Reducing CAN latencies by use of weak synchronization between stations

Hugo Daigmorte[1], Marc Boyer[1], Jörn Migge[2]
[1]ONERA, Université de Toulouse, France
[2]RealTime-at-Work, France

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. Maintaining a global synchronization amongst the stations induces substantial overhead and complexity in networks not providing a global time service such as CAN. Indeed, on CAN, a global clock is rarely implemented in practice and each station possesses its own local clock. Without a global clock, the de-synchronization between the streams of frames created by offsets remains local to each station and thus less efficient. In a previous paper [1], we developed a method to compute latency upper bounds for set of messages with offsets when the inter-node synchronization is not perfect. On a simplified test case, we obtained a reduction of 65% of the delay using a clock accuracy of only 1ms. In this article, we extend the method to consider a realistic case study (mixing periodic and asynchronous flows, considering errors and tacking into account the synchronization protocol).

1. Introduction

1.1 Context

Controller Area Network (CAN) is a serial communication bus network that was initially developed for automotive applications in the mid 90s. Due to the many advantages of CAN, including its high reliability and cost effectiveness, it has found application in other industries. Real-time distributed applications increasingly use CAN for transmitting real-time information. These applications often require to respect temporal constraints and so to bound the communication latencies of the frames, also called the worst case response times (WCRT).

It is well known (e.g., see [9]) that the use of offsets reduces frame response times and increases the possible bus utilization level. Indeed, offsets allow the workload to be better balanced over time which reduces contention for the bus access and, as a result, decrease the frame response times and allow a better bandwidth utilization. However implementing offsets requires a clock. In distributed systems there are two main solutions: all nodes share a global clock or each node has its own local clock. In both cases, each message is sent at an certain offset with regard to a clock.

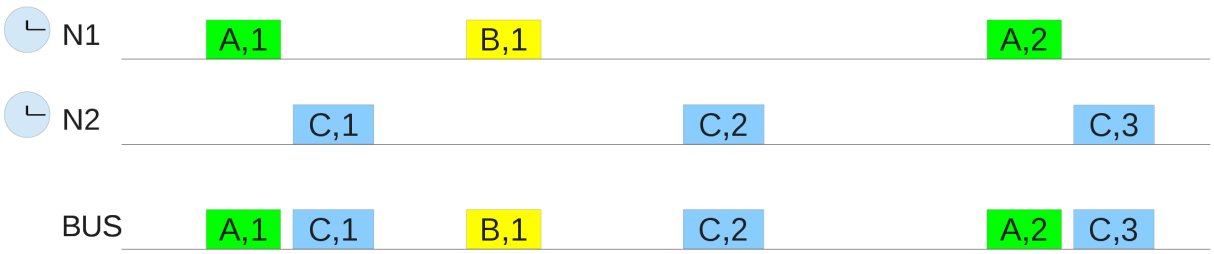

Figure 1: Schedule example with a global clock

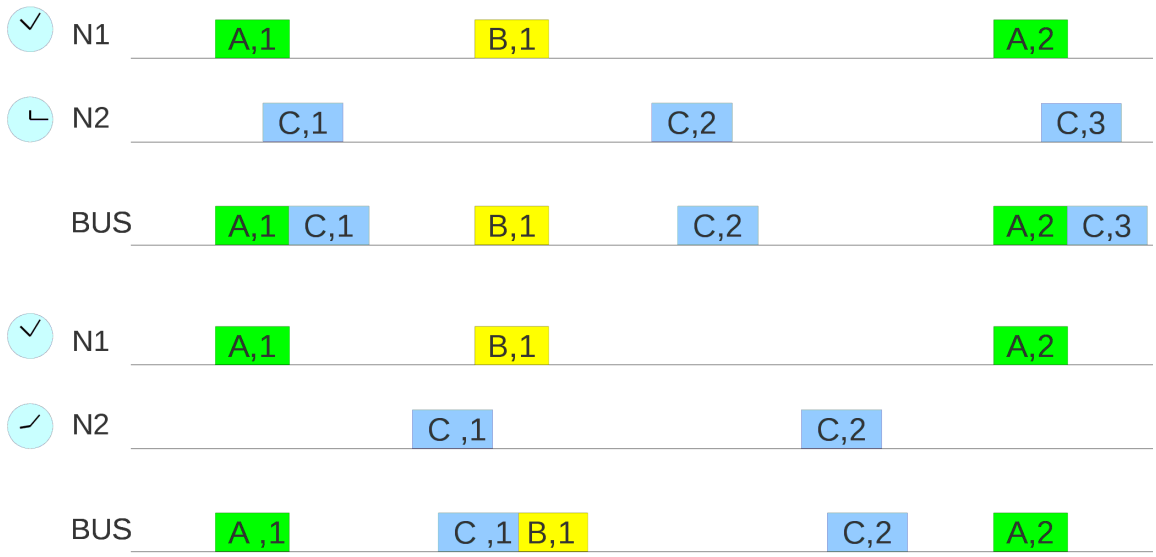

Figure 2: Schedule examples with local clocks

Weak synchronization

1.2 Weak synchronization

In case of global clock all nodes have (up to a certain precision) the same clock value, and with the proper time-triggered frame schedule no contentions occur, neither between the flows from the same node, nor from different nodes. However global clock requires synchronization mechanisms, and the clock precision must be much smaller than the sending time of one frame such that there is no contention. An example of such a schedule is given in Figure 1. A time slot is dedicated to each message, and no contention occurs between the 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 example schedules are given in Figure 2. Contentions between 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 trade-off 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. The phases between nodes is not perfectly known but bounded, and some contentions can be avoided. An example 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) between N1 and N2 is small enough, no contention can

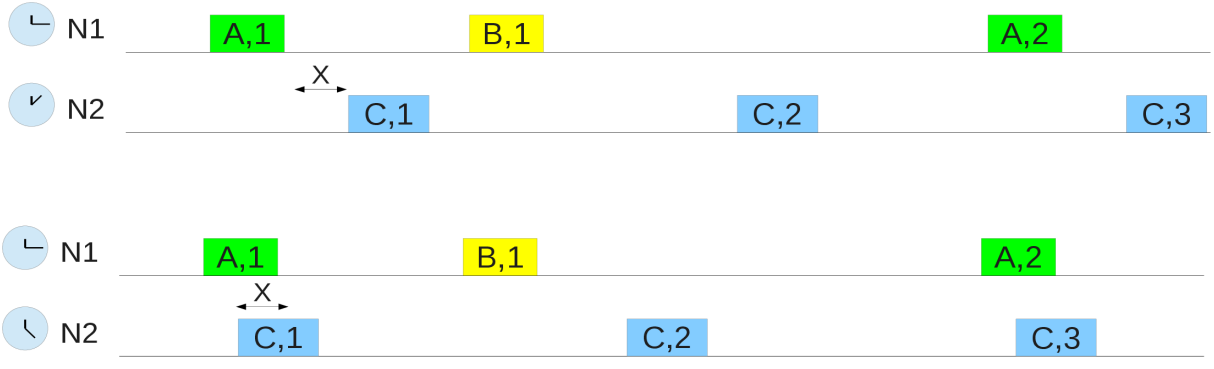

Figure 3: Schedule example with bounded phases

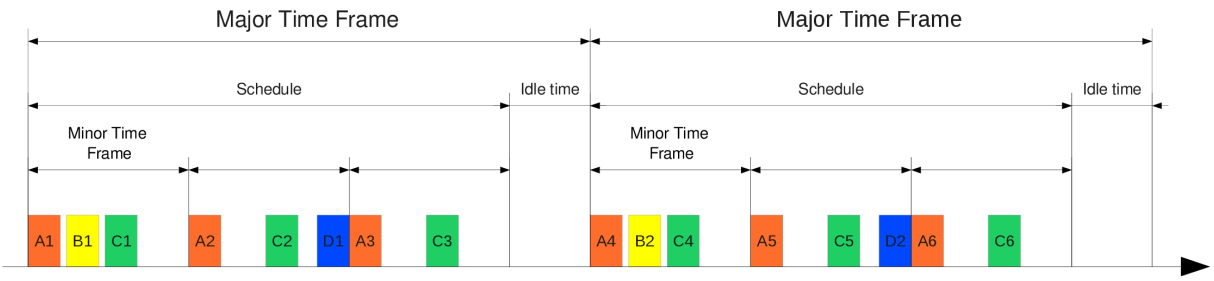

Figure 4: Example of scheduling using Major and Minor Time Frame

occur between flows B and C. This shows that it is possible to benefit from some of the advantages of a global clock with fewer constraints on the synchronization between nodes.

1.3 Contribution

We have shown in [1] that important gains with respect to the communication latencies can be achieved if we implement bounded clock de-synchronization. For the sake of understanding, some simplifications were done in [1]:

- the traffic associated to the synchronization mechanism had not been considered,
- no event-triggered traffic was considered,
- only standard CAN 2.0A was considered,
- transmission errors were not taken into account.

In this article, we propose to show an acceptable synchronization mechanism, how this method can be used in a context mixing asynchronous flows and periodic flows with offsets, how to take into account CAN FD traffic and how errors can be considered.

2. Computing an upper bound with network calculus

Network calculus is a theory to derive deterministic upper bounds on the communication latencies in networks. In [6] it has been shown that application of Network Calculus can bound the worst case response times for CAN bus. 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 $D \leq A$, 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 determined 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 [5].

Definition 1 *(Arrival curve)*: Let A be a flow, and α be a non decreasing function. Then, α is an arrival curve for flow A, iff :

$$\forall (t, d) \in \mathbb{R}^2 , A(t + d) - A(t) \leq \alpha(d)$$

Definition 2 (Service curve): A server S offers a strict service β iff for all input/output A,D and for all busy period (s,t]

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

Knowing the arrival and the service curve for a flow and a server it is possible to deduce a bound for the worst case traversal time. More details on network calculus can be found in [5] and for this specific case in [1].

3. Contribution

3.1 Synchronization protocol

Several synchronization protocols have been proposed, but their implementation on COTS components can be costly, and the use of dedicated hardware is not always possible. The one presented in this section describes an acceptable method. This method is based on a minor/major time frame (MIF/MAF) concept where all the nodes share the same minor/major time frame period, see Figure 4. At the end of each Major time frame an idle time of variable duration exists. Due to hardware and software latency, it may be hard to start all MAF at exactly the same time, but it could be possible to have a bounded phase between them.

The synchronization protocols that we envisage is based on a master that sends a frame at the beginning of the each Major Time Frame Cycle. This message is then used by each node on the bus in order to define a reference point for their own local clock. This message has to be sent not only at the beginning of the synchronization but at every new Major Time Frame because of the local clocks drifting apart.

However even if every node uses the same message to define the origin of their clocks it does not guarantee that they will be perfectly synchronized. A message transmission delay can be broken into four parts: a preprocessing time (Tpre), a waiting time (Twait), a transmission time (Tx) and a postprocessing time (Tpost), see Figure 5. And these times may vary for the same message considering different destination nodes.

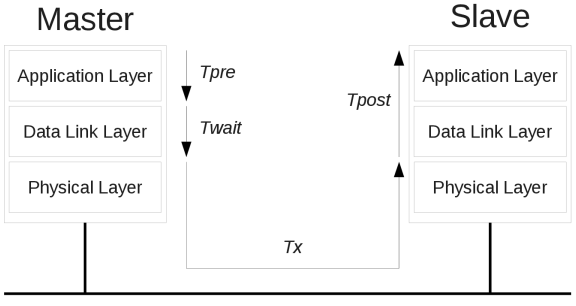

Figure 5: A timing diagram showing time spent sending a message from a source node to a destination node. (Fig 2 in [2])

The preprocessing time is the time required to acquire data from the environment and encode them into network data whereas the postprocessing time is the time required to decode the network data and transmit them to the environment. These times are unknown at design and depend on the device software and hardware characteristics, however they can be bounded. In this study we use results presented in [2] and consider that 0,5 ms < Tpre + Tpost < 1 ms. The waiting time is the time spent in the queue at the sender buffer. Even if we consider that our synchronization messages have the highest priority, CAN bus use a non-preemptive policy, due to asynchronous flows this time is unknown at design. However it is possible to bound it, in our case we consider that 0 ms < Twait < 0,5 ms. Finally the transmission time is the time required to physically transmit the message on the bus, it depends on the data rate, the message size and the distance between nodes. All these values are known at design time and so this transmission time can be calculated at design, in our case we consider Tx = 0,26 ms.

To summarize the time required to send a message from the master to a slave is unknown at design and may be different for each slave, however it is possible to bound it :

0.76 ms < Tpre+Twait+Tx+Tpost < 1,76 ms

This variable delay will lead to a weak synchronization between nodes, in our example for example local clocks of two different nodes may have at least a difference of 1ms, this difference between local clocks will be referred as “phase” thereafter.

Due to clock drift, a new synchronization message has to be sent periodically at each Major Time Frame by the master. The main point is that all the frames scheduled in a Major time Frame have to be sent before a node starts the next Major time Frame. Each node starts a new Major time Frame when receiving this synchronization message. This means that the master has to avoid to send this message too soon, and so a minimal idle time has to be defined at the end of the Major time Frame, see Figure 4.

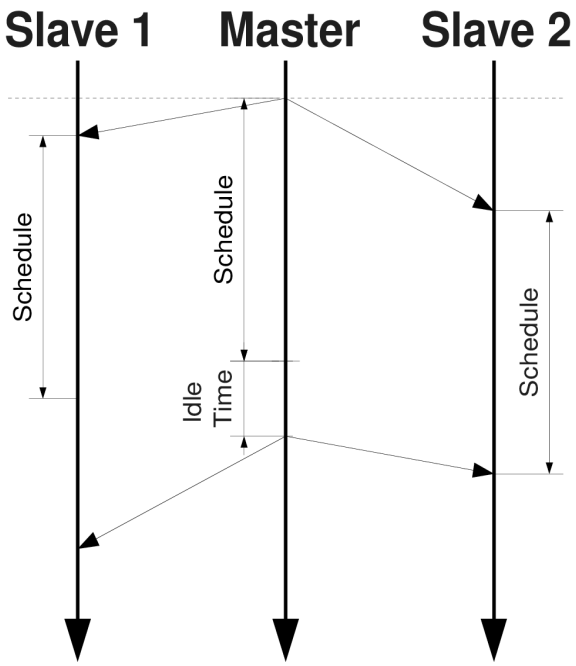

Figure 6: Minimal Major time Frame for the master

First, without considering clock drift, as the message transmission delay may vary (see above), the master has to take into account the worst case. This case is illustrated Figure 6. In this case the Slave 2 receives the first transmission frame with the maximal delay, so it starts its Major frame time later than Slave 1. The master has to wait in order to avoid that, if the following transmission message is sent with the minimal traversal time, Slave 2 receives it before the end of its transmissions. The slave has to receive these messages with a minimal time between them large enough to send all the frames. This imposes for the master a minimum idle time equal to the maximal phase between the master and the slave: ϕmax. In this case it ensures that all the slaves finish their transmission before the new major time frame, then they may have an idle time (Slave 1) or not (Slave 2).
Secondly, due to clock drift, the accuracy of the clock: ε, has to be taken into account. If the clock of the master is faster than the reference time and the clock of the slave slower, during a time T, their difference is bounded by: 2εT.

It lead us to:

$$\text{Idle Time} \geq 2\varepsilon \, \text{MAF} + \varphi_{max}$$

3.2 Sporadic/asynchronous flows and alarms

The method presented in [1] only considers periodic flows with offsets: messages are sent periodically with a known offset. However in practice many systems also contain sporadic/asynchronous flows: messages are sent as soon as specificevents occur, respecting a minimal duration between two successive frames. Such transmission can be triggered by alarms that, by definition, cannot be scheduled, or for instance the period of the flows sent by the engine in automotive networks depends on the engine frequency. However, in order to be able to respect timing constraints, asynchronous flows has a bandwidth limit defined by two parameters: a minimal duration between two successive frames, the Minimum Update Time (MUT), and a maximal frame size. These parameters can be used to define an arrival curve and so compute a worst case response time.

3.3 CAN FD

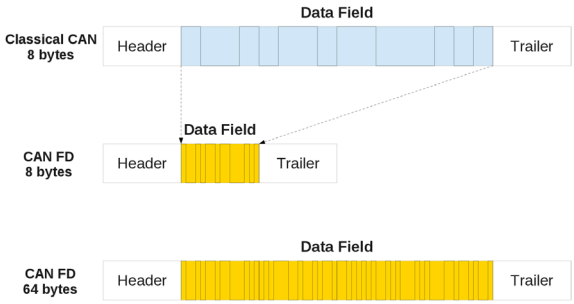

Figure 7: CAN FD data frame format

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 [3] (CAN with Flexible Data-rate). CAN FD modifies the CAN frame format by increasing the maximal number of data bytes per CAN frame up to 64 and by permitting to switch the bit rate to faster value inside the CAN frame. In Network Calculus we are interested in the frame

size from the point of view of the network, i.e. the duration of bus occupancy. In order to represent CAN FD in network calculus, it is sufficient to consider it as a classical CAN frame with the same bus occupancy duration, see Figure 7.

3.4 Transmission errors

Real-time distributed applications have often the obligation to respect stringent temporal constraints. It may be essential to take into account of transmission errors in Network Calculus in order to ensure that time constraints are respected. Transmission errors are a random phenomenon, and so it cannot be forecast. However Tindell and Burns, in [4], have introduced the idea that the number of errors can be upper bounded during a given time period. This upper bound is characterized by:

- Nerror, the burst errors, it is the maximal number of errors that could occur back-to-back
- Terror, the residual error period.

The number of transmission errors during time t is thus:

$$Nerror + \left\lceil \frac{t}{Terror} \right\rceil - 1$$

This result can then be used in Network Calculus for adapting the service curve. If β is the service curve without considering errors then the service curve with transmission errors is:

$$\beta error = \beta - \left(Nerror + \left\lceil \frac{t}{Terror} \right\rceil - 1 \right).(Lmax+Lerror)$$

Where Lmax is the maximal frame size, because each errors can lead to the loss of a complete frame, in the worst-case the largest frame of the system. And Lerror is the size of the error frame (23 bytes), see [10].

5. Case-studies

In order to show the usability of our method we decide to adapt a real CAN bus configuration presented in [8].

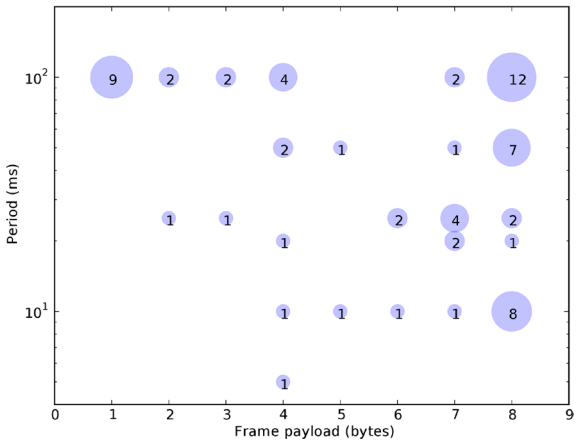

Figure 8: Distribution of the size of the data payloads and periods for the tested configuration.

The system model consists of 6 identical nodes that are connected to a single CAN network. There are 69 messages in the system. Figure 8 shows for the set of frames the number of flows per period and payload, as indicated by the size of circle and the number inside. For example the number 8 in the right bottom corner means that there are 8 flows with a period of 10ms and a payload of 8 bytes. The CAN bus data rate is 500 kbps and its utilization is 60,25 %. In a first step, we decide to consider the 69 messages as periodic. The frame offset assignment that is used in this study is the SOPA algorithm available in the RTaW-Pegase software from the company RTaW. SOPA algorithm has been chosen because our experience is that it consistently outperforms the few other offset algorithms available [7][9], and thus provides us with an estimate of the best possible gains that can be achieved in practice with offsets. Moreover we have supposed that the number of errors can be upper bounded as explained previously with Nerror = 2 and Terror = 100 ms.

The first experiment, see Figure 9, considers that all flows are purely periodic. In order to evaluate the gain due to the bounded synchronization we compare the delay bounds obtained without using offset and with only local clocks. The results with phases take into account an additional flow used to maintain the synchronization considered with the highest priority. Results can be compared on Figure 9, and reveal an average gain of around 53 % compared with

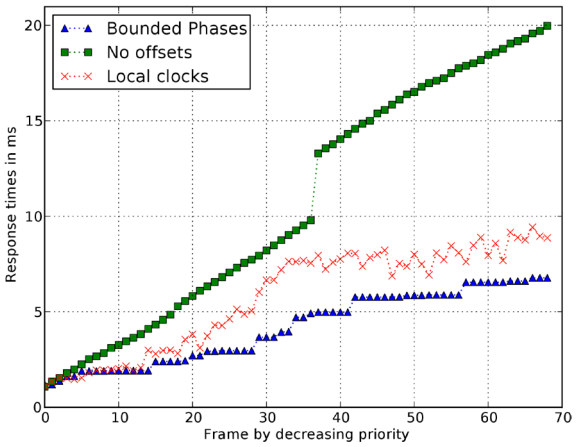

Figure 9: Delay bounds with only periodic messages (phase = 1 ms)

a system without offsets and an average gain of around 22 % compared with a system with local clocks. For very high priority frames (1-15) this gain is far lower because maintaining the synchronization requires to add a flow which increases the worst case response time as it will delay the others flows. However the gains of synchronization outweigh this disadvantage.

Table 1: Distribution between periodic and sporadic messages

		Periodic	Sporadic
Priorities	**1-17**	50 %	50 %
	18-34	75 %	25 %
	35-69	100 %	0 %
Part of the load		60 %	40 %

In a second step, we decide to consider that our systems does not only contain periodic flows but also sporadic messages. We consider that sporadic messages have a higher priority than periodic messages as they are used for alarms, so we decide to set as sporadic 50 % of messages from priority 1 to 17 and 25 % from priority 18 to 34; the rest of the traffic remains periodic. The distribution is summarized in Table 1.

It is important to notice that highest priority flows are also flows with the smallest period, that is why in this system, more than 40 % of the total load have been changed to sporadic traffic.

Results are presented Figure 10. For high priority frames (1-30) the gain of using offsets is limited because an important part of the traffic is sporadic, moreover as previously mentioned there is an additional flow to maintain the synchronization. However for frames of low priority (40-75) the gain due to offset remains very important, 45 % compared with a system without offset and 17 % compared with a system with local clocks.

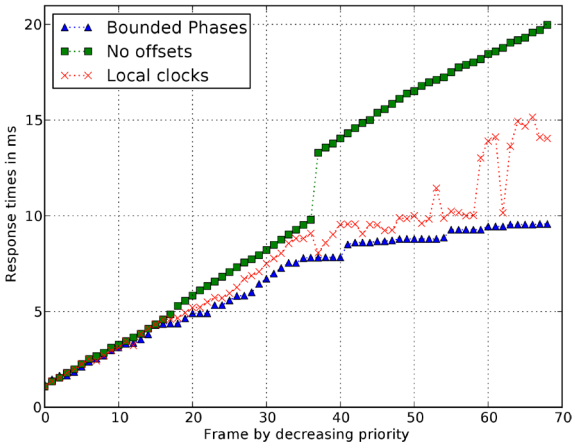

Figure 10: Delay bounds with 40 % of sporadic messages

6. Conclusion

The major contribution of this paper is to show the applicability in a realistic case-study of the new approch presented in [1]: using bounded clock desynchronization, which offers a trade-off between a global clock and local clocks. Using a global clock requires synchronization mechanisms with a precision much smaller than the frame sending time but their implementation on a COTS can be costly. Using local clocks does not avoid inter-nodes contentions.
In this paper we propose a simple synchronization mechanism to establish a system with bounded phases between nodes, but results presented also apply with any of them. We used the method developed in [1] to bound the delay of CAN with bounded desynchronization and show how this method can be used in a context mixing asynchronous flows and periodic flows with offsets. Furthermore we have extended the technique presented in [1] to take into account errors.

The experiments have brought insights on the beneficial impact of bounded phases, with, on our case-studies, an average delay reduction of around 50 % when all the traffic is periodic. Even when an important part of the traffic is sporadic the used of bounded phases remains very beneficial.

References

[1] DAIGMORTE, Hugo and BOYER, Marc. Traversal time for weakly synchronized CAN bus. In : Proceedings of the 24th International Conference on Real-Time Networks and Systems. ACM, 2016. p. 35-44.

[2] LIAN, Feng-Li, MOYNE, James, and TILBURY, Dawn. Network design consideration for distributed control systems. IEEE Transactions on Control Systems Technology, 2002, vol. 10, no 2, p. 297-307.

[3] HARTWICH, Florian. CAN with flexible datarate. In : Proc. ICC. 2012.

[4] TINDELL, Ken et BURNS, Alan. Guaranteed message latencies for distributed safety-critical hard real-time control networks. Dept. of Computer Science, University of York, 1994.

[5] LE BOUDEC, Jean-Yves and THIRAN, Patrick. Network calculus: a theory of deterministic queuing systems for the internet. Springer Science & Business Media, 2001.

[6] SOFACK, William Mangoua and BOYER, Marc. 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 Berlin Heidelberg, 2012. p. 258-272.

[7] GRENIER, Mathieu, GOOSSENS, Joël, and NAVET, Nicolas. Near-optimal fixed priority preemptive scheduling of offset free systems. In: 14th International Conference on Real-Time and Networks Systems (RTNS'06). 2006. p. 35--42.

[8] ZENG, Haibo, DI NATALE, Marco, GIUSTO, Paolo, et al. Using statistical methods to compute the probability distribution of message response time in controller area network. IEEE Transactions on Industrial Informatics, 2010, vol. 6, no 4, p. 678-691.

[9] GRENIER, Mathieu, HAVET, Lionel and NAVET, Nicolas. Pushing the limits of CAN – Scheduling frames with offsets provides a major performance boost, In: 4th European Congress Embedded Real Time Software (ERTS 2008), Toulouse, France, January 29 – February 1, 2008.

[10] NAVET, Nicolas, SONG, Y.-Q., et SIMONOT, Françoise. Worst-case deadline failure probability in real-time applications distributed over controller area network. Journal of systems Architecture, 2000, vol. 46, no 7, p. 607-617.

Daigmorte Hugo, Boyer Marc
ONERA
2, avenue E. Belin
FR-31055 Toulouse Cedex
Tel.: (33) 5.62.25.26.36
Fax: (33) 5.62.25.26.93
www.onera.fr/staff/marcboyer
www.onera.fr/staff/hugodaigmorte

Jörn Migge
RealTimeatWork
Immeuble Thiers, 4 rue Piroux
FR-54000 Nancy
Tel.: (33) 3.83.85.00.03
Fax: (33) 3.83.30.45.98
jorn.migge@realtimeatwork.com