Weakly Hard Real-time Constraints on Controller Area Network

Ian Broster, Guillem Bernat and Alan Burns
Real-Time Systems Research Group,
Department of Computer Science, University of York, UK
{ianb,bernat,burns}@cs.york.ac.uk

Abstract

For priority based buses such as CAN, worst case response time analysis is able to determine whether messages always meet their deadlines. This can include system models with bounded network faults. However, the worst-case scenario (the critical instant) used by the analysis is extremely pessimistic compared to the rest of the invocations in the hyperperiod. We use weakly hard constraints to provide upper bounds on the maximum number of missed deadlines under fault conditions. By allowing some deadlines to be missed (around the critical instant) the weakly-hard schedulability of the system can be guaranteed at much higher levels of faults. This paper presents a response time based formulation that provides a guarantee on the weakly-hard schedulability of messages. Simulation results based on CAN and the Latest Send Time-CAN protocol show that because of the pessimism of the approach, in fact almost all messages meet their deadlines.

1. Motivation and Outline

The aim of the work is to show that an event-triggered priority based bus such as CAN is able to provide reliable communication in real-time systems with faults. This paper draws from two areas of previous work: *weakly hard real-time systems* [2, 1] and a variation of the CAN [5] protocol called *latest send time CAN* [6]. We proceed as follows:

1. Firstly, hard worst case response time analysis of CAN is used to guarantee that all deadlines are met in the presence of known worst case faults.
2. Next, weakly hard analysis is applied to CAN to show that where occasional deadline misses can be tolerated, guarantees are possible for far greater levels of faults.
3. Simulations are used to show that weakly hard response time with faults is conservative: the number of missed deadlines is significantly lower than the guaranteed level.
4. LST-CAN is used to further reduce the number of missed deadlines and guarantee that the system is *predictable* even if faults exceed the level used in analysis.

Weakly hard real-time systems concerns the specification and analysis of real-time systems where a *specified* number of deadlines may be missed. The motivation for a weakly hard system is that in practical engineering contexts, the occasional missed deadline can be tolerated. For example, control loops are generally able to work effectively if a small number of data values are lost (vacant sampling) [11]. However, it is still important to be able to guarantee that *sufficient* data values are delivered.

A weakly hard system is stronger than a soft real-time system since in a weakly hard system, we may specify exactly how many deadlines may be missed in the worst case. For example: "*in any 20 consecutive deadlines the process must always meet at least 18 of them and it must never miss any 2 consecutively*". Weakly hard response time analysis may then be used to show that all the weakly hard constraints are met. This is covered in more detail in section 3.

LST-CAN is an extension to the controller area network (CAN) protocol that provides a *predictable* real-time protocol where guarantees can be made about the timeliness of messages, even under *unspecified* levels of network faults (caused by electrical interference, poor connectors *etc.*). In this paper, we use LST-CAN at run-time to guard against unanticipated network faults.

We assume that we cannot know *a priori* the worst case level of network faults. Therefore in the presence of unpredictable network interference, normal CAN provides no *guarantees* about real-time behaviour because faults can introduce unknown delays onto the bus. The LST-CAN protocol considers that late messages (which have no value) can be abandoned in order to release bandwidth on the bus so that the bus can *catch up* and hence meet future deadlines. Generally, only a *very* small number of frames are lost, even when the faults are *much greater* than the fault model used for analysis. LST-CAN is explained in more detail in section 4.

1.1. Link

The link between weakly-hard systems and LST-CAN is that they both exploit the fact that in an event triggered system, tasks (or message frames) tend to suffer different levels of interference on each invocation. The interference depends on the relative periods of higher priority tasks (frames). It can be seen by measurement, simulation and analysis that where the periods of each task are non-harmonic there is huge variation in the response times for each task. In general, the average case response time is *much* lower than the worst case response time.

As an example of this variation, figure 1 shows the distribution of worst case response times for a frame on a CAN bus from the simulations later in this paper. The frame is priority 2 (low) of the task set described in table 1, assuming no faults. The worst case response time is 23.016ms but as can be clearly seen, the worst case at other invocations within the hyperperiod is much shorter. Notice that there are only a relatively small number of peaks.

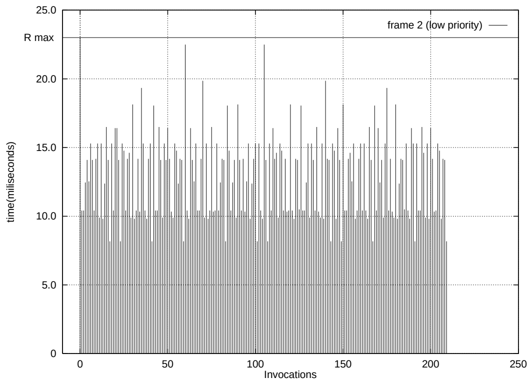


Figure 1. Typical Distribution of response times (Max $R_i = 23.016ms$)

LST-CAN exploits the same property of event triggered systems. Instances of a frame suffer more interference (from higher priority frames) at the critical instant. Where faults also affect frames near the critical instant, frames may be pushed beyond the worst case response time. However, away from the critical instant, the same pattern of faults may not be sufficient to delay the frames beyond the worst case response time. Therefore, although the system may be able to function safely with some faults for most of the time, when the faults occur near the critical instant there is insufficient time to recover from them. It is these messages near the critical instant that the LST-CAN protocol does not transmit. As explained previously this only happens rarely.

1.2. Model and Nomenclature

We consider a networked real-time system where nodes send messages through a bus. The system consists of periodic messages which are characterised by the following parameters: T_i, the period; D_i, the deadline ($D_i \leq T_i$); B_i, the worst case blocking time, this is the maximum time a message may need to wait due to a lower priority message on the bus; J_i is the worst case jitter of message release times; C_i, the worst case transmission time, not including the inter-frame space; P_i, the priority of the message, τ is one bit time (8 μs at the bit rate used later in this); S is the time to transmit an inter-frame space (3 bit times).

Note that we use a higher number to indicate a higher priority, *e.g.* priority 1 is the lowest priority. The pattern of release times of all messages is repeated every *hyperperiod* which is $H = \mathsf{lcm}(T_i)\forall i$ units long. A message at priority P_i will only suffer interference from higher priority messages. Therefore the pattern of releases repeats every *hyperperiod at level* i (given by $H_i = \mathsf{lcm}(T_i)\forall i \in \mathsf{hep}(i)$, where $\mathsf{hep}(i)$ is the set of messages of higher or equal priority to M_i). The number of invocations of a message in a hyperperiod is given by $A_i = H/T_i$, and the number of invocations of a message in the hyperperiod at level i is $a_i = H_i/T_i$.

We require a mechanism to keep local clocks synchronised within a known upper bound to ensure that the pattern of release times is repeated cyclically with a maximum difference between real and theoretical release times bounded by the maximum clock drift. For instance three techniques are discussed in [20, 9, 16]. It is reasonable to assume a maximum clock drift of 16 μs, (2 bit times). However, for simplicity this paper assumes that there is perfect clock synchronisation.

2. CAN

The Controller Area Network (CAN) protocol [5, 10] is a bus protocol with many desirable properties for embedded and real-time systems. CAN is inexpensive and widely used in factory automation and in vehicles.

2.1. Hard Real-time Analysis

CAN is a priority based protocol where collisions are avoided by using priorities for bus arbitration. In the absence of faults, worst case timing analysis of CAN is straightforward [19]. CAN is similar to a non-preemptive scheduler, allowing worst case response time analysis to be performed for each frame in the same way as can be done for non-preemptable processes. The following equations for the worst case transmission times R_i are based on the response time analysis in [19] and [6] .

$$R_i = J_i + t_i \tag{1}$$

$$t_i = B_i + C_i + I_i(t_i) \tag{2}$$

$$B_i = \max_{\forall k \in \mathsf{lp}(i)} (C_k) + S \tag{3}$$

I_i is the worst case interference from higher priority frames, given by:

$$I_i(t) = \sum_{j \in \mathsf{hp}(i)} \left\lceil \frac{t - C_i + J_j + \tau}{T_j} \right\rceil \hat{C}_j \tag{4}$$

where $\hat{C}_i$, is $C_i + S$, *i.e.* the time to transmit a frame and one inter-frame space, $\mathsf{lp}(i)$ is the set of messages with lower priority than i and $\mathsf{hp}(i)$ is the set of messages with higher priority than i. Equation (2) may be solved iteratively by forming a recurrence relation with $t_i^0 = C_i$ which terminates when $t_i^{n+1} = t_i^n$ or fails when $t_i^{n+1} > D_i - J_i$ where $D_i \leq T_i$.

If there is a solution R_i $\forall i$ and $R_i \leq D_i$ then the analysis above will guarantee that all messages will always meet their deadlines, provided that there are no faults.

2.2. Faults

In normal CAN, we may assume that all network faults are detected[7]. Any corrupted frames are re-queued for transmission: when the bus is idle again, they compete for access to the bus using the same priorities. The effect of network faults may be included in response time analysis [19, 14]. In any given time interval, t, if a value for the worst case overhead due to network faults and extra frames can be calculated, $E_i(t)$, it can be included in worst case response time analysis equation (2) as shown in equation (5) .

$$t_i = B_i + C_i + I_i(t_i) + E_i(t_i) \tag{5}$$

A simple fault model [19] considers single-bit errors that have a minimum inter-arrival time, T_f. Therefore an expression for the worst case overhead due to errors during an interval t is

$$E_i(t) = \left\lceil \frac{t}{T_f} \right\rceil \max_{k \in \mathsf{hep}(i)} (C_k + E + S) \tag{6}$$

where E, is the longest possible error frame from a single error.

This will guarantee that all deadlines are always met provided that errors do not exceed the model specified in $E_i(t)$.

However, the problems with using this technique are:

1. it is hard to determine a realistic function $E_i(t)$ for network faults;
2. always assuming the worst case faults is very pessimistic: so much spare bandwidth must be reserved that there is very little available for normal messages;
3. at run time, there is no guarantee that the faults will actually conform to the assumptions used for analysis.

In this paper, we do not attempt to address the first point. More accurate fault models have been proposed [14], but it is necessary to move to a probabilistic $E_i(t)$ for more realistic modelling. Solutions to the second and third problems appear in the following sections.

3. Weakly-hard Constraints

In the previous section we showed that schedulability tests exist that can determine whether or not under a particular fault model all messages will arrive on time. However, the assumption that all deadlines have to be met is unrealistic. Further, the worst case scenario on which the schedulability test is based is pessimistic as it happens very rarely. By assuming that a bounded number of deadlines can be missed, the schedulability of the system can be significantly increased.

For this purpose we use weakly-hard constraints to specify the tolerable number of missed messages. A weakly-hard constraint specifies a guaranteed upper bound on the maximum number of missed deadlines (late messages) during a window of time. Four basic constraints are defined:

$\binom{n}{m}$ for any m consecutive messages, at least n messages must arrive and arrive on time (by the deadline). No constraint is put on whether these n messages arrive consecutively or non-consecutively.

$\overline{\binom{n}{m}}$ for any m consecutive messages, at most n deadlines can arrive late (or not arrive at all). This is the dual of the previous constraint. $\binom{n}{m}$ is equivalent to $\overline{\binom{m-n}{m}}$.

$\left\langle {n \atop m} \right\rangle$ for any m consecutive messages, at least n *consecutive* messages need to arrive and on time. This is clearly more demanding than just meeting n in a window.

$\overline{\left\langle {n \atop m} \right\rangle}$ it is not the case that more that n consecutive messages are missed (or not sent) in a row in m consecutive invocations. The parameter m is redundant, we could also write $\overline{\langle n \rangle}$, which is equivalent to $\overline{\left\langle {n \atop n} \right\rangle}$.

3.1. μ-patterns

We represent a message arriving on time with a 1 and a message arriving late or not being sent as a 0. Therefore we can characterise all the invocations of a message stream by a string of zeros and ones. We call this string the message μ-pattern and we denote it by $\mu_i(k)$ where $k = 0, 1, 2, \ldots$ is the invocation number of message i. For example 01111111 means that the first deadline was missed and then seven deadlines were met.

Two interpretations of the μ-pattern are possible:

- a past μ-pattern describes the past execution of the task, for example the result of a simulation run or the run-time monitoring of a system;
- a future μ-pattern describes the *worst-case* possible behaviour of the future invocations of the message.

One important result is that the worst case future μ-pattern is made up of the cyclic repetition of a μ-pattern of length p where p is the number of invocations in the hyperperiod. Therefore, checking whether a constraint is satisfied requires checking p windows of size m including wrapping around the ends of the pattern. This means that the worst case behaviour of a particular message stream (even if the stream is infinitely long) can be fully described by a finite μ-pattern.

3.2. Analysing μ-patterns

Weakly-hard constraints provide a powerful mechanism to capture the resilience of a task to missed and met deadlines. Given a system we shall be interested in determining the following properties of the system:

P1 — Given a weakly-hard constraint λ and the past μ-pattern α, determine whether the constraint is satisfied.

P1' — If P1 is not satisfied, determine how many times the constraint is not satisfied.

P2 — Given a μ-pattern α, determine the set of pairs n, m so that the constraint $\binom{n}{m}$ (or $\left\langle {n \atop m} \right\rangle$, $\overline{\binom{n}{m}}$, $\overline{\langle n \rangle}$) are satisfied.

P2' — Determine how many times each constraint $\binom{n}{m}$ (or $\left\langle {n \atop m} \right\rangle$, $\overline{\binom{n}{m}}$, $\overline{\langle n \rangle}$) is satisfied / not satisfied for any pair of values n, m.

P2' is particularly interesting as it provides an insight of the resilience of the system to faults. If the pattern under analysis is a past μ-pattern (from a very long simulation or monitoring of the execution of the system) it gives a detailed insight of not only how many messages miss their deadlines, but also how spread-out they are. For future μ-patterns P2' describes a guaranteed worst case behaviour.

Note that checking the satisfiability of a μ-pattern of length p against a weakly hard constraint can be done with simple algorithms of cost $O(p)$. For further details on properties of weakly-hard constraints and algorithms for solving P1-P2 see [3], [4] and [1].

4. Latest Send Time CAN

The *latest send time* extension[6] to CAN was introduced to provide predictable real-time behaviour in the presence of unknown and unspecified network faults. LST-CAN shifts the focus of error handling from *reliability of delivery* to *timeliness of delivery*: messages are delivered reliably only when time allows in a best-effort manor. Timeliness becomes *independent* of errors. Reliability of delivery therefore becomes *dependent* on the effect of any unanticipated faults. However, due to the variation of interference in event triggered systems, only a very small fraction of messages need to be aborted, making LST-CAN far more tolerant to faults than a time-triggered bus (for example TTCAN[8] where messages only have one shot at transmission.

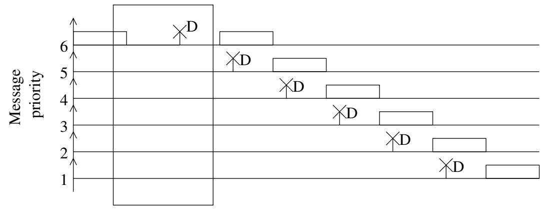
The LST-CAN protocol works by not sending any late messages in order to release bandwidth on the bus so that the bus can *catch up* and hence more meet future deadlines. *Never attempting to send a frame if it cannot arrive in time* ensures that the bus does not 'waste' time transmitting a message that is not useful. It guarantees that the bus can never lag behind such that messages are queued for indefinite times and therefore ensures that buffer overflow cannot occur due to faults on the network.

Figure 2 provides an example of how the protocol can ensure the deadlines of other messages on the bus after a continuous block of interference. In 2a, a burst of network interference prevents any message from appearing correctly on the bus for some time, *all* messages are delayed and arrive after their deadlines. Figure 2b shows how the protocol deals with errors: messages 6 and 5 cannot possibly arrive before **D**, therefore they are abandoned, allowing all other messages to arrive on time.

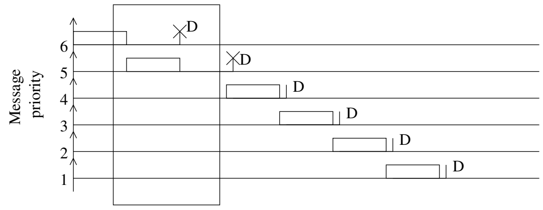
LST-CAN requires the application to specify a *latest send time*, $L_{i,k}$, for each message frame, which marks the latest time that transmission may begin for that frame. Until $L_{i,k}$, the protocol works exactly as for normal CAN, with arbitration and retransmissions. If a frame has not started transmission by $L_{i,k}$ then the frame is removed from the transmission queue, so is never sent. There are several possible policies for setting $L_{i,k}$. Useful values are between the worst case response time and the deadline. For example:

$$L_{i,k} = O_i + kT_i + D_i - C_i \tag{7}$$

LST-CAN is effective at keeping the bus running 'up to speed' and reducing the number of deadlines missed, even for extremely high levels of faults that far exceed the fault



a. Network error delays all messages



b. Network error loses 2 messages, all others on time

Figure 2. Example of losing messages in order to maintain other deadlines

model used for analysis. We use LST-CAN in this paper as a run-time guard to ensure that the system is predictable even under unspecified fault conditions.

5. Weakly-hard Analysis of CAN

Section 3 introduced the concept of μ-patterns and weakly-hard constraints. Past μ-patterns can be obtained from simulation or monitoring of a system, however they are not guaranteed upper bounds. In order to provide levels of guarantees, determination of the worst case future μ-pattern needs to be performed.

This is done by computing the worst case response time, $R_{i,k}$, for each frame at each possible invocation k, $k = 0, 1, 2, \ldots$, under some fault model. This is done with an extension of the worst-case response time formulation technique for several invocations[2] which has been updated to consider interference from CAN error frames and consider the non-preemptive nature of CAN data frames.

One important result is that not only is the pattern of releases repeated every hyperperiod at level i, but also the worst case response times are repeated cyclically with a period equal to a_i invocations. This result holds as long as the assumption of a synchronisation between nodes holds. The interference at invocations $k, k + a_i, k + 2a_i, \ldots$ is effectively the same. Therefore we only need to compute the worst case response time at invocations $k = 1, \ldots a_i$. In fact, as there may be overrun after the first hyperperiod, the worst case response time is actually $R_{i,k}$ for the invocations in the *second* hyperperiod (or any hyperperiod, except the first one). This can be found by computing $R_{i,k}$ in the first hyperperiod at level i and then examining r invocations into the second hyperperiod at level i until $R_{i,r} \leq R_{i,a_i+r}$.

5.1. Computing $R_{i,k}$

The computation of the exact worst case response time of a message i at invocation k for the general case is an NP-hard problem [15]. The conditions that lead to the worst case at a particular invocation k may not be the ones that lead to the worst case at invocation $k+1$, in order to find the exact worst case response time all possible invocation of all tasks release times shall need to be considered, which is clearly infeasible.

For this reason we aim to compute an upper bound on the worst case response time at invocation $R_{i,k}$ by determining the worst case scenario at invocation k independently of the other invocations. We repeat this analysis for all invocations of the frame within the hyperperiod at level i. The worst case response time at invocation k of frame i happens when the frame receives the maximum blocking from lower priority frames at the release $S_{i,k}$ ($S_{i,k} = kT_i$ for $k = 0, 1, \ldots$), and the maximum number of faults hit that frame. So, $R_{i,k} = w + J_i - S_{i,k}$ where w is the smallest integer greater than zero which is a solution to the fixed-point equation (8).

$$w = k\hat{C}_i + kB_i + \delta_{i,k} + I_i(w) + E_{i,k}(w) \qquad (8)$$

where $\delta_{i,k}$ is the amount of idle time at level i between 0 and $S_{i,k}$. w can be found using a recurrence relation, as described in section 2.

The amount of idle time at level i can be computed by introducing a virtual pre-emptable task $\bar{\tau} = (\bar{T}, \bar{D}, \bar{C})$ with period $\bar{T} = \bar{D} = S_{i,k}$, at a priority level just below i and finding the largest $\bar{C}$ that makes the task schedulable (see figure 3):

$$\delta_{i,k} = \max\{\bar{C} | \bar{R}(\bar{C}) \leq S_{i,k}\}$$

where $\bar{R}(\bar{C}) = u$ is the solution to the fixed-point equation:

$$u = (k-1)\hat{C}_i + (k-1)B_i + \bar{C} + \sum_{j \in \mathsf{hep}(i)} \left\lceil \frac{u + J_j}{T_j} \right\rceil \hat{C}_j + E_{i,k}(u) \qquad (9)$$

and where the error function is such that an error hits exactly at $S_{i,k}$:

$$E_{i,k}(w) = \left\lceil \frac{w + f}{T_f} \right\rceil \max_{k \in \mathsf{hep}(i)} (C_k + E + S) \qquad (10)$$

where

$$f = S_{i,k} - \left\lfloor \frac{S_{i,k}}{Tf} \right\rfloor T_f \qquad (11)$$

Note that $\delta_{i,k}$ in equation (8) does not depend on w, therefore it does not need to be reevaluated at every iteration.

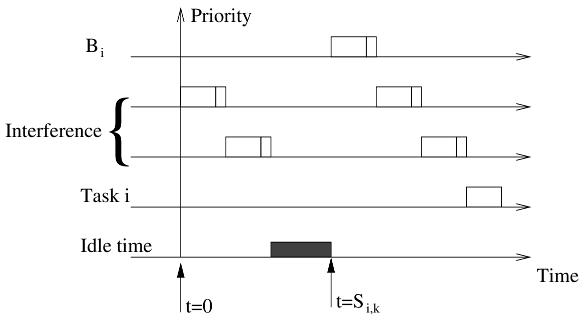


Figure 3. Virtual task to calculate the idle time

6. Evaluation

In order to show the benefits of using weakly hard constraints on CAN, a suitable task set was generated based on a well known benchmark and the following were done in logical progression:

- Hard worst case response time analysis was performed at various fault levels. At high levels of faults, the analysis indicates that no guarantees can be given.
- Weakly hard worst case response time analysis was performed at the same levels of faults. This provides guarantees for a minimum number of frames at much higher fault levels.
- Simulation of CAN bus and μ-pattern analysis. This is compared to the weakly hard worst case response time analysis guarantees and shows that the weakly hard analysis is still pessimistic.
- Simulation of LST-CAN and μ-pattern analysis. This is compared to the normal CAN simulations and shows that LST-CAN outperforms CAN, especially at very high levels of faults.

6.1. Message Set

The Society of Automotive Engineers (SAE) produced a benchmark [18] of messages that might be transmitted within a vehicle. The SAE benchmark has messages with harmonic periods. Weakly hard analysis of the SAE benchmark does indeed show considerable improvement over hard response time analysis. However, it is clear that the periods of the messages have been designed to be harmonic (perhaps in order to facilitate cyclic scheduling). In an event triggered system, there is no need to require the periods to be so constrained, indeed it is better if they are not. Therefore, we have exploited the the flexibility of event-triggered systems and altered the benchmark slightly such that the periods of the messages are not as harmonic.

The reason why non-harmonic messages produce better results in an event-triggered system, especially with weakly hard analysis is straightforward: the worst case response time occurs when all the messages are released close together so that the messages suffer maximum interference from higher priority messages. This occurs once per hyperperiod. If the periods are non-harmonic, then the hyperperiod is much larger, therefore the worst case (and other bad cases) occur infrequently.

This message set used to demonstrate the concepts in this paper appears in table 1. It provides a bus utilisation of about 70% with a bus speed of 125kbits/second. The utilisation is fairly high and therefore provides a good basis for fault injection. The rightmost column of the table (R_i) refers to the (hard) worst case response time, calculated from equations (1) and (2) without any faults.

6.2. Hard Worst Case Response Time Analysis

Hard worst case response time analysis was performed on the task set, using equations (1) and (5). Fault models

Table 1. CAN messages used for simulation

Pri	Bytes	C_i (ms)	T_i (ms)	J_i (ms)	D_i (ms)	R_i (ms)
17	1	0.496	1000.0	0.2	4.0	1.616
16	2	0.576	4.5	0.2	4.5	2.216
15	1	0.496	5.0	0.2	5.0	2.736
14	2	0.576	6.0	0.2	6.0	3.336
13	1	0.496	8.0	0.2	8.0	3.856
12	2	0.576	9.0	0.2	9.0	4.456
11	6	0.896	10.0	0.2	10.0	5.216
10	1	0.496	12.0	0.2	12.0	7.456
9	2	0.576	14.0	0.2	14.0	8.056
8	2	0.576	16.0	0.2	16.0	9.176
7	1	0.496	18.0	0.2	18.0	12.336
6	4	0.736	120.0	0.3	120.0	14.236
5	1	0.496	140.0	0.3	140.0	16.476
4	1	0.496	160.0	0.3	160.0	18.116
3	3	0.656	1000.0	0.4	1000.0	18.736
2	1	0.496	1200.0	0.4	1200.0	23.016
1	1	0.496	1400.0	0.4	1400.0	23.040

Table 2. Hard Response Time Analysis

	R_i in ms, varying the faults per Second						
Pri	0	60	80	160	200	320	640
17	1.616	2.360	2.360	2.360	2.360	2.360	3.104
16	2.216	3.040	3.040	3.040	3.040	3.040	4.688
15	2.736	3.560	3.560	3.560	3.560	4.384	7.456
14	3.336	4.160	4.160	4.160	4.160	4.984	+
13	3.856	4.680	4.680	4.680	4.680	*8.048	+
12	4.456	6.400	6.400	6.400	7.824	*9.168	+
11	5.216	8.080	8.080	9.744	9.744	+	+
10	7.456	9.120	9.120	*15.248	*17.432	+	+
9	8.056	12.360	12.360	*17.408	*18.552	+	+
8	9.176	15.280	*17.464	*22.992	*29.840	+	+
7	12.336	16.320	*20.384	*29.816	*39.848	+	+
6	14.236	23.124	23.124	36.540	69.452	+	+
5	16.476	24.244	24.244	47.668	69.972	+	+
4	18.116	26.924	29.868	48.188	79.900	+	+
3	18.736	27.544	30.488	48.808	89.928	+	+
2	23.016	29.864	34.768	60.456	107.624	+	+
1	23.040	29.888	34.792	60.480	107.648	+	+

*=Missed Deadline
+= Unbounded

of $f \in \{0, 60, 80, 160, 200, 320, 640\}$ single bit errors per second were used, assuming a minimum inter-arrival time $T_F = \frac{1}{f}$.

This fault model (minimum inter-arrival time) is not realistic: in real life, faults occur unpredictably. However, we justify the use of this fault model in the analysis as follows:

- recall from section 2 that it is difficult to find a more suitable function $E_i(t)$ because the very nature of faults is that they are unpredictable; this is a simple, understandable model;
- it is impossible to give a *worst case* response time without using a fault model with similar bounding restrictions (such as minimum inter-arrival time, or a limit in a period of time); using a non-bounded fault model means that the worst case is that no message ever arrives);
- later, we use LST-CAN as protection against the faults at run-time exceeding the level of faults used in the fault model for analysis.

The hard worst case response times are shown in table 2. Note that at higher fault levels, many of the frames are not schedulable (they would arrive after their deadlines or not at all).

Using the formulation given in section 2 it can be shown that the system becomes unschedulable for error rates larger than 62 faults/second ($T_f = 16.120$ms). In that case, frame 7 misses its deadline in the worst case.

6.3. Weakly Hard Analysis

Although the system is not strongly-hard schedulable for faults rates greater than 62 faults/sec it is weakly-hard schedulable. For the same fault models μ-patterns of the frames were produced. Table 3 shows the percentage of times the constraint $\binom{n}{m}$ is not satisfied in the worst case for frame 7 under a fault rate of 200 errors per second. Even under these extreme error conditions, only 22% of the deadlines can be missed, moreover, there are no $\binom{1}{3}$ missed, which means that no 3 consecutive deadlines will be missed.

Table 3. Percentage of missed weakly hard constraints $\binom{n}{m}$ by analysis,

	Frame 7 at 200 faults/sec.									
m	n=1	2	3	4	5	6	7	8	9	10
1	22.3									
2	3.2	41.4								
3	0.0	9.3	57.7							
4	0.0	1.0	18.8	69.4						
5	0.0	0.0	2.1	29.4	80.5					
6	0.0	0.0	0.0	4.8	41.0	88.2				
7	0.0	0.0	0.0	0.2	10.3	53.7	92.2			
8	0.0	0.0	0.0	0.0	1.0	16.9	65.1	95.8		
9	0.0	0.0	0.0	0.0	0.0	1.9	27.6	72.1	99.4	
10	0.0	0.0	0.0	0.0	0.0	0.1	6.4	37.7	79.2	100.0

Weakly hard analysis allows certain schedulability guarantees at much higher levels of faults than hard analysis (contrast table 2 and table 3). However, the pessimism of the analysis is still high as the following section that presents results based on simulation shows.

The pessimism of the analysis comes from two factors. One is the assumption that a frame receives a blocking at each invocation. In this sense, the analysis is equivalent to a message set in which there is one additional message in the bus at higher priority. The other source of pessimism is the fault model. The analysis assumes that if n errors may hit a window of time w, then those errors will always hit the longest message and just at the last bit. This is not only pessimistic, but in most cases impossible, This assumption is made in order to keep the analysis tractable.

6.4. Simulation

A software CAN bus simulator has been constructed which simulates messages and errors on a CAN bus. The simulator also implements the LST-CAN protocol. The same message set was simulated for a period of 18000 seconds. This is approximately 71 hyperperiods of the message set. For LST-CAN, L_i was set according to the deadline of the frame, as in equation (7).

Faults were injected onto the bus according to a random Poisson error distribution. Note that this is different to the bounded fault model used for analysis in section 6.2. One aim of the simulation is to see how robust the network can be to the faults varying from the simple fault model.

The simulations showed:

- nearly all guarantees made by the weakly hard analysis held; note that where the constraints were broken, it was because the simulations used a random distribution of errors yet the analysis assumed bounded faults. The constraints were broken only occasionally (when there were several faults close together).
- in most cases, the weakly hard guarantees were pessimistic.

For comparison, the simulation results were converted to μ-patterns and analysed for weakly hard constraints, see table 4. The third column in table 4 is produced by weakly-hard analysis; it shows the values of n such that $\binom{n}{25}$ is satisfied for a fault rate of 200 faults/sec. Note that only frames 7 is only able to guarantee $\binom{5}{25}$, yet the worst observed case was $\binom{22}{25}$. Further analysis of the μ-patterns (not shown in this paper) shows that no two consecutive deadlines can be missed in a row for any of the frames.

As can be seen, the CAN simulation usually performs better than the weakly hard analysis. Where the simulations do break a constraint from the weakly-hard analysis, an entry is shown in the table as a percentage of the number of times that the constraint was not satisfied per the number of times the constraint was evaluated[1]. The number of times that the constraints are broken is extremely small. This indicates that the simple bounded fault model is not wholly inappropriate. At low fault rates, the LST-CAN performs very similarly to normal CAN (as measured by weakly hard constraints). Minor differences are due to the random nature of the fault injection.

At higher error rates, *e.g.* 320, 640 errors/second, the weakly-hard analysis could guarantee very few deadlines, see table 5. However, the simulations show that due to the pessimism of the analysis, many messages are still schedulable. At these very high fault rates the LST-CAN protocol shows considerable improvement over CAN. For example, at 640 errors/second, messages 9 and below are queued more often than they can be transmitted. Under CAN, the whole network begins to lag and no messages below priority 9 arrive on time and no message below priority 5 ever arrives at all. LST-CAN does not attempt to send messages that are late and therefore prevents excessive delays on the bus. As can be seen from table 5, LST-CAN does ensure that a very large number of messages actually arrive on time. The worst affected message was priority 7 which in the worst case only managed 5 in a window of 25 messages. On average, 60% of priority 7 frames arrived on time using LST-CAN. This is considerably better than CAN, where the same message suffered unbounded queueing.

[1]Note that where there is one single deadline missed in the whole simulation run, this corresponds to 25 constraint failures. This is because the constraint will be tested in all possible positions—the size of the constraint window is 25 frames and therefore it can be placed in 25 different ways over the one missed deadline, one missed deadline is counted a total of 25 times.

Table 4. Comparison of weakly-hard constraints and simulations at 200 Errors/sec

P_i	Analysis		Simulation		Simulation	
	Hard	Weak	CAN		LST CAN	
	R_i (ms)	$\binom{n}{25}$	$\binom{n}{25}$	% missed	$\binom{n}{25}$	% missed
17	2.360	25	25		25	
16	3.040	25	25		25	
15	3.560	25	24	0.002	24	0.002
14	4.160	25	24	0.050	24	0.040
13	4.680	25	25		24	0.002
12	7.824	25	24	0.040	24	0.031
11	9.744	25	24	0.578	24	0.590
10	*17.432	21	23		23	
9	*18.552	20	24		23	
8	*29.840	17	23		23	
7	*39.848	5	22		22	
6	69.452	25	25		25	
5	69.972	25	25		25	
4	79.900	25	25		25	
3	89.928	25	25		25	
2	107.624	25	25		25	
1	107.648	25	25		25	

*=Missed Deadline

'% missed' columns show the percentage of missed deadlines (or lost frames for LST-CAN) only when the simulation gave worse results than the analysis. This is due to the more realistic probabilistic fault model used by the simulator.

Table 5. Comparison of weakly-hard constraints at 640 Errors/sec

P_i	Analysis		Simulation		Simulation	
	Hard	Weak	CAN		LST CAN	
	R_i (ms)	$\binom{n}{25}$	$\binom{n}{25}$	% missed	$\binom{n}{25}$	% missed
17	2.666	25	25		25	
16	4.031	25	23	0.9812	23	0.9937
15	4.551	25	23	2.0467	23	1.8125
14	-	0	22		22	
13	-	0	22		22	
12	-	0	19		21	
11	-	0	13		15	
10	-	0	3		14	
9	-	0	0		13	
8	-	0	0		10	
7	-	0	0		5	
6	-	0	0		20	
5	-	0	0		21	
4	-	0	0		18	
3	-	0	0		25	
2	-	0	0		25	
1	-	0	0		25	

7. Conclusion

In this paper we have explained why the formulation for computing the worst-case response time for hard schedulability is pessimistic. We have provided a formulation to compute whether a message set on a CAN bus is weakly-hard schedulable in the presence of bounded faults and showed that even with very high error rates, guarantees on the temporal behaviour of the system can be given.

The CAN weakly hard analysis is pessimistic, and although the analysis may predict that deadlines can be missed, simulation experiments show that even with fault rates close to the limit for weakly-hard schedulability, very few deadlines are missed. Moreover, even at much higher error rates, the simulation still shows that many weakly-hard constraints are satisfied.

The operation of the LST protocol provides even better performance than the normal CAN protocol because it discards late messages effectively reducing the load on the bus. This approach results in a system where the number of times a particular weakly-hard constraint is satisfied is much higher than under CAN.

The contributions of this paper are twofold. Firstly, the illustration of how to use weakly-hard constraints to reason about guarantees of minimum behaviour of the system under high load conditions. Secondly, the demonstration through a case study of the large difference between the critical instant and the rest of the invocations in the hyperperiod.

8. Future Work

The work presented here has made some simplifying assumptions that may be addressed in future work. The first is that nodes are perfectly synchronised. We believe that the worst case response time formulation $R_{i,k}$ can be adapted to cater for a maximum clock drift ϵ, however the hypothesis of having the nodes synchronised (within ϵ) is essential and can not be lifted.

The fault model used in the schedulability analysis has its limitations. The formulation relies on having an upper bound on the number of errors in a window of time, however the more realistic fault model used for simulation shows that even with lower fault rates there is a possibility that more faults occur than considered in the formulation. This is a common problem of all related papers that use this type of bounded error function. In a future work we will consider probabilistic error models in the schedulability tests and compare them to the simulations.

Finally, we have taken a particular priority assignment and used it throughout the analysis. It has been shown [2] that for weakly hard constraints, DMA is not necessary the optimal priority assignment (although it is a good heuristic). Within the approach presented here there is a circular argument. A given priority assignment results in some weakly-hard constraints being satisfied. A better priority assignment can be found that satisfies even more demanding constraints, which in turn may result in a different priority ordering. We conjecture that this iterative process will eventually converge.

References

[1] G. Bernat. *Specification and Analysis of Weakly Hard Real-time Systems*. PhD thesis, Universitat de les Illes Balears, January 1998.

[2] G. Bernat. Weakly hard real-time systems. *IEEE Transactions on Computers*, 50(3):308–321, March 2001.

[3] G. Bernat and A. Burns. Weakly-hard temporal constraints. Tech. Report YCS, Department of Computer Science. University of York, 2000.

[4] G. Bernat and R. Cayssials. Guaranteed on-line weakly-hard real-time systems. In *22nd IEEE Real-Time Systems Symposium*, London, UK, December 2001.

[5] Bosch, Postfach 50, D-700 Stuttgart 1. *CAN Specification*, version 2.0 edition, 1991.

[6] I. Broster and A. Burns. Timely use of the CAN protocol in critical hard real-time systems with faults. In *Proceedings of the 13th Euromicro Conference on Real-time Systems*, Delft, The Netherlands, June 2001. IEEE.

[7] J. Charzinski. Performance of the error detection mechanisms in CAN. In *Proceedings of the 1st International CAN Conference*, pages 20–29, Mainz, Sept 1994.

[8] T. Fuhrer, B. Muller, W. Dieterle, F. Hartwich, R. Hugel, and M. Walther. Time triggered communication on CAN. Technical report, Robert Bosch GmbH, http://www.can.bosch.com/, 2000.

[9] M. Gergeleit and H. Steich. Implementing a distributed high-resolution real-time clock using the can bus. In *Proceedings of the 1st International CAN Conference*. CiA, 1994.

[10] International Standards Organisation. *ISO 11898. Road Vehicles – Interchange of digital information – Controller area network (CAN) for high speed communication*, 1993.

[11] J. Nilsson, B. Bernhardsson, and B. Wittenmark. Some topics in real-time control. In *Proc. 17th American Control Conference*, pages 2386–2390, Philadelphia, Pennsylvania, June 1998.

[12] L. M. Pinho and F. Vasques. Timing analysis of reliable real-time communication in CAN networks. In *Proceedings Euromicro Conference on Real-time Systems*, Delft, The Netherlands, 2001.

[13] J. Proenza and J. Miro-Julia. MajorCAN: A modification to the Controller Area Network protocol to achieve Atomic Broadcast. In *IEEE Int. Workshop on Group Communications and Computations. IWGCC. Taipei, Taiwan*, April 2000.

[14] S. Punnekkat, H. Hansson, and C. Norstrom. Response time analysis under errors for CAN. In *Proceedings of RTAS 2000*, pages 258–265, Washington DC, 2000. IEEE.

[15] G. Quan and X. S. Hu. Enhanced fixed-priority scheduling with (m,k)-firm guarantee. In *21st IEEE Real-Time Systems Symposium. Florida. USA*, November 2000.

[16] L. Rodrigues, M. Guimarães, and J. Rufino. Fault-tolerant clock syncronization in CAN. In *Proceedings of the 19th Real-Time Systems Symposium*, pages 420–429. IEEE, December 1998.

[17] J. Rufino, P. Veríssimo, G. Arroz, C. Almeida, and L. Rodrigues. Fault-tolerant broadcasts in CAN. In *Digest of Papers, The 28th International Symposium on Fault-Tolerant Computing Systems*, pages 150–159, Munich, Germany, June 1998. IEEE.

[18] SAE. Class C application requirement considerations. Technical Report J2056/1, Society of Automotive Engineers, 1993.

[19] K. 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. Turski. A global time system for CAN networks. In *Proceedings of the 1st International CAN Conference*. CiA, 1994.