

What CAN Go Wrong in CAN (Timing Analysis)

Marco Di Natale
Scuola Superiore S. Anna

Copyright © 2009 SAE International

ABSTRACT

The advent of active-safety and safety-critical functions, including by-wire systems, and the interdependency of these functions is rapidly changing the scenario of automotive systems. OEMs need to understand and control functional and timing properties, including end-to-end latencies of distributed computations. The evaluation of the timing behavior can be very complex, considering the communication and synchronization model between application tasks, middleware, and network drivers, and the scheduling choices for tasks and messages. In this view, the timing behavior of CAN messages is of very high importance. In this paper we present some of the challenges in the evaluation of CAN message latencies.

INTRODUCTION

The CAN protocol adopts a collision detection and resolution scheme, where the message to be transmitted is chosen according to its identifier. When multiple nodes need to transmit over the bus, the lowest identifier message is selected for transmission. This MAC arbitration protocol allows encoding the message priority into the identifier field and implementing priority-based real-time scheduling of periodic and aperiodic messages.

Predictable scheduling of real-time messages on the CAN bus is then made possible by adapting existing real-time scheduling algorithms to the MAC arbitration protocol or by superimposing a higher-level purposely designed scheduler.

Starting from the early 90's solutions have been derived for the worst case latency evaluation. The analysis method, commonly known in the automotive world as Tindell's analysis (from [12]) is very popular in the academic community and had a substantial impact on

the development of industrial tools and automotive communication architectures. The original paper has been cited more than 200 times, its results influenced the design of on-chip CAN controllers like the Motorola msCAN and have been included in the development of the early versions of Volcano's Network Architect tool. Volvo used these tools and the timing analysis results from Tindell's theory to evaluate communication latency in several car models, including the Volvo S80, XC90, S60, V50 and S40 [2]. The analysis was later found to be flawed, although under quite high network load conditions, that are very unlikely to occur in practical systems. Davis and Bril provided evidence of the problem as well as a set of formulas for the exact or approximate evaluation of the worst case message response times (or latencies) in [3].

The real problem with the existing analysis methods, however, are a number of assumptions on the architecture of the CAN communication stack that are seldom true in practical automotive systems. These assumptions include the existence of a perfect priority-based queue at each node for the outgoing messages, the availability of one output register for each message (or preemptability of the transmit registers) and immediate (zero-time) or very fast copy of the highest priority message from the software queue used at the driver or middleware level, to the transmit register(s) or TxObjects at the source node as soon as they are made available by the transmission of a message.

When these assumptions do not hold, as unfortunately is the case for many systems, the latency of the messages can be significantly larger than what is predicted by the analysis. A relatively limited number of studies have attempted the analysis of the effect of additional priority inversion and blocking caused by limited TxObject availability at the CAN adapter. The problem has been

discussed first in [13] where two peripheral chips (Intel 82527 and Philips 82C200) are compared with respect to the availability of TxObjects. The possibility of unbounded priority inversion is shown for the single TxObject implementation with preemption and message copy times larger than the time interval that is required for the transmission of the interframe bits.

The effect of multiple TxObject availability is discussed in [8] where it is shown that even when the hardware transmits the messages in the TxObjects according to the priorities (identifiers) of the messages, the availability of a second TxObject is not sufficient to bound priority inversion. It is only by adding a third TxObject, under these assumptions, that priority inversion can be avoided.

However, sources of priority inversion go well beyond the limited availability of TxObjects at the bus adapter. In the next section we review the time analysis of the ideal system configuration and later, the possible causes of priority inversion, as related to implementation choices at all levels in the communication stack. We will detail and analyze all possible causes of priority inversion and describe their expected impact on message latencies. Examples derived from actual message traces will be provided to show typical occurrences of the presented cases.

IDEAL BEHAVIOR AND WORST CASE RESPONSE TIME ANALYSIS

This section starts from the description of the original analysis method by Tindell. Although formally not correct in the general case (later, the flaw is discussed, together with the fix proposed in [3]), it is still valid when the worst-case response time (or latency) of a message is not caused by a busy period that extends beyond the message period (or interarrival time), and provides a very good introduction to the general analysis method.

NOTATION - We assume the system is composed by a set of *periodic* or *sporadic* messages with *queuing jitter*, that is, messages are enqueued at periodic time instants or two consecutive instances of the same message are enqueued with a *minimum interarrival time*. In addition, the actual queuing instants can be delayed with respect to the reference periodic event stream by a variable amount of time, upper bounded by the *queuing jitter*. For the purpose of schedulability analysis, a periodic or sporadic message stream M_i is characterized by the t-uple

$$M_i = \{ m_i, id_i, N_i, T_i, J_i, D_i \}$$

where m_i is the length of the message in bits. CAN frames can only contain a data content that is multiple of 8 bits. Hence, m_i is the smallest multiple of 8 bits that is larger than the actual payload. id_i is the CAN identifier of the message, N_i the index of its sending node, T_i its period or minimum interarrival time, J_i (in case the

message is periodic, $J_i=0$ if the message is sporadic) its arrival jitter and D_i its deadline (relative to the release time of the message). In almost all cases of practical interest it must be $D_i < T_i$ because a new instance of a message will overwrite the data content of the old one if it has not been transmitted yet.

Messages are indexed according to their identifier priority, that is, $i < j \rightarrow id_i < id_j$. B_r is the bit rate of the bus, and p is the number of protocol bits in a frame. In the case of a standard frame format (11-bit identifier), then $p=46$, if an extended format (29-bit identifier) is used, then $p=65$. In the following, we assume a standard frame format, but, the formulas can be easily adapted to the other case. The worst case message length needs to account for bit stuffing (only 34 of the 46 protocol bits are subject to stuffing) and e_i is the time required to transmit the message provided that it wins the contention on the bus

$$e_i = \frac{m_i + p + \left\lfloor \frac{m_i + 34}{4} \right\rfloor}{B_r}$$

Finally, w_i is the *queuing delay*, that is, the worst case interval, from the time message M_i is enqueued (at the middleware level) to the time it starts transmission on the bus.

Please note that, in reality, messages are enqueued by tasks. In most cases, it is actually a single task, conceptually at the middleware level, that is called TxTask. If this is the case, the queuing jitter of each message can be assumed to be the worst case response time of the queuing TxTask.

In modelling the scheduling problem for CAN, one assumption is commonly used. Each time a new arbitration starts, the CAN controller at each node enters the highest priority message that is available. In other words, it is assumed that the peripheral TxObjects always contain the highest priority messages in the middleware queue, and the controller selects the highest priority (lowest id) among them. If this is the case, the worst case response time of M_i is given by

$$R_i = J_i + w_i + e_i$$

The queuing delay term w_i consists of two terms:

- blocking B_i due to a lower priority message that is transmitted when M_i is enqueued
- interference I_i due to higher priority messages that are transmitted before M_i on the bus

The queuing delay w_i is part of the busy period of level i , defined as follows: a contiguous interval of time that starts at the time t_s when a message of priority higher than i is queued for transmission and there are no other higher priority messages waiting to be transmitted that

were queued before t_s . During the busy period, only messages with priority higher than i are transmitted. It ends at the earliest time t_e when the bus becomes idle or a message with priority lower than i is transmitted.

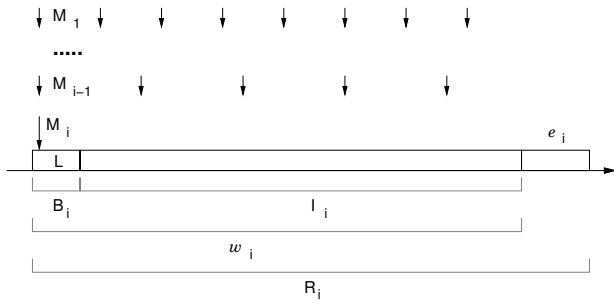


Figure 1: Worst case response time, busy period and time critical instant for M_i .

The worst-case queuing delay for message M_i occurs for some instance of M_i queued within a level- i busy period that starts immediately after the longest lower priority frame is transmitted on the bus. The busy period that corresponds to the worst case response time must occur at the critical instant for M_i , that is, when M_i is queued at the same time together with all the other higher priority messages in the network. Subsequently, these messages are queued with their highest possible rate (if sporadic). The situation is represented in figure 1. Formally, this requires the evaluation of the following fixed-point formula for the worst-case queuing delay:

$$w_i = B_i + \sum_{k \in hp(i)} \left\lceil \frac{w_i + J_k + \tau_{bit}}{T_k} \right\rceil e_k \quad (1)$$

where $hp(i)$ is the set of messages with priorities higher than i and τ_{bit} is the time for the transmission of one bit on the network. The solution to the fixed-point formula can be computed considering that the right hand side is a monotonic non-decreasing function of w_i . The solution can be found iteratively using the following recurrence.

$$w_i^{(m+1)} = B_i + \sum_{k \in hp(i)} \left\lceil \frac{w_i^{(m)} + J_k + \tau_{bit}}{T_k} \right\rceil e_k$$

A suitable starting value is $w_i^{(0)} = B_i$, or $w_i^{(0)} = \varepsilon$, with $\varepsilon \ll C_j \forall j$ for the lowest priority message. The recurrence relation iterates until, either $w_i^{(m+1)} > D_i$, in which case the message is not schedulable, or $w_i^{(m+1)} = w_i^{(m)}$ in which case the worst-case response time is simply given by $w_i^{(m)}$. The flaw in the above analysis is that, given the constraint $D_i \leq T_i$, it implicitly assumes that if M_i is schedulable, then the priority level- i busy period will end at or before T_i .

Please note that this assumption fails only when the network is loaded to the point that the response time of M_i is very close to its period. Given the typical load of

CAN networks (seldom beyond 65%) this is almost never the case.

When there is such a possibility, the analysis needs to be corrected as follows. To explain the problem and the fix, we refer to the same example in [3] with three messages (A, B and C) with very high utilization. All messages have a transmission time of 1 unit and periods of, respectively, 2.5, 3.5 and 3.5 units.

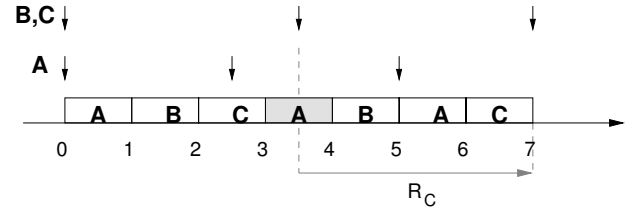


Figure 2: An example showing the need for a fix in the evaluation of the worst-case response time.

The critical instant is represented in Figure 2. Because of the impossibility of preempting the transmission of the first instance of C, the second instance of message A is delayed, and, as a result, it pushes back the execution of the second instance of C. The result is that the worst case response time for message C does not occur for the first instance (3 time units), but for the second one, with an actual response time of 3.5 time units. The fix can be found observing that the worst case response time is always inside the busy period for message C. Hence, to find the correct worst case, the formula to be applied is a small modification of (1) that checks all the q instances of message M_i that fall inside the busy period starting from the critical instant. Analytically, the worst-case queuing delay for the q -th instance in the busy period is:

$$w_i(q) = B_i + qe_i + \sum_{k \in hp(i)} \left\lceil \frac{w_i(q) + J_k + \tau_{bit}}{T_k} \right\rceil e_k \quad (3)$$

and its response time is

$$R_i(q) = J_i + w_i(q) - q T_i + e_i$$

where q ranges from 0 to the last instance of M_i inside the busy period, that is, until $R_i(q) < T_i$. After the end of the busy period, the worst case is simply found as

$$R_i = \max_q \{ R_i(q) \}$$

Alternatively, an upper bound to the worst case response time may simply be found by substituting the worst case frame transmission time in place of B_i in formula (1). Please note that the previous analysis methods are based on the assumption that the highest priority active message at each node is considered for bus arbitration.