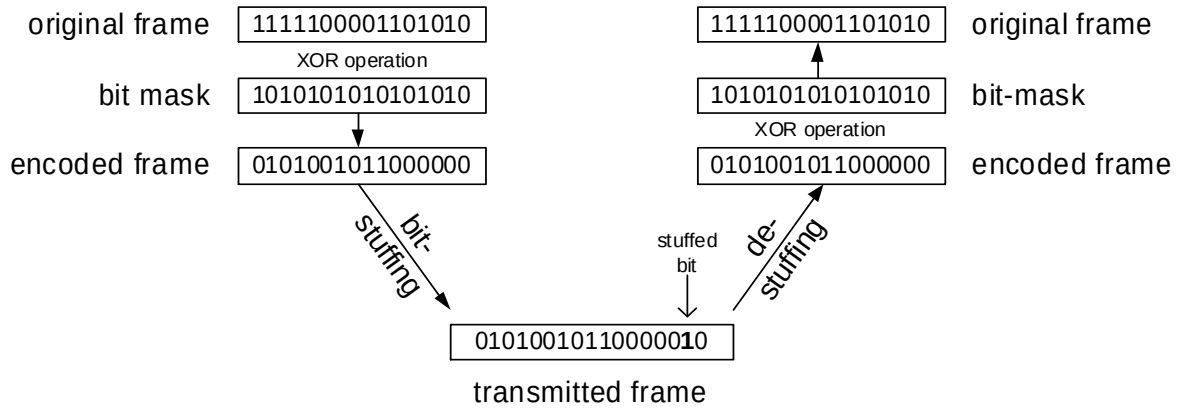

Figure 5. Encoding/decoding process for the proposed method.

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

We have applied the simple XOR-coding to our 25 000 automotive CAN frames. The result is presented in Figure 4 (”Real traffic using XOR”). Here we compare the number of stuff-bits in a frame of size 64 bits, i.e., 8 bytes. Our 50/50 independent model gives 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 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 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.

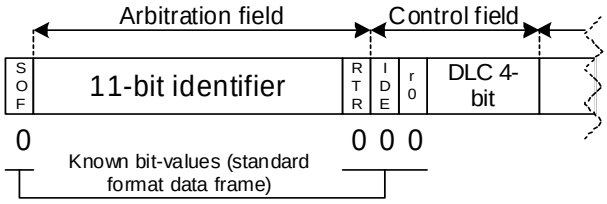

Figure 6. CAN frame header, the first 6 fields of the CAN frame (standard format).

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

Table 1. Amount of remaining priorities for various data lengths and their corresponding number of stuff-bits (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 6), which are subject to bit-stuffing before the frame is actually transmitted.

By carefully selecting priorities we can avoid the effect of stuff-bits in the frame header, i.e., by excluding the identifiers that lead to bit-stuffing we can *à priori* make sure that there will be no stuff-bits in any of the fields shown in Figure 6. 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

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}).

field, see Figure 6, 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 6).

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 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.

6 Combination of techniques

The methods described in Section 4 and Section 5 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 4 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 2.3 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.

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 $x \times 10^y$).

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

We must also remember that these values are based on our model. When using our method to de-

crease the number of stuff-bits in a real system the actual number of stuff-bits can be even smaller, as shown in Figure 4.

7 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 2.3, and without manipulation, are shown in Figure 7. What we can read from the figure is that the actual worst-case number of stuff-bits has dropped from 17 to 7, this as a result of removing patterns of consecutive bits in the data part of the CAN frame. We used the same bit-pattern for the mask, as shown in Figure 5. 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 5 The result of this is shown in Figure 8 along with the independent model described in Section 2.3 (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

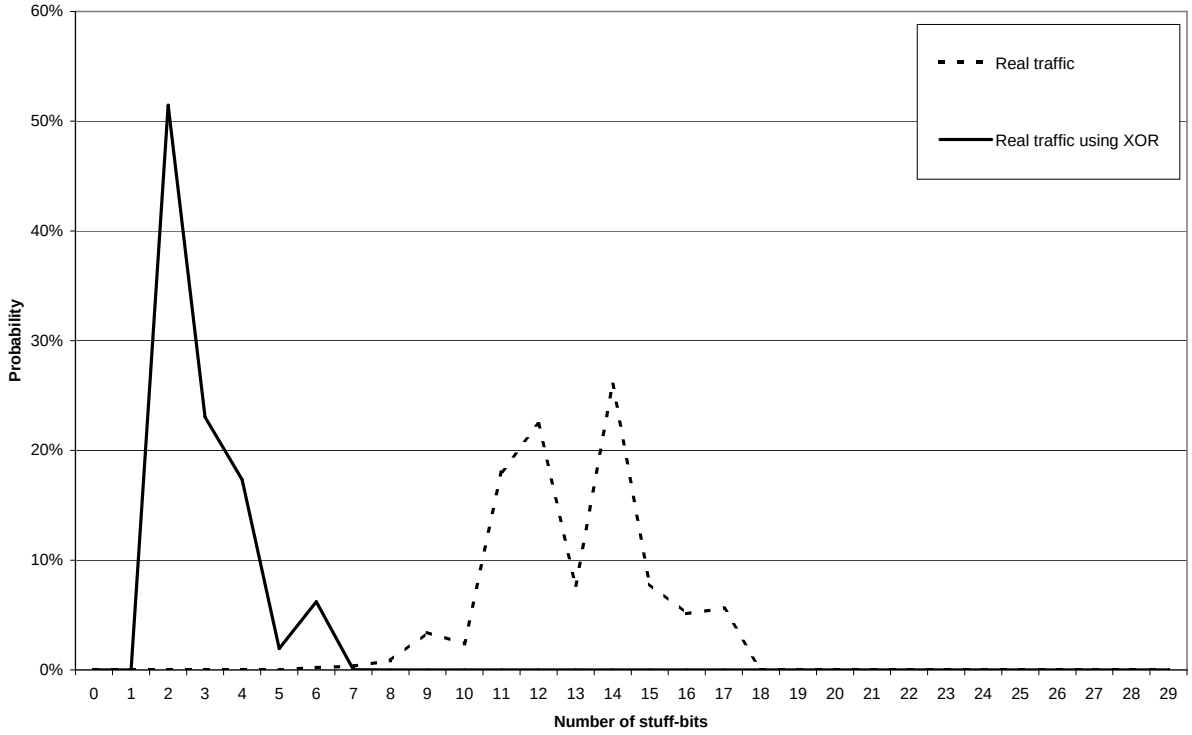

Figure 7. Probability density functions, PDF:s, showing the number of stuff-bits in a CAN frame (extended format). We show here real traffic along with the same traffic but manipulated with XOR.

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.

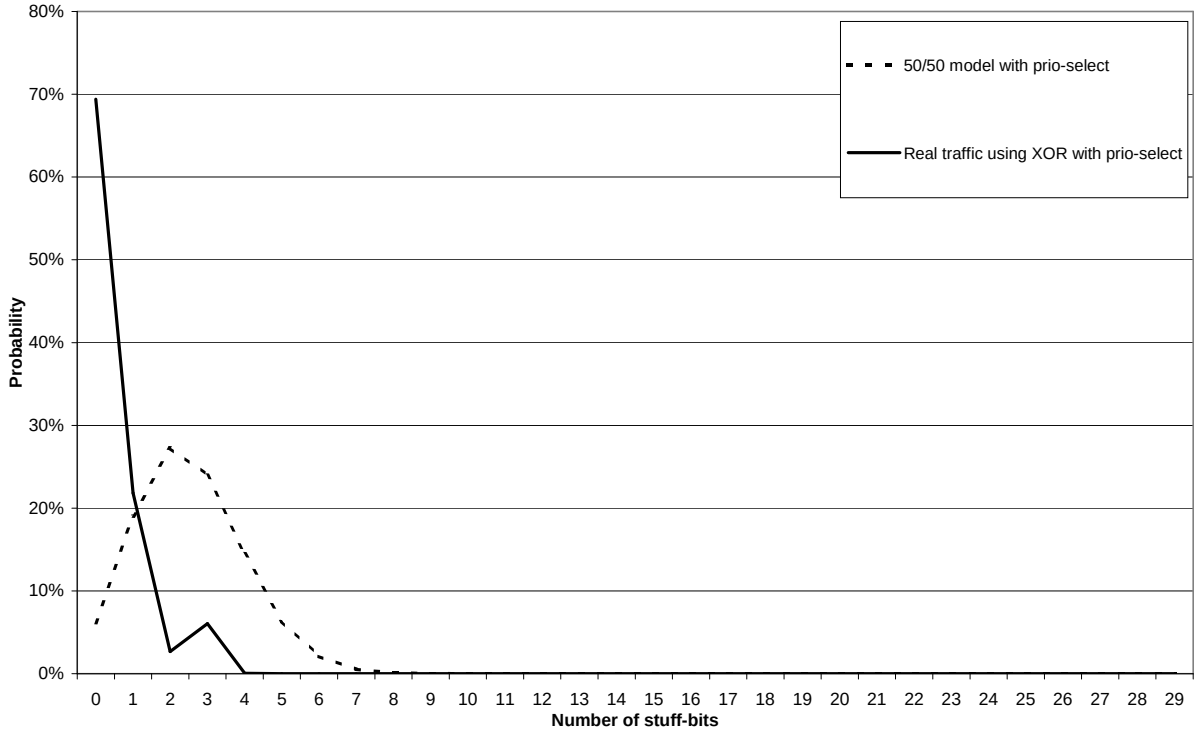

Figure 8. Probability density functions, PDF:s, showing the number of stuff-bits in a CAN frame (extended format). We show here real traffic manipulated with XOR and careful priority selecting. Our independent model is also shown with respect to the careful priority select.

from 17 to 4, as can be seen in Figure 8.

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.

8 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, in the data part of the frame, 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.

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.

Furthermore, we achieve, by applying these methods, an improvement in terms of jitter. We have significantly reduced the jitter caused by the variations of the number of stuff-bits in a CAN frame. This has been achieved by lowering the maximum number of stuff-bits that can occur in a frame.

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 the exact effects of this further, including the integration of bit-stuffing effects in the framework for analysing reliability and timing trade-offs presented in [9]. It would be interesting to see 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.

Our ultimate goal is of course 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 author wishes to express his gratitude to Hans Hansson and Christer Norström for their support and useful discussions.

The work presented in this paper was supported by the Swedish Foundation for Strategic Research (SSF) via the 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 - 3^{rd} 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 - 7^{th} International CAN Conference*, October 2000.

[8] J.D. 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*, 49(6), December 2002.

[10] H. Hansson, C. Norström, and S. Punnekkat. Integrating Reliability and Timing Analysis of CAN-based Systems. *Proceedings of WFCS'2000 - 3^{rd} 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)*, 6^{th} 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. In *Proceedings of IEEE Real-Time Technology and Applications Symposium (RTAS 2000)*, pages 258–265. IEEE Computer Society, 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.P. Lehoczky. Priority Inheritance Protocols: An Approach to Real-Time Synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, September 1990.

[17] A. Stothert and I.M. MacLeod. Effect of Timing Jitter on Distributed Computer Control System Performance. *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). In *Proceedings 15th IEEE Real-Time Systems Symposium*, pages 259–265. IEEE Computer Society, December 1994.

[21] M. Törngren. Fundamentals of implementing real-time control applications in distributed computer systems. *Real-Time Systems Journal*, 14(3):219–250, May 1998.

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

A Probability of having exactly n stuff-bits when data is m bit long

Number of bits in data	Probability of having exactly n stuff-bits								
	0	1	2	3	4	5	6	7	8
1-4	1	0	0	0	0	0	0	0	0
5	0.9375	0.0625	0	0	0	0	0	0	0
6	0.90625	0.09375	0	0	0	0	0	0	0
7	0.875	0.125	0	0	0	0	0	0	0
8	0.84375	0.15625	0	0	0	0	0	0	0
9	0.8125	0.183594	0.003906	0	0	0	0	0	0
10	0.783203	0.208984	0.007813	0	0	0	0	0	0
11	0.754883	0.232422	0.012695	0	0	0	0	0	0
12	0.727539	0.253906	0.018555	0	0	0	0	0	0
13	0.701172	0.273438	0.025146	0.000244	0	0	0	0	0
14	0.675781	0.29126	0.032349	0.00061	0	0	0	0	0
15	0.651306	0.307434	0.0401	0.00116	0	0	0	0	0
16	0.627716	0.322052	0.048309	0.001923	0	0	0	0	0
17	0.60498	0.335205	0.056885	0.002914	1.53E-05	0	0	0	0
18	0.583069	0.346985	0.065758	0.004143	4.58E-05	0	0	0	0
19	0.561951	0.357471	0.07486	0.005619	9.92E-05	0	0	0	0
20	0.541597	0.366743	0.084127	0.007349	0.000183	0	0	0	0
21	0.521981	0.374874	0.093504	0.009335	0.000305	9.54E-07	0	0	0
22	0.503076	0.381935	0.102938	0.011576	0.000473	3.34E-06	0	0	0
23	0.484855	0.387993	0.112381	0.01407	0.000693	8.11E-06	0	0	0
24	0.467294	0.393111	0.121793	0.016813	0.000973	1.65E-05	0	0	0
25	0.450369	0.397349	0.131132	0.019798	0.001321	2.98E-05	5.96E-08	0	0
26	0.434057	0.400764	0.140367	0.023018	0.001744	4.98E-05	2.38E-07	0	0
27	0.418336	0.403411	0.149464	0.026464	0.002247	7.82E-05	6.41E-07	0	0
28	0.403184	0.405341	0.158397	0.030124	0.002836	0.000117	1.42E-06	0	0
29	0.388581	0.406601	0.16714	0.033988	0.003518	0.000169	2.76E-06	3.73E-09	0
30	0.374507	0.407238	0.175673	0.038045	0.004296	0.000236	4.94E-06	1.68E-08	0
31	0.360943	0.407296	0.183976	0.04228	0.005176	0.00032	8.27E-06	4.94E-08	0
32	0.34787	0.406816	0.192032	0.046682	0.006162	0.000425	1.31E-05	1.18E-07	0
33	0.33527	0.405836	0.199828	0.051237	0.007256	0.000552	2E-05	2.46E-07	2.33E-10
34	0.323127	0.404393	0.207351	0.055932	0.008462	0.000705	2.93E-05	4.69E-07	1.16E-09
35	0.311424	0.402522	0.214591	0.060752	0.009781	0.000887	4.18E-05	8.3E-07	3.73E-09
36	0.300144	0.400257	0.22154	0.065683	0.011216	0.001101	5.81E-05	1.39E-06	9.55E-09
37	0.289273	0.397627	0.22819	0.070712	0.012767	0.001348	7.89E-05	2.23E-06	2.13E-08
38	0.278796	0.394663	0.234538	0.075826	0.014435	0.001633	0.000105	3.43E-06	4.28E-08
39	0.268699	0.391392	0.240579	0.08101	0.016219	0.001958	0.000138	5.11E-06	8.01E-08
40	0.258967	0.38784	0.24631	0.086252	0.01812	0.002326	0.000177	7.41E-06	1.41E-07
41	0.249587	0.384032	0.25173	0.091538	0.020137	0.002739	0.000225	1.05E-05	2.36E-07
42	0.240547	0.379991	0.25684	0.096856	0.022267	0.0032	0.000283	1.45E-05	3.8E-07
43	0.231835	0.375739	0.26164	0.102194	0.02451	0.003711	0.000351	1.97E-05	5.91E-07
44	0.223438	0.371297	0.266131	0.107538	0.026862	0.004276	0.000431	2.63E-05	8.92E-07
45	0.215345	0.366683	0.270315	0.112879	0.029322	0.004895	0.000524	3.47E-05	1.31E-06
46	0.207546	0.361917	0.274197	0.118204	0.031885	0.005572	0.000631	4.5E-05	1.88E-06
47	0.200029	0.357015	0.27778	0.123504	0.03455	0.006308	0.000754	5.76E-05	2.65E-06
48	0.192784	0.351993	0.281068	0.128767	0.037311	0.007105	0.000895	7.3E-05	3.66E-06
49	0.185801	0.346867	0.284066	0.133985	0.040166	0.007965	0.001053	9.14E-05	4.98E-06
50	0.179072	0.34165	0.28678	0.139147	0.04311	0.008888	0.001232	0.000113	6.66E-06
51	0.172586	0.336356	0.289214	0.144246	0.046139	0.009877	0.001433	0.000139	8.8E-06
52	0.166335	0.330997	0.291377	0.149273	0.049248	0.010932	0.001656	0.00017	1.15E-05
53	0.160311	0.325586	0.293273	0.154219	0.052432	0.012055	0.001904	0.000206	1.48E-05
54	0.154504	0.320132	0.294909	0.159079	0.055686	0.013245	0.002178	0.000247	1.89E-05
55	0.148908	0.314646	0.296293	0.163845	0.059006	0.014504	0.002479	0.000294	2.38E-05
56	0.143515	0.309137	0.297431	0.16851	0.062386	0.015831	0.002809	0.000349	2.98E-05
57	0.138317	0.303615	0.298331	0.173069	0.065821	0.017228	0.003169	0.000411	3.7E-05
58	0.133307	0.298087	0.299	0.177517	0.069306	0.018693	0.00356	0.000481	4.55E-05

Table 5. Probability of having exactly 0-8 stuff-bits when data is 1-58 bit long.

Number of bits in data	Probability of having exactly n stuff-bits								
	0	1	2	3	4	5	6	7	8
59	0.128479	0.292562	0.299445	0.181848	0.072835	0.020227	0.003985	0.00056	5.56E-05
60	0.123826	0.287045	0.299675	0.186058	0.076402	0.021829	0.004443	0.000649	6.75E-05
61	0.119341	0.281545	0.299695	0.190143	0.080004	0.023498	0.004938	0.000748	8.13E-05
62	0.115018	0.276066	0.299515	0.1941	0.083633	0.025234	0.005469	0.000859	9.74E-05
63	0.110853	0.270615	0.299141	0.197924	0.087286	0.027037	0.006037	0.000982	0.000116
64	0.106838	0.265197	0.298581	0.201613	0.090955	0.028904	0.006645	0.001117	0.000137
65	0.102968	0.259816	0.297843	0.205165	0.094637	0.030834	0.007293	0.001267	0.000162
66	0.099239	0.254477	0.296933	0.208577	0.098327	0.032827	0.007981	0.001431	0.000189
67	0.095644	0.249184	0.29586	0.211848	0.102018	0.03488	0.008712	0.00161	0.000221
68	0.09218	0.24394	0.29463	0.214976	0.105706	0.036992	0.009484	0.001806	0.000256
69	0.088842	0.238749	0.29325	0.21796	0.109386	0.039161	0.010301	0.00202	0.000296
70	0.085624	0.233615	0.291728	0.220799	0.113053	0.041386	0.011161	0.002251	0.000341
71	0.082523	0.228539	0.29007	0.223493	0.116703	0.043663	0.012066	0.002501	0.000391
72	0.079534	0.223524	0.288284	0.226042	0.12033	0.045992	0.013015	0.002772	0.000447
73	0.076653	0.218573	0.286376	0.228445	0.123932	0.048369	0.014011	0.003063	0.000509
74	0.073877	0.213688	0.284352	0.230703	0.127502	0.050793	0.015052	0.003375	0.000577
75	0.071201	0.208869	0.282218	0.232815	0.131037	0.053261	0.016138	0.00371	0.000652
76	0.068622	0.204119	0.279982	0.234784	0.134534	0.05577	0.017271	0.004068	0.000735
77	0.066137	0.199439	0.277649	0.23661	0.137988	0.058318	0.01845	0.004451	0.000826
78	0.063741	0.194831	0.275225	0.238294	0.141395	0.060902	0.019676	0.004858	0.000925
79	0.061433	0.190295	0.272715	0.239837	0.144752	0.063521	0.020947	0.005291	0.001034
80	0.059208	0.185831	0.270126	0.241241	0.148055	0.06617	0.022264	0.00575	0.001152
81	0.057063	0.181442	0.267463	0.242508	0.151302	0.068847	0.023626	0.006236	0.00128
82	0.054996	0.177126	0.264731	0.243639	0.15449	0.07155	0.025033	0.00675	0.001418
83	0.053005	0.172885	0.261935	0.244636	0.157614	0.074275	0.026485	0.007293	0.001568
84	0.051085	0.168719	0.259081	0.245502	0.160673	0.07702	0.027981	0.007864	0.00173
85	0.049235	0.164628	0.256172	0.246238	0.163665	0.079782	0.029521	0.008465	0.001904
86	0.047451	0.160612	0.253214	0.246847	0.166585	0.082557	0.031103	0.009097	0.002091
87	0.045733	0.156671	0.250211	0.247331	0.169434	0.085345	0.032727	0.009759	0.002292
88	0.044076	0.152805	0.247168	0.247694	0.172207	0.08814	0.034392	0.010452	0.002506
89	0.04248	0.149013	0.244089	0.247936	0.174904	0.090941	0.036097	0.011177	0.002736
90	0.040941	0.145296	0.240977	0.248062	0.177523	0.093745	0.037841	0.011934	0.00298
91	0.039458	0.141652	0.237837	0.248073	0.180061	0.096549	0.039623	0.012723	0.00324
92	0.038029	0.138082	0.234672	0.247973	0.182518	0.09935	0.041442	0.013545	0.003517
93	0.036652	0.134584	0.231486	0.247764	0.184892	0.102146	0.043297	0.0144	0.003811
94	0.035324	0.131159	0.228283	0.247449	0.187182	0.104933	0.045186	0.015287	0.004122
95	0.034045	0.127805	0.225065	0.247032	0.189387	0.10771	0.047108	0.016208	0.004451
96	0.032812	0.124522	0.221836	0.246514	0.191506	0.110473	0.049062	0.017163	0.004799
97	0.031623	0.12131	0.218599	0.245899	0.193539	0.113221	0.051047	0.01815	0.005165
98	0.030478	0.118166	0.215357	0.24519	0.195484	0.11595	0.05306	0.019171	0.005552

Table 6. Probability of having exactly 0-8 stuff-bits when data is 59-98 bit long.

Number of bits in data	Probability of having exactly n stuff-bits							
	9	10	11	12	13	14	15	16
1-36	0	0	0	0	0	0	0	0
37	1.46E-11	0	0	0	0	0	0	0
38	8E-11	0	0	0	0	0	0	0
39	2.76E-10	0	0	0	0	0	0	0
40	7.57E-10	0	0	0	0	0	0	0
41	1.79E-09	9.09E-13	0	0	0	0	0	0
42	3.8E-09	5.46E-12	0	0	0	0	0	0
43	7.46E-09	2.02E-11	0	0	0	0	0	0
44	1.38E-08	5.89E-11	0	0	0	0	0	0
45	2.41E-08	1.47E-10	5.68E-14	0	0	0	0	0
46	4.04E-08	3.29E-10	3.69E-13	0	0	0	0	0
47	6.54E-08	6.76E-10	1.46E-12	0	0	0	0	0
48	1.02E-07	1.3E-09	4.51E-12	0	0	0	0	0
49	1.56E-07	2.38E-09	1.19E-11	3.55E-15	0	0	0	0
50	2.32E-07	4.15E-09	2.78E-11	2.49E-14	0	0	0	0
51	3.38E-07	6.96E-09	5.98E-11	1.05E-13	0	0	0	0
52	4.82E-07	1.13E-08	1.2E-10	3.41E-13	0	0	0	0
53	6.75E-07	1.78E-08	2.28E-10	9.41E-13	2.22E-16	0	0	0
54	9.31E-07	2.74E-08	4.12E-10	2.31E-12	1.67E-15	0	0	0
55	1.27E-06	4.11E-08	7.16E-10	5.17E-12	7.44E-15	0	0	0
56	1.7E-06	6.04E-08	1.2E-09	1.08E-11	2.55E-14	0	0	0
57	2.25E-06	8.72E-08	1.96E-09	2.13E-11	7.36E-14	1.39E-17	0	0
58	2.94E-06	1.24E-07	3.11E-09	3.99E-11	1.88E-13	1.11E-16	0	0
59	3.82E-06	1.73E-07	4.8E-09	7.17E-11	4.4E-13	5.24E-16	0	0
60	4.89E-06	2.38E-07	7.27E-09	1.24E-10	9.53E-13	1.88E-15	0	0
61	6.22E-06	3.24E-07	1.08E-08	2.09E-10	1.94E-12	5.69E-15	8.67E-19	0
62	7.84E-06	4.35E-07	1.57E-08	3.41E-10	3.77E-12	1.52E-14	7.37E-18	0
63	9.8E-06	5.77E-07	2.26E-08	5.43E-10	7E-12	3.68E-14	3.66E-17	0
64	1.22E-05	7.59E-07	3.19E-08	8.45E-10	1.25E-11	8.26E-14	1.38E-16	0
65	1.5E-05	9.87E-07	4.45E-08	1.29E-09	2.17E-11	1.74E-13	4.35E-16	5.42E-20
66	1.83E-05	1.27E-06	6.12E-08	1.93E-09	3.64E-11	3.49E-13	1.2E-15	4.88E-19
67	2.23E-05	1.63E-06	8.32E-08	2.84E-09	5.95E-11	6.68E-13	3.03E-15	2.55E-18
68	2.69E-05	2.06E-06	1.12E-07	4.11E-09	9.52E-11	1.23E-12	7.04E-15	1E-17
69	3.24E-05	2.6E-06	1.49E-07	5.87E-09	1.49E-10	2.19E-12	1.53E-14	3.29E-17
70	3.87E-05	3.24E-06	1.96E-07	8.26E-09	2.29E-10	3.78E-12	3.17E-14	9.46E-17
71	4.6E-05	4.02E-06	2.56E-07	1.15E-08	3.45E-10	6.36E-12	6.25E-14	2.46E-16
72	5.44E-05	4.96E-06	3.32E-07	1.58E-08	5.11E-10	1.04E-11	1.18E-13	5.91E-16
73	6.41E-05	6.08E-06	4.26E-07	2.15E-08	7.46E-10	1.67E-11	2.17E-13	1.33E-15
74	7.51E-05	7.4E-06	5.43E-07	2.89E-08	1.07E-09	2.63E-11	3.84E-13	2.83E-15
75	8.77E-05	8.97E-06	6.87E-07	3.85E-08	1.53E-09	4.06E-11	6.62E-13	5.74E-15
76	0.000102	1.08E-05	8.63E-07	5.09E-08	2.14E-09	6.15E-11	1.11E-12	1.12E-14
77	0.000118	1.3E-05	1.08E-06	6.66E-08	2.98E-09	9.19E-11	1.83E-12	2.1E-14
78	0.000136	1.55E-05	1.34E-06	8.66E-08	4.09E-09	1.35E-10	2.94E-12	3.82E-14
79	0.000157	1.84E-05	1.65E-06	1.12E-07	5.56E-09	1.96E-10	4.64E-12	6.75E-14
80	0.000179	2.17E-05	2.02E-06	1.43E-07	7.49E-09	2.81E-10	7.2E-12	1.16E-13
81	0.000205	2.56E-05	2.47E-06	1.82E-07	1E-08	3.98E-10	1.1E-11	1.95E-13
82	0.000233	3E-05	3E-06	2.29E-07	1.32E-08	5.58E-10	1.65E-11	3.21E-13
83	0.000265	3.5E-05	3.62E-06	2.88E-07	1.74E-08	7.74E-10	2.45E-11	5.18E-13
84	0.000299	4.08E-05	4.35E-06	3.59E-07	2.26E-08	1.06E-09	3.58E-11	8.21E-13
85	0.000338	4.73E-05	5.21E-06	4.46E-07	2.93E-08	1.44E-09	5.16E-11	1.28E-12
86	0.00038	5.47E-05	6.2E-06	5.5E-07	3.77E-08	1.95E-09	7.37E-11	1.96E-12
87	0.000427	6.3E-05	7.36E-06	6.76E-07	4.81E-08	2.6E-09	1.04E-10	2.96E-12
88	0.000477	7.24E-05	8.71E-06	8.26E-07	6.1E-08	3.45E-09	1.46E-10	4.42E-12
89	0.000533	8.28E-05	1.03E-05	1E-06	7.69E-08	4.54E-09	2.01E-10	6.5E-12

Table 7. Probability of having exactly 9-16 stuff-bits when data is 1-89 bit long.

Number of bits in data	Probability of having exactly n stuff-bits							
	9	10	11	12	13	14	15	16
90	0.000594	9.46E-05	1.2E-05	1.21E-06	9.65E-08	5.93E-09	2.76E-10	9.46E-12
91	0.00066	0.000108	1.41E-05	1.46E-06	1.2E-07	7.7E-09	3.76E-10	1.36E-11
92	0.000731	0.000122	1.64E-05	1.76E-06	1.49E-07	9.93E-09	5.07E-10	1.93E-11
93	0.000809	0.000138	1.9E-05	2.1E-06	1.84E-07	1.27E-08	6.78E-10	2.73E-11
94	0.000893	0.000156	2.2E-05	2.5E-06	2.27E-07	1.62E-08	9.01E-10	3.8E-11
95	0.000984	0.000176	2.54E-05	2.96E-06	2.77E-07	2.05E-08	1.19E-09	5.26E-11
96	0.001082	0.000198	2.93E-05	3.5E-06	3.37E-07	2.59E-08	1.56E-09	7.21E-11
97	0.001187	0.000221	3.36E-05	4.13E-06	4.09E-07	3.24E-08	2.03E-09	9.8E-11
98	0.001301	0.000248	3.84E-05	4.84E-06	4.94E-07	4.04E-08	2.62E-09	1.32E-10

Table 8. Probability of having exactly 9-16 stuff-bits when data is 90-98 bit long.

Number of bits in data	Probability of having exactly n stuff-bits							
	17	18	19	20	21	22	23	24
1-68	0	0	0	0	0	0	0	0
69	3.39E-21	0	0	0	0	0	0	0
70	3.22E-20	0	0	0	0	0	0	0
71	1.76E-19	0	0	0	0	0	0	0
72	7.23E-19	0	0	0	0	0	0	0
73	2.47E-18	2.12E-22	0	0	0	0	0	0
74	7.35E-18	2.12E-21	0	0	0	0	0	0
75	1.98E-17	1.21E-20	0	0	0	0	0	0
76	4.9E-17	5.18E-20	0	0	0	0	0	0
77	1.14E-16	1.83E-19	1.32E-23	0	0	0	0	0
78	2.49E-16	5.66E-19	1.39E-22	0	0	0	0	0
79	5.19E-16	1.57E-18	8.3E-22	0	0	0	0	0
80	1.04E-15	4.02E-18	3.69E-21	0	0	0	0	0
81	2E-15	9.59E-18	1.35E-20	8.27E-25	0	0	0	0
82	3.73E-15	2.16E-17	4.32E-20	9.1E-24	0	0	0	0
83	6.74E-15	4.62E-17	1.24E-19	5.67E-23	0	0	0	0
84	1.19E-14	9.49E-17	3.26E-19	2.61E-22	0	0	0	0
85	2.04E-14	1.87E-16	7.99E-19	9.92E-22	5.17E-26	0	0	0
86	3.43E-14	3.57E-16	1.85E-18	3.27E-21	5.95E-25	0	0	0
87	5.66E-14	6.62E-16	4.06E-18	9.65E-21	3.85E-24	0	0	0
88	9.15E-14	1.19E-15	8.54E-18	2.61E-20	1.84E-23	0	0	0
89	1.45E-13	2.09E-15	1.73E-17	6.59E-20	7.22E-23	3.23E-27	0	0
90	2.27E-13	3.6E-15	3.37E-17	1.56E-19	2.45E-22	3.88E-26	0	0
91	3.5E-13	6.05E-15	6.39E-17	3.52E-19	7.46E-22	2.61E-25	0	0
92	5.32E-13	9.98E-15	1.18E-16	7.59E-19	2.08E-21	1.29E-24	0	0
93	7.97E-13	1.62E-14	2.11E-16	1.57E-18	5.38E-21	5.23E-24	2.02E-28	0
94	1.18E-12	2.58E-14	3.7E-16	3.14E-18	1.31E-20	1.83E-23	2.52E-27	0
95	1.73E-12	4.04E-14	6.34E-16	6.07E-18	3.02E-20	5.73E-23	1.76E-26	0
96	2.5E-12	6.26E-14	1.07E-15	1.14E-17	6.66E-20	1.64E-22	9.02E-26	0
97	3.58E-12	9.55E-14	1.76E-15	2.09E-17	1.41E-19	4.35E-22	3.77E-25	1.26E-29
98	5.08E-12	1.44E-13	2.86E-15	3.74E-17	2.88E-19	1.09E-21	1.36E-24	1.64E-28

Table 9. Probability of having exactly 17-24 stuff-bits when data is 1-98 bit long.

B Number of combinations with exactly n stuff-bits when data is m bit long (where the total number of combinations for a m bit data is 2^m)

Number of bits in data	Number of combinations with exactly n stuff-bits							
	1	2	3	4	5	6	7	8
1-4	0	0	0	0	0	0	0	0
5	2	0	0	0	0	0	0	0
6	6	0	0	0	0	0	0	0
7	16	0	0	0	0	0	0	0
8	40	0	0	0	0	0	0	0
9	94	2	0	0	0	0	0	0
10	214	8	0	0	0	0	0	0
11	476	26	0	0	0	0	0	0
12	1040	76	0	0	0	0	0	0
13	2240	206	2	0	0	0	0	0
14	4772	530	10	0	0	0	0	0
15	10074	1314	38	0	0	0	0	0
16	21106	3166	126	0	0	0	0	0
17	43936	7456	382	2	0	0	0	0
18	90960	17238	1086	12	0	0	0	0
19	187418	39248	2946	52	0	0	0	0
20	384558	88214	7706	192	0	0	0	0
21	786168	196092	19576	640	2	0	0	0
22	1601952	431752	48552	1982	14	0	0	0
23	3254720	942724	118028	5812	68	0	0	0
24	6595304	2043340	282076	16332	276	0	0	0
25	13332818	4400076	664324	44342	1000	2	0	0
26	26894846	9419844	1544732	117020	3340	16	0	0
27	54144944	20060704	3551884	301534	10496	86	0	0
28	1.09E+08	42519268	8086356	761304	31444	380	0	0
29	2.18E+08	89732710	18247372	1888524	90622	1484	2	0
30	4.37E+08	1.89E+08	40850188	4613114	252922	5306	18	0
31	8.75E+08	3.95E+08	90796504	11116360	687018	17752	106	0
32	1.75E+09	8.25E+08	2E+08	26465658	1823310	56366	506	0
33	3.49E+09	1.72E+09	4.4E+08	62331028	4742396	171530	2116	2
34	6.95E+09	3.56E+09	9.61E+08	1.45E+08	12118760	503876	8052	20
35	1.38E+10	7.37E+09	2.09E+09	3.36E+08	30487844	1436542	28532	128
36	2.75E+10	1.52E+10	4.51E+09	7.71E+08	75637968	3991624	95572	656
37	5.46E+10	3.14E+10	9.72E+09	1.75E+09	1.85E+08	10845968	305802	2922
38	1.08E+11	6.45E+10	2.08E+10	3.97E+09	4.49E+08	28896770	941834	11778
39	2.15E+11	1.32E+11	4.45E+10	8.92E+09	1.08E+09	75658748	2808282	44016
40	4.26E+11	2.71E+11	9.48E+10	1.99E+10	2.56E+09	1.95E+08	8143114	154944
41	8.44E+11	5.54E+11	2.01E+11	4.43E+10	6.02E+09	4.96E+08	23045000	519462
42	1.67E+12	1.13E+12	4.26E+11	9.79E+10	1.41E+10	1.24E+09	63835000	1672034
43	3.31E+12	2.3E+12	8.99E+11	2.16E+11	3.26E+10	3.09E+09	1.73E+08	5198738
44	6.53E+12	4.68E+12	1.89E+12	4.73E+11	7.52E+10	7.58E+09	4.64E+08	15688292
45	1.29E+13	9.51E+12	3.97E+12	1.03E+12	1.72E+11	1.84E+10	1.22E+09	46123526
46	2.55E+13	1.93E+13	8.32E+12	2.24E+12	3.92E+11	4.44E+10	3.16E+09	1.33E+08
47	5.02E+13	3.91E+13	1.74E+13	4.86E+12	8.88E+11	1.06E+11	8.11E+09	3.73E+08
48	9.91E+13	7.91E+13	3.62E+13	1.05E+13	2E+12	2.52E+11	2.05E+10	1.03E+09
49	1.95E+14	1.6E+14	7.54E+13	2.26E+13	4.48E+12	5.93E+11	5.15E+10	2.8E+09
50	3.85E+14	3.23E+14	1.57E+14	4.85E+13	1E+13	1.39E+12	1.28E+11	7.5E+09
51	7.57E+14	6.51E+14	3.25E+14	1.04E+14	2.22E+13	3.23E+12	3.14E+11	1.98E+10
52	1.49E+15	1.31E+15	6.72E+14	2.22E+14	4.92E+13	7.46E+12	7.65E+11	5.17E+10
53	2.93E+15	2.64E+15	1.39E+15	4.72E+14	1.09E+14	1.71E+13	1.85E+12	1.33E+11
54	5.77E+15	5.31E+15	2.87E+15	1E+15	2.39E+14	3.92E+13	4.45E+12	3.4E+11
55	1.13E+16	1.07E+16	5.9E+15	2.13E+15	5.23E+14	8.93E+13	1.06E+13	8.59E+11
56	2.23E+16	2.14E+16	1.21E+16	4.5E+15	1.14E+15	2.02E+14	2.51E+13	2.15E+12
57	4.38E+16	4.3E+16	2.49E+16	9.49E+15	2.48E+15	4.57E+14	5.92E+13	5.33E+12

Table 10. Number of combinations with exactly 0-8 stuff-bits when data is 1-57 bit long.

Number of bits in data	Number of combinations with exactly n stuff-bits							
	1	2	3	4	5	6	7	8
58	8.59E+16	8.62E+16	5.12E+16	2E+16	5.39E+15	1.03E+15	1.39E+14	1.31E+13
59	1.69E+17	1.73E+17	1.05E+17	4.2E+16	1.17E+16	2.3E+15	3.23E+14	3.21E+13
60	3.31E+17	3.46E+17	2.15E+17	8.81E+16	2.52E+16	5.12E+15	7.48E+14	7.78E+13
61	6.49E+17	6.91E+17	4.38E+17	1.84E+17	5.42E+16	1.14E+16	1.73E+15	1.88E+14
62	1.27E+18	1.38E+18	8.95E+17	3.86E+17	1.16E+17	2.52E+16	3.96E+15	4.49E+14
63	2.5E+18	2.76E+18	1.83E+18	8.05E+17	2.49E+17	5.57E+16	9.05E+15	1.07E+15
64	4.89E+18	5.51E+18	3.72E+18	1.68E+18	5.33E+17	1.23E+17	2.06E+16	2.53E+15
65	9.59E+18	1.1E+19	7.57E+18	3.49E+18	1.14E+18	2.69E+17	4.67E+16	5.96E+15
66	1.88E+19	2.19E+19	1.54E+19	7.26E+18	2.42E+18	5.89E+17	1.06E+17	1.4E+16
67	3.68E+19	4.37E+19	3.13E+19	1.51E+19	5.15E+18	1.29E+18	2.38E+17	3.26E+16
68	7.2E+19	8.7E+19	6.34E+19	3.12E+19	1.09E+19	2.8E+18	5.33E+17	7.57E+16
69	1.41E+20	1.73E+20	1.29E+20	6.46E+19	2.31E+19	6.08E+18	1.19E+18	1.75E+17
70	2.76E+20	3.44E+20	2.61E+20	1.33E+20	4.89E+19	1.32E+19	2.66E+18	4.03E+17
71	5.4E+20	6.85E+20	5.28E+20	2.76E+20	1.03E+20	2.85E+19	5.91E+18	9.24E+17
72	1.06E+21	1.36E+21	1.07E+21	5.68E+20	2.17E+20	6.15E+19	1.31E+19	2.11E+18
73	2.06E+21	2.7E+21	2.16E+21	1.17E+21	4.57E+20	1.32E+20	2.89E+19	4.8E+18
74	4.04E+21	5.37E+21	4.36E+21	2.41E+21	9.59E+20	2.84E+20	6.38E+19	1.09E+19
75	7.89E+21	1.07E+22	8.8E+21	4.95E+21	2.01E+21	6.1E+20	1.4E+20	2.46E+19
76	1.54E+22	2.12E+22	1.77E+22	1.02E+22	4.21E+21	1.3E+21	3.07E+20	5.55E+19
77	3.01E+22	4.2E+22	3.58E+22	2.09E+22	8.81E+21	2.79E+21	6.73E+20	1.25E+20
78	5.89E+22	8.32E+22	7.2E+22	4.27E+22	1.84E+22	5.95E+21	1.47E+21	2.8E+20
79	1.15E+23	1.65E+23	1.45E+23	8.75E+22	3.84E+22	1.27E+22	3.2E+21	6.25E+20
80	2.25E+23	3.27E+23	2.92E+23	1.79E+23	8E+22	2.69E+22	6.95E+21	1.39E+21
81	4.39E+23	6.47E+23	5.86E+23	3.66E+23	1.66E+23	5.71E+22	1.51E+22	3.09E+21
82	8.57E+23	1.28E+24	1.18E+24	7.47E+23	3.46E+23	1.21E+23	3.26E+22	6.86E+21
83	1.67E+24	2.53E+24	2.37E+24	1.52E+24	7.18E+23	2.56E+23	7.05E+22	1.52E+22
84	3.26E+24	5.01E+24	4.75E+24	3.11E+24	1.49E+24	5.41E+23	1.52E+23	3.35E+22
85	6.37E+24	9.91E+24	9.53E+24	6.33E+24	3.09E+24	1.14E+24	3.27E+23	7.37E+22
86	1.24E+25	1.96E+25	1.91E+25	1.29E+25	6.39E+24	2.41E+24	7.04E+23	1.62E+23
87	2.42E+25	3.87E+25	3.83E+25	2.62E+25	1.32E+25	5.06E+24	1.51E+24	3.55E+23
88	4.73E+25	7.65E+25	7.67E+25	5.33E+25	2.73E+25	1.06E+25	3.23E+24	7.76E+23
89	9.22E+25	1.51E+26	1.53E+26	1.08E+26	5.63E+25	2.23E+25	6.92E+24	1.69E+24
90	1.8E+26	2.98E+26	3.07E+26	2.2E+26	1.16E+26	4.68E+25	1.48E+25	3.69E+24
91	3.51E+26	5.89E+26	6.14E+26	4.46E+26	2.39E+26	9.81E+25	3.15E+25	8.02E+24
92	6.84E+26	1.16E+27	1.23E+27	9.04E+26	4.92E+26	2.05E+26	6.71E+25	1.74E+25
93	1.33E+27	2.29E+27	2.45E+27	1.83E+27	1.01E+27	4.29E+26	1.43E+26	3.77E+25
94	2.6E+27	4.52E+27	4.9E+27	3.71E+27	2.08E+27	8.95E+26	3.03E+26	8.16E+25
95	5.06E+27	8.92E+27	9.79E+27	7.5E+27	4.27E+27	1.87E+27	6.42E+26	1.76E+26
96	9.87E+27	1.76E+28	1.95E+28	1.52E+28	8.75E+27	3.89E+27	1.36E+27	3.8E+26
97	1.92E+28	3.46E+28	3.9E+28	3.07E+28	1.79E+28	8.09E+27	2.88E+27	8.18E+26
98	3.74E+28	6.82E+28	7.77E+28	6.2E+28	3.67E+28	1.68E+28	6.08E+27	1.76E+27

Table 11. Number of combinations with exactly 0-8 stuff-bits when data is 58-98 bit long.

Number of bits	Number of combinations with exactly n stuff-bits							
in data	9	10	11	12	13	14	15	16
1-36	0	0	0	0	0	0	0	0
37	2	0	0	0	0	0	0	0
38	22	0	0	0	0	0	0	0
39	152	0	0	0	0	0	0	0
40	832	0	0	0	0	0	0	0
41	3930	2	0	0	0	0	0	0
42	16714	24	0	0	0	0	0	0
43	65644	178	0	0	0	0	0	0
44	242064	1036	0	0	0	0	0	0
45	847814	5170	2	0	0	0	0	0
46	2844270	23122	26	0	0	0	0	0
47	9198530	95150	206	0	0	0	0	0
48	28820970	366542	1270	0	0	0	0	0
49	87835110	1337798	6674	2	0	0	0	0
50	2.61E+08	4666882	31298	28	0	0	0	0
51	7.6E+08	15664902	134598	236	0	0	0	0
52	2.17E+09	50857094	540382	1536	0	0	0	0
53	6.08E+09	1.6E+08	2050750	8476	2	0	0	0
54	1.68E+10	4.93E+08	7423910	41574	30	0	0	0
55	4.56E+10	1.48E+09	25814542	186420	268	0	0	0
56	1.22E+11	4.35E+09	86686678	778388	1836	0	0	0
57	3.24E+11	1.26E+10	2.82E+08	3065608	10612	2	0	0
58	8.49E+11	3.57E+10	8.95E+08	11495900	54320	32	0	0
59	2.2E+12	9.97E+10	2.77E+09	41340858	253456	302	0	0
60	5.64E+12	2.75E+11	8.39E+09	1.43E+08	1098612	2172	0	0
61	1.43E+13	7.46E+11	2.49E+10	4.82E+08	4482608	13120	2	0
62	3.62E+13	2.01E+12	7.26E+10	1.57E+09	17384896	69946	34	0
63	9.04E+13	5.33E+12	2.08E+11	5.01E+09	64560430	338996	338	0
64	2.24E+14	1.4E+13	5.89E+11	1.56E+10	2.31E+08	1522846	2546	0
65	5.53E+14	3.64E+13	1.64E+12	4.76E+10	7.99E+08	6427516	16040	2
66	1.35E+15	9.39E+13	4.52E+12	1.42E+11	2.68E+09	25744200	88904	36
67	3.29E+15	2.4E+14	1.23E+13	4.19E+11	8.79E+09	98593988	446824	376
68	7.95E+15	6.09E+14	3.3E+13	1.21E+12	2.81E+10	3.63E+08	2077160	2960
69	1.91E+16	1.53E+15	8.79E+13	3.46E+12	8.79E+10	1.29E+09	9056444	19414
70	4.57E+16	3.83E+15	2.32E+14	9.75E+12	2.7E+11	4.47E+09	37413532	111690
71	1.09E+17	9.5E+15	6.05E+14	2.71E+13	8.13E+11	1.5E+10	1.48E+08	581264
72	2.57E+17	2.34E+16	1.57E+15	7.46E+13	2.41E+12	4.92E+10	5.59E+08	2792488
73	6.05E+17	5.74E+16	4.02E+15	2.03E+14	7.05E+12	1.58E+11	2.05E+09	12561300
74	1.42E+18	1.4E+17	1.03E+16	5.46E+14	2.03E+13	4.96E+11	7.26E+09	53460274
75	3.31E+18	3.39E+17	2.59E+16	1.46E+15	5.77E+13	1.53E+12	2.5E+10	2.17E+08
76	7.71E+18	8.16E+17	6.52E+16	3.84E+15	1.62E+14	4.65E+12	8.41E+10	8.45E+08
77	1.78E+19	1.96E+18	1.63E+17	1.01E+16	4.5E+14	1.39E+13	2.76E+11	3.17E+09
78	4.12E+19	4.67E+18	4.04E+17	2.62E+16	1.24E+15	4.09E+13	8.89E+11	1.15E+10
79	9.47E+19	1.11E+19	9.97E+17	6.75E+16	3.36E+15	1.19E+14	2.81E+12	4.08E+10
80	2.17E+20	2.63E+19	2.45E+18	1.73E+17	9.05E+15	3.4E+14	8.7E+12	1.4E+11
81	4.96E+20	6.18E+19	5.97E+18	4.39E+17	2.42E+16	9.63E+14	2.66E+13	4.72E+11
82	1.13E+21	1.45E+20	1.45E+19	1.11E+18	6.4E+16	2.7E+15	7.98E+13	1.55E+12
83	2.56E+21	3.39E+20	3.5E+19	2.79E+18	1.68E+17	7.48E+15	2.36E+14	5.01E+12
84	5.79E+21	7.89E+20	8.41E+19	6.95E+18	4.38E+17	2.05E+16	6.92E+14	1.59E+13
85	1.31E+22	1.83E+21	2.01E+20	1.73E+19	1.13E+18	5.58E+16	2E+15	4.95E+13
86	2.94E+22	4.23E+21	4.8E+20	4.26E+19	2.91E+18	1.51E+17	5.7E+15	1.52E+14
87	6.6E+22	9.75E+21	1.14E+21	1.05E+20	7.44E+18	4.02E+17	1.61E+16	4.59E+14
88	1.48E+23	2.24E+22	2.69E+21	2.56E+20	1.89E+19	1.07E+18	4.5E+16	1.37E+15
89	3.3E+23	5.13E+22	6.35E+21	6.21E+20	4.76E+19	2.81E+18	1.25E+17	4.02E+15

Table 12. Number of combinations with exactly 9-16 stuff-bits when data is 1-89 bit long.

Number of bits in data	Number of combinations with exactly n stuff-bits							
	9	10	11	12	13	14	15	16
90	7.35E+23	1.17E+23	1.49E+22	1.5E+21	1.19E+20	7.34E+18	3.42E+17	1.17E+16
91	1.63E+24	2.66E+23	3.48E+22	3.62E+21	2.98E+20	1.91E+19	9.3E+17	3.37E+16
92	3.62E+24	6.05E+23	8.11E+22	8.7E+21	7.39E+20	4.92E+19	2.51E+18	9.58E+16
93	8.01E+24	1.37E+24	1.88E+23	2.08E+22	1.83E+21	1.26E+20	6.72E+18	2.7E+17
94	1.77E+25	3.09E+24	4.36E+23	4.95E+22	4.49E+21	3.21E+20	1.78E+19	7.53E+17
95	3.9E+25	6.97E+24	1.01E+24	1.17E+23	1.1E+22	8.13E+20	4.71E+19	2.08E+18
96	8.57E+25	1.57E+25	2.32E+24	2.78E+23	2.67E+22	2.05E+21	1.23E+20	5.71E+18
97	1.88E+26	3.51E+25	5.32E+24	6.54E+23	6.48E+22	5.13E+21	3.21E+20	1.55E+19
98	4.12E+26	7.85E+25	1.22E+25	1.53E+24	1.57E+23	1.28E+22	8.3E+20	4.19E+19

Table 13. Number of combinations with exactly 9-16 stuff-bits when data is 90-98 bit long.

Number of bits in data	Number of combinations with exactly n stuff-bits							
	17	18	19	20	21	22	23	24
1-68	0	0	0	0	0	0	0	0
69	2	0	0	0	0	0	0	0
70	38	0	0	0	0	0	0	0
71	416	0	0	0	0	0	0	0
72	3416	0	0	0	0	0	0	0
73	23286	2	0	0	0	0	0	0
74	138846	40	0	0	0	0	0	0
75	747228	458	0	0	0	0	0	0
76	3705264	3916	0	0	0	0	0	0
77	17175924	27702	2	0	0	0	0	0
78	75227536	170962	42	0	0	0	0	0
79	3.14E+08	950266	502	0	0	0	0	0
80	1.26E+09	4858110	4462	0	0	0	0	0
81	4.84E+09	23182964	32710	2	0	0	0	0
82	1.8E+10	1.04E+08	208678	44	0	0	0	0
83	6.52E+10	4.47E+08	1196618	548	0	0	0	0
84	2.3E+11	1.83E+09	6300578	5056	0	0	0	0
85	7.9E+11	7.25E+09	30921548	38360	2	0	0	0
86	2.66E+12	2.77E+10	1.43E+08	252686	46	0	0	0
87	8.76E+12	1.02E+11	6.29E+08	1493268	596	0	0	0
88	2.83E+13	3.69E+11	2.64E+09	8089948	5700	0	0	0
89	9E+13	1.3E+12	1.07E+10	40795810	44704	2	0	0
90	2.81E+14	4.45E+12	4.18E+10	1.94E+08	303732	48	0	0
91	8.67E+14	1.5E+13	1.58E+11	8.73E+08	1848000	646	0	0
92	2.63E+15	4.94E+13	5.82E+11	3.76E+09	10292084	6396	0	0
93	7.9E+15	1.6E+14	2.09E+12	1.56E+10	53284330	51796	2	0
94	2.34E+16	5.1E+14	7.32E+12	6.21E+10	2.59E+08	362618	50	0
95	6.85E+16	1.6E+15	2.51E+13	2.4E+11	1.2E+09	2269456	698	0
96	1.98E+17	4.96E+15	8.45E+13	9.04E+11	5.28E+09	12982350	7146	0
97	5.68E+17	1.51E+16	2.79E+14	3.31E+12	2.23E+10	68950550	59692	2
98	1.61E+18	4.56E+16	9.07E+14	1.18E+13	9.12E+10	3.44E+08	430204	52

Table 14. Number of combinations with exactly 17-24 stuff-bits when data is 1-98 bit long.