CAN Bus Experiments of Real-Time Communications

Fernando G. Tinetti*, Fernando L. Romero, Alejandro D. Pérez

Instituto de Investigación en Informática LIDI (III-LIDI)
Facultad de Informática, UNLP, 50 y 120, La Plata, Argentina
{fernando, fromero}@lidi.info.unlp.edu.ar, perez.alejandrodaniel@gmail.com

Abstract. This paper presents a benchmark development and the analysis of communication times in a CAN (Controller Area Network) bus. Preemptive and fixed priority scheduling is taken as departure point for the initial experiments and analysis. This method would be adapted for programming CAN messages without preemption of the shared communications channel. It is also complemented by a specific priority assignment algorithm, suitable for non-preemptive systems. A real CAN network is used for the experiments, and results are analyzed from the point of view of real-time performance and CAN specification. We have found well known, deterministic, and correct performance communication times. Messages and signals are delivered within deadlines, a fundamental requirement for a real-time system.

Keywords: CAN Bus Implementation, Real-Time Experiments, Instrumentation and Monitoring, Real-Time Schedulability

1 Introduction

The initial development of a CAN time analysis is based on [6], which defines a preemptive and fixed priority task scheduling (mainly) for single-processor systems. This method is adapted for the programming of CAN messages and several enhancements defined in [2]. Basically, an optimal prioritization algorithm introduced in [1] is applicable to non-proprietary systems. Enhancements and optimizations can be experimentally verified (at least in part) by building a real CAN network where specific communication time measurements can be made. Time measurements/sampling can be then used for evaluating the fulfillment of CAN real time requirements.

A CAN bus is arbitrated using priorities, and once the bus access is obtained, the message is transmitted in full. Thus, the bus appropriative policy prevents another message from gaining access to the bus while a message is being transmitted. The priority inversion problem allowed by the CAN apropriativeness should be addressed. Regardless of priority inversion, every message has its own

* Comisión de Investigaciones Científicas, Prov. de Bs. As., corresponding author

deadline, and the bus arbitration will impose some delay that should be minimized in order to avoid real-time deadline violations. This work will be focused on the behavior of the Data Frames transmission of the CAN standard.

Messages in the CAN bus are handled by controllers, and a single controller may handle several CAN nodes. Also, each controller implements the bus arbitration while it maintains a queue of pending messages. The controller queue is managed according to fixed priority and non-preemptive scheduling. Controllers are synchronized at the start of a message with the clock of the sending node (rising edge). A transmission is allowed to start after the bus is idle for a 3-bit (interframe) period. A controller begins to transmit with a dominant bit ("'0"), which causes the rest of the controllers to synchronize with the rising edge of this bit and become receivers. Any message that is ready to be transmitted must wait for another arbitration cycle to start when the bus is again idle. The CAN protocol requires the transmitter to insert a bit with opposite polarity every 5 bits of the same polarity, a technique known as "bit stuffing". Thus, a stuffing bit may be added every 4 bits of the message. Taking into account the message bit length, mbl, the constant number of bits k needed for the node identification, the maximum number of bits added for stuffing, and the bit transmission time, τ_{bit}, the maximum transmission time of the message would be given by Mt_m in (1).

$$Mt_m = \left(mbl + k + \left\lfloor \frac{mbl + k - 1}{4} \right\rfloor \right) \tau_{bit} \tag{1}$$

2 Waiting, Errors, and Maximum Time

Given a specific application or experiment, the message types are predefined, i.e. each message m has a unique and fixed identifier and priority. Messages are queued in controllers by a software routine. Message queueing needs a run time between 0 and Jm, known as queuing the message jitter, and is inherited from the response time of the task, including the delay by polling [6]. The message time period Tm is considered for events that trigger the message queueing. There are three types of events associated with Tm: a) Events with exact period Tm, b) Events that occur sporadically with a minimum separation time Tm, and c) Events that occur only when the system starts. Each message has a hard deadline Dm, the maximum time from the initial event that generates the message to be queued to the actual message arrival at destination.

The worst case for response time Rm of a message can be defined as the longest time since the start event occurs until the message begins to be received by the node/s that require it. A message can be called "schedulable" if and only if its worst case response time is less than or equal to Dm: $Rm \leq Dm$. The whole system is "schedulable" if and only if all the messages are schedulable. Rm is basically the aggregation of: 1) The delay time Wm (also called Window time), which is the longest message waiting time in the CAN controller, and 2) The worst message transmission time, Mt_m. In turn, Wm is a function of: 1) Blocking time, Bm, given by messages of lower priority in the transmission process, and

2) Interference time, where a message with higher priority is transmitted earlier than m. The message blocking time can be calculated as $Bm = max(C_k)$ with $k \in lp(m)$, and $lp(m)$ is the set of messages with lower priority than m. The interference time is closely related to the concept of "busy period" introduced in [5], where a busy period of *level i* is defined as the time interval [a, b] within which the tasks with priority i or greater are processed within this interval and there are no tasks with priority i in the intervals $(a - t, a)$ and $(b, b + t)$ with $t > 0$. Adapting this definition to the CAN protocol, we have a busy period of priority m. The period starts at a time Ts where a message of priority m or greater is queued to transmit, and there are no queued messages of priority m or greater than m. It is a continuous interval of time, during which any priority message less than m is not transmitted. Finally, the period ends at time Te, when the bus becomes ready for the next transmission and arbitration cycle and there are no messages with priority greater than m.

The key characteristic of the busy period is that all messages of priority m or higher are transmitted within the period. In mathematical terms, the busy periods can be seen as intervals $[Ts, Te)$. Furthermore, the end of a busy period may coincide with the beginning of another busy period [5]. The busy period elapsed time can be calculated as the maximum of t_m^{n+1} given in equation 2, where $gep(m)$ is the set of messages with priority greater than or equal to m, and $f(t_m^n, t_j)$ is the function that combines t_m^n with the times related to message j: J_j (jitter) and T_j (time period).

$$t_m^{n+1} = B_m + \sum_{j \in gep(m)} \left\lceil f(t_m^n, tr_j) \right\rceil C_j \tag{2}$$

The busy period time given by the recurrence in equation 2 monotonically grows from B_m up to one of the following values: a) $t_m^n + C_m > D_m$, which makes the message non-schedulable or unschedulable, or b) $t_m^{n+1} = t_m^n$, where the worst response time of the first instance of the message is reached in the busy period determined by $t_m^n + C_m$

Errors and their related recovery times have to be taken into account when dealing with noisy environments. Each controller detecting an error starts a recovery process by transmitting an error signal. The resynchronization process takes at least 29 bit transmission times. Furthermore, resynchronization is only the first step of the error recovery process. Other tasks (and their corresponding times) involved in the recovery process are those required to the retransmission of the message affected by errors. The equation including errors and their associated times in noisy environments is given in equation 3, where E_m involves at least the 29-bit resynchronization and message retransmission, which is usually considered in the analysis as the worst case overhead time as given in equation 4.

$$t_m = B_m + E_m + \sum_{j \in gep(m)} \left\lceil f(t_m^n, tr_j) \right\rceil C_j \tag{3}$$

$$Over_m = \max_{j \in gep(m)} (C_j) + 29\tau_{bit} \tag{4}$$

There are other details (and their corresponding times) involved in dealing with errors, but the current work is focused on the basic analysis of CAN bus messages and delay times. Thus, we will focus on a real CAN network and experimentation in the next section without taking into account errors and error recovery time/s.

3 Specific Benchmark

The Society of Automotive Engineers (SAE) provides a set of test data to evaluate communication technologies handling the seven different subsystems in an electric car prototype [6]. Signal data details are used to show a case of "real" use for the timing model of the previous section, with the 53 signals given by the SAE and explained in [4]. The seven subsystems that generate signals include the brakes, the driver himself, the engine, etc. In total there are 53 signals, characterized not only by the subsystem that generates them but by other important details, such as the amount of information bits, if they are periodic or sporadic, jitter, etc. Every signal implies handling a message with specific real-time latency/deadline.

The order in which the messages in a controller queue are transmitted will be determined by their priorities, so priority definition will determine the schedulability of a system with specific signal periods. Also, priority assignment will be directly related to the robustness of the communication system when facing errors. Priority assignment "by deadline" (deadline monotonic priority assignment) is proposed in [6], i.e. those messages in the queue nearer their deadline will be assigned greater priority. This priority assignment policy is optimal for systems with preemptive and fixed priority task scheduling, assuming deadlines are no longer than message periods. A non-optimal policy does not mean it is useless, it actually means that another option should be chosen under (more) critical times/deadlines. Table 1 shows an example similar to that found in [2] in which an unschedulable system is generated by: a) assuming no jitter, 0 jitter time, b) the channel is already being used by 1ms by a lower priority message, LM, that gained access to the bus before messages M1, M2, and M3 arrive at the controller, and c) deadline monotonic priority assignment. Given that priorities

Table 1. Timing Example

Message	Period & Deadline	Trans. time
LM	N/A	1 ms
M1	3.0 ms	1.1 ms
M2	4.0 ms	1.1 ms
M3	4.5 ms	0.5 ms

assignment determine the order M1, M2, M3, if the channel is (already) busy by 1ms, then messages M1 and M2 use the channel by a total amount of 2.2ms.

Thus, while message M2 is being transmitted, a new instance of message M1, is queued and this new instance is the one with highest priority. Then, while the new instance of message M1 is being transmitted a new instance of message M2 is queued, and message M3 is delayed beyond its deadline. The sequence of messages in the channel would be

LM => M1, after 1ms => M2, after 2.1ms => M1', after 3.2ms

because the channel is not initially preempted from the lower priority message/s (represented by LM in the sequence above) and priorities are fixed and determined by deadline/s. However, if the order of messages is determined as M1, M3, and M2 the worst response times are R_{M1} = 2.1ms, R_{M2} = 2.6ms, and R_{M3} = 3.7ms. Thus, this *new* priority assignment determines a schedulable system, where response times $R_m < D_m$, $\forall m$. The new *priority assignment* imply the following sequence of messages in the communication channel

LM => M1, after 1ms => M3, after 2.1ms => M2, after 2.6ms

and no new instances of messages M1 and M2 will generate any delay on message M3. In this context, the priority assignment algorithm given in [1] is claimed to be optimal in non-preemptive systems in [3]. In general, the algorithm given in [1] is useful as long as the worst case of response time of a message: a) does not depend on an ordering of the highest priority messages, and b) does not change its length (it is not made longer, in particular) when having a higher priority.

The length of the delay queue and the length of the busy period do not depend on a specific order of the high priority messages. The blocking time, B_m, can be made longer by changing the priority, but at the same time the interference is decreased. The priority assignment algorithm given in [1] performs a maximum of $n(n-1)/2$ evaluations for n messages, and guarantees message scheduling when possible. It should be borne in mind that the algorithm does not specify an order in which messages should be analyzed at each priority level. This order greatly influences the priority assignment if there is more than one scheduling, and a *poor* choice of initial order may result in a non-optimal scheduling.

Given a set of CAN messages, it is possible to assign the corresponding priorities by following a sequence of simple steps

1. Define the message characteristics: size, identifier, bus speed, etc.
2. Sort messages by some criterion, whether those given in [6] or [1].
3. Apply the timing model to each message. Usually, it is possible to analyze the minimum transmission rate at which the system is schedulable.

In the case of the SAE signals and data, the "D - J" method given in [6] will be used as a guide, in which higher priority is assigned to signals with lower waiting time, (that is, a smaller D - J value, being D and J Deadline time and Waiting time, respectively). Once signals have their priorities, it is possible to analize the system's schedulability with the timing model given above and considering the data rate of the communication channel. Besides, the channel utilization is may be enhanced with a technique referred to as *"piggyback"*. Basically, the piggyback technique takes advantage of periodic signals that can be grouped in a single message. Table 2 shows the first 10 signals defined by the SAE, all of them of a single-byte periodic signals with deadline time (D) equal to period

(P), and jitter time given as J, where it is possible to identify 4 signals with the same period from the same source. More specifically, signals 1, 2, 4, and 6 from the Battery subsystem are sent every 100 ms. Instead of sending 4 one-byte messages, a single message is sent with 4 bytes using the piggyback technique. Piggybacking reduces bus overhead of three messages. The transmission of a data byte can require the transmission of up to 63 bits. This technique can be extended to periodic signals that have different periods, as long as the minimum time period is a divisor of the other period times. For example, signals 29, 30, and 32 have the same source subsystem, with periods 10 ms, 10 ms, and 5 ms respectively. Thus, every 2 messages of signal 32, the signals 29 and 30 can be added to the same message with piggybacking.

Table 2. First 10 SAE Signals

#S	Signal	From	D = P	J
1	Hi&Lo Contactor Open/Close	Battery	100 ms	0.6 ms
2	Brake Pressure, Line	Battery	100 ms	0.7 ms
3	Processed Motor Speed	Battery	1000 ms	1.0 ms
4	Torque Command	Battery	100 ms	0.8 ms
5	Brake Pressure, Master Cylinder	Battery	1000 ms	1.1 ms
6	Accelerator Position	Battery	100 ms	0.9 ms
7	Torque Measured	Driver	5 ms	0.1 ms
8	Transaction Clutch Line Pressure	Brakes	5 ms	0.1 ms
9	Clutch Pressure Control	Brakes	5 ms	0.2 ms
10	High Contactor Control	Trans	100 ms	0.2 ms

The piggyback technique can also be applied to sporadic signals. These signals can be sent in a "server" message where the station sending the message checks for the occurrence of the sporadic signal before queuing the message. With this approach, a sporadic signal can be delayed no longer than the polling period time, plus the worst latency time of the "server" message. For a message with deadline of 20 ms, a server message with a 15 ms polling period and 5 ms worst case latency the technique would be good enough. Combining piggybacking with "server" messages for the sporadic signals the SAE benchmark/system is schedulable with a data rate of 125 Kb/s. Piggybacking provides most of the optimization, making possible channel utilization below 100%.

4 Experiments on a Real CAN Network

Fig. 1-a) schematically shows the CAN network we built to validate the previous analysis in a real network, specifically for experimentation. Real-time generation signals/messages generation is achieved by using Arduino development cards (UNO and Mega). Fig. 1-b) shows several runtime network data, in particular the Arduino output with timestamped events. The timestamped events are received

in the PC at runtime from the Arduino Mega. Data shown on the screen is only for visual monitoring, the real-time and precise timing calculations are made in a spreadsheet after each specific experiment.

Fig. 2 follows the same construction scheme of the experimental CAN network, including the Kinetis K70 development card. The Kinetis K70 includes a CAN interface, so it was immediately available to interoperate with the Arduino-based CAN network.

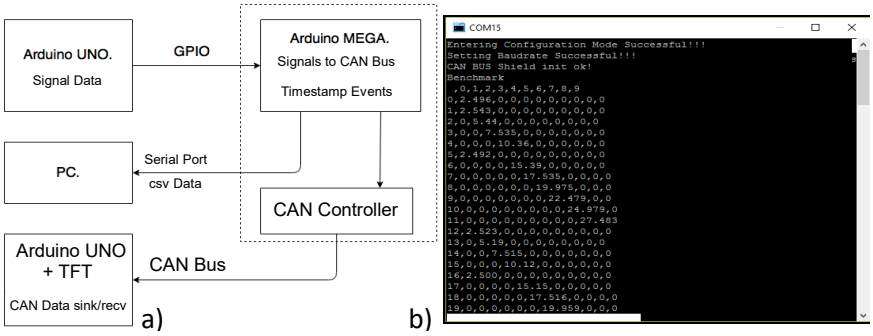

Fig. 1. CAN Network: Schematic and Monitoring

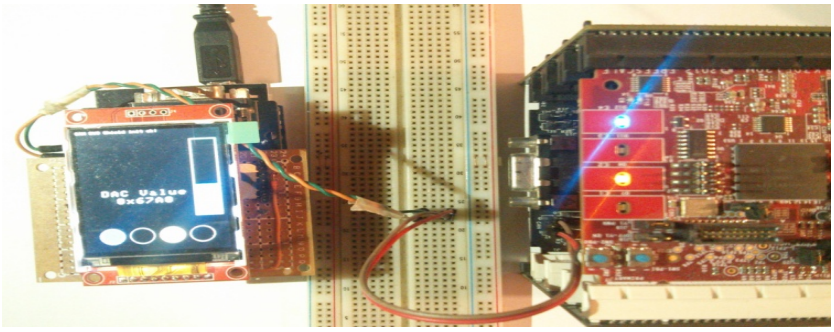

Fig. 2. CAN Network Including a Kinetis K70 Node

Although the data presented in Table 3 corresponds to a very short elapsed time of the experiments it is possible to verify that messages in the real network are sent sequentially, each one adding its transmission time to the waiting time of the next messages to be transmitted. Thus, it resembles the incremental sequence of message times as in equation 2 of the timing model given in a previous section. Experimentation on the real network allows us to verify that:

– The timing model is accurate in the sense that it represents the actual behavior of a real network, i.e. there is a *busy period* in which messages with low priority have to wait for a message with high priority, implying monotonically increasing waiting times.
– The actual writing time of a specific sequence of messages, where the waiting time is not always the maximum possible, because messages are delivered as soon as the communication channel is available.

Table 3. Timing Measurements in a Real CAN Network

	0	1	2	3	4	5	6	7	8	9
0	2.576	0	0	0	0	0	0	0	0	0
1	2.579	0	0	0	0	0	0	0	0	0
2	2.584	0	0	0	0	0	0	0	0	0
3	0	2.66	0	0	0	0	0	0	0	0
4	0	0	5.248	0	0	0	0	0	0	0
5	0	0	0	7.84	0	0	0	0	0	0
6	0	0	0	0	10.432	0	0	0	0	0
7	0	0	0	0	0	13.27	0	0	0	0
8	0	0	0	0	0	0	15.564	0	0	0
9	0	0	0	0	0	0	0	18.163	0	0
10	0	0	0	0	0	0	0	0	0	20.752

Fig. 3 shows the sequence of message arrivals to the CAN controller, and waiting times calculated from the measurements taken in the real CAN network while processing the SAE benchmark data. As schematically shown in Fig. 1, the Arduino UNO generated the signal data (those defined by the SAE benchmark), which the Arduino Mega handles for

– Message transmission to the CAN Bus, i.e. queueing and the real transmission.
– Message time instrumentation and monitoring, i.e. timestamping events and sending monitoring data to an external data analizer (a PC in this case).

Each vertical bar in Fig. 3 identifies a particular message or message instance (where the signal type is identified by a color or gray level) numbering them from 0 onwards (on the x-axis). The height of each bar is directly proportional to the message elapsed time from the event of reaching the CAN controller until it is received at destination. Looking in detail at the first 10 bars of Fig. 3, it is possible to identify that:

– The CAN messages 0 to 2 belong to the same signal, 0, which has a period of 30 ms, so the sender can send the 3 messages before the arrival of signal 1, which has a period of 100ms.

– After sending the first 3 messages, in the same instant (at 100ms) data from signals 1 to 8 arrive. These all enter at the same time, so there are incremental times of messages 3 to 10 in the graph, depending on the CAN behavior: messages are sent taking into account the corresponding priorities.
– Messages that arrived at the same instant are sent in less than 30ms and then a new message of signal 0 arrives. It is important to note that none of the messages had a latency higher than the worst time previously calculated with the timing model. Thus, it can be said that the model, as well as its implementation have been successfully tested.

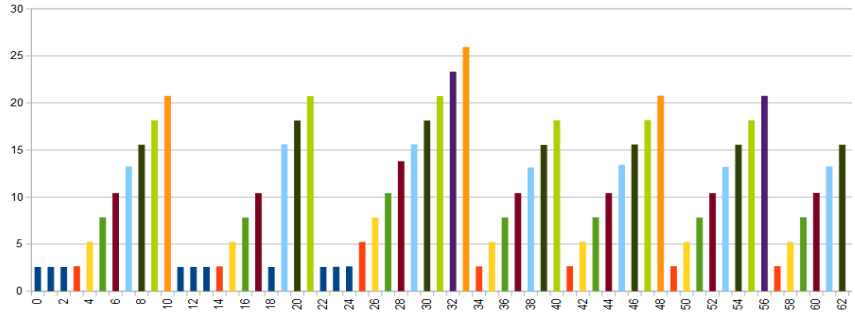

Fig. 3. Busy Periods in the CAN Network

5 Conclusions and Further Work

It has been possible to carry out the experimental corroboration of the timing model analysis proposed in previous works regarding latency and response times in the transmission of messages using the CAN protocol. The mathematical representation of each of the phenomena present on the bus has been analyzed with a set of signals of a hypothetical system, based on the SAE benchmark. This approach has been extensively used by automobile and machinery manufacturers when considering their CAN-based systems. Thus the analysis in this work is useful in itself and allows for new research and approaches. A real CAN network has been implemented and validated, building not only the hardware on which to carry out the experimentation but also including the necessary instrumentation for the collection of data to verify with the mathematical model.

As part of the future work, a careful study should be considered to increase the experimentation hardware environment, mainly in terms of having a CAN network closer to the real ones regarding the number of controllers. Analogously, the bus bandwidth and the number of message sources (types of signals involved) that each CAN controller must handle when accessing the CAN bus must be carefully determined.

References

1. Audsley, N.: Optimal priority assignment and feasibility of static priority tasks with arbitrary start times (1991)
2. Davis, R.I., Burns, A., Bril, R.J., Lukkien, J.J.: Controller area network (can) schedulability analysis: Refuted, revisited and revised. Real-Time Systems 35(3), 239–272 (Apr 2007), https://doi.org/10.1007/s11241-007-9012-7
3. George, L., Rivierre, N., Spuri, M.: Preemptive and Non-Preemptive Real-Time UniProcessor Scheduling. Research Report RR-2966, INRIA (1996), https://hal.inria.fr/inria-00073732, projet REFLECS
4. Kopetz, H.: A solution to an automotive control system benchmark. In: 1994 Proceedings Real-Time Systems Symposium. pp. 154–158 (Dec 1994)
5. Lehoczky, J.P.: Fixed priority scheduling of periodic task sets with arbitrary deadlines. In: [1990] Proceedings 11th Real-Time Systems Symposium. pp. 201–209 (Dec 1990)
6. Tindell, K., Burns, A., Wellings, A.: Calculating controller area network (can) message response times. Control Engineering Practice 3(8), 1163 – 1169 (1995), http://www.sciencedirect.com/science/article/pii/0967066195001128