Integrating Mixed Transmission and Practical Limitations with the Worst-Case Response-Time Analysis for Controller Area Network

Saad Mubeen[a,b], Jukka Mäki-Turja[a,b], Mikael Sjödin[a]

Contact: saad.mubeen@mdh.se, +46 704 274 019

[a]*Mälardalen Real-Time Research Centre (MRTC), Mälardalen University, Västerås, Sweden*

[b]*Arcticus Systems AB, Järfälla, Sweden*

Abstract

The existing worst-case response-time analysis for Controller Area Network (CAN) calculates upper bounds on the response times of messages that are queued for transmission either periodically or sporadically. However, it does not support the analysis of mixed messages. These messages do not exhibit a periodic activation pattern and can be queued for transmission both periodically and sporadically. They are implemented by several higher-level protocols based on CAN that are used in the automotive industry. We extend the existing analysis to support worst-case response-time calculations for periodic and sporadic as well as mixed messages. Moreover, we integrate the effect of hardware and software limitations in the CAN controllers and device drivers such as abortable and non-abortable transmit buffers with the extended analysis. The extended analysis is applicable to any higher-level protocol for CAN that uses periodic, sporadic and mixed transmission modes.

Keywords: Distributed embedded systems, controller area network, CAN protocol, real-time network, response-time analysis, schedulability analysis.

1. Extended version

This paper extends our previous works that are published in the conferences as a full paper in [1] and two work-in-progress papers (discussing basic ideas and preliminary work) in [2] and [3] respectively. To be precise, the work in this paper generalizes the response-time analysis for Controller Area

Network (CAN) [4] developed in [1] by extending the proposed analyses in [2] and [3]. In addition, we conduct a case study to show a detailed comparative evaluation of the extended analyses.

2. Introduction

CAN is a multi-master, event-triggered, serial communication bus protocol supporting speeds of up to 1 Mbit/s. It has been standardized in ISO 11898-1 [5]. It is a widely used protocol in the automotive domain. It also finds its applications in other domains, e.g., industrial control, medical equipments and production machinery [6]. There are several higher-level protocols for CAN that are developed for many industrial applications such as CANopen, J1939, Hägglunds Controller Area Network (HCAN) and CAN for Military Land Systems domain (MilCAN). CAN is often used in hard real-time systems that have stringent deadlines on the production of their responses. The need for safety criticality in most of these systems requires evidence that the actions by them will be provided in a timely manner, i.e., each action will be taken at a time that is appropriate to the environment of the system. For this purpose, *a priori* analysis techniques such as schedulability analysis [7, 8, 9] have been developed. Response Time Analysis (RTA) [10] is a powerful, mature and well established schedulability analysis technique. It is a method to calculate upper bounds on the response times of tasks or messages in a real-time system or a network respectively. RTA applies to systems (or networks) where tasks (or messages) are scheduled with respect to their priorities and which is the predominant scheduling technique [11].

2.1. Motivation and related work

Tindell et al. [12] developed RTA for CAN which has been implemented in the industrial tools, e.g., VNA tool [13]. Davis et al. [14] refuted, revisited and revised the analysis by Tindell et al. The revised analysis is also implemented in an industrial tool suite Rubus-ICE [15, 16]. The analysis in [12, 14] assumes that each node picks up the highest priority message from its transmit buffers when entering into the bus arbitration. This assumption may not hold in some cases due to different types of queueing policies and hardware limitations in the CAN controllers [6, 17, 18]. The different types of queueing polices in the CAN device drivers and communications stacks, internal organization, and hardware limitations in CAN controllers may have significant impact on the timing behavior of CAN messages.

Various practical issues and limitations due to deviation from the assumptions made in the seminal work [12, 14] are discussed in [19] and analyzed by means of message traces in [6]. A few examples of these limitations that are considered in RTA for CAN are controllers implementing First-In, First-Out (FIFO) and work-conserving queues [20, 18], limited number of transmit buffers [21, 22], copying delays in transmit buffers [23], transmit buffers supporting abort requests [17], device drivers lacking abort request mechanisms in transmit buffers [23], and protocol stack prohibiting transmission abort requests in some configurations, e.g., AUTOSAR [24].

In [18, 20], Davis et al. extended the analysis of CAN with FIFO and work-conserving queues while supporting arbitrary deadlines of messages. In [21], it is proved that the priority inversion due to limited buffers can be avoided if the CAN controller implements at least three transmit buffers. However, RTA in [21] does not account the timing overhead due to copying delay in abortable transmit buffers. Khan et al. [17] integrated this extra delay with RTA for CAN [12, 14]. RTA for CAN with non-abortable transmit buffers is extended in [23, 22]. However, none of the above analyses support messages that are scheduled with offsets. The worst-case RTA for CAN messages with offsets is developed in several works including [25, 26, 27].

However, all these analyses assume that the messages are queued for transmission either periodically or sporadically. They do not support mixed messages which are simultaneously time (periodic) and event (sporadic) triggered. Mixed messages are implemented by several higher-level protocols for CAN that are used in the automotive industry. Mubeen et al. [1] extended the seminal RTA [12, 14] to support mixed messages in CAN where nodes implement priority-based queues. Mubeen et al. [28] further extended the RTA to support mixed messages in the network where some nodes implement priority queues while others implement FIFO queues. Mubeen et al. also extended the existing RTA for CAN to support periodic and mixed messages that are scheduled with offsets [29, 30]. In [2] and [3] we presented the basic idea for analyzing mixed messages in CAN with controllers implementing abortable and non-abortable transmit buffers respectively.

2.2. Paper contributions

We extend and generalize the RTA for periodic, sporadic and mixed messages in CAN by integrating it with the effect of buffer limitations in the CAN controllers namely abortable and non-abortable transmit buffers. The relationship between the existing and extended RTA for CAN is shown in

Figure 1. The analyses enclosed within the dashed-line box in Figure 1 are the focus of this paper. The extended analysis is able to analyze network communications in not only homogeneous systems, but also heterogeneous systems where:

1. CAN-enabled Electronic Control Units (ECUs) are supplied by different tier-1 suppliers such that some of them implement abortable transmit buffers, some implement non-abortable transmit buffers, while others may not have buffer limitations because they implement very large but finite number of transmit buffers;
2. any higher-level protocol based on CAN is employed that uses periodic, sporadic and mixed transmission modes for messages.

It should be noted that the main contribution in this paper, compared to the contributions in [1, 2, 3], is that the extended analysis is also applicable to the heterogeneous systems. Moreover, we conduct a case study to show the applicability of the extended analyses. We also carry out a detailed comparative evaluation of the extended analyses.

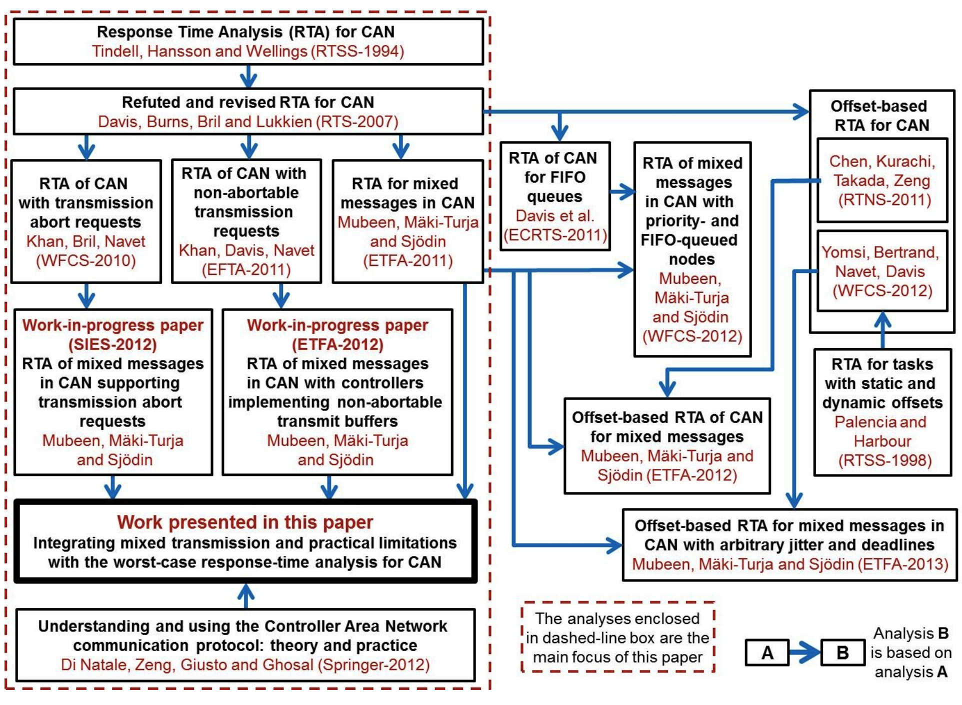

Figure 1: Relation between the existing and extended response-time analyses for CAN

2.3. Paper layout

The rest of the paper is organized as follows. In Section 3, we discuss the mixed messages. Section 4 describes the system model. In Section 5, we present the extended RTA for mixed messages without buffer limitations. Sections 6 and 7 discuss the extended RTA for mixed messages in the case of abortable and non-abortable transmit buffers respectively. Section 8 presents a case study and evaluation. Section 9 concludes the paper.

3. Mixed messages implemented by the higher-level protocols

Traditionally, it is assumed that the tasks queueing CAN messages are invoked either periodically or sporadically. However, there are some higher-level protocols for CAN in which the task that queues the messages can be invoked periodically as well as sporadically. If a message can be queued for transmission periodically as well as sporadically then the transmission type of the message is said to be mixed. In other words, a mixed message is simultaneously time (periodic) and event triggered (sporadic). We identified three types of implementations of mixed messages used in the industry.

Consistent terminology. We use the terms message and frame interchangeably because we only consider messages that fit into one frame (maximum 8 bytes). We term a CAN message as periodic, sporadic or mixed if it is queued by an application task that is invoked periodically, sporadically or both (periodically and sporadically) respectively. If a message is queued for transmission at periodic intervals, we use the term "Period" to refer to its periodicity. A sporadic message is queued for transmission as soon as an event occurs that changes the value of one or more signals contained in the message provided the Minimum Update Time (MUT) between queueing of two successive sporadic messages has elapsed. We overload the term "MUT" to refer to the "Inhibit Time" in the CANopen protocol [31] and the "Minimum Delay Time (MDT)" in AUTOSAR communication [32].

3.1. Method 1: implementation in the CANopen protocol

A mixed message in the CANopen protocol [31] can be queued for transmission at the arrival of an event provided the Inhibit Time has expired. The Inhibit Time is the minimum time that must be allowed to elapse between the queueing of two consecutive messages. The mixed message can also be queued periodically at the expiry of the Event Timer. The Event Timer is reset every time the message is queued. Once a mixed message is queued, any additional queueing of it will not take place during the Inhibit Time [31]. The transmission pattern of mixed message in the CANopen is illustrated in Figure 2. The first instance of the mixed message is queued as soon as the event A arrives. Both the Event Timer and Inhibit Time are reset. As soon as the Event Timer expires, instance 2 is queued due to periodicity and both the Event Timer and Inhibit Time are reset again. When the event B arrives, instance 3 is immediately queued because the Inhibit Time has already expired. Note that the Event Timer is also reset at the same time when instance 3 is queued as shown in Figure 2. Instance 4 is queued because of the expiry of the Event Timer. There exists a dependency relationship between the Inhibit Time and the Event Timer, i.e., the Event Timer is reset not only with every periodic transmission but also with every sporadic transmission.

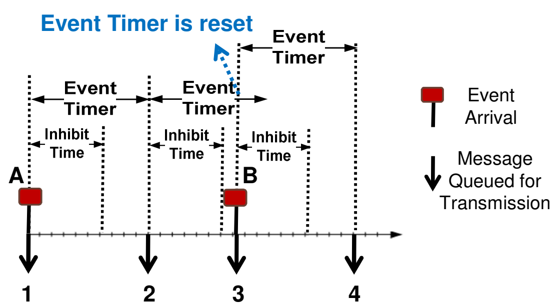

Figure 2: Mixed transmission pattern in the CANopen protocol

3.2. Method 2: implementation in the AUTOSAR communications

In AUTOSAR [32], a mixed message can be queued for transmission repeatedly with a period equal to the mixed transmission mode time period. The mixed message can also be queued at the arrival of an event provided the MDT timer has been expired. However, each transmission of the mixed message, regardless of being periodic or sporadic, is limited by the MDT timer. This means that both periodic and sporadic transmissions are delayed until the MDT timer expires. The transmission pattern of a mixed message implemented by AUTOSAR is illustrated in Figure 3. The first instance of the mixed message is queued (the MDT timer is started) because of partly periodic nature of the mixed message. When the event A arrives, instance 2 is queued immediately because the MDT timer has already expired. The next periodic transmission is scheduled 2 time units after the transmission of instance 2. However, the next two periodic transmissions corresponding to instances 3 and 4 are delayed because the MDT timer is not expired. The periodic transmissions corresponding to instances 5 and 6 occur at the scheduled times because the MDT timer is already expired in both cases.

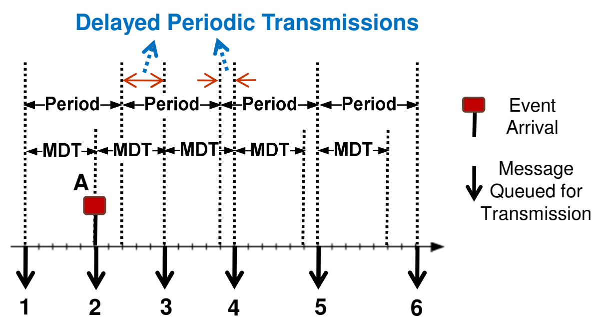

Figure 3: Mixed transmission pattern in the AUTOSAR Communications

3.3. Method 3: implementation in the HCAN protocol

A mixed message in the HCAN protocol [33] contains signals out of which some are periodic and some are sporadic. A mixed message is queued for transmission not only periodically but also as soon as an event occurs that changes the value of one or more event signals, provided the MUT timer between the queueing of two successive sporadic instances of the mixed message has elapsed. Hence, the transmission of the mixed message due to arrival of events is constrained by the MUT timer. The transmission pattern of the mixed message is illustrated in Figure 4. The first instance of the mixed message is queued because of periodicity. As soon as event A arrives, instance 2 is queued. When event B arrives it is not queued immediately because the MUT timer is not expired yet. As soon as the MUT timer expires, instance 3 is queued which contains the signal changes that correspond to event B. Similarly, an instance is not immediately queued when the event C arrives because the MUT timer is not expired. Instance 4 is queued because of the periodicity. Although, the MUT timer is not expired, the event signal corresponding to event C is packed in instance 4 and queued as part of the periodic instance. Hence, there is no need to queue an additional sporadic instance when the MUT timer expires. This indicates that the periodic transmission of the mixed message cannot be interfered by its sporadic transmission. When the event D arrives, a sporadic instance of the mixed message denoted by 5 is immediately queued because the MUT timer has already expired. Instance 6 is queued due to periodicity.

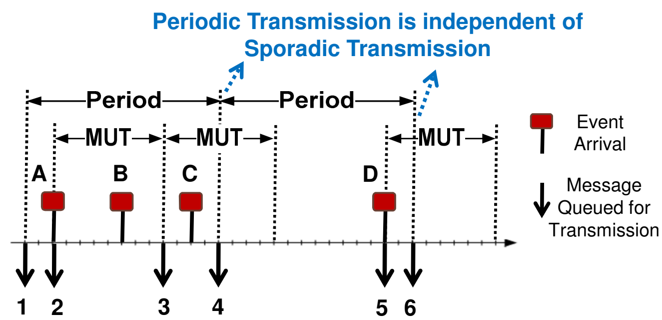

Figure 4: Mixed transmission pattern in the HCAN protocol

3.4. Discussion

In the first method [31], the Event Timer is reset every time the mixed message is queued for transmission. The implementation of mixed message in method 2 [32] is similar to method 1 to some extent. The main difference is that in method 2, the periodic transmission can be delayed until the expiry of the MDT timer. Whereas in method 1, the periodic transmission is not delayed, in fact, the Event Timer is restarted with every sporadic transmission. The MDT timer is started with every periodic or sporadic transmission of the mixed message. Hence, the worst-case periodicity of the mixed message in methods 1 and 2 can never be higher than the Inhibit Timer and MDT timer respectively. This means that the models of mixed messages in the first and second implementation methods reduce to the classical sporadic model. Therefore, the existing analyses for CAN [12, 14, 17, 23] can be used for analyzing mixed messages in the first and second implementation methods.

However, the periodic transmission is independent of the sporadic transmission in the third method [33]. The periodic timer is not reset with every sporadic transmission. The mixed message can be queued for transmission even if the MUT timer is not expired. Hence, the worst-case periodicity of the mixed message is neither bounded by period nor by the MUT timer. Therefore, the analyses in [12, 14, 17, 23] cannot be used for analyzing mixed messages in the third implementation method. This implies the need for the extension of existing analyses to support the mixed messages.

4. System model

The system model is an extension of the model in [12, 17, 23] in such a way that it supports mixed messages along with periodic and sporadic messages. The system consists of a number of CAN controllers (nodes[1]) denoted by $CC_1, CC_2, ...CC_n$. The nodes are connected to a single CAN network. The nodes implement priority-ordered queues, i.e., the highest priority message in each node enters into the bus arbitration. The set of messages in the system is defined as $\aleph$. Let $\aleph_c$ defines the set of messages sent by CC_c. We assume that each CAN controller has a finite number of transmit buffers (at least three). Let K_c denote the transmit buffers in the CAN controller CC_c. The number of transmit buffers in CC_c is returned by the function $Sizeof(K_c)$.

Each CAN message m_m has a unique identifier and a priority denoted by ID_m and P_m respectively. The priority of a message is assumed to be equal to its ID. The priority of m_m is considered higher than the priority of m_n if $P_m < P_n$. Let the sets $hp(m_m)$, $lp(m_m)$, and $hep(m_m)$ contain the messages with priorities higher, lower, and equal and higher than m_m respectively. Although the priorities of CAN messages are unique, the set $hep(m_m)$ is used in the case of mixed messages. $FRAME\_TYPE$ specifies whether the frame is a standard or an extended CAN frame. The difference between the two is that the standard CAN frame uses an 11-bit identifier whereas the extended CAN frame uses a 29-bit identifier. In order to keep the notations simple and consistent, we define a function ξ_m that denotes the transmission type of a message. ξ_m specifies whether m_m is periodic (P), sporadic (S) or mixed (M). Formally, the domain of ξ_m can be defined as follows.

$$\xi_m \in [P, \quad S, \quad M]$$

Each message m_m has a transmission time C_m and queueing jitter J_m which is inherited from the sending task. We assume that J_m can be smaller, equal or greater than T_m or MUT_m. Each message can carry a data payload denoted by s_m that ranges from 0 to 8 bytes. In the case of periodic transmission, m_m has a transmission period which is denoted by T_m. Whereas in the case of sporadic transmission, m_m has the MUT_m time. B_m denotes the blocking time of m_m which refers to the largest amount of time m_m has to wait for the transmission of a lower priority message.

[1]We overload the terms node and CAN controller throughout the paper

Each mixed message m_m is duplicated, i.e., it is treated as two separate messages: one periodic and the other sporadic. The duplicates share all the attributes except the T_m and MUT_m. The periodic copy inherits T_m while the sporadic copy inherits the MUT_m. Each message has a worst-case response time, denoted by R_m, and defined as the longest time between the queueing of the message (on the sending node) and the delivery of the message to the destination buffer (on the destination node). m_m is deemed schedulable if its R_m is less than or equal to its deadline D_m. The system is considered schedulable if all of its messages are schedulable.

We consider the deadlines to be arbitrary which means that they can be greater than the periods or MUTs of corresponding messages. We assume that the CAN controllers are capable of buffering more than one instance of a message. The instances of a message are assumed to be transmitted in the same order in which they are queued (i.e., FIFO policy).

5. Extended worst-case RTA for CAN without buffer limitations

In this section, we extend the existing RTA for CAN [12, 14] to support all types of messages namely periodic, sporadic, and mixed. However, we do not consider the buffer limitations in this section. That is, the CAN controllers are assumed to implement very large but finite number of transmit buffers such that there is no need to abort transmission requests[2]. Let m_m be the message under analysis. First, we discuss few terms that are used in the extended analysis. In order to calculate the worst-case response time of m_m, the maximum busy period [12, 14] for priority level-m should be known.

Maximum busy period. It is the longest contiguous interval of time during which m_m is unable to complete its transmission due to two reasons. First, the bus is occupied by the higher priority messages, i.e., at least one message of priority level-m or higher has not completed its transmission. Second, a lower priority message already started transmission when m_m is queued for transmission. This period starts at the so-called critical instant.

Critical instant. For a system where messages are scheduled without offsets, the critical instant corresponds to the point in time when all higher priority messages in the system are assumed to be queued simultaneously

[2]We use the term "no buffer limitations" consistently throughout the paper

with m_m while their subsequent instances are assumed to be queued after the shortest possible interval of time [14].

We analyze m_m differently based on its transmission type. Intuitively, we consider two different cases: (1) periodic and sporadic, and (2) mixed.

5.1. Case 1: When the message under analysis (m_m) is periodic or sporadic

Consider m_m to be a periodic or sporadic message. Since we consider arbitrary deadlines for messages, there can be more than one instance of m_m that may become ready for transmission before the end of priority level-m busy period. There can be another reason to check if more than one instance of m_m is queued for transmission in the priority level-m busy period. Since, the message transmission in CAN is non-preemptive, the transmission of previous instance of m_m could delay the current instance of a higher priority message that may add to the interference received by the current instance of m_m. This phenomenon is identified by Davis et al. [14] and termed as "push-through interference". Due to this interference, a higher priority message may be waiting for its transmission before the transmission of the current instance of m_m finishes. Hence, the length of busy period may extend beyond T_m or MUT_m. Therefore, we need to calculate the response time of each instance of m_m within priority level-m busy period. The maximum value among the response times of all instances of m_m is considered as the worst-case response-time of m_m. Let q_m be the index variable to denote instances of m_m. The worst-case response time of m_m is given by:

$$R_m = max\{R_m(q_m)\} \tag{1}$$

According to the existing analysis [12, 14], the worst-case response-time of any instance of m_m consists of three parts as follows.

1. The queueing jitter denoted by J_m. It is inherited from the sending task. Basically, it represents the maximum variation in time between the release of the sending task and queuing of the message in the transmit queue (buffers). It is calculated by taking the difference between the worst- and best-case response times of the sending task.
2. The queueing delay denoted by ω_m. It is equal to the longest time that elapses between the instant m_m is queued in the transmit queue and the instant when m_m is about to start its successful transmission. In other words, ω_m is the interference caused by other messages to m_m.

3. The worst-case transmission time denoted by C_m. It represents the longest time it takes for m_m to be transmitted over the network.

Thus, the worst-case response time of any instance q_m of a periodic or sporadic message m_m is given by the following set of equations.

$$R_m(q_m) = \begin{cases} J_m + \omega_m(q_m) - q_m T_m + C_m, & \text{if } \xi_k = \text{P} \\ J_m + \omega_m(q_m) - q_m MUT_m + C_m, & \text{if } \xi_k = \text{S} \end{cases} \tag{2}$$

The terms $q_m T_m$ and $q_m MUT_m$ in (2) are used to support the response-time calculations for multiple instances of m_m.

5.1.1. Calculations for the worst-case transmission time C_m

The worst-case transmission time of m_m is calculated according to the method derived in [12] and later adapted by [14]. For the standard CAN identifier frame format, C_m is calculated as follows.

$$C_m = \left(47 + 8s_m + \left\lfloor \frac{34 + 8s_m - 1}{4} \right\rfloor\right)\tau_{bit} \tag{3}$$

Where τ_{bit} denotes the time required to transmit a single bit of data on the CAN network. Its value depends upon the speed of the network. In (3), *47* is the number of bits due to protocol overhead. It is composed of start of frame bit (1-bit), arbitration field (12-bits), control field (6-bits), Cyclic Redundancy Check (CRC) field (16-bits), acknowledgement (ACK) field (2-bits), End of Frame (EoF) field (7-bits), and inter-frame space (3-bits). The number of bits due to protocol overhead in the case of extended CAN frame format is equal to *67*.

In [34], Broster identified that the analysis in [12, 14] uses *47*-bits instead of *44*-bits as the protocol overhead for a standard CAN identifier frame format. This is because the analysis in [12, 14] accounts 3-bit inter-frame space as part of the CAN frame. The 3-bit inter-frame space must be considered when calculating the interferences or blocking from other messages. However, Broster argued that this adds slight amount of pessimism to the response time of the message under analysis if the 3-bit inter-frame space is also considered in its transmission time. This is because the destination node can access the message before the inter-frame space. In order to avoid this pessimism, we subtract 3-bit time from the response time of the instance of the message under analysis.

The term $\left\lfloor \frac{\mathit{34}+\mathit{8}s_m-\mathit{1}}{\mathit{4}} \right\rfloor$ in (3) is added to compensate for the extra time due to *bit stuffing*. It should be noted that the bit sequences *000000* and *111111* are used for error signals in CAN. In order to be unambiguous in non-erroneous transmission, a stuff bit of opposite polarity is added whenever there are five bits of the same polarity in the sequence of bits to be transmitted [14]. The value *34* indicates that only 34-bits out of 47-bits protocol overhead are subjected to bit stuffing. The term $\lfloor \frac{a}{b} \rfloor$ is the notation for *floor* function. It returns the largest integer that is less than or equal to $\frac{a}{b}$.

Similarly, C_m is calculated for the extended frame format as follows.

$$C_m = \left(67 + 8s_m + \left\lfloor \frac{54 + 8s_m - 1}{4} \right\rfloor \right)\tau_{bit} \tag{4}$$

The calculations for C_m in (3) can be simplified as follows.

$$C_m = (55 + 10s_m)\tau_{bit} \tag{5}$$

Similarly, the calculations for C_m in (4) can be simplified as follows.

$$C_m = (80 + 10s_m)\tau_{bit} \tag{6}$$

5.1.2. Calculations for the queueing delay ω_m

In (2), the queueing delay for any instance of m_m consists of two elements.

1) Blocking delay. If any lower priority message just starts its transmission when m_m is queued for transmission then m_m has to wait in the transmit queue and is said to be blocked by the lower priority message. The lower priority message cannot be preempted during its transmission because CAN uses fixed-priority non-preemptive scheduling. Since we consider arbitrary deadline, m_m can also be blocked from its own previous instance due to push-through blocking [14] as discussed in Subsection 5.1. It should be noted that a CAN message can be blocked either by only one message in the set of lower priority messages or by only one of its previous instances. Moreover, the message under analysis can only be blocked by either the periodic copy or the sporadic copy of any lower priority mixed message (both copies of a mixed message have the same transmission time, C_m). Therefore, the maximum blocking delay is equal to the largest transmission time in the set of lower priority messages including the message itself. The maximum blocking delay for m_m denoted by B_m is calculated as follows.

$$B_m = \max_{\forall m_j \in lep(m_m)} \{C_j\} \tag{7}$$

Since we consider arbitrary deadlines, m_m can also be blocked from its own previous instance due to push-through blocking [14] as discussed in Subsection 5.1. That is the reason why (7) includes the function $lep(m_m)$ instead of $lp(m_m)$. It is important to point out that the blocking delay for the lowest priority message in the system is equal to zero if (7) is used. However, Broster [34] identified that lowest priority message can be blocked for 3-bits of time due to inter-frame space before it. Therefore, we consider $3\tau_{bit}$ time as the blocking delay for only the lowest priority message.

2) Delay due to interference from higher priority messages. Since CAN uses fixed-priority non-preemptive scheduling, a message cannot be interfered by higher priority messages during its transmission. Whenever we use the term interference, it refers to the amount of time m_m has to wait in the transmit queue because the higher priority messages in the system win the arbitration, i.e., the right to transmit before m_m. We adapt the calculations for the interference from higher priority messages from the existing analysis [12, 14]. However, the existing analysis considers the interference from only higher-priority periodic or sporadic messages. As we discussed in the system model that a mixed message is duplicated as two messages (one periodic and the other sporadic), each higher-priority mixed message should contribute interference from both the duplicates.

Thus, the queueing delay sums up the interferences due to higher priority messages, previous instances of the same message and the blocking delay. The queueing delay ω_m for the instance q_m of m_m can be calculated by solving the following equation.

$$\omega_m^{n+1}(q) = B_m + q_m C_m + \sum_{\forall m_k \in hp(m_m)} I_k C_k \tag{8}$$

(8) is an iterative equation. It is solved iteratively until two consecutive solutions become equal or the solution exceeds the message deadline in which case the message is deemed unschedulable. The starting value for ω_m^n can be selected equal to $B_m + q_m C_m$. In (8), I_k is calculated differently for different values of ξ_k (k is the index of any higher priority message) as shown below.

$$I_k = \begin{cases} \left\lceil \frac{\omega_m^n(q_m)+J_k+\tau_{bit}}{T_k} \right\rceil, & \text{if } \xi_k = \text{P} \\ \left\lceil \frac{\omega_m^n(q_m)+J_k+\tau_{bit}}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{S} \\ \left\lceil \frac{\omega_m^n(q_m)+J_k+\tau_{bit}}{T_k} \right\rceil + \left\lceil \frac{\omega_m^n(q_m)+J_k+\tau_{bit}}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{M} \end{cases} \tag{9}$$

The term $\lceil \frac{a}{b} \rceil$ is the notation for *ceil* function. It returns the smallest integer that is greater than or equal to $\frac{a}{b}$. The three terms $\left\lceil \frac{\omega_m^n(q_m)+J_k+\tau_{bit}}{T_k} \right\rceil$, $\left\lceil \frac{\omega_m^n(q_m)+J_k+\tau_{bit}}{MUT_k} \right\rceil$ and $\left\lceil \left\lceil \frac{\omega_m^n(q_m)+J_k+\tau_{bit}}{T_k} \right\rceil + \left\lceil \frac{\omega_m^n(q_m)+J_k+\tau_{bit}}{MUT_k} \right\rceil \right\rceil$ in (9) represent the maximum number of instances of higher-priority periodic, sporadic and mixed messages that are queued for transmission in the maximum busy period respectively. It is evident that the interference from a higher-priority mixed message contains the contribution from both of its duplicates.

5.1.3. Calculations for the length of priority level-m busy period

In order to calculate the worst-case response time of m_m, the number of instances of q_m that become ready for transmission before the end of the priority level-m busy period should be known. The length of the priority level-m busy period, denoted by t_m, can be calculated by adapting the existing analysis [14] as follows.

$$t_m^{n+1} = B_m + \sum_{\forall m_k \in hep(m_m)} I'_k C_k \tag{10}$$

where I'_k is given by the following relation. Note that the contribution from both the duplicates of the mixed message m_k is taken into account, provided it belongs to the set of equal or higher priority messages with respect to m_m.

$$I'_k = \begin{cases} \left\lceil \frac{t_m^n+J_k}{T_k} \right\rceil, & \text{if } \xi_k = \text{P} \\ \left\lceil \frac{t_m^n+J_k}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{S} \\ \left\lceil \frac{t_m^n+J_k}{T_k} \right\rceil + \left\lceil \frac{t_m^n+J_k}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{M} \end{cases} \tag{11}$$

In order to solve the iterative equation (10), C_m can be used as the initial value of t_m^n. The right hand side of (10) is a monotonic non-decreasing function of t_m. The iterative equation (10) is guaranteed to converge if the bus utilization for messages of priority level-m and higher, denoted by U_m, is less than 1. That is,

$$U_m < 1 \tag{12}$$

where U_m is calculated as follows.

$$U_m = \sum_{\forall m_k \in hep(m_m)} C_k I_k'' \tag{13}$$

where I_k'' is given by the following relation:

$$I_k'' = \begin{cases} \frac{1}{T_k}, & \text{if } \xi_k = \text{P} \\ \frac{1}{MUT_k}, & \text{if } \xi_k = \text{S} \\ \frac{1}{T_k} + \frac{1}{MUT_k}, & \text{if } \xi_k = \text{M} \end{cases} \tag{14}$$

In the above equation, the contribution from both copies of all mixed messages that are included in the set of equal and higher priority messages with respect to m_m is taken into account while calculating the bus utilization.

Calculations for the number of instances of m_m. The number of instances of m_m, denoted by Q_m, that become ready for transmission before the busy period ends is calculated as follows.

$$Q_m = \begin{cases} \left\lceil \frac{t_m + J_m}{T_m} \right\rceil, & \text{if } \xi_k = \text{P} \\ \left\lceil \frac{t_m + J_m}{MUT_m} \right\rceil, & \text{if } \xi_k = \text{S} \end{cases} \tag{15}$$

The range for the index variable q_m for the number of instances of m_m queued in the priority level-m busy period is given as follows.

$$0 \leq q_m \leq Q_m - 1 \tag{16}$$

The response times of all instances of m_m in the range shown in (16) should be calculated while the largest among them represents the worst-case response time of m_m.

5.2. Case 2: When the message under analysis is mixed

When the message under analysis is mixed, we treat the message as two separate message streams, i.e., the mixed message is duplicated as the periodic and sporadic messages. The response time of both the duplicates is calculated separately. Consider m_m to be a mixed message. For simplicity, we denote the periodic and sporadic copies of m_m by m_{m_P} and m_{m_S} respectively. Let the worst-case response times of m_{m_P} and m_{m_S} are denoted by R_{m_P} and R_{m_S} respectively. The worst-case response time of m_m is the largest value between R_{m_P} and R_{m_S} as given by the following equation.

$$R_m = max\{R_{m_P},\ \ R_{m_S}\} \tag{17}$$

Where R_{m_P} and R_{m_S} are equal to the maximum value among the response times of their respective instances. Let q_{m_P} and q_{m_S} be the index variables to denote the instances of m_{m_P} and m_{m_S} respectively. The calculations for R_{m_P} and R_{m_S} are adapted from the periodic and sporadic cases (discussed in the previous subsection) respectively as follows.

$$R_{m_P} = max\{R_{m_P}(q_{m_P})\}, \qquad \forall\ 0 \leq q_{m_P} \leq (Q_{m_P} - 1) \tag{18}$$

$$R_{m_S} = max\{R_{m_S}(q_{m_S})\}, \qquad \forall\ 0 \leq q_{m_S} \leq (Q_{m_S} - 1) \tag{19}$$

Where, Q_{m_P} and Q_{m_S} represent the number of instances of m_{m_P} and m_{m_S} that are queued in the priority level-m busy period respectively. We will come back to these two terms later in this subsection.

In (18) and (19), R_{m_P} and R_{m_S} for each respective instance are calculated separately by adapting the response-time calculations for the periodic and sporadic messages (from previous subsection) as follows.

$$R_{m_P}(q_{m_P}) = J_m + \omega_{m_P}(q_{m_P}) - q_{m_P}T_m + C_m \tag{20}$$

$$R_{m_S}(q_{m_S}) = J_m + \omega_{m_S}(q_{m_S}) - q_{m_S}MUT_m + C_m \tag{21}$$

The queueing jitter, J_m, is the same (equal) in both the equations (20) and (21). The worst-case transmission time, C_m, is also the same in these equations and is calculated using (5) or (6) depending upon the type of CAN frame identifier. Although, m_{m_P} and m_{m_S} inherit same J_m and C_m from m_m, they experience different amount of queueing delay caused by other messages.

5.2.1. Calculations for the queueing delay

The queueing delay experienced by m_{m_P} and m_{m_S} is denoted by ω_{m_P} and ω_{m_S} in (20) and (21) respectively. ω_{m_P} and ω_{m_S} can be calculated by adapting the calculations for the queueing delay in (8). However, in this equation we need to add the effect of self interference in a mixed message. By self interference, we mean that the periodic copy of a mixed message can be interfered by the sporadic copy and vice versa. Since, both m_{m_P} and m_{m_S} have equal priorities, any instance of m_{m_S} queued ahead of m_{m_P} will contribute an extra delay to the queueing delay experienced by m_{m_P}. A similar argument holds in the case of m_{m_S}. Let the self interference experienced by m_{m_P} due to one or more instances of m_{m_S} be denoted by $SI^P_{m_S}$. Similarly, $SI^S_{m_P}$ represents the self interference experienced by m_{m_S} due to one or more instances of m_{m_P}. Hence, ω_{m_P} and ω_{m_S} can be calculated as follows.

$$\omega^{n+1}_{m_P}(q_{m_P}) = B_m + q_{m_P} C_m + \sum_{\forall m_k \in hp(m_m)} I_{k_P} C_k + SI^P_{m_S} \tag{22}$$

$$\omega^{n+1}_{m_S}(q_{m_S}) = B_m + q_{m_S} C_m + \sum_{\forall m_k \in hp(m_m)} I_{k_S} C_k + SI^S_{m_P} \tag{23}$$

Calculations for the self interference in a mixed message. In order to derive the contribution of one copy of a mixed message to the queueing delay of the other, consider three different cases, depicting the transmission pattern of a mixed message m_m, shown in Figure 5. In the first case, we assume T_m to be greater than MUT_m. That is, there can be more transmissions of m_{m_S} compared to that of m_{m_P}. Since, the maximum update time between the queueing of any two instances of m_{m_S} can be arbitrarily very long, it is possible to have fewer sporadic transmissions than periodic transmissions of m_m. In the second case, we assume that T_m is equal to MUT_m. In this case, there are equal number of transmissions of m_{m_P} and m_{m_S}. In the third case, we assume T_m to be smaller than MUT_m. This implies that the number of sporadic transmissions will be less than the periodic transmissions of m_m.

It is important to note that in the example shown in Figure 5, there is a small offset between the first periodic and sporadic transmission of m_m. This offset is used to maximize the queueing delay. If this offset is removed then only one message will be queued corresponding to the first instance of both

m_{m_P} and m_{m_S}. Moreover, the larger value between T_m and MUT_m is the integer multiple of the smaller in all the cases. This relationship along with offset between T_m and MUT_m ensures that periodic and sporadic transmissions of m_m will not overlap, there by, maximizing the queueing delay.

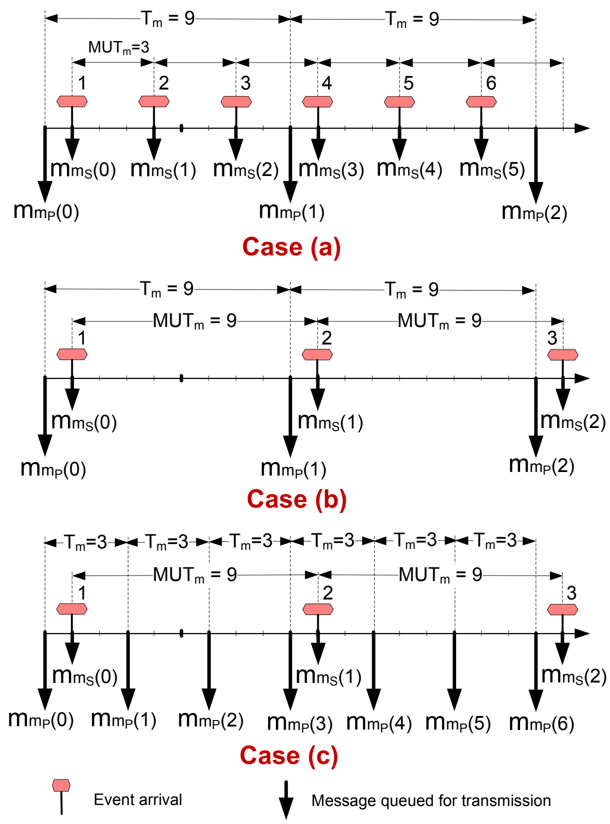

Figure 5: Self interference in a mixed message: (a) $\mathbf{T_m > MUT_m}$, (b) $\mathbf{T_m = MUT_m}$, (c) $\mathbf{T_m < MUT_m}$

.

Case (a): $\mathbf{T_m > MUT_m}$

Let the message under analysis be m_{m_P} and consider case (a) in Figure 5. An application task queues m_m periodically with a period T_m (equal to 9 time units). Moreover, the same task can also queue m_m sporadically at the arrival of events (labeled with numbers 1-6). The queueing of m_{m_S} is constrained by MUT_m (equal to 3 time units). The first instance of m_{m_P} ($q_{m_P} = 0$) is queued for transmission as shown by $m_{m_P}(0)$ in Figure 5. If event *1* had arrived at the same time as the queueing of $m_{m_P}(0)$ then the signals in $m_{m_S}(0)$ would have been updated as part of $m_{m_P}(0)$. In that case, $m_{m_S}(0)$ would not have been queued separately (this is the property of the mixed message in the HCAN protocol). In order to maximize the contribution of m_{m_S} on the queueing delay of m_{m_P}, $m_{m_S}(0)$ is queued just after the queueing of $m_{m_P}(0)$ as shown in all the cases in Figure 5. Therefore, $m_{m_S}(0)$ and subsequent instances of m_{m_S} will have no contribution in the worst-case queueing delay of the first instance of m_{m_P} denoted by $m_{m_P}(0)$.

Consider the second instance of m_{m_P}. All those instances of m_{m_S} that are queued ahead of $m_{m_P}(1)$ will contribute to its queueing delay. It can be observed in the case (a) that the first three instances of m_{m_S} are queued ahead of $m_{m_P}(1)$. Similarly, there are six instances of m_{m_S} that are queued ahead of $m_{m_P}(2)$. Let $Q^P_{m_S}$ denotes the total number of instances of m_{m_S} that are queued ahead of the $q^{th}_{m_P}$ instance of m_{m_P}. We can generalize $Q^P_{m_S}$ for the case (a) as follows.

$$Q^P_{m_S} = \left\lceil \frac{q_{m_P} T_m}{MUT_m} \right\rceil \tag{24}$$

For example, consider again the queueing of different instances of m_{m_S} and m_{m_P} in the case (a). Equation (24) yields the set $\{Q^P_{m_S} = 0, 3, 6, ...\}$ for the corresponding values in the set $\{q_{m_P} = 0, 1, 2, ...\}$. Thus the total number of instances of m_{m_S} queued ahead of each instance of m_{m_p} calculated by (24) are consistent with the case (a) in Figure 5.

Case (b): $\mathbf{T_m = MUT_m}$

Consider case (b) in which T_m is equal to MUT_m. It can be observed from Figure 5 that there are 0, 1, and 2 instances of m_{m_S} that are queued ahead of $m_{m_P}(0)$, $m_{m_P}(1)$ and $m_{m_P}(2)$ respectively. When (24) is applied in case (b), we get the set $\{Q^P_{m_S} = 0, 1, 2, ...\}$ for the corresponding values in the set $\{q_{m_P} = 0, 1, 2, ...\}$. Therefore, (24) is also applicable on case (b).

Case (c): $\mathbf{T_m < MUT_m}$

Now, consider case (c) in which T_m (3 time units) is smaller than MUT_m (9 time units). The first instance of m_{m_S} denoted by $m_{m_S}(0)$ will be queued ahead of $m_{m_P}(1)$, $m_{m_P}(2)$ and $m_{m_P}(3)$. Similarly, the two instances of m_{m_S} denoted by $m_{m_S}(0)$ and $m_{m_S}(1)$ will contribute to the queueing delay of $m_{m_P}(4)$, $m_{m_P}(5)$ and $m_{m_P}(6)$. (24) yields the set $\{Q^P_{m_S} = 0, 1, 1, 1, 2, 2, 2, ...\}$ for the corresponding values in the set $\{q_{m_P} = 0, 1, 2, 3, 4, 5, 6, ...\}$. Thus the total number of instances of m_{m_S} queued ahead of each instance of m_{m_P}

calculated by (24) are consistent with the case (c) in Figure 5.

Now we consider the effect of jitter on the instances of m_{m_S} prior to $m_{m_S}(0)$ which can be queued just ahead of $m_{m_P}(0)$ and contribute to the queueing delay of m_{m_P}. We assume FIFO queueing policy among the instances of the same message. By considering the jitter of m_{m_S} in $Q^P_{m_S}$, (24) can be generalized for the three cases as follows.

$$Q^P_{m_S} = \left\lceil \frac{q_{m_P} T_m + J_m}{MUT_m} \right\rceil \tag{25}$$

The self interference experienced by m_{m_P} due to one or more instances of m_{m_S} is the product of $Q^P_{m_S}$ and worst-case transmission time of m_{m_S}.

$$SI^P_{m_S} = Q^P_{m_S} C_m = \left\lceil \frac{q_{m_P} T_m + J_m}{MUT_m} \right\rceil C_m \tag{26}$$

The total number of instances of m_{m_P} that are queued ahead of the $q^{th}_{m_S}$ instance of m_{m_S}, denoted by $Q^S_{m_P}$, can be derived in a similar fashion. Thus, $Q^S_{m_P}$ can be calculated by the following equation.

$$Q^S_{m_P} = \left\lceil \frac{q_{m_S} MUT_m + J_m}{T_m} \right\rceil \tag{27}$$

The self interference experienced by m_{m_S} due to one or more instances of m_{m_P} is the product of $Q^S_{m_P}$ and worst-case transmission time of m_{m_P}.

$$SI^S_{m_P} = Q^S_{m_P} C_m = \left\lceil \frac{q_{m_S} MUT_m + J_m}{T_m} \right\rceil C_m \tag{28}$$

From (26) and (28) it is obvious that when q_{m_P} and q_{m_S} are zero (i.e., zeroth instances of m_{m_P} and m_{m_S}) as well as J_m is also zero then $SI^P_{m_S}$ and $SI^S_{m_P}$ are also zero respectively. However, even if J_m is zero, the zeroth instance of m_{m_P} can be interfered by one instance of m_{m_S}. Similar argument holds for the zeroth instance of m_{m_E}. For example, consider Case (a) in Figure 5. Let m_m be the highest priority message. Let $m_{m_S}(0)$ is queued just after the queueing of $m_{m_P}(0)$. The instance $m_{m_P}(0)$ can be blocked by any lower priority message. However, $m_{m_S}(0)$ cannot start its transmission unless $m_{m_P}(0)$ is transmitted. Therefore, we have to consider this specific case for the calculation of self interference in (26) and (28) as follows. This specific case is not considered for the calculations of self interference in [1]. It should be noted that this specific case may not occur if we consider holistic view of a distributed system using CAN network. This is because a message inherits its release jitter (most often non-zero) that is equal to the difference between worst- and best-case response times of the sending task.

$$SI_{m_S}^{P} = \begin{cases} \left\lceil \frac{q_{m_P} T_m + J_m + \tau_{bit}}{MUT_m} \right\rceil C_m, & \text{if } (q_{m_P} = 0) \;\&\&\; (J_m = 0) \\ \left\lceil \frac{q_{m_P} T_m + J_m}{MUT_m} \right\rceil C_m, & \text{otherwise} \end{cases} \tag{29}$$

$$SI_{m_P}^{S} = \begin{cases} \left\lceil \frac{q_{m_S} MUT_m + J_m + \tau_{bit}}{T_m} \right\rceil C_m, & \text{if } (q_{m_S} = 0) \;\&\&\; (J_m = 0) \\ \left\lceil \frac{q_{m_S} MUT_m + J_m}{T_m} \right\rceil C_m, & \text{otherwise} \end{cases} \tag{30}$$

(22) and (23) are solved iteratively until two consecutive solutions of each equation become equal or the solution exceeds the message deadline in which case the message is deemed unschedulable. The starting values for $\omega_{m_P}^{n}$ and $\omega_{m_S}^{n}$ can be selected equal to $B_m + q_{m_P} C_m$ and $B_m + q_{m_S} C_m$ respectively. The blocking time B_m is calculated using (7). The calculations for I_{k_P} and I_{k_S} are adapted from (9) separately for m_{m_P} and m_{m_S} as follows.

$$I_{k_P} = \begin{cases} \left\lceil \frac{\omega_{m_P}^{n}(q_{m_P}) + J_k + \tau_{bit}}{T_k} \right\rceil, & \text{if } \xi_k = \text{P} \\ \left\lceil \frac{\omega_{m_P}^{n}(q_{m_P}) + J_k + \tau_{bit}}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{S} \\ \left\lceil \frac{\omega_{m_P}^{n}(q_{m_P}) + J_k + \tau_{bit}}{T_k} \right\rceil + \left\lceil \frac{\omega_{m_P}^{n}(q_{m_P}) + J_k + \tau_{bit}}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{M} \end{cases} \tag{31}$$

$$I_{k_S} = \begin{cases} \left\lceil \frac{\omega_{m_S}^{n}(q_{m_S}) + J_k + \tau_{bit}}{T_k} \right\rceil, & \text{if } \xi_k = \text{P} \\ \left\lceil \frac{\omega_{m_S}^{n}(q_{m_S}) + J_k + \tau_{bit}}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{S} \\ \left\lceil \frac{\omega_{m_S}^{n}(q_{m_S}) + J_k + \tau_{bit}}{T_k} \right\rceil + \left\lceil \frac{\omega_{m_S}^{n}(q_{m_S}) + J_k + \tau_{bit}}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{M} \end{cases} \tag{32}$$

5.2.2. Calculations for the length of priority level-m busy period

The length of priority level-m busy period, denoted by t_m, can be calculated using (10) that was developed for the periodic and sporadic messages. This is because (10) takes into account the effect of queueing delay from all higher and equal priority messages. Since, the duplicates of a mixed message inherit the same priority from it, the contribution of queueing delay from the duplicate is also covered in (10). Therefore, there is no need to calculate t_m for m_{m_P} and m_{m_S} separately. In fact, t_m should be calculated only once for the mixed message that is under analysis.

Although the length of priority level-m busy period is the same for m_{m_P} and m_{m_S}, the number of instances of both these messages that become ready for transmission just before the end of the busy period, denoted by Q_{m_P} and Q_{m_S} respectively, may be different. The reason is that the calculations for Q_{m_P} and Q_{m_S} require T_m and MUT_m respectively and which may have different values. Q_{m_P} and Q_{m_S} can be calculated by adapting (15) that was derived for the calculations for the number of instances of periodic and sporadic messages. Q_{m_P} and Q_{m_S} are given by the following equations.

$$Q_{m_P} = \left\lceil \frac{t_m + J_m}{T_m} \right\rceil \tag{33}$$

$$Q_{m_S} = \left\lceil \frac{t_m + J_m}{MUT_m} \right\rceil \tag{34}$$

6. Integrating the effect of abortable transmit buffers with the extended worst-case RTA for CAN

In this section, we integrate the effect of abortable transmit buffers in the CAN controllers with the extended RTA of CAN for periodic, sporadic and mixed messages. We assume that the CAN controllers implement limited number of transmit buffers and support transmission abort requests. In order to avoid multiple priority inversions [21], we assume the controllers to implement at least 3 transmit buffers.

6.1. Priority inversion in the case of abortable transmit buffers

Additional delay and jitter due to priority inversion. In order to demonstrate the additional delay due to priority inversion when CAN controllers support transmission abort requests, consider the example of transmission of a message set as shown in Figure 6. Assume there are three nodes

CC_c, CC_j and CC_k in the system and each node has three transmit buffers. m_1 is the highest priority message in the node CC_c as well as in the system.

When m_1 becomes ready for transmission in the message queue, a lower priority message m_6 belonging to node CC_k is already under transmission. This represents the blocking delay for m_1. At this point in time, all transmit buffers in CC_c are occupied by the lower priority messages (say m_3, m_4 and m_5). The device drivers signal an abort request for the lowest priority message in K_c (transmit buffers in CC_c) that is not under transmission.

Hence, m_5 is aborted and copied from the transmit buffer to the message queue whereas m_1 is moved to the vacated transmit buffer. The time required to do this swapping is identified as *swapping time* in Figure 6. During the swapping time a series of events occur: m_6 finishes its transmission, new arbitration round starts, another message m_2 belonging to node CC_j and having priority lower than m_1 wins the arbitration and starts its transmission. Thus m_1 has to wait in the transmit buffer until m_2 finishes its transmission. This results in the priority inversion and adds an extra delay to the response time of m_1. In [17], Khan et al. pointed out that this extra delay of the higher priority message appears as its additional jitter to the lower priority messages, e.g., m_5 in Figure 6.

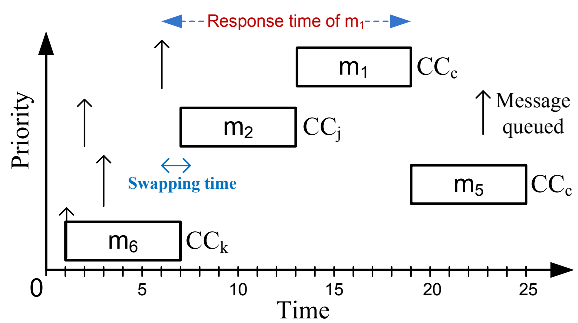

Figure 6: Demonstration of priority inversion in the case of abortable transmit buffers

Calculations for the additional jitter. The calculations for the additional jitter are adapted from the analysis in [17]. Let m_m be the message under analysis that belongs to the node CC_c. Let K_c denote the transmit buffer queue in CC_c. Let CT_m denotes the maximum between the time required to copy m_m from the message queue to the transmit buffer and from transmit buffer to the message queue. As noted in [17], these two times are very similar to each other in practice. Let the additional jitter of m_m as seen by the lower priority messages due to priority inversion (discussed above) be denoted by AJ_m^A. Where AJ stands for "Additional Jitter" while the superscript "A" stands for Abortable transmit buffer. The maximum jitter of m_m denoted by $\hat{J}_m$ is the summation of its original jitter J_m and the additional jitter due to priority inversion. Mathematically, the additional jitter of m_m that is seen by lower priority messages is calculated as follows.

$$\hat{J}_m = J_m + AJ_m^A \tag{35}$$

The additional jitter for m_m depends upon the following three elements.

1. The largest copy time of a message in the set of lower priority messages that belong to the same node CC_c.
2. The largest value among the worst-case transmission times of all those messages whose priorities are lower than the priority of m_m but higher than the highest priority message in K_c.
3. Since the original blocking time B_m for m_m is separately considered as part of the queueing delay, it should be subtracted from the additional delay.

In other words, the additional jitter of m_m (seen by lower priority messages) is equal to the sum of largest copy time of a message in the set of lower priority messages that belong to the same node CC_c; and the difference between transmission time of the message that won arbitration during the swapping process and the original blocking time B_m. Consider again the example of transmission of the message set in Figure 6. There are two cases with respect to message swapping time. In the first case, the swapping time window completes at or before the time 7 (completion of the transmission time of m_6 , i.e., the blocking message). Consequently, m_1 is already in the transmit buffer and ready to participate in bus arbitration at the start of new arbitration round. Hence, there is no additional delay of m_1. Intuitively, the additional jitter of m_1 as seen by lower priority messages is zero in the first case.

In the second case, which also depicts worst-case scenario, the swapping time window starts before the time 7 and completes after the time 7. In this case, the additional delay of m_1 is equal to the sum of (1) largest copy time of a message in the set of lower priority messages and (2) the difference between the blocking time due to priority inversion (transmission time of m_2) and the original blocking time B_m (transmission time of m_6). This extra delay of m_m

is in addition to its original blocking delay B_m. This additional delay for m_m appears as its additional jitter as seen by lower priority messages. It should be noted that the transmission time of blocking message (due to priority inversion, e.g., m_2) is always smaller than or equal to B_m. This is because B_m is the maximum transmission time among all lower priority messages than m_m (see equation 7); while the blocking message due to priority inversion can be one of the lower priority messages. Therefore, AJ_m^A is calculated as follows:

$$AJ_m^A = \max(0, \max_{\forall m_l \in CC_c \wedge m_l \in lep(m_m)}(CT_l) + \max_{P_m < P_l \leq P_{h_{K_c}}}(C_l) - B_m) \qquad (36)$$

From (36), it is clear that the additional jitter of m_m is due to variation in its release (start of transmission) because of CT_l. If B_m and C_l are equal then the additional jitter is equal to CT_l. On the other hand, if C_l is smaller than B_m then the additional jitter is less than CT_l even equal to zero in the best case. In (36), we consider only half of the swapping time, i.e., the time to move a lower priority message (to vacate space for m_m) from the transmit buffer to the message queue. This is because it is the only factor that may cause additional variation in time when m_m is queued in the transmit buffer depending upon whether the transmit buffer queue is full or not. The rest of the swapping time, i.e., the time to copy m_m in the transmit buffer is not considered as part of the additional jitter since m_m is copied to the transmission buffer anyway. It is considered as part of the worst-case queueing delay (e.g., see equation 43). In (36), $m_{h_{K_c}}$ is the highest priority message in K_c. We will come back to the calculations for finding the priority of $m_{h_{K_c}}$ in the next subsection.

Calculations for the blocking delay. When m_m is subjected to priority inversion, it experiences an extra amount of blocking in addition to the original blocking delay B_m. Let the total blocking delay for m_m due to priority inversion be denoted by $\hat{B}_m$. Mathematically, it is equal to the sum of the original blocking delay and the largest copy time of a message in the set of lower priority messages that belong to the same node CC_c.

$$\hat{B}_m = \max_{\forall m_j \in lep(m_m)}\{C_j\} + \max_{\forall m_l \in CC_c \wedge m_l \in lep(m_m)}(CT_l) \qquad (37)$$

Since we consider arbitrary deadlines, m_m can also be blocked from its own previous instance due to push-through blocking [14] as discussed in Subsection 5.1. That is the reason why (37) includes the function $lep(m_m)$ instead of $lp(m_m)$.

6.2. Extended RTA

The work in [17] noted that not all messages in a node suffer from priority inversion. Therefore we consider two different cases for calculating response times of periodic, sporadic and mixed messages in CAN with abortable transmit buffers. In this section, first we determine which messages are free from priority inversion. After that we extend the analysis from Section 5 by adapting the analysis in [17].

6.2.1. Calculations for the number of messages free from priority inversion

If we assume that multiple instances of a message cannot occupy transmit buffers then the number of lowest priority messages equal to the number of transmit buffers in a node will be safe from priority inversion. Whereas, the rest of the messages in the same node may suffer from priority inversion. This can be explained by a simple example. Let there be 4 transmit buffers in a node. Let there be 6 messages m_1, m_2, m_3, m_4, m_5 and m_6 in this node. m_1 has the highest priority, while m_6 has the lowest priority. Assume m_3 arrives in the message queue when 3 out of 4 transmit buffers are occupied by the three lowest priority messages m_6, m_5 and m_4. The fourth transmit buffer can either be empty or occupied by one of the higher priority messages m_1 or m_2. If the fourth transmit buffer is empty then m_3 is immediately copied to it. On the other hand, m_3 has to wait in the message queue because at least one transmit buffer contains a higher priority message. In both cases there is no need to abort any transmission. This implies that m_6, m_5, m_4 and m_3 will be safe from priority inversion, whereas m_1 and m_2 may undergo priority inversion. In this case, $m_{h_{K_c}}$ is represented by message m_3. This means that the set of lower priority messages whose size is equal to the number of transmit buffers will be free from priority inversion. However, this condition may become invalid if we assume that multiple instances of a message can occupy transmit buffers at the same time. Hence, we need to find out the worst-case scenario where messages are free from priority inversion.

Worst-case scenario for $m_{h_{K_c}}$. For convenience, assume that N_c represents the number of messages sent by the node CC_c. Intuitively, we can assume that the lowest priority message belonging to CC_c can be indexed as $m_m^{N_c-1}$. It should be noted that $N_c - 1$ does not represent the priority of the message. Similarly, the second lowest priority message belonging to CC_c can be indexed as $m_m^{N_c-2}$.

Let the total number of instances of all messages occupying the transmit buffers in CC_c be denoted by Ω_c. Assume that the maximum number of instances of $m_m^{N_c-1}$ occupying transmit buffers ahead of $m_m^{N_c-2}$ is denoted by $\Omega_c^{N_c-1_2}$. Its value depends upon three factors.

1. Periods of these two messages. If the period of $m_m^{N_c-2}$ is higher than the period of $m_m^{N_c-1}$, there can be more than one instance of $m_m^{N_c-1}$ that may occupy transmit buffers in CC_c ahead of $m_m^{N_c-2}$.
2. Due to jitter of $m_m^{N_c-1}$, more than one instance of $m_m^{N_c-1}$ may occupy transmit buffers in CC_c.
3. Transmission type of $m_m^{N_c-1}$. If $m_m^{N_c-1}$ is a mixed message, we need to consider the contribution of its periodic as well as sporadic part.

The value of $\Omega_c^{N_c-1_2}$ can be calculated with a similar intuition that we used in (25) as follows.

$$\Omega_c^{N_c-1_2} = \begin{cases} \left\lceil \frac{T_m{}^{N_c-2} + J_m{}^{N_c-1}}{T_m{}^{N_c-1}} \right\rceil, & \text{if } \xi_m{}^{N_c-1} = \text{P} \\ \left\lceil \frac{T_m{}^{N_c-2} + J_m{}^{N_c-1}}{MUT_m{}^{N_c-1}} \right\rceil, & \text{if } \xi_m{}^{N_c-1} = \text{S} \\ \left\lceil \frac{T_m{}^{N_c-2} + J_m{}^{N_c-1}}{T_m{}^{N_c-1}} \right\rceil + \left\lceil \frac{T_m{}^{N_c-2} + J_m{}^{N_c-1}}{MUT_m{}^{N_c-1}} \right\rceil, & \text{if } \xi_m{}^{N_c-1} = \text{M} \end{cases} \tag{38}$$

In this case, Ω_c is equal to $\Omega_c^{N_c-1_2}$ because we consider only two lowest priority messages. It should be noted that we consider period or minimum update time of $m_m^{N_c-2}$ if it is periodic or sporadic. However, if $m_m^{N_c-2}$ is mixed then we select the maximum between its period and minimum update time in (38).

Let us consider three lowest priority messages in CC_c denoted by $m_m^{N_c-1}$, $m_m^{N_c-2}$ and $m_m^{N_c-3}$. We denote the maximum number of instances of $m_m^{N_c-1}$ occupying the transmit buffers ahead of $m_m^{N_c-3}$ by $\Omega_c^{N_c-1_3}$. Similarly, the maximum number of instances of $m_m^{N_c-2}$ occupying the transmit buffers ahead of $m_m^{N_c-3}$ be denoted by $\Omega_c^{N_c-2_3}$. The calculations for $\Omega_c^{N_c-1_3}$ and $\Omega_c^{N_c-2_3}$ are adapted from (38) as follows.

$$\Omega_c^{N_c-1_3} = \begin{cases} \left\lceil \frac{T_m{}^{N_c-3}+J_m{}^{N_c-1}}{T_m{}^{N_c-1}} \right\rceil, & \text{if } \xi_m{}^{N_c-1} = \text{P} \\ \left\lceil \frac{T_m{}^{N_c-3}+J_m{}^{N_c-1}}{MUT_m{}^{N_c-1}} \right\rceil, & \text{if } \xi_m{}^{N_c-1} = \text{S} \\ \left\lceil \frac{T_m{}^{N_c-3}+J_m{}^{N_c-1}}{T_m{}^{N_c-1}} \right\rceil + \left\lceil \frac{T_m{}^{N_c-3}+J_m{}^{N_c-1}}{MUT_m{}^{N_c-1}} \right\rceil, & \text{if } \xi_m{}^{N_c-1} = \text{M} \end{cases} \tag{39}$$

$$\Omega_c^{N_c-2_3} = \begin{cases} \left\lceil \frac{T_m{}^{N_c-3}+J_m{}^{N_c-2}}{T_m{}^{N_c-2}} \right\rceil, & \text{if } \xi_m{}^{N_c-2} = \text{P} \\ \left\lceil \frac{T_m{}^{N_c-3}+J_m{}^{N_c-2}}{MUT_m{}^{N_c-2}} \right\rceil, & \text{if } \xi_m{}^{N_c-2} = \text{S} \\ \left\lceil \frac{T_m{}^{N_c-3}+J_m{}^{N_c-2}}{T_m{}^{N_c-2}} \right\rceil + \left\lceil \frac{T_m{}^{N_c-3}+J_m{}^{N_c-2}}{MUT_m{}^{N_c-2}} \right\rceil, & \text{if } \xi_m{}^{N_c-2} = \text{M} \end{cases} \tag{40}$$

In this case, Ω_c is equal to the sum of $\Omega_c^{N_c-1_3}$ and $\Omega_c^{N_c-2_3}$ as follows.

$$\Omega_c = \Omega_c^{N_c-1_3} + \Omega_c^{N_c-2_3} \tag{41}$$

Similarly, the maximum number of instances for any arbitrary number Z of lower priority messages occupying transmit buffers in CC_c can be calculated using the following equation. We assume Z to be smaller than or equal to N_c.

$$\Omega_c = \Omega_c^{N_c-1_Z} + \Omega_c^{N_c-2_Z} + \Omega_c^{N_c-3_Z} + ... + \Omega_c^{N_c-(Z-1)_Z} \tag{42}$$

In this manner, we need to keep on calculating the number of instances of lower priority messages occupying transmit buffers in CC_c until the value of Ω_c exceeds $Sizeof(K_c)$. The starting value for Z is 2. Once we have reached this condition, the highest priority message in this set of low priority messages is designated as $m_{h_{K_c}}$.

6.2.2. Case1: When message under analysis is free from priority inversion

Let the message under analysis be m_m and it belongs to the node CC_c. Once again, m_m is treated differently in the extended RTA based on its transmission type. In this case, we consider that m_m is free from priority inversion, i.e., its priority is smaller than or equal to the priority of $m_{h_{K_c}}$.

Case 1(a): When (m_m) is periodic or sporadic

Most of the equations to calculate response time of m_m from Subsection 5.1 are applicable in this case. However, the only difference lies in the calculations for the queueing delay ω_m and the length of priority level-m busy period t_m. The calculations for ω_m should take into account two more elements.

1. The copying delay (from the message queue to the transmit buffer) denoted by CT_m for every instance of m_m in the priority level-m busy period.
2. Additional jitter of higher priority messages that is experienced by m_m.

Adding these elements to (8) and (9), ω_m can be calculated as follows.

$$\omega_m^{n+1}(q) = B_m + q_m C_m + (q_m + 1) CT_m + \sum_{\forall m_k \in hp(m_m)} I_k C_k \tag{43}$$

$$I_k = \begin{cases} \left\lceil \frac{\omega_m^n(q_m) + \hat{J}_k + \tau_{bit}}{T_k} \right\rceil, & \text{if } \xi_k = \text{P} \\ \left\lceil \frac{\omega_m^n(q_m) + \hat{J}_k + \tau_{bit}}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{S} \\ \left\lceil \frac{\omega_m^n(q_m) + \hat{J}_k + \tau_{bit}}{T_k} \right\rceil + \left\lceil \frac{\omega_m^n(q_m) + \hat{J}_k + \tau_{bit}}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{M} \end{cases} \tag{44}$$

The calculations for t_m should take into account only one more element, i.e., the additional jitter of higher priority messages that is experienced by m_m. Adding it to (10) and (11), t_m can be calculated as follows.

$$t_m^{n+1} = B_m + \sum_{\forall m_k \in hep(m_m)} I'_k C_k \tag{45}$$

$$I'_k = \begin{cases} \left\lceil \frac{t_m^n + \hat{J}_k}{T_k} \right\rceil, & \text{if } \xi_k = \text{P} \\ \left\lceil \frac{t_m^n + \hat{J}_k}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{S} \\ \left\lceil \frac{t_m^n + \hat{J}_k}{T_k} \right\rceil + \left\lceil \frac{t_m^n + \hat{J}_k}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{M} \end{cases} \tag{46}$$

In (44) and (46), $\hat{J}_k$ is calculated by replacing m with k in (35) and (36).

Case 1(b): When (m_m) is mixed

Similar to Case 1(a), most of the equations to calculate response time of m_m from Subsection 5.2 are applicable in this case. The only difference lies in the calculations for ω_m and t_m. The same arguments from Case 1(a) hold for the calculations of ω_m. In this case, t_m can be calculated using (45) and (46). However, the queueing delay should be calculated separately for periodic (m_{m_P}) and sporadic (m_{m_S}) copies of m_m by integrating CT_m and $\hat{J}_m$ in (22), (23), (31) and (32) as follows.

$$\omega_{m_P}^{n+1}(q_{m_P}) = B_m + q_{m_P}C_m + (q_{m_P}+1)CT_m + \sum_{\forall m_k \in hp(m_m)} I_{k_P}C_k + SI_{m_S}^{P} \tag{47}$$

$$\omega_{m_S}^{n+1}(q_{m_S}) = B_m + q_{m_S}C_m + (q_{m_S}+1)CT_m + \sum_{\forall m_k \in hp(m_m)} I_{k_S}C_k + SI_{m_P}^{S} \tag{48}$$

$$I_{k_P} = \begin{cases} \left\lceil \frac{\omega_{m_P}^{n}(q_{m_P})+\hat{J}_k+\tau_{bit}}{T_k} \right\rceil, & \text{if } \xi_k = \text{P} \\ \left\lceil \frac{\omega_{m_P}^{n}(q_{m_P})+\hat{J}_k+\tau_{bit}}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{S} \\ \left\lceil \frac{\omega_{m_P}^{n}(q_{m_P})+\hat{J}_k+\tau_{bit}}{T_k} \right\rceil + \left\lceil \frac{\omega_{m_P}^{n}(q_{m_P})+\hat{J}_k+\tau_{bit}}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{M} \end{cases} \tag{49}$$

$$I_{k_S} = \begin{cases} \left\lceil \frac{\omega_{m_S}^{n}(q_{m_S})+\hat{J}_k+\tau_{bit}}{T_k} \right\rceil, & \text{if } \xi_k = \text{P} \\ \left\lceil \frac{\omega_{m_S}^{n}(q_{m_S})+\hat{J}_k+\tau_{bit}}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{S} \\ \left\lceil \frac{\omega_{m_S}^{n}(q_{m_S})+\hat{J}_k+\tau_{bit}}{T_k} \right\rceil + \left\lceil \frac{\omega_{m_S}^{n}(q_{m_S})+\hat{J}_k+\tau_{bit}}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{M} \end{cases} \tag{50}$$

In (49) and (50), $\hat{J}_k$ is calculated by replacing $_m$ with $_k$ in (35) and (36).

6.2.3. Case2: When message under analysis is subjected to priority inversion

In this case, we consider that m_m can undergo priority inversion, i.e., its priority is greater than the priority of $m_{h_{K_c}}$.

Case 2(a): When (m_m) is periodic or sporadic

Most of the equations to calculate response time of m_m from Subsection 5.1 are applicable in this case. However, the only difference lies in the calculations for the queueing delay ω_m, blocking delay B_m, and the length of priority level-m busy period t_m. The calculations for ω_m should take into account three more elements.

1. The copying delay (from the message queue to the transmit buffer) denoted by CT_m for every instance of m_m in the priority level-m busy period.
2. Additional jitter of higher priority messages that is experienced by m_m.
3. Additional blocking delay as shown in (37).

Adding these elements to (8) and (9), ω_m can be calculated as follows.

$$\omega_m^{n+1}(q) = \hat{B}_m + q_m C_m + (q_m + 1)CT_m + \sum_{\forall m_k \in hp(m_m)} I_k C_k \tag{51}$$

It should be noted that B_m is replaced with $\hat{B}_m$ which is calculated using (37). I_k in (51) is calculated differently for different values of ξ_k (k is the index of any higher priority message) using (44).

The value of priority level-m busy period t_m is calculated similar to Case 1(a) in Subsection 5.1. However, the calculations for t_m should take into account two more elements.

1. Additional jitter of higher priority messages that is experienced by m_m.
2. Additional blocking delay as shown in (37).

Adding these elements to (10) and (11), t_m can be calculated as follows.

$$t_m^{n+1} = \hat{B}_m + \sum_{\forall m_k \in hep(m_m)} I'_k C_k \tag{52}$$

I'_k in (52) is calculated differently for different values of ξ_k (k is the index of any higher priority message) using (46).

Case 2(b): When (m_m) is mixed

Similar to Case 2(a), most of the equations to calculate response time of m_m from Subsection 5.2 are applicable in this case. The only difference lies in the calculations for ω_m, B_m and t_m. In this case, t_m can be calculated using (52) and (46). However, the queueing delay should be calculated separately for periodic (m_{m_P}) and sporadic (m_{m_S}) copies of m_m by integrating CT_m, $\hat{B}_m$, $\hat{J}_m$ in (22) and (23).

$$\omega_{m_P}^{n+1}(q_{m_P}) = \hat{B}_m + q_{m_P} C_m + (q_{m_P} + 1) CT_m + \sum_{\forall m_k \in hp(m_m)} I_{k_P} C_k + SI_{m_S}^{P} \tag{53}$$

$$\omega_{m_S}^{n+1}(q_{m_S}) = \hat{B}_m + q_{m_S} C_m + (q_{m_S} + 1) CT_m + \sum_{\forall m_k \in hp(m_m)} I_{k_S} C_k + SI_{m_P}^{S} \tag{54}$$

Where I_{k_P} and I_{k_S} are calculated using (49) and (50) respectively.

7. Integrating the effect of non-abortable transmit buffers with the extended worst-case RTA for CAN

We integrate the effect of non-abortable transmit buffers in the CAN controllers with the extended RTA of CAN for periodic, sporadic and mixed messages. Basically, we extend the analysis from Section 5 by adapting the analysis in [23]. We assume that the CAN controllers do not support transmission abort requests. In order to avoid multiple priority inversions [21], we assume the controllers to implement at least 3 transmit buffers.

7.1. Additional delay and jitter due to priority inversion

When CAN controllers do not support transmission abort requests, a higher priority message may suffer from priority inversion and this, in turn, adds extra delay to its response time [23]. Consider an example of three controllers CC_c, CC_j, CC_k connected to a single CAN network in Figure 7. Let m_1, belonging to CC_c, be the highest priority message in the system. Assume that when m_1 is ready to be queued, all transmit buffers in CC_c are occupied by lower priority messages which cannot be aborted because the controllers implement non-abortable transmit buffers. In addition, m_1 can be blocked by any lower priority message because the lower priority message already started its transmission. In this example m_1 is blocked by m_5 that belongs to node CC_k. Since all transmit buffers in CC_c are full, m_1 has to wait in the message queue until one of the messages in K_c is transmitted. Let m_4 be the highest priority message in K_c. m_4 can be interfered by higher priority messages (m_2 and m_3) belonging to other nodes. Hence, it can be seen that priority inversion takes place because m_1 cannot start its transmission before m_4 finishes its transmission while m_4 has to wait until messages m_2 and m_3 are transmitted. This adds an additional delay to the worst-case response time of m_1. Let this additional delay for m_1 be denoted by AD_1. In this example, AD_1 is the sum of the worst-case transmission times of m_2, m_3 and m_4. Generally, this additional delay is denoted by AD_m^N for any message m_m. As we discussed in Subsection 6.1, this additional delay appears as additional jitter of m_m as seen by the lower priority messages. Let the additional jitter be denoted by AJ_m^N. Where AJ stands for "Additional Jitter" while the superscript "N" stands for Non-abortable transmit buffer.

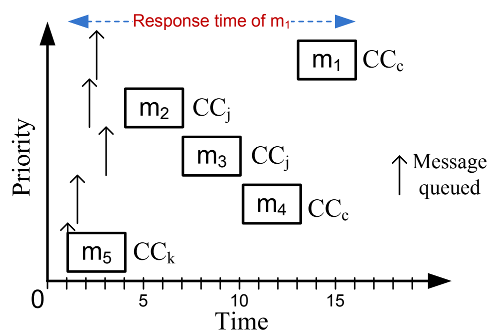

Figure 7: Demonstration of priority inversion in the case of non-abortable transmit buffers

7.2. Calculations for the additional delay, jitter and blocking

The calculations for the additional delay, additional jitter and extra blocking due to priority inversion (discussed in the above subsection) are adapted[3] from the existing analysis [23] to support mixed messages as well.

Calculations for the additional delay. Let $m_{h_{K_c}}$ be the highest priority message in the transmit buffers of CC_c denoted by K_c. The calculations to

[3] the existing analysis [23] does not support mixed messages

determine the priority of $m_{h_{K_c}}$ can be adapted from Section 7.3. Let m_m be the message under analysis whose priority is higher than $m_{h_{K_c}}$ and belongs to the same node CC_c. Assume all transmit buffers are occupied by lower priority messages when m_m becomes ready for transmission. So m_m has to wait until $m_{h_{K_c}}$ is transmitted. This waiting time for m_m depends upon the response time of $m_{h_{K_c}}$. Let us term the response time of $m_{h_{K_c}}$ without its jitter as the modified response time and denote it by $R^*_{h_{K_c}}$. Mathematically,

$$R^*_{h_{K_c}} = \omega^*_{h_{K_c}} + C_{h_{K_c}} \tag{55}$$

where, $C_{h_{K_c}}$ and $\omega^*_{h_{K_c}}$ denote the the worst-case transmission time and queueing delay of $m_{h_{K_c}}$ respectively. The reason for not considering jitter of $m_{h_{K_c}}$ as part of its modified response time is that $m_{h_{K_c}}$ is already in transmit buffer and hence its jitter will have no impact on the response time of m_m.

The message $m_{h_{K_c}}$ can be blocked by either one message in the set of lower priority messages belonging to other nodes or from its previous instance due to push-through blocking (discussed in Subsection 5.1). The queueing delay for $m_{h_{K_c}}$ is calculated as follows.

$$\omega^{*(n+1)}_{h_{K_c}} = B_{h_{K_c}} + \sum_{\forall m_k \in hp(m_{h_{K_c}})} I^*_k C_k \tag{56}$$

In (56), I^*_k is calculated differently for different values of ξ_k (k is the index of any higher priority message) as shown below.

$$I^*_k = \begin{cases} \left\lceil \frac{\omega^{*(n)}_{h_{K_c}} + \hat{J}_k + \tau_{bit}}{T_k} \right\rceil, & \text{if } \xi_k = \text{P} \\ \left\lceil \frac{\omega^{*(n)}_{h_{K_c}} + \hat{J}_k + \tau_{bit}}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{S} \\ \left\lceil \frac{\omega^{*(n)}_{h_{K_c}} + \hat{J}_k + \tau_{bit}}{T_k} \right\rceil + \left\lceil \frac{\omega^{*(n)}_{h_{K_c}} + \hat{J}_k + \tau_{bit}}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{M} \end{cases} \tag{57}$$

In (57), $\hat{J}_k$ is the additional jitter of higher priority message m_k as seen by $m_{h_{K_c}}$. We will come back to its calculations later.

Once $m_{h_{K_c}}$ is in K_c, it cannot be interfered by $hp_c(m_{h_{K_c}})$ (i.e., the set of messages that belong to CC_c and have priorities higher than the priority of $m_{h_{K_c}}$) because the buffers are non-abortable. Let this interference be denoted $IF^c_{h_{K_c}}$. However, the messages in $hp_c(m_{h_{K_c}})$ can indirectly interfere with $m_{h_{K_c}}$ before it occupies a buffer in K_c by interfering with the messages in the set $hp(m_{h_{K_c}})$ belonging to other nodes. Let the interference received by $m_{h_{K_c}}$ from the messages in the set $hp(m_m)$ belonging to all nodes other than CC_c be denoted by $IF^m_{h_{K_c}}$. The additional delay for m_m will be equal to the difference between the modified response time $R^*_{h_{K_c}}$ of $m_{h_{K_c}}$ and the two combined interferences $IF^c_{h_{K_c}}$ and $IF^m_{h_{K_c}}$. m_m can receive this additional delay from any message in node CC_c whose priority is smaller than m_m and greater or equal to $m_{h_{K_c}}$. Hence, we need to calculate all these delays and select the maximum among them as the additional delay for m_m as follows.

$$AD^N_m = \max_{\forall m_l \in CC_c \wedge (P_m < P_l \leq P_{h_{K_c}})} (R^*_{h_l} - IF^c_{h_{K_c}} - IF^m_{h_{K_c}}) \tag{58}$$

Where the interferences $IF^c_{h_{K_c}}$ and $IF^m_{h_{K_c}}$ are calculated as follows.

$$IF^c_{h_{K_c}} = \sum_{\forall m_i \in CC_c \wedge (1 \leq P_i < P_l)} I_{k_1} C_i \tag{59}$$

$$IF^m_{h_{K_c}} = \sum_{\forall m_j \notin CC_c \wedge (1 \leq P_j < P_m)} I_{k_2} C_j \tag{60}$$

In (59) and (60), the values for I_{k_1} and I_{k_2} are calculated differently for different values of ξ_i and ξ_j respectively as follows.

$$I_{k_1} = \begin{cases} \left\lceil \frac{R^*_{h_l} - C_l + \hat{J}_i + \tau_{bit}}{T_i} \right\rceil, & \text{if } \xi_i = \text{P} \\ \left\lceil \frac{R^*_{h_l} - C_l + \hat{J}_i + \tau_{bit}}{MUT_i} \right\rceil, & \text{if } \xi_i = \text{S} \\ \left\lceil \frac{R^*_{h_l} - C_l + \hat{J}_i + \tau_{bit}}{T_i} \right\rceil + \left\lceil \frac{R^*_{h_l} - C_l + \hat{J}_i + \tau_{bit}}{MUT_i} \right\rceil, & \text{if } \xi_i = \text{M} \end{cases} \tag{61}$$

$$I_{k_2} = \begin{cases} \left\lceil \frac{R^*_{h_l} - C_l + \hat{J}_j + \tau_{bit}}{T_j} \right\rceil, & \text{if } \xi_j = \text{P} \\ \left\lceil \frac{R^*_{h_l} - C_l + \hat{J}_j + \tau_{bit}}{MUT_j} \right\rceil, & \text{if } \xi_j = \text{S} \\ \left\lceil \frac{R^*_{h_l} - C_l + \hat{J}_j + \tau_{bit}}{T_j} \right\rceil + \left\lceil \frac{R^*_{h_l} - C_l + \hat{J}_j + \tau_{bit}}{MUT_j} \right\rceil, & \text{if } \xi_j = \text{M} \end{cases} \tag{62}$$

Calculations for the additional jitter. The total jitter of m_m denoted by $\hat{J}_m$ as seen by the lower priority messages is the sum of its original jitter J_m and the additional jitter due to priority inversion as follows.

$$\hat{J}_m = J_m + AJ_m^N \tag{63}$$

The additional jitter AJ_m^N is calculated similar to the additional delay AD_m^N. However, we need to subtract only interference $IF^c_{h_{K_c}}$ from $R^*_{h_l}$ because m_{h_l} cannot be interfered by higher priority messages from the same node after it has been transferred to the transmit buffer. Therefore, AJ_m^N is calculated as follows:

$$AJ_m^N = \max_{\forall m_l \in CC_c \wedge (P_m < P_l \leq P_{h_{K_c}})} (R^*_{h_l} - IF^c_{h_{K_c}}) \tag{64}$$

Where, $IF^c_{h_{K_c}}$ is calculated using (59) and (61).

Calculations for the blocking delay. When m_m is subjected to priority inversion due to non-abortable transmit buffers, it experiences an extra amount of blocking in addition to the original blocking delay B_m. The total blocking delay for m_m denoted by $\hat{B}_m$ is the maximum value between the original blocking delay B_m and additional delay AD_m^N. B_m is calculated using (7) while $\hat{B}_m$ is calculated as follows.

$$\hat{B}_m = \max(B_m, AD_m^N) \tag{65}$$

It is important to note that equations (55), (58), and (63) are implicitly dependent on each other. Therefore, they are solved simultaneously and iteratively until two consecutive solutions of each equation become equal or the solutions exceed the message deadline in which case the message is deemed unschedulable. For convenience, the calculations for the total jitter and additional delay are depicted in Algorithms 1 and 2. The inputs required by this algorithm are the sets of all messages and all CAN controllers along with the number of transmit buffers in each controller.

7.3. Extended RTA

As discussed in the example given in Section , some messages will be safe from priority inversion, whereas other messages in the same node may suffer from priority inversion. Therefore, we consider two different cases for calculating response times of messages in CAN with non-abortable transmit

Algorithm 1 Procedure for the calculations of $\hat{J}_m$ and AD_m^N. It is used in Algorithm 2

begin
for all CAN_controllers_in_the_system i: 1...N **do**
    $K_i \leftarrow$ NR_OF_TRANSMIT_BUFFERS_IN_CC$_i$ () ▷ The number of transmit buffers in each CAN controller should be available as input
    $S1_i \leftarrow$ CALCULATE_MAX_NR_OF_MESSAGES_IN_K$_i$ () ▷ Use eq. (42)
end for
procedure CALCULATE_$\hat{J}_m$_AND_AD_m^N ()
    for all messages_in_the_system m: 1...M **do**
        if $m_m \in S1_m$ **then**
            $AD_m^N\_new = 0$ ▷ m_m is safe from priority inversion
        else
            CALCULATE_AD_m^N_NEW () ▷ Use eq. (58)
        end if
        CALCULATE_$R^*_{h_{K_m}}$_NEW () ▷ Use eq. (55)
        CALCULATE_AJ_m^N_NEW () ▷ Use eq. (64)
        $\hat{J}_m \leftarrow J_m + AJ_m^N$
    end for
end procedure
end

Algorithm 2 Iterative algorithm for the calculations of $\hat{J}_m$ and AD_m^N.

begin
$Repeat \leftarrow TRUE$
$Schedulable \leftarrow FALSE$
for all messages_in_the_system m: 1...M **do**
    $AD_m^N\_old \leftarrow 0$ ▷ Initialize the additional delay
    $AJ_m^N\_old \leftarrow 0$ ▷ Initialize the additional jitter
    $R^*_{h_{K_m}}\_old \leftarrow C_{K_m}$ ▷ Initialize the modified response times
end for
while $Repeat = TRUE$ **do**
    CALCULATE_$\hat{\mathrm{J}}_m$_AND_AD_m^N () ▷ Use Algorithm 1
    for all messages_in_the_system m: 1...M **do**
        if ($R^*_{h_{K_m}}$_new $> D_m$) **then** ▷ The modified response time of m_m exceeds its deadline, hence the system is unschedulable
            $Repeat \leftarrow FALSE$
            $Schedulable \leftarrow FALSE$
        else
            if (AD_m^N_new = AD_m^N_old) && (AJ_m^N_new = AJ_m^N_old) && ($R^*_{h_{K_m}}$_new = $R^*_{h_{K_m}}$_old) **then**
                $Repeat \leftarrow FALSE$
                $Schedulable \leftarrow TRUE$
            else
                $Repeat \leftarrow TRUE$
            end if
        end if
    end for
end while
end

buffers: Case(1) when message under analysis is free from priority inversion, and Case (2) when message under analysis is subjected to priority inversion. In each of these cases, we treat the message under analysis differently based on its transmission type: Case (a) when message under analysis is periodic or sporadic, and Case (b) when message under analysis is mixed. This is exactly similar to the extended analysis for periodic, sporadic and mixed messages in CAN with abortable transmit buffers that is discussed in the previous section. All equations for the response-time calculations from (43) to (54) from the previous section are applicable with the following changes.

1. Since the controllers implement non-abortable transmit buffers, there will be no copying delays. Therefore, the copying delay denoted by CT_m should be neglected. The following changes should be made in the analysis from the previous section:
   (a) $(q_m + 1)CT_m$ should be removed from equations (43) and (51),
   (b) $(q_{m_P} + 1)CT_m$ should be removed from equations (47) and (53),
   (c) $(q_{m_S} + 1)CT_m$ should be removed from equations (48) and (54).
2. The total jitter of m_m denoted by $\hat{J}_m$ as seen by the lower priority messages should be calculated using (63) instead of (35).
3. Additional delay should be calculated using (58).
4. The total blocking delay for m_m denoted by $\hat{B}_m$ should be calculated using (65) instead of (37).

8. Comparative evaluation

We perform a number of tests on a message set consisting of 50 messages to evaluate and compare the three extended analyses. The message set is generated using the NETCARBENCH tool [35]. In all these tests, the system consists of 5 ECUs which are connected to a single CAN network that runs at 250 Kbit/s. The buffer limitations in the ECUs are different in each test. Each message in the generated message set has a unique priority. The highest priority is 1, whereas the lowest priority is 50. It should be noted that the NETCARBENCH tool cannot generate mixed messages. We randomly selected 20 mixed, 15 periodic and 15 sporadic messages from the generated message set. The messages are equally distributed among the ECUs, i.e., each ECU transmits 4 mixed, 3 periodic and 3 sporadic messages.

8.1. *Comparison of the extended analyses*

In the first test, we consider three different cases: (i) all ECUs are assumed to have no buffer limitations in the CAN controllers, (ii) each ECU implements three transmit buffers in the CAN controller and the buffers are abortable, and (iii) each ECU implements three transmit buffers in the CAN controller and the buffers are non-abortable. In the case (i), we analyze the message set with the extended analysis that does not take into account buffer limitations in the CAN controllers (the analysis from Section 5). In the case (ii), we analyze the same message set with the extended analysis that considers abortable transmit buffers (the analysis from Section 6). Finally in the case (iii), we analyze the same message set with the extended analysis that considers non-abortable transmit buffers (the analysis from Section 7). Figure 8 depicts the bar graph that shows the response times of messages that are calculated with three different analyses discussed above.

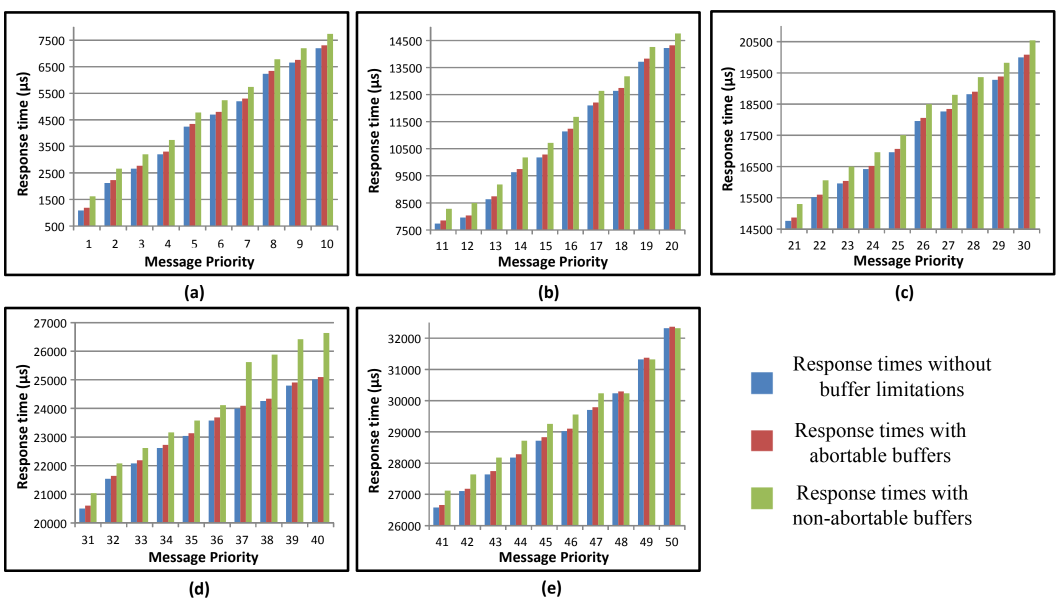

Figure 8: Comparison of message response times that are calculated with the extended analyses (i) without buffer limitations, (ii) with abortable transmit buffers, and (iii) with non-abortable transmit buffers.

The results indicate that message response times are always lower when there are no buffer limitations in the CAN controllers. Apart from those lowest priority messages that are equal to the number of transmit buffers in each CAN controller (three lowest priority messages in this case), the response times of messages are smaller if CAN controllers implement abortable transmit buffers compared to non-abortable transmit buffers. On the other hand, the response times of the three lowest priority messages in the system with non-abortable transmit buffers is smaller compared to the system with abortable transmit buffers because the three lowest priority messages are free from priority inversion. In fact, their response times in the system with non-abortable transmit buffers match their response times when there are no buffer limitations in the CAN controllers. The message set is selected in a way that there are no multiple instances of most of the lower priority messages. It can be concluded that it is more feasible to use CAN controllers with abortable transmit buffers compared to non-abortable transmit buffers. Moreover, it is important to use the RTA that matches the actual limitations and constraints in the hardware, device drivers and protocol stack. Otherwise, the calculated response times can be optimistic.

8.2. Application of the extended analyses to heterogeneous systems

In the second test, we consider the case of a heterogeneous system in addition to the three cases from the first test. By heterogeneous system, we mean that the ECUs have different buffer limitations. That is, two ECUs implement abortable transmit buffers, two implement non-abortable transmit buffers while there are no buffer limitations in one ECU. Those ECUs that have buffer limitations implement three transmit buffers. We use the same message set in the heterogeneous system. In this case the messages that belong to the ECUs without buffer limitations are analyzed with the analysis from Section 5. The messages that belong to the ECUs that implement abortable transmit buffers are analyzed with the analysis from Section 6. Similarly, the messages that belong to the ECUs that implement non-abortable transmit buffers are analyzed with the analysis from Section 7.

Figure 9 depicts the bar graph that shows the calculated response times of messages in four different cases. The results indicate that the message response times in the heterogeneous system are always greater than the message response times when the ECUs have no buffer limitations or the ECUs implement abortable transmit buffers. However, the response times of the 47 highest priority messages in the heterogeneous system are smaller than their response times when the ECUs implement non-abortable transmit buffers. Whereas, this trend is reversed for the three lowest priority messages because these messages are free from priority inversion. The message set is selected in a way that there are no multiple instances of most of the lower priority messages.

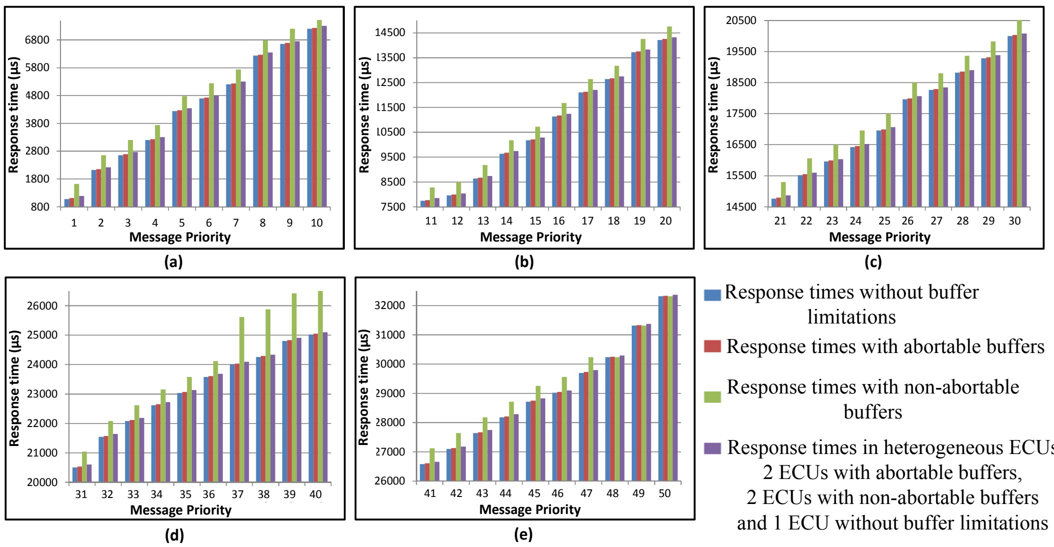

Figure 9: Comparison of message response times that are calculated with the analyses (i) without buffer limitations, (ii) with abortable buffers, (iii) with non-abortable buffers, and (iv) all three analysis in (i), (ii) and (iii) are applied on a heterogeneous system.

8.3. Effect of copy times of messages on their response times

In the third test, we explore the effect of message copy times on their response times in the systems where ECUs implement three transmit buffers which are of abortable type. We use the same message set that we used in the previous tests. In this test, we consider six different cases with respect to the amount of message copy times: (i) copy time of all messages is four times the transmission time of a single bit of data over CAN (1-bit more time than the time required for inter-frame space of 3-bits), (ii) copy time of each message is 5% of its transmission time, (iii) copy time of each message is 10% of its transmission time, (iv) copy time of each message is 15% of its transmission time, (v) copy time of each message is 20% of its transmission time, and (vi) copy time of each message is 25% of its transmission time.

We analyze the message set in all these cases with the extended analysis from Section 6. The calculated response times are depicted in the bar graph in Figure 10. The results indicate that the increase in the response times of messages is directly proportional to the increase in the amount of message copy times. If the message copy time is less than the inter-frame space (time required to transmit 3-bits of data on CAN), the response times of messages in the system with abortable transmit buffers converge to the response times of same messages in the system with no buffer limitations.

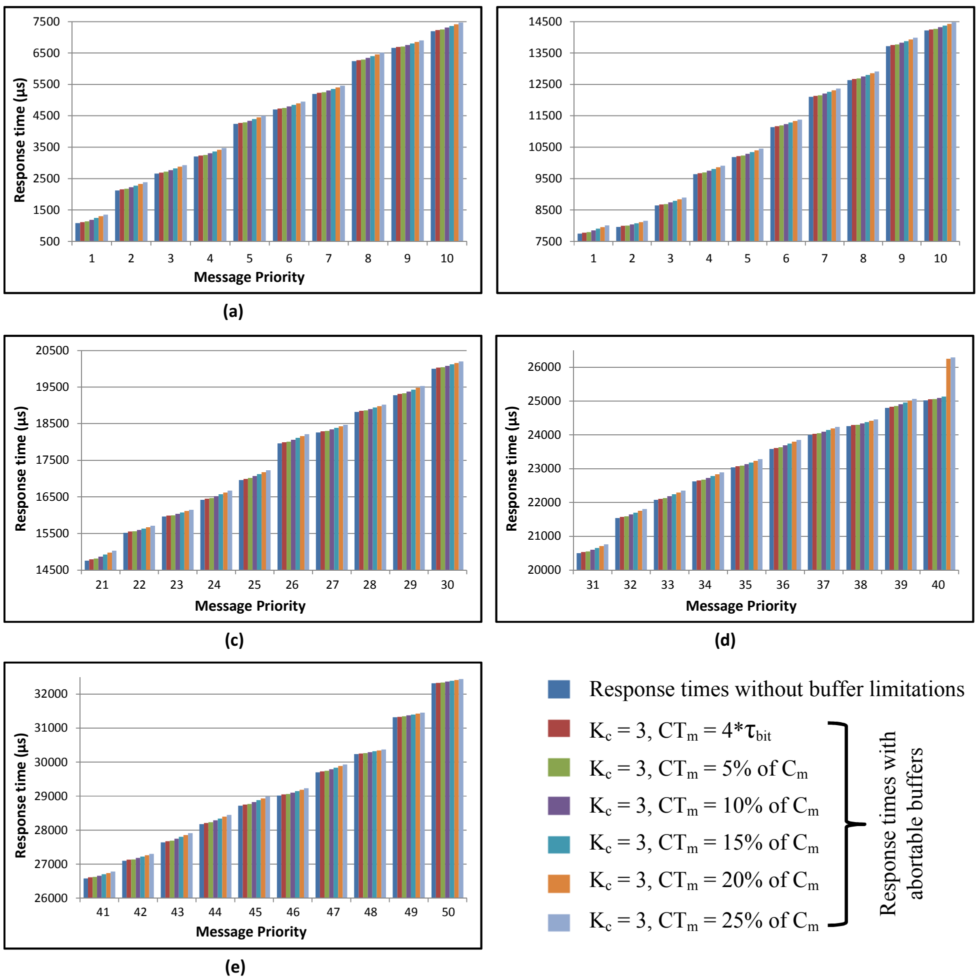

Figure 10: Comparison of message response times that are calculated with the extended analysis with abortable transmit buffers with different amount of message copy times.

8.4. *Effect of the number of transmit buffers on message response times*

In the fourth test, we explore the effect of the number of transmit buffers on message response times in the systems where ECUs implement non-abortable transmit buffers. Once again, the same message set is used. In this test, we consider nine different cases with respect to the number of transmit buffers in the CAN controllers, i.e., the number of transmit buffers in each ECU is equal to: (i) very large, (ii) ten, (iii) nine, (iv) eight, (v) seven, (vi) six, (vii) five, (viii) four, and (ix) three. We analyze the message set in all these cases separately with the extended analysis from Section 7. The calculated response times are depicted in the bar graph in Figure 11.

As expected, the response times of messages in the system with no buffer limitations are always smaller than or equal to their response times when the ECUs in the system implement non-abortable transmit buffers. Let's consider the three lowest priority messages (priorities 48, 49 and 50). The response times of these messages are equal in all the cases because there are at least 3 transmit buffers in every ECU in each case. Therefore, these messages are free from priority inversion. This also shows that the message set is selected in a way that there are no multiple instances of most of the lower priority messages. Now consider the message with priority equal to 47. This message has the highest response time when ECUs contain 3 transmit buffers as shown by the last bar in Figure 11(e). Since, it is fourth lowest priority message in the system, it is not save from priority inversion when there are three transmit buffers in each ECU. Similarly, for the message with priority equal to 46, the message has higher response times in the system where ECUs implement 3 and 4 transmit buffers as shown by the the second last and last bars in Figure 11(e) respectively. This trend of increasing response times with priorities 45, 44, 43, 42, 42, and 40 continues as the number of transmit buffers in the ECUs keeps on increasing from 5 to 10.

9. Conclusion

The existing worst-case Response Time Analysis (RTA) for Controller Area Network (CAN) does not support mixed messages. Mixed messages can be queued for transmission both periodically and sporadically. They are implemented by some of the higher-level protocols and commercial extensions of CAN that are used in the automotive industry. We extended the existing analysis to support mixed messages. The extended analysis is able to calculate upper bounds on the response times of CAN messages with all types of transmission patterns, i.e., periodic, sporadic and mixed. Furthermore, we integrated the effect of hardware and software limitations in the CAN controllers and device drivers such as abortable and non-abortable trans-

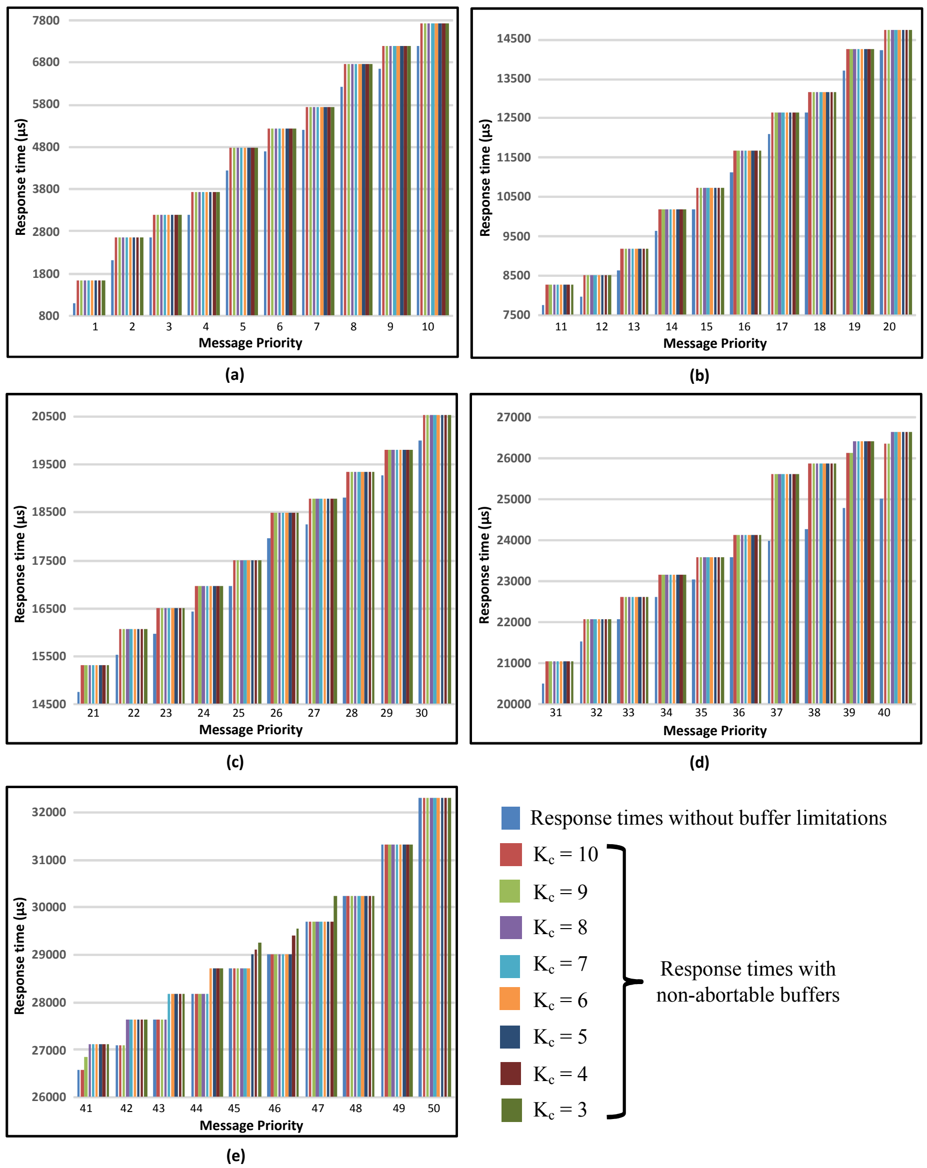

Figure 11: Comparison of message response times that are calculated with the extended analysis with non-abortable transmit buffers with different size of transmit buffers in the CAN controllers.

mit buffers with the extended analysis for mixed messages. The extended analyses are also applicable to heterogeneous types of systems where ECUs are supplied by different tier-1 suppliers. These ECUs may have different limitations in the CAN controllers, device drivers and protocol stack.

We also conducted a case study to show the applicability of the extended analyses and performed the comparative evaluation of the extended analyses. The evaluation results indicate that if there are limited number of transmit buffers in the CAN controllers and the effect of buffer limitations is not considered in the RTA, the calculated response times can be optimistic. Hence, it is important to use the RTA that matches the actual limitations and constraints in the hardware, device drivers and protocol stack.

Acknowledgement

This work is supported by the Swedish Research Council (VR) within the project TiPCES and the Swedish Knowledge Foundation (KKS) within the projects FEMMVA and EEMDEF. The authors thank the industrial partners Arcticus Systems AB, BAE Systems Hägglunds and Volvo CE, Sweden.

References

[1] S. Mubeen, J. Mäki-Turja, M. Sjödin, Extending schedulability analysis of Controller Area Network (CAN) for mixed (periodic/sporadic) messages, in: 16th IEEE Conference on Emerging Technologies and Factory Automation (ETFA), Sep., 2011.

[2] S. Mubeen, J. Mäki-Turja, M. Sjödin, Response time analysis for mixed messages in CAN supporting transmission abort requests, in: 7th IEEE International Symposium on Industrial Embedded Systems (SIES), Jun., 2012.

[3] S. Mubeen, J. Mäki-Turja, M. Sjödin, Extending response-time analysis of mixed messages in CAN with controllers implementing non-abortable transmit buffers, in: 17th IEEE Conference on Emerging Technologies and Factory Automation (ETFA), Sep., 2012.

[4] Robert Bosch GmbH, CAN specification version 2.0 (1991). Postfach 30 02 40, D-70442 Stuttgart.

[5] ISO 11898-1, Road Vehicles interchange of digital information controller area network (CAN) for high-speed communication, ISO Standard-11898, Nov. (1993).

[6] Marco Di Natale, Haibo Zeng, Paolo Giusto, Arkadeb Ghosal, Understanding and Using the Controller Area Network Communication Protocol, Springer, 2012.

[7] N. Audsley, A. Burns, M. Richardson, K. Tindell, A. J. Wellings, Applying new scheduling theory to static priority pre-emptive scheduling, Software Engineering Journal, 8 (1993) 284–292.

[8] N. Audsley, A. Burns, R. Davis, K. Tindell, A. Wellings, Fixed priority pre-emptive scheduling:an historic perspective, Real-Time Systems, 8 (1995) 173–198.

[9] L. Sha, T. Abdelzaher, K.-E. A. rzén, A. Cervin, T. P. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. P. Lehoczky, A. K. Mok, Real time scheduling theory: A historical perspective, Real-Time Systems, 28 (2004) 101–155.

[10] M. Joseph, P. Pandya, Finding response times in a real-time system, Computer Journal, 29 (1986) 390–395.

[11] M. Nolin, J. Mäki-Turja, K. Hänninen, Achieving industrial strength timing predictions of embedded system behavior, in: International Conference on Embedded Systems and Applications, 2008, pp. 173–178.

[12] K. Tindell, H. Hansson, A. Wellings, Analysing real-time communications: controller area network (CAN), in: Real-Time Systems Symposium (RTSS), 1994, pp. 259 –263.

[13] Volcano Network Architect. Mentor Graphics, http://www.mentor.com/products/vnd/communication-management/vna, accessed on Feb. 05, 2014.

[14] R. Davis, A. Burns, R. Bril, J. Lukkien, Controller Area Network (CAN) schedulability analysis: refuted, revisited and revised, Real-Time Systems, 35 (2007) 239–272.

[15] Rubus-ICE: Integrated component Development Environment, http://www.arcticus-systems.com, accessed on Feb. 05, 2014.

[16] S. Mubeen, J. Mäki-Turja, M. Sjödin, Support for end-to-end response-time and delay analysis in the industrial tool suite: Issues, experiences and a case study, Computer Science and Information Systems 10 (2013).

[17] D. Khan, R. Bril, N. Navet, Integrating hardware limitations in CAN schedulability analysis, in: 8th IEEE International Workshop on Factory Communication Systems (WFCS), May, 2010, pp. 207 –210.

[18] R. Davis, S. Kollmann, V. Pollex, F. Slomka, Schedulability analysis for controller area network (CAN) with FIFO queues priority queues and gateways, Real-Time Systems 49 (2013) 73–116.

[19] Marco Di Natale and Haibo Zeng, Practical issues with the timing analysis of the Controller Area Network, in: 18th IEEE Conference on Emerging Technologies and Factory Automation (ETFA), Sep., 2013.

[20] R. Davis, N. Navet, Controller area network (CAN) schedulability analysis for messages with arbitrary deadlines in FIFO and work-conserving queues, in: 9th IEEE International Workshop on Factory Communication Systems (WFCS), May, 2012, pp. 33 –42.

[21] A. Meschi, M. Di Natale, M. Spuri, Priority inversion at the network adapter when scheduling messages with earliest deadline techniques, in: Eighth Euromicro Workshop on Real-Time Systems, 1996, pp. 243–248.

[22] M. D. Natale, Evaluating message transmission times in Controller Area Networks without buffer preemption, in: 8th Brazilian Workshop on Real-Time Systems, 2006.

[23] D. Khan, R. Davis, N. Navet, Schedulability analysis of CAN with non-abortable transmission requests, in: 16th IEEE Conference on Emerging Technologies Factory Automation (ETFA), Sep., 2011.

[24] Transmit Cancellation in AUTOSAR Specification of CAN Driver, Rel. 4.1, Rev. 3, Ver. 4.3.0. March, 2014. http://www.autosar.org/download /R4.1/AUTOSAR_SWS_CANDriver.pdf, accessed on May 05, 2014.

[25] A. Szakaly, Response Time Analysis with Offsets for CAN, Master's thesis, Department of Computer Engineering, Chalmers University of Technology, 2003.

[26] Y. Chen, R. Kurachi, H. Takada, G. Zeng, Schedulability comparison for CAN message with offset: Priority queue versus FIFO queue, in: 19th International Conference on Real-Time and Network Systems (RTNS), Sep., 2011, pp. 181–192.

[27] P. Yomsi, D. Bertrand, N. Navet, R. Davis, Controller Area Network (CAN): Response time analysis with offsets, in: 9th IEEE International Workshop on Factory Communication Systems (WFCS), May, 2012.

[28] S. Mubeen, J. Mäki-Turja and M. Sjödin, Response-time analysis of mixed messages in Controller Area Network with priority- and FIFO-queued nodes, in: 9th IEEE International Workshop on Factory Communication Systems (WFCS), May, 2012.

[29] S. Mubeen, J. Mäki-Turja, M. Sjödin, Worst-case response-time analysis for mixed messages with offsets in Controller Area Network, in: 17th IEEE Conference on Emerging Technologies and Factory Automation (ETFA), Sep., 2012.

[30] S. Mubeen, J. Mäki-Turja, M. Sjödin, Extending offset-based response-time analysis for mixed messages in Controller Area Network, in: 18th IEEE Conference on Emerging Technologies and Factory Automation (ETFA), Sep., 2013.

[31] CANopen Application Layer and Communication Profile. CiA Draft Standard 301. Ver. 4.02. Feb., 2002. http://www.can-cia.org/index.php?id=440, accessed on Feb. 05, 2014.

[32] AUTOSAR Requirements on Communication, Rel. 4.1, Rev. 3, Ver. 3.3.1, Mar., 2014. www.autosar.org/download/R4.1/AUTOSAR_SRS_COM.pdf.

[33] Hägglunds Controller Area Network (HCAN), Network Implementation Specification, BAE Systems Hägglunds, Sweden (2009).

[34] I. Broster, Flexibility in Dependable Real-time Communication, Ph.D. thesis, University of York, 2003.

[35] C. Braun, L. Havet, N. Navet, NETCARBENCH: A benchmark for techniques and tools used in the design of automotive communication systems, in: 7th IFAC International Conference on Fieldbuses & Networks in Industrial & Embedded Systems, Nov., 2007.