Evaluation of admissible CAN bus load with weak synchronization mechanism

Hugo Daigmorte
ONERA – The French aerospace lab
2, av Edouard Belin
Toulouse, France 31055
hugo.daigmorte@onera.fr

Marc Boyer
ONERA – The French aerospace lab
2, av Edouard Belin
Toulouse, France 31055
marc.boyer@onera.fr

ABSTRACT

Scheduling frames with offsets has been shown in the literature to be very beneficial for reducing response times in real-time networks because it allows the workload to be better spread over time and thus to reduce peaks of load. In the specific case of CAN, the response time is mainly related to the priority assignment, but offsets can still improve the achievable bus load. When it exists a global clock, a good offsets assignment leads to a TDMA medium access. When each node have its own local clock the use of offsets still spreads the workload over time.

However, on CAN, global clock is hardly implemented in practice since using a global clock often requires dedicated hardware and complicates the sharing of the bus with non-synchronized nodes.

That is why, we previously introduce the notion of bounded phases, a tradeoff between global and local clocks. Bounded phases allows an affordable synchronization with standard CAN controllers and reduces delays with regard to local clocks. Through an experiment on 5,000 configurations, we have shown that the maximal bus load that can be reached is 80% in the case of bounded phases.

CCS CONCEPTS

• **General and reference** → **Performance**; • **Mathematics of computing** → *Numerical analysis*; • **Computer systems organization** → *Embedded and cyber-physical systems*; *Real-time systems*;

KEYWORDS

worst-case traversal times, network calculus, CAN bus

Hugo Daigmorte and Marc Boyer. 2017. Evaluation of admissible CAN bus load with weak synchronization mechanism. In *Proceedings of 25th International Conference on Real-Time Networks and Systems, Grenoble, France, October 4–6, 2017 (RTNS '17),* 10 pages.
https://doi.org/10.1145/3139258.3139261

1 INTRODUCTION

Controller Area Network (CAN), a serial communication bus network, was initially developed for automotive applications. Today mainstream family cars contain around 30 Electronics Control Units (ECUs) which often communicate using CAN [13]. Due to the many advantages of CAN, including its high reliability and cost effectiveness, it has found applications in other industries. In particular the standard ARINC 825, was developed to standardize the use of CAN in aerospace domain [7]. And so CAN bus is increasingly used for transmitting real-time information. These real-time applications often require to respect temporal constraints (deadlines) and so to bound the communication latencies of the frames.

It has been shown that the use of offsets reduces response time and increases the bus usage [21, 37], because their use allows the workload to be spread over time and thus to reduce peaks load and avoid contentions and so to reduce the worst-case response times and to permit a better bandwidth utilisation [21].

One may wonder why to use offsets since it already exists efficient algorithms assigning priority to messages and allowing to run to a load close to 100% [1, 30]. The reason is that, in an industrial context, priorities are constrained by design constraints. The main one is related to reusability, and the fact that the CAN label both fixes the identity of the message (its content) and its priority. A car is the assembly of different components, and car manufacturers try to maximize the reuse of components between different cars. It simplifies the conception and debug if the same data has the same label (and same priority) on different cars. Moreover component outsourcing often implies that the set of data labels given to a sub-system is set at early design stage, before the integration and response-time analysis. Furthermore, upgrades and extensions should not change already existing labels. Last standards may constraint the label assignment [7]. For example, Davis et al. [13] have addressed the case when a subset of priorities is fixed by design.

Then, the use of offsets is another parameter that can be used to reduce response time and increase bus load, in complement or independently of the priorities assignment.

However using offsets requires a clock. In distributed systems there are two main solutions: each node having a *local clock* or all nodes sharing a *global clock*. In both, each message is sent at an offset with regard to a clock. In case of *global clock* each node has (up to some precision) the same clock value, and no contention occurs, neither between flows from the same node, nor from different nodes. Theoretically with a good offsets assignment (also known as Time-Triggered schedule) it allows to run to a load close to 100%. There are several competitive time-triggered solutions of control busses using a global clocks, such as TTP [22], FlexRay [8], or TTCAN

[26]. However these solutions require specific hardware devices and synchronization mechanisms which have a cost. In case of *local clocks*, the scheduling remains local. Using local clocks avoids the contentions between flows from the same node, and reduces the contentions between flows from different nodes.

In a previous paper [9], we have presented a tradeoff between global clock and local clocks: *bounded phases*. This solution, that will be presented in details in Section 3, consists in using offsets on a network with a weak synchronization between the nodes clocks. Bounded phases allow an affordable synchronization with standard CAN controllers and reduce delays with regard to local clocks.

The contribution of this paper is twofold. First, since [9] was limited to purely periodic flows on a perfect CAN bus, then we show how the notion of bounded phases allows to mix on the same CAN bus periodic and sporadic flows, and how to consider the impact of losses. Second, the gain related to offsets (in the case of a purely periodic system and CAN bus mixing periodic and sporadic flows) is evaluated on 5,000 configurations, with random or efficient priority assignment. One goal is to get the "breakdown utilisation factor" of this method, *i.e.* a coarse evaluation of the admissible load of a CAN bus using bounded phases.

Section 2 presents the context of this study (the CAN bus, application and error model). Section 3 presents in details the concept of bounded phases. Section 4 presents an overview of the related work, in the areas of synchronized networks, CAN priority assignment, the response time analysis methods. Section 5 shows how to extend the results of [9] to consider the error model and the sporadic flows. Then, Section 6 shows the result of an experimental evaluation on 5,000 configurations.

2 CONTEXT

2.1 CAN bus

Controller Area Network is a non-preemptive serial bus standard [33] for connecting Electronic Control Units (ECUs) also known as nodes. CAN was initially developed by Robert Bosch Gmbh for the automotive industry in the mid 90s.

One reason for this success is the bit-wise non-destructive arbitration mechanism of CAN. CAN is a broadcast bus, with a priority-based access to the medium. Each message contains an identifier, unique to the whole system, that is used to assign a priority to the transmission and allows the message to be filtered at the reception. The CAN specifications use the terms "dominant" bits and "recessive" bits. If two nodes try to access the bus at the same time, they will first send the identifier of their message. If one node transmits a dominant bit and another node transmits a recessive bit then there is a collision and the dominant bit "wins". The node transmitting the lower priority message automatically stops transmitting and waits until the bus becomes idle again before attempting to re-transmit the message.

This mechanism implements in a distributed way a non-preemptive static priority medium access policy.

In CAN, six consecutive bits of the same polarity are used for error signaling. Moreover due to "Non Return to Zero" (NRZ) coding used with CAN, drift in the receiver's clock can occur when a long sequence of identical bits has been transmitted. To avoid these special bit patterns in transmitted frames, a bit of opposite polarity is inserted after five consecutive bits of the same polarity. All receiving nodes remove the "inserted" bits to recover the original data. 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 a additional transmission delay which need to be considered in the analysis. Note that not all bits in a message are subject to bit-stuffing (CRC, ACK and the end of frame field), but only the one of the payload. So, if the payload is made of n_{bytes} bytes, the frame length is upper bounded by $55 + n_{bytes} * 8 * 1.25$ bits.

In the case of probabilistic schedulability analysis a lower bound could have been used as it is shown in [31].

2.2 Model application

In this study we consider that there is a fixed set of data flows. Each flow is characterized by a maximum payload size, a priority and a sender. These flows can be either *periodic* or *sporadic*. A periodic flow is characterized by a period, $P > 0$. A frame is sent periodically, that is to say, the distance between the release time of two consecutive frames is equal to the period P. A sporadic flow is characterized by a minimum update time, $MUT > 0$. A frame is sent as soon as specific events occur, however the minimal distance between the release time of two consecutive frames is not less than the minimum update time MUT.

Usually it is considered that each message has to respect an implicit deadline equal to its period for periodic flows and equal to its minimal update time for sporadic flows.

2.3 Error model

CAN has a very efficient error detection mechanism. An "error flag" can be transmitted by each node which detects an error. The error flag consists of six consecutive dominant bits and violates the rule of bit stuffing. After receiving the error flag, the node transmitting the corrupted message automatically stops transmitting and the message will re-enter arbitration. Transmission errors are a random phenomenon, and so it cannot be forecast. However Tindell and Burns, in [35], have introduced the idea that the number of errors can be upper bounded during a given time period. This upper bound is characterized by:

- N_{error}, the burst errors, is the maximal number of errors that could occur back-to-back.
- T_{error}, the residual error period.

The number of transmission errors during the duration d is thus bounded by: $N_{error} + \left\lceil \frac{d}{T_{error}} \right\rceil - 1$.

2.4 CAN FD

The increasing system complexity requires to increase the bandwidth. The classic CAN's bit rate is limited to 1Mbps due to its arbitration mechanism for media access control, and the number of data per CAN frame is limited to 8 bytes. In order to overcome these limitations while keeping most of the software and hardware unchanged, R. Bosch GmbH introduced in 2012 CAN FD [23] (CAN with Flexible Data-rate). CAN FD modifies the CAN frame format by increasing the maximal payload size per CAN frame up to 64

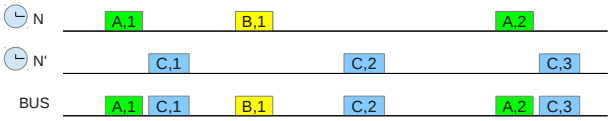

Figure 1: Schedule example with a global clock.

bytes and by permitting to switch the bit rate to faster value inside the CAN frame.

That is to say CAN FD does not really increase the number of messages that can be sent on the bus but increases their payload. And so the issue of evaluating the admissible bandwidth remains similar. This is why this study focuses only on a classical CAN configurations, but the results can be extended to CAN FD.

3 BOUNDED PHASES

3.1 Presentation

A periodic flow can be characterized by its period P, and an offset O: the k-th frame of the flow i is released when the local clock is equal to $O_i + kP_i$. Considering a set of periodic flows, the choice of the offset value O_i of each flow has an impact on the response time, [21].

Nevertheless, using offsets requires a clock. In distributed systems there are two main solutions: each node having a *local clock* or all nodes sharing a *global clock*. In both, the offset value is relative to the considered clock. Let c_N be the local clock of the node N, the k-th frame of the flow i is released at an instant t such that $c_N(t) = O_i + kP_i$.

In case of global clock all nodes have (up to a certain precision) the same clock value ($\forall N, N', t : c_N(t) \approx c_{N'}(t)$). Using this common clock and with the proper time-triggered frame schedule no contention occurs, neither between the flows from the same node, nor from different nodes. An example of such a schedule is given in Figure 1. A time slot is dedicated to each message, and no contention occurs between frames from flows A, B, C.

In case of local clocks, the scheduling remains local. Using local clocks avoids the contentions between flows from the same node, and reduces the contentions between flows from different nodes. Two examples of schedule are given in Figure 2. Contentions between frames from flows A and B from node 1 cannot happen. However contentions between flows from different nodes can happen: between A and C (upper case) and between B and C (lower case). Nevertheless, offsets with local clocks create some traffic shaping and reduce contentions between nodes: C can be delayed by at most A or B but never both of them.

We introduce the notion of *bounded phases* as a tradeoff between global clock and local clocks: a system with a global clock but a weak precision, that can also be seen as a system with local clocks, where the phases between the clocks are bounded: a bound φ such that $\forall N, N', t : |c_N(t) - c_{N'}(t)| \leq \varphi$. The phases between nodes are not perfectly known but bounded, and some contentions can be avoided. An example of schedule is shown in Figure 3. Like in the case of local clocks, no contention will occur between the flows A and B. But if the phase $X = c_N(t) - c_{N'}(t)$ between N and N' is small enough, no contention can occur between flows B and C.

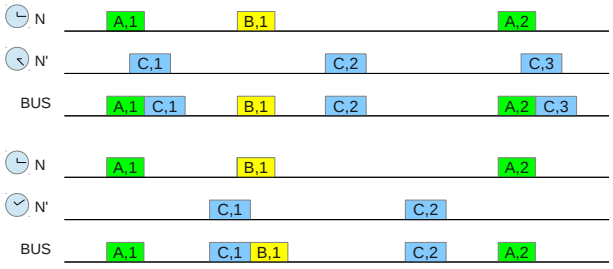

Figure 2: Schedule examples with local clocks.

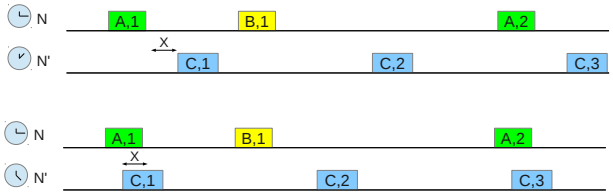

Figure 3: Schedule example with bounded phases.

The interest of using bounded phases is that it is possible to benefit from some of the advantages of a global clock with fewer constraints on the synchronization. First as it have been shown previously in Figure 3 there is less contentions and so a better worst latency with regard to local clocks. Moreover it is possible to use standard CAN controller whereas time-triggered systems require in general to use dedicated devices [1]. Finally it is compatible with any synchronization mechanism. This synchronization does not have to be perfect but the weaker it is, the weaker the gain on delay will be.

3.2 Mixing synchronous and asynchronous flows

In the previous part, we only considered periodic flows and did not take into account sporadic flows. How can periodic and sporadic flows be mixed? We consider two sub-cases: either a node sends both periodic and sporadic flows, or a node sends only one type of flows.

In the case of global clock, for both of these cases, sporadic flows may create contentions that had been avoided by using a proper time trigger frame schedule. And so a contention resolution mechanism has to be established. An alternative solution consists in reserving specific time slots for asynchronous flows. With this case asynchronous flows will only be released during their time slots and so contentions can occur only between asynchronous flows, however asynchronous flows have to wait until their next time slot before being released which increases their delay.

In the case of local clocks and bounded phases, contentions between frames sent from different nodes were already possible. However in the case where a node sends both periodic and sporadic flows, intra-node contention can occur, sporadic flows add the obligation to manage these intra-node contentions.

[1] A time-triggered system requires a bound on the clock difference $c_N - c_{N'}$ smaller that the duration of a few bits, around several micro-seconds.

In the case of CAN bus, the non preemptive static priority policy already manages this case, however sporadic flows have an impact on the performances.

4 STATE OF THE ART

4.1 Synchronized bus

CAN is an asynchronous serial data bus that was designed as a simple and robust broadcast bus. In order to increase the maximal bus load and in order to design a highly dependable protocol several competitive time-triggered solutions of control bus, such as TTP [22], FlexRay [8], or TTCAN [26] have been designed. Most of them use a medium access mechanism different from CAN, the Time Division Multiple Access (TDMA). The TDMA bandwidth allocation scheme subdivides the time domain into different time slots. Network nodes are assigned time windows during which, they have the full transmission capacity of the medium for the duration of this window: this allows multiple stations to share the same transmission medium. The TTCAN uses the classical CAN access and collision resolution mechanism, and adds TDMA on top of the CAN protocol. Flexray mixes periodic and non periodic flows. Flexray is composed of static and dynamic segments, that are repeated periodically. The first one is used for periodic flows, the second one for non periodic flows (as presented in section 3.2). TTCAN by using the classical CAN access mechanism can mix periodic and non periodic flows, without reserving a specific time slot for non periodic flows.

In time-triggered systems, all actions are derived from a global notion of time which requires to synchronize the local clocks of each node. Flexray utilizes the concept of *microticks* and *macroticks*. Microticks correspond to the node's own internal time base, and is not synchronized with rest of system. While macroticks represent the global notion of time used to trigger actions and to order events, it's an integral number of microticks, but not necessarily the same number of microticks per node. Each node synchronizes its macrotick by dynamically increasing or decreasing the number of microticks per macrotick, according to a clock synchronization algorithm.

In TTP all the nodes know the schedule. Each node measures the difference between the a priori known expected and the observed arrival time of a message. Then a clock synchronization algorithm corrects the local clock in order to maintain the same global time.

The TTCAN synchronization mechanism is based on a master/slave principle: a master node provides a global reference time from which all other nodes on the network derive their own local clock.

4.2 Priorities and offsets assignment algorithm

In the preemptive case, it have first been shown in [30] that, in the case where deadline equals period, the rate-monotonic priority assignment, which assigns priorities in a monotonic relation to their their period, is optimum among all possible assignments. For the more general case where deadlines are equal or less to periods, this algorithm can be adapted to the deadline monotonic ordering and it remains optimal, see [27]. However, in the general case where no relation exists between periods and deadlines, these algorithms are no more optimal, an optimal priority assignment procedure was provided in [1], known as the Audsley priority assignment.

Contrary to the preemptive case, in the non-preemptive case even when deadlines are less or equal to periods, the deadline monotonic assignment is no longer optimal. It has been first assumed that if the size of higher priority frame is less or equal to the size of lower priority frame then deadline monotonic assignment is an optimal priority assignment [18, Theorem 16]. However this theorem has been proven incorrect as shown by the counter example in [11]. The Audsley priority assignment remains optimal in the general case [1, 18].

The specific case where some of the priorities are fixed, and a subset may be freely assigned have been studied by Davis et al. [13].

We have only presented the necessary work for our case study, however a lot of work have been done on the issue of priority assignment, see [14] for a more detailed overview.

Additionally to the priorities assignment a very important point is the offsets assignment. The problem of choosing the best offsets has been shown in [19] to have a complexity that grows exponentially with the periods of the tasks. In [19], an optimal offsets assignment is proposed, however due to its complexity offsets assignment heuristics non optimal but with lower complexity have been proposed in [19, 20]. Importance of choosing offsets has been shown in [21].

4.3 Analysis methods

In order to evaluate the maximal bus load, it is necessary to be able to bound delay. The timing analysis of CAN has been the object for various studies in the past. The worst case response times were first provided in [36] and then revisited in [12] but without considering offsets. They gave the exact response time for sporadic messages. In [24, 32] it has been shown that network calculus can be also used to compute upper bounds on CAN, but without proof of being tight (that is to say "exact"). Response times on CAN with offsets and local clocks have also been studied first with approximate but lower-complexity forms of analyse in [34] and then an effective worst-case response time analysis in the non-preemptive case with offsets has been given in [37]. Multi-hop systems with local clocks and offsets have also been studied using network calculus in the case of AFDX [28, 29]. Finally response time in the case of bounded phases using network calculus have been studied in [9, 10].

5 BOUNDING DELAYS WITH NETWORK CALCULUS

In order to compare the use of bounded phases with regards to systems without offsets or only local clock, we need response time analysis for each case.

Section 5 recalls some network calculus background and presents some new properties, required for mixing sporadic flows, errors and bounded phases. Then we will present network calculus results for local clocks.

5.1 Network Calculus reminds

Network calculus is a theory to get deterministic upper bounds in networks. Network calculus mainly handles non decreasing functions, null before 0. It is mathematically based on the (min,+) dioid and beyond classical operations like addition or minimum, network calculus relies on two basics operators the convolution and the deconvolution.

More details, and in particular the proofs of Property 1 and Theorem 5.4 can be found in [6, 25].

Definition 5.1. The min-plus convolution $*$ and deconvolution $\oslash$ of two functions f and g are defined by

$$(f * g)(t) = \inf_{0 \le s \le t} \{f(t-s) + g(s)\}$$
$$(f \oslash g)(t) = \sup_{0 \le u} \{f(t+u) - g(u)\}$$

The non-decreasing non-negative closure is defined by

$$[f]^{+}_{\uparrow}(t) = \max_{0 \le s \le t} (f(s), 0)$$

In network calculus, input and output flows of data are modeled by cumulative functions which represent the amount of data produced by the flow up to time t. The servers are just relations between some input and output flows: a server S receives an arrival/input flow, $A(t)$, and delivers the data after some delay, it is the departure/output flow, $D(t)$. We always have the relation $D \le A$, meaning that data can only go out after their arrival. However the exact input/output data flows are in general unknown at design time, or too complex, and the calculus of these cumulative functions cannot be obtained. Nevertheless, the evolution of input/output data flows can be bounded considering contracts on the traffics and the services in the network. For this purpose, Network Calculus provides the concepts of arrival curve and service curve, that have been more widely described in [6, 25].

Definition 5.2 (Arrival curve). Let A be a flow, and α be a function. Then, α is said to be an arrival curve for flow A, iff

$$\forall (t, d) \in \mathbb{R}^2_+,\ A(t+d) - A(t) \le \alpha(d) \tag{1}$$

Eq. (1) is equivalent to $A \le A * \alpha$, see [25].

Property 1. *Let A_1 and A_2 be two flows, and α_1 (resp. α_2) an arrival curve of A_1 (resp. A_2).*

- *If $\alpha' \ge \alpha_1$ then α' is an arrival curve of A_1;*
- *$A_1 \oslash A_1$ is the "best" arrival curve for A_1, i.e. $A_1 \oslash A_1$ is an arrival curve and $A_1 \oslash A_1 \le \alpha_1$;*
- *$\alpha_1 + \alpha_2$ is an arrival curve of $A_1 + A_2$.*

Definition 5.3 (Minimal services). A server S offers a strict minimal service curve β iff for all input/output A,D and for all backlogged period (or busy period) (s,t]

$$D(t) - D(s) \ge \beta(t-s)$$

Let us now present the main network calculus result which allow, considering contracts, to compute bounds on delay.

Theorem 5.4 (Delay bound). *Let S be a server offering a min-plus minimal service curve β. If the input flow A has an arrival curve α, then, the delay can be bounded by*

$$delay = hDev(A, D) \le hDev(\alpha, \beta)$$

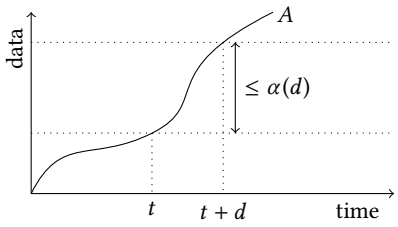

Figure 4: Arrival curve

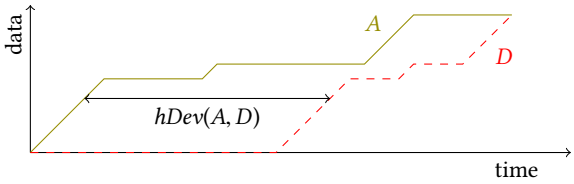

Figure 5: Delay of the flow A

where hDev is the horizontal deviation (see Figure 5 for an illustration, and [6, 25] for the definition).

Theorem 5.5. *[Non-preemptive static priority, [2]] Let S be a server offering a strict minimal service β, shared by three flows, A, A_H, A_L, A_H having a higher priority than A, and A_L a lower. Then, if α_H is an arrival curve for A_H, L^{max} is an upper bound on the frame size of A and $L^{\max}_L$ is an upper bound on the frame size of A_L, the flow A receives a strict service curve β^A:*

$$\beta^A = \left[\beta - \alpha_H - \max(L^{max}, L^{\max}_L)\right]^{+}_{\uparrow}$$

5.2 Mixing asynchronous flows, errors and bounded phases

In a previous paper [9], we have developed three different methods to bound the delay in the case of bounded phases. Each method assumes that the bus offers, to the set of periodic flows, a strict service of curve β. When there are only periodic flows, the service is a constant rate $\beta(t) = C.t$, where C is the link speed.

When periodic flows share the bus with some sporadic flows, and errors, one have to define another β function representing the residual service offered to the periodic flows. In the case of CAN bus with non-preemptive static priority arbitration we are going to use theorem 5.5.

To capture the influence of sporadic flows, let $S \subset [1, n]$ be the subset of flows with sporadic behavior. Let $i \in S$ be such sporadic flow, of maximal frame size s_i and minimal update time MUT_i. It admits as arrival curve $\alpha_i(t) = s_i \left\lceil \frac{t}{MUT_i} \right\rceil$. Now considering a set of sporadic flows, using property 1 it is possible to deduce an arrival curve for the set as the sum of the arrival curve of the individual flows.

To capture the influence of errors, considering the error model presented in section 2.3, one may consider that errors are virtual frames of some virtual additional asynchronous flow of maximum priority and a larger frame size (to take into account the error frame). This virtual flow has arrival curve

$$\alpha_{error}(d) = (L_{max} + L_{error}) \left(N_{error} + \left\lceil \frac{d}{T_{error}} \right\rceil - 1 \right) \tag{2}$$

Then, when considering the periodic flow of priority i, the theorem 5.5 allows to use the results presented in [9] using $\beta = \lambda_R - \alpha_{error} - \sum_{j<i, j\in S} \alpha_j - \max_{j=i..n}(L_j)$, since it is a residual service offered to the flow i.

5.3 Local clocks with network calculus

In the case of local clocks, as offsets rely on local clocks, flows have to be aggregated by sender node in order to evaluate a correct arrival curve.

Theorem 5.6 (Arrival curve for local clock scheduling). *Let $A_1, \ldots, A_n$ be a set of flows, and $N_1, \ldots, N_m$ a set of nodes. For any of A_i, if $s(i)$ is the sender of the flow, then an arrival curve of $\sum_{k=1}^{n} A_k$ is:*

$$\sum_{l=1}^{m} \left(\sum_{k=1, s(k)=l}^{n} A_k \right) \oslash \left(\sum_{k=1, s(k)=l}^{n} A_k \right) \tag{3}$$

Proof. Considering $A'_1, \ldots, A'_{n'}$ a set of flow sent by the same sender. From Property 1 we can deduce that $\left(\sum_{k=1}^{n'} A'_k\right) \oslash \left(\sum_{k=1}^{n'} A'_k\right)$ is an arrival curve of $\sum_{k=1}^{n'} A'_k$.

Now using Property 1, we know that the sum of the arrival curves is an arrival curve of the sum of flows. And so $\sum_{l=1}^{m} \left(\sum_{k=1, s(k)=l}^{n} A_k\right) \oslash \left(\sum_{k=1, s(k)=l}^{n} A_k\right)$ is an arrival curve of $\sum_{k=1}^{n} A_k$ □

Since the cumulative curve of a periodic flow of period P, frame size s and offset o is known ($A(t) = s\left\lceil \frac{t-o}{P} \right\rceil$), the arrival curve of the aggregate flow can be computed. It is then possible to bound the delay using network calculus.

6 EXPERIMENTATION

6.1 Configuration Pattern

This section evaluates the effect that *Bounded Phases* between nodes have on the maximum achievable bus load. In order to evaluate this impact, we would have first to define an experimental configuration representative of a "common" CAN configuration. However, there is no "common" CAN configuration. CAN is widely used and so there is huge set of configurations: only time-triggered flows, only asynchronous, presence of a gateways that generates a large amount of the network load... There is no universal CAN configuration.

Our choice inspired by [17, 38], considers a configuration pattern with:

- 16 nodes connected via a single CAN bus, exchanging periodic messages (except experiments 6.7).
- Each priority was randomly, with uniform distribution, allocated to a message (except experiments 6.4).
- The periods of the frames were uniformly chosen from the set {20, 25, 40, 50, 100, 200}ms.
- An 8 bytes payload and an 11-bit identifier.
- There were 35-55 messages (more details further).
- The assignment of offsets that was used in this study is the SOPA algorithm available in the RTaW-Pegase software from the company RTaW [4].
- The deadlines of each message was set equal to their period, *i.e.* implicit deadlines (except experiment 6.4).
- Each message was randomly, with uniform distribution, allocated to one of the source (except experiment 6.6).

6.2 Methods for delay evaluation

We are going to compare the gain given by the use of bounded phases w.r.t. local clocks and no offset. Here are presented the methods used to do the evaluation of delay for each solution (already presented in section 4.3).

For bounded phases, three methods based on Network Calculus (NC) have been defined to compute upper bound on delays in [9]. None is better than the others, but all the three give a valid upper bound. Then, in this paper, the three methods have been aggregated into a single one: the three analysis were run but only the best (*i.e.* smaller) result was kept. Results will be plotted in red.

For local clocks, the reference algorithm is the one of [37]. It computes a worst response time exact up to one frame length. We used the implementation available in RTaW-Pegase tool [4]. Results will be plotted in blue.

Nevertheless, this computation takes a lot of time, so we also used Theorem 5.6 which computes an upper bound. Results will be plotted in green.

When the computation time was acceptable, both have been run. When it was not, only the NC-based was run. For example, in the second experiment that will be presented (section 6.5), the mean computation time of RTaW-Pegase was 435s per configuration (over 5,000 configurations), whereas the NC-based evaluation (computing both local clocks and bounded phases delays) took only 65s. For the first experiment, that will be presented in section 6.4, we run the RTaW-Pegase tool only on 100 configurations, and gave up, since the mean computation time was 2h10mn per configuration, whereas the NC-based code took only 72s. When both can be run, their results are quite close (1-2% in average).

For the cases where no offsets are used, the classical response time algorithm was used [12]. It computes an exact worst response time. Results will be plotted in black.

6.3 Breakdown utilisation

Now that we have defined the characteristics of our reference CAN configuration pattern, we have to define a criterion to evaluate what is the nominal load of a CAN bus. In order to answer this question we use the *breakdown utilisation* [15–17]. The breakdown utilisation is defined as the part of the minimal link speed required to guarantee that all deadlines are respected. For example, if nodes send on the network 150 kilobytes of data per second and if, in order to meet all the deadlines, the link speed has to be at least 300kbit/s then the breakdown utilisation for this configuration is 50%.

Now that we have a criterion to compare no offsets, local clocks and bounded phases we can describe the experiment. First, 5,000 configurations will be generated. For each of them the total load sent on the bus is 150kbit/s, *i.e.* the total bus load on the network at 500kbit/s is 30%. Then for each configuration we compute the delay for each flow with all the different methods described in the previous section, first with a link speed equal to 1000 kbit/s. At 1000 kbit/s all the messages meet their deadline. Then we decrease the link speed, and compute the new delay for each message in

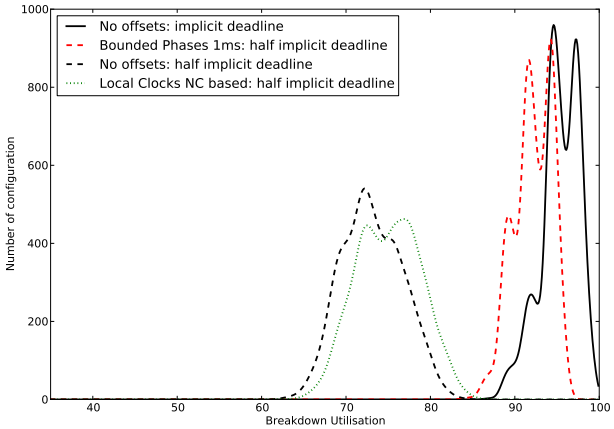

Figure 6: Breakdown utilisation in case of deadline monotonic assignment.

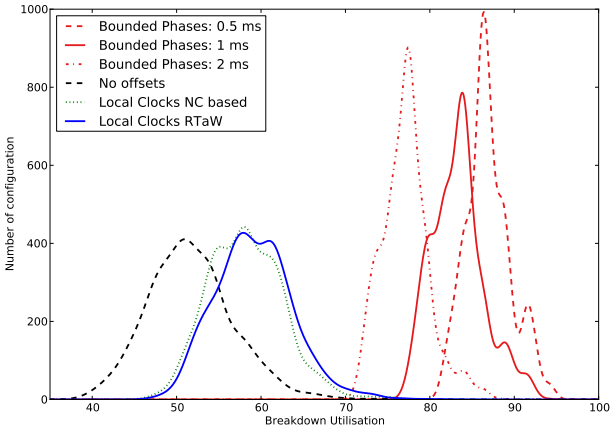

Figure 7: Breakdown utilisation, with phases bounded by 0.5ms, 1ms and 2ms.

order to obtain, for each method, the breakdown utilisation [2] (up to 5kbit/s).

Such an experiment results lead to an histogram. Nevertheless for ease of readability, we will not plot the histograms but a continuous approximation call "kernel density estimation" in python (*scipy.stats.gaussian_kde*), like the one on Figure 7 (notice that, for readability, only range 35%-100% is plotted).

We want to compare the different methods, so in order to compare them (for example Figure 7) we used on the same 5,000 configurations the different methods. Each of them will result on different breakdown utilisations and so we can compare them.

Note that since the bounded phases principle requires a synchronization flow, such a flow with a period of 200ms is added to each configuration when considering bounded phases.

6.4 First Experiment: deadline monotonic

Let start with an experiment where priorities are assigned using deadline monotonic priority assignment.

First consider the common case where the deadline of each flow is equal to its period (implicit deadline). In Figure 6 is plotted the breakdown utilisation factor without offsets (black plain line). For the 5,000 configurations, the breakdown utilisation is always bigger than 85%. And the mean value of the bus load is 95%, confirming the performances of this strategy. In these conditions, the use of offsets would not be very useful

Second, consider a more challenging constraint, setting the deadline to be half of the period. In this case, three strategies are used: no offsets, local clocks and bounded phases (with a 1ms clock accuracy).

Without offsets, the deadline monotonic assignment allows to reach a 73% mean load. The use of local clock slightly increases it, to 75%. With a phases between nodes bounded by 1ms, the bandwidth utilisation is close to 100%, between 85% and 97%.

This first experiment shows that priority assignment and the use of offsets are complementary.

[2] We used a binary search for the RTaW method and a linear search for the network calculus methods due to implementation constraints.

In order to evaluate the gain related to offsets only, the others experiments will consider random priority assignment and implicit deadlines.

Last, note that the delays with local clocks were evaluated only using the NC-based methods, since the RTaW implementation was too long in this case (cf. section 6.2).

6.5 Second Experiment: uniform sources

The second experiment will consider randomly assigned priorities (to model a choice due to some unknown design criteria), with different inter nodes clock accuracy.

The gain due to a network *weakly synchronized* is directly dependent on the bound on the phase between clocks. The weaker the synchronization, the lower the gain. The bound on the phase between clocks is the consequence of the synchronization mechanism chosen. Our method does not assume a specific synchronization mechanism, we consider that it requires at most sending one frame at each hyper period (in this case 200ms). In [10] we demonstrate that with a simple synchronization mechanism a precision around 1ms can be achieved. That why, we will compare the breakdown utilisation of three solutions: no offsets, local clocks and bounded phases with precisions of 0.5ms, 1ms and 2ms. Results are presented in Figure 7.

First of all we will take an interest in the case where no offsets are used. It has to be compared to the result obtained when a deadline monotonic scheduling is used. The maximum bus load is much lower, between 40% and 65%, with a mean value around 50%, which is almost two times less than previously. Once more, it confirms the folk result that the CAN load has to be less than 35%, to ensure that all deadlines are met [5, 16] (keep in mind that the 35% load considers "typical" error rate, not considered here).

Second, consider the case of local clocks, where bounds are computed using two methods. We used two methods because even if network calculus is not exact it requires a much lower calculation time. The maximum bus load is bigger, between 55% and 70%, with a mean value around 58%. It means that using local offsets permits to increase the number of messages sent on the network by more

than 20% with regard to the maximal number of messages than can be sent in order to ensure that all deadlines are met without using offsets.

Third, consider bounded phases. With a bound on phases between nodes of 1ms, the maximum bus load is much bigger, between 75% and 95%, with a mean value of 83%. This gain is very important. It means for example that it is possible to ensure the respect of all deadlines with 50% of additional messages comparing to offset with only local clocks.

As the precision of the synchronization in the case of bounded phases is primordial, other calculations have been done with a precision of 0.5ms and 2ms (see Figure 7). As expected the weaker the synchronization (*i.e.* larger bound), the lower the gain. However even with a synchronization of 2ms, the gains remain important (about 15% more with regard to local clocks). And the maximal bus load achievable with bounded phases is always better than the maximal bus load achievable without synchronization between nodes.

The results are summarised in Table 1.

Table 1: Breakdown utilisation, uniform sources.

Breakdown Utilisation (%)	Minimum	Maximum	Mean
No offsets (periodic or sporadic)	38.79	79.57	51.67
Local clocks (NC)	45.86	82.45	58.18
Local clocks (RTaW)	46.59	86.68	59.16
Bounded phases 0.5ms	81.36	97.11	86.94
Bounded phases 1ms	77.19	94.5	83.35
Bounded phases 2ms	70.63	88.54	77.15
No offsets, errors	37.37	69.34	49.51
Local clocks, errors	43.92	69.65	55.41
Local clocks, 20% sporadic	45.54	69.65	57.64
Local clocks, errors, 20% sporadic	44.22	69.65	54.97
Bounded phases, errors	71.67	91.22	77.55
Bounded phases, 20% sporadic	59.03	91.23	73.23
Bounded phases, errors, 20% sporadic	55.75	86.4	69.13

6.6 Third Experiment: Gateway

In the next experiment we decide to examine the impact of a non uniform distribution of senders. It is often the case in practice: it can be a gateway for example that sends a large part of the traffic. Thereafter this node will be design as *the gateway*.

In the first case we consider that the gateway sends 30% of the traffic (70% shared by the 15 other nodes), whereas in a second case, the gateway generates 60% of the traffic. The results are presented in Figure 8.

Using a master slightly increases the breakdown utilisation in the case of local clocks (from 58% to 61% in the case of 30% gateway traffic and 67 % in the case of 60% gateway traffic).

Considering bounded phases, it has a negligible impact: in Figure 8 the three red plain, dotted and dashed lines (representing no gateway, 30% gateway and 60% gateway) quite collapse in a single one.

This result was predictable, since in the case of local clocks it reduces the contentions of messages from different flows (as they are mainly sent by the gateway) and so the delays. Whereas in the case of bounded phases, contentions between messages from

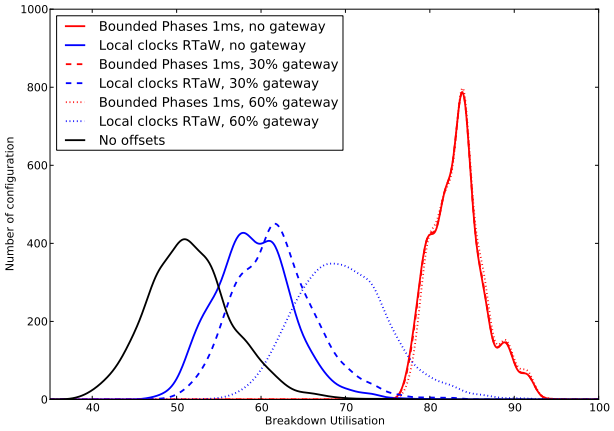

Figure 8: Breakdown utilisation, with a gateway sending 30% or 60% of the traffic.

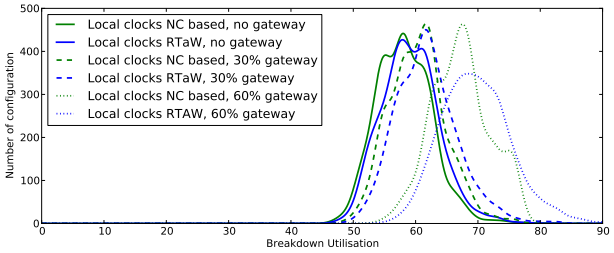

Figure 9: Comparison of methods for local clocks on same configurations that Figure 8.

different flows have already been reduced. However even with more than half of the traffic sent by a single node, using bounded phases remains the best solution in term of bus load.

For readability, NC based results for local clocks were not presented in Figure 8. They are presented in Figure 9, together with ones provided by RTaW-Pegase. Once more, results from the NC based method show it gives a good approximation of the delays for local clocks.

6.7 Fourth Experiment: errors and sporadic messages

In this last experiment we decide to examine the impact of errors and sporadic messages. To capture the influence of errors, we used the model presented in section 2.3, with $N_{error} = 2$ and $T_{error} = 100ms$. And to capture the influence of sporadic flows, we used the same set of configuration than previously but changing 20% of the periodic messages into sporadic messages. This lead us to 3 experiments: one with only periodic flows and errors (Figure 10), one with sporadic flows and no errors (Figure 11) and a last one with errors and sporadic flows (Figure 12). The results are summarised in Table 1.

First consider the case with errors (Figure 10). Obviously taking into account errors increases the delay and so reduces the admissible bus utilisation, around 2% for no offset, 3.5% for local clocks and 6% for bounded phases. This result was predictable: in the case

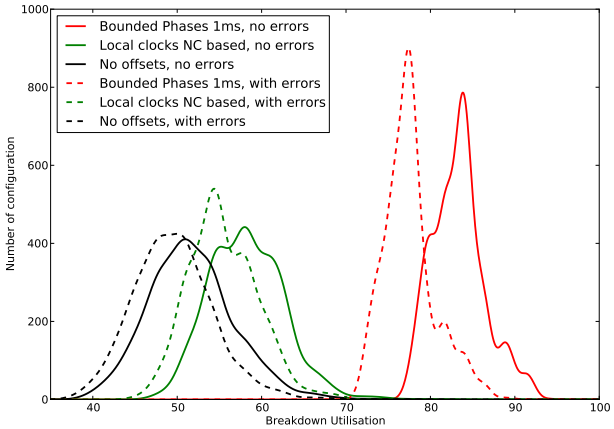

Figure 10: Breakdown utilisation: errors, no sporadic.

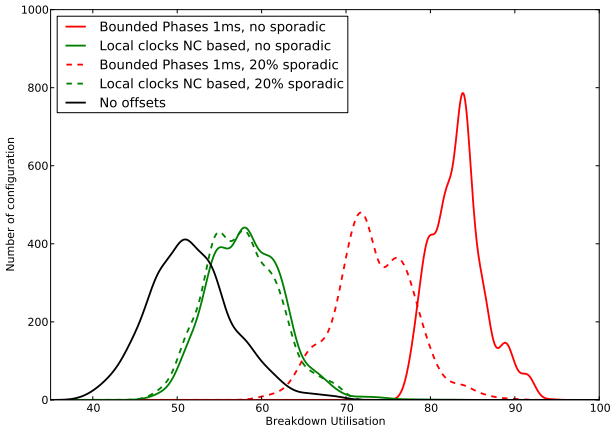

Figure 11: Breakdown utilisation: no errors, 20% sporadic.

of bounded phases, offsets have been chosen in order to avoid contentions without considering errors. With errors, some contentions may occurs. It is the same phenomenon that occurs for local clocks, but only intra-nodes contentions, and so the bus utilisation is less impacted.

Then consider the case without errors and 20% of sporadic messages (Figure 11). In the case of no offsets, it has of course no impact, because the use of no offsets is equivalent to 100% sporadic. It has a very small impact on local clocks (1.5%), but a significant one on bounded phases (10%).

In the case of local clocks, offsets are used in order to avoid intra-node contentions and in order to spread the traffic over time, but inter-node contentions are not avoided in the case of local clocks. And so changing a flow from periodic to sporadic will mostly impact the messages sent by the same node. Whereas in the case of bounded phases, changing a flow from periodic to sporadic will impact all the messages of the system.

Last consider the case with errors and sporadic messages (Figure 12). The perturbations are accumulated (5% for local clocks, 14% for bounded phases) but the use of offsets, either with local clocks or bounded phases, still improves the admissible load.

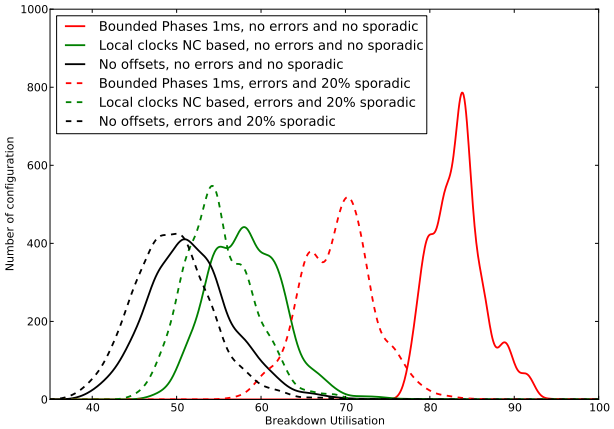

Figure 12: Breakdown utilisation: errors, 20% sporadic.

7 CONCLUSION

A data flow, periodic or sporadic, is characterized by several parameters: size, period/minimum update time (MUT), priority and deadline. The period/MUT, the size and the deadline are applicative constraints and can not be freely assigned whereas priority can be assigned. The priority assignment is one way to meet deadline, a lot of work have been done on this subject [1, 14, 30].

In the case of periodic flows, it exists another parameter that also greatly influences the bus latency: the offset [21]. This notion of offset is related to a clock value. This clock is commonly either local to each node, or global to all nodes (implying some synchronization mechanism). In a previous paper [9], we have proposed a new mechanism, a kind of trade-off between both: the notion of "bounded phases", the use of offsets with a weak inter-node clock synchronization. We also have given an analysis method for bounded phases, based on network calculus.

The analysis method have been extended in this paper in three directions:

(1) it can now analyse systems where periodic and sporadic flows share the same bus,
(2) errors are also modelled (as a virtual sporadic flow),
(3) an approximation of the delay in the case of local clocks has been proposed.

Then, we have evaluated the performances of bounded phases with regards to other mechanisms: when offsets are related to a local clock, and when no offsets are used. Given a configuration pattern, we have generated 5,000 configurations to see what is the maximal admissible load, for each method, requiring that each flow respects its deadline.

Our experiments confirm well known results: with implicit deadline, deadline-monotonic allows to reach 95% bus load, whereas random priority assignment only allows about 50%. We also consider the case where the deadline of each flow is set to half of its period. In this case, deadline-monotonic priority allocation and bounded phases can be combined to reach a 90% load.

Since some design constraints can prevent from a free choice of priorities, the other experiments assume randomly assigned priorities, and implicit deadlines.

Then, on the same configurations, the use of bounded phases allows to get a mean load of 83%, with a clock precision of 1ms (87% with a 0.5ms precision, and 77% with a 2ms precision).

The previous experiments assume a uniform distribution of the load per node. We also investigate the common case where one node generates more traffic than others (up to 60%), and found that it does not change dramatically the performances.

Last, we consider to take into account the errors than may occur on the bus, and to change 20% of the periodic messages into sporadic messages. These cases reduce the benefits due to offsets, since they increase the number of contentions. However we have shown that bounded phases allows to get a load of 70% even in these conditions.

On a large set of experiments, our approximation of delays in the case of local clocks have been compared with the analyse method presented in [37]. The results are very similar, but our approximation is about 10 times faster for experiments in section 6.5 (5,000 runs) and more than 100 times faster for experiments in section 6.4 (100 runs only).

To sum up, it is well known that the use of offsets can improve the bus utilisation. In a previous work [9], we have proposed a new way to use offsets, called bounded phases, and in this paper, we have shown that it is very beneficial, by itself as well as in combination with priorities assignment.

In the future, we would like first to enhance the analysis method by considering the case where some message offsets are fixed and only a subset may be freely assigned. One may in particular consider the case where the offsets are set by the task scheduling, as in [3]. We would like also to develop a method that combines the priorities assignment and the offsets assignment instead of studying them separately.

REFERENCES

[1] Neil C Audsley. 2001. On priority assignment in fixed priority scheduling. *Inform. Process. Lett.* 79, 1 (2001), 39–44.

[2] Anne Bouillard, Laurent Jouhet, and Eric Thierry. 2009. *Service curves in Network Calculus: dos and don'ts.* Research Report RR-7094. INRIA. 24 pages. http://hal.inria.fr/inria-00431674/en/

[3] Marc Boyer and David Doose. 2012. Combining network calculus and scheduling theory to improve delay bound. In *Proc of the 20th International Conference on Real-Time and Network Systems (RTNS 2012)*. Pont á Mousson, France.

[4] Marc Boyer, Jörn Migge, and Marc Fumey. 2011. PEGASE, a robust and efficient tool for worst case network traversal time. In *Proc. of the SAE 2011 AeroTech Congress & Exhibition, Toulouse, France.*

[5] Darren Buttle. 2012. Real-time in the prime-time. In *Keynote speech at the 24th Euromicro Conference on Real-Time Systems.*

[6] Cheng-Shang Chang. 2000. *Performance Guarantees in communication networks.* Springer.

[7] Airlines Electronic Engineering Committee et al. 2011. ARINC Specification 825-2: General standardization of CAN (Controller Area Network) bus protocol for airborne use. *Annapolis, Maryland* (2011).

[8] FlexRay Consortium et al. 2005. FlexRay communications system-protocol specification. *Version* 2, 1 (2005), 198–207.

[9] Hugo Daigmorte and Marc Boyer. 2016. Traversal time for weakly synchronized CAN bus. In *Proc. of the 24th International Conference on Real-Time Networks and Systems (RTNS 2016).* ACM, 35–44.

[10] Hugo Daigmorte, Marc Boyer, and Jörn Migge. 2016. Reducing CAN latencies by use of weak synchronization between stations. (2016).

[11] Robert I Davis and Alan Burns. 2009. Robust priority assignment for messages on Controller Area Network (CAN). *Real-Time Systems* 41, 2 (2009), 152–180.

[12] Robert I Davis, Alan Burns, Reinder J Bril, and Johan J Lukkien. 2007. Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems* 35, 3 (2007), 239–272.

[13] Robert I Davis, Alan Burns, Victor Pollex, and Frank Slomka. 2015. On priority assignment for controller area network when some message identifiers are fixed. In *Proceedings of the 23rd International Conference on Real Time and Networks Systems.* ACM, 279–288.

[14] Robert I Davis, Liliana Cucu-Grosjean, Marko Bertogna, and Alan Burns. 2016. A review of priority assignment in real-time systems. *Journal of systems architecture* 65 (2016), 64–82.

[15] Robert I Davis, Steffen Kollmann, Victor Pollex, and Frank Slomka. 2011. Controller area network (can) schedulability analysis with fifo queues. In *Real-Time Systems (ECRTS), 2011 23rd Euromicro Conference on.* IEEE, 45–56.

[16] Robert I Davis, Steffen Kollmann, Victor Pollex, and Frank Slomka. 2013. Schedulability analysis for Controller Area Network (CAN) with FIFO queues priority queues and gateways. *Real-Time Systems* 49, 1 (2013), 73–116.

[17] Robert I Davis and Nicolas Navet. 2012. Controller area network (CAN) schedulability analysis for messages with arbitrary deadlines in FIFO and work-conserving queues. In *Factory Communication Systems (WFCS), 2012 9th IEEE International Workshop on.* IEEE, 33–42.

[18] Laurent George, Nicolas Rivierre, and Marco Spuri. 1996. *Preemptive and non-preemptive real-time uniprocessor scheduling.* Technical Report.

[19] Joël Goossens. 2003. Scheduling of offset free systems. *Real-Time Systems* 24, 2 (2003), 239–258.

[20] Mathieu Grenier, Joël Goossens, and Nicolas Navet. 2006. Near-optimal fixed priority preemptive scheduling of offset free systems. In *14th International Conference on Real-Time and Networks Systems (RTNS'06).* 35–42.

[21] Mathieu Grenier, Lionel Havet, and Nicolas Navet. 2008. Pushing the limits of CAN-scheduling frames with offsets provides a major performance boost. In *4th European Congress on Embedded Real Time Software (ERTS 2008).*

[22] TTA Group et al. 2001. Time Triggered Protocol (TTP/C), Version 1.0. (2001).

[23] Florian Hartwich et al. 2012. CAN with flexible data-rate. In *Proc. iCC.* 1–9.

[24] Ulrich Klehmet, Thomas Herpel, Kai-Steffen Hielscher, and Reinhard German. 2008. Delay bounds for CAN communication in automotive applications. In *Measuring, Modelling and Evaluation of Computer and Communication Systems (MMB), 2008 14th GI/ITG Conference-.* VDE, 1–15.

[25] Jean-Yves Le Boudec and Patrick Thiran. 2001. *Network calculus: a theory of deterministic queuing systems for the internet.* Vol. 2050. Springer Science & Business Media.

[26] Gabriel Leen and Donal Heffernan. 2002. TTCAN: a new time-triggered controller area network. *Microprocessors and Microsystems* 26, 2 (2002), 77–94.

[27] Joseph Y-T Leung and Jennifer Whitehead. 1982. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance evaluation* 2, 4 (1982), 237–250.

[28] Xiaoting Li, Jean-Luc Scharbarg, and Christian Fraboul. 2010. Improving end-to-end delay upper bounds on an AFDX network by integrating offsets in worst-case analysis. In *IEEE Conf. on Emerging Technologies and Factory Automation (ETFA 2010).* IEEE, 1–8.

[29] Xiaoting Li, Jean-Luc Scharbarg, Christian Fraboul, and Frédéric Ridouard. 2011. Existing offset assignments are near optimal for an industrial AFDX network. In *Proc. of the 10th International Workshop on Real-time Networks (RTN 2011).* Porto, Portugal.

[30] Chung Laung Liu and James W Layland. 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)* 20, 1 (1973), 46–61.

[31] Thomas Nolte, Hans Hansson, Christer Norström, and Sasikumar Punnekkat. 2001. Using bit-stuffing distributions in CAN analysis. In *IEEE Real-Time Embedded Systems Workshop at the Real-Time Systems Symposium.*

[32] William Mangoua Sofack and Marc Boyer. 2012. Non preemptive static priority with network calculus: Enhancement. In *International GI/ITG Conference on Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance.* Springer, 258–272.

[33] ISO Standard. 1993. 11898: Road Vehicles-Interchange of Digital Information-Controller Area Network (CAN) for High-Speed Communication. *International Standards Organization, Switzerland* (1993).

[34] Ken Tindell. 1994. *Adding time-offsets to schedulability analysis.* Technical Report. University of York, England.

[35] Ken Tindell and Alan Burns. 1994. Guaranteed message latencies for distributed safety-critical hard real-time control networks. *Dept. of Computer Science, University of York* (1994).

[36] KW Tindell, Hans Hansson, and Andy J Wellings. 1994. Analysing Real-Time Communications: Controller Area Network (CAN). In *Real-Time Systems Symposium, 1994., Proceedings.* IEEE, 259–263.

[37] Patrick Meumeu Yomsi, Dominique Bertrand, Nicolas Navet, and Robert I Davis. 2012. Controller area network (CAN): Response time analysis with offsets. In *Factory Communication Systems (WFCS), 2012 9th IEEE International Workshop on.* IEEE, 43–52.

[38] Haibo Zeng, Marco Di Natale, Paolo Giusto, and Alberto Sangiovanni-Vincentelli. 2010. Using statistical methods to compute the probability distribution of message response time in controller area network. *IEEE Transactions on Industrial Informatics* 6, 4 (2010), 678–691.