MPS-CAN Analyzer: Integrated Implementation of Response-Time Analyses 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

We present a new response-time analyzer for Controller Area Network (CAN) that integrates and implements a number of response-time analyses which address various transmission modes and practical limitations in the CAN controllers. The existing tools for the response-time analysis of CAN support only periodic and sporadic messages. They do not analyze mixed messages which are partly periodic and partly sporadic. These messages are implemented by several higher-level protocols based on CAN that are used in the automotive industry. The new analyzer supports periodic, sporadic as well as mixed messages. It can analyze the systems where periodic and mixed messages are scheduled with offsets. It also supports the analysis of all types of messages while taking into account several queueing policies and buffer limitations in the CAN controllers such as abortable or non-abortable transmit buffers. Moreover, the tool supports the analysis of mixed, periodic and sporadic messages in the heterogeneous systems where Electronic Control Units (ECUs) implement different types of queueing policies and have different types of buffer limitations in the CAN controllers. We conduct a case study of a heterogeneous application from the automotive domain to show the usability of the tool. Moreover, we perform a detailed evaluation of the implemented analyses.

Keywords: Controller Area Network, Real-time networks, schedulability analysis, response-time analysis, mixed messages, buffer limitations.

1. Introduction

The Controller Area Network (CAN) [1] is a widely used real-time network protocol in the automotive domain. In 2003, it was standardized by the International Organization for Standardization in ISO 11898-1 [2]. It is a multi-master, event-triggered, serial communication protocol supporting bus speeds of up to 1 megabits per second. Over 850 million CAN enabled controllers were sold in 2011 according to the CAN in Automation (CiA) [3] estimate. Over 2 billion controllers have been sold to date and most of them have been used in the automotive industry. The CAN protocol also finds its applications in other domains, e.g., industrial control, medical equipments, maritime electronics, and production machinery. There are several higher-level protocols for CAN that are developed for many industrial applications such as CAN Application Layer (CAL), CANopen, J1939, Hägglunds Controller Area Network (HCAN), and CAN for Military Land Systems domain (MilCAN).

Often, CAN is used in hard real-time systems. The providers of these systems are required to ensure that the systems meet their deadlines. In order to provide evidence that each action by the system will be provided in a timely manner, *a priori* analysis techniques, such as schedulability analysis [4, 5, 6], have been developed by the research community. Response-Time Analysis (RTA) [4, 5, 6, 7] 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.

1.1. Paper contribution

There is a limitation with the existing response-time analyses for CAN and the corresponding tools that implement these analyses. That is, they support only periodic and sporadic messages. They do not support the analysis of mixed messages which are partly periodic and partly sporadic. Mixed messages are simultaneously time- and event-triggered and are implemented by several higher-level protocols based on CAN that are used in the automotive industry today. To the best our knowledge, there is no freely-available tool that implements the analysis of mixed messages (a commercial tool Rubus-ICE implements basic analysis of mixed messages in CAN). In this paper we present a new response-time analyzer for CAN namely MPS-CAN analyzer (MPS stands for Mixed, Periodic and Sporadic). It supports the analysis of periodic, sporadic and mixed messages. It implements several extensions of RTA for CAN taking into account the following aspects:

- analysis of mixed messages;
- analysis of messages scheduled with or without offsets;
- analysis of messages having arbitrary jitter and deadlines;
- analysis of network with CAN controllers implementing different queueing policies, e.g., priority or First-In, First-Out (FIFO),
- analysis of network with no buffer limitations in the CAN controllers, i.e., the controllers implement such a large (but finite) number of transmit buffers that there is no need to abort transmission requests;
- analysis of network with limitations in CAN controllers, e.g., the controllers implement abortable or non-abortable transmit buffers.

The tool also supports the analysis of mixed, periodic and sporadic messages in heterogeneous systems where Electronic Control Units (ECUs) implement different types of queueing policies and have different types of buffer limitations in the CAN controllers. In these systems, the tool treats each message differently depending upon its transmission type, and the type of queueing policy and buffer limitations in the sender ECU. We also conduct a case study in which we analyze the CAN messages in the heterogeneous system to show usability of the tool. Moreover, we perform a detailed evaluation of the implemented analyses.

1.2. Paper layout

The remainder of the paper is organized as follows. In Section 2, we discuss mixed transmission patterns supported by several higher-level protocols. In Section 3, we discuss the practical limitations in the CAN controllers. Section 4 discusses the related works. Section 5 discusses the implemented analyses, layout and usability of the MPS-CAN analyzer. Section 6 presents the case study and evaluation. Finally, Section 7 concludes the paper.

2. Mixed transmission supported by higher-level protocols

The analysis implemented in the MPS-CAN analyzer supports periodic and sporadic as well as mixed messages. In this section, we discuss and compare the implementation of mixed messages by several higher-level protocols for CAN. Traditionally, it is assumed that the tasks queueing CAN messages are invoked either by periodic or sporadic events. 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[1]) between the queueing of two successive sporadic messages has elapsed. However, there are some higher-level protocols and commercial extensions of CAN in which the tasks that queue the messages can be invoked periodically as well as sporadically. If a message can be queued for transmission periodically as well as sporadically, it is said to be mixed. In other words, a mixed message is simultaneously time- and event-triggered. We identified three different types of implementations of mixed messages used in the industry.

2.1. Method 1: Implementation in the CANopen protocol

The CANopen protocol [8] supports mixed transmission that corresponds to the Asynchronous Transmission Mode coupled with the Event Timer. The Event Timer is used to transmit an asynchronous message cyclically. A mixed message 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. A mixed message can also be queued periodically when the Event Timer expires. The Event Timer is reset every time the message is queued. Once a mixed message is queued, any additional queueing of this message will not take place during the Inhibit Time [8]. The transmission pattern of a mixed message in CANopen is illustrated in Figure 1(a). The down-pointing arrows symbolize the queueing of messages while the upward lines (labeled with alphabetic characters) represent arrival of the events. Message 1 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, message 2 is queued

[1] We overload the term "MUT" to refer to the Inhibit Time in the CANopen protocol and the Minimum Delay Time (MDT) in the AUTOSAR communication.

due to periodicity and both the Event Timer and Inhibit Time are reset again. When the event B arrives, message 3 is immediately queued because the Inhibit Time has already expired. Note that the Event Timer is also reset at the same time when message 3 is queued as shown in Figure 1(a). Message 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 with every sporadic transmission.

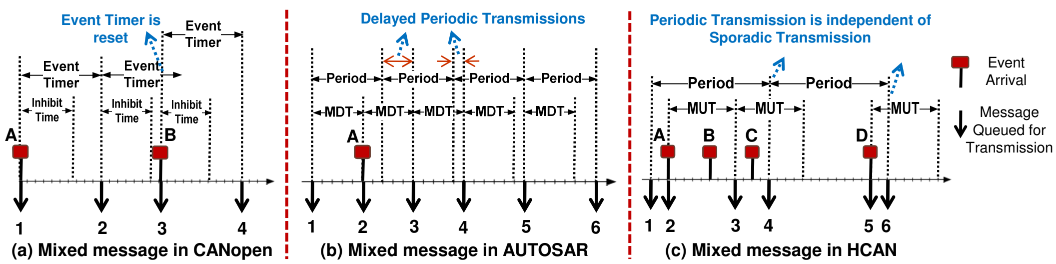

Figure 1: Mixed transmission pattern in higher-level protocols for CAN

2.2. Method 2: Implementation in the AUTOSAR communications

AUTOSAR (AUTomotive Open System ARchitecture) [9] can be viewed as a higher-level protocol if it uses CAN for network communication. Mixed transmission mode in AUTOSAR is widely used in practice. In AUTOSAR, 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 Minimum Delay Time (MDT) has been expired. However, each transmission of a mixed message, regardless of being periodic or sporadic, is limited by the MDT. This means that both periodic and sporadic transmissions are delayed until the MDT expires. The transmission pattern of a mixed message implemented by AUTOSAR is illustrated in Figure 1(b). Message 1 is queued (the MDT is started) because of partly periodic nature of a mixed message. When the event A arrives, message 2 is queued immediately because the MDT has already expired. The next periodic transmission is scheduled 2 time units after the transmission of message 2. However, the next two periodic transmissions corresponding to messages 3 and 4 are delayed because the MDT is not expired. This is indicated by the text "Delayed Periodic Transmissions" in Figure 1(b). The periodic transmissions corresponding to messages 5 and 6 take place at the scheduled times because the MDT is already expired in both cases.

2.3. Method 3: Implementation in the HCAN protocol

A mixed message in the HCAN protocol [10] 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 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. The transmission pattern of the mixed message is illustrated in Figure 1(c). Message 1 is queued because of periodicity. As soon as event A arrives, message 2 is queued. When event B arrives it is not queued immediately because the MUT is not expired yet. As soon as the MUT expires, message 3 is queued. Message 3 contains the signal changes that correspond to event B. Similarly, a message is not immediately queued when the event C arrives because the MUT is not expired. Message 4 is queued because of the periodicity. Although, the MUT was not expired, the event signal corresponding to event C was packed in message 4 and queued as part of the periodic message. Hence, there is no need to queue an additional sporadic message when the MUT expires. This indicates that the periodic transmission of a mixed message cannot be interfered by its sporadic transmission. This is a unique property of the HCAN protocol. When the event D arrives, a sporadic instance of the mixed message is immediately queued as message 5 because the MUT has already expired. Message 6 is queued due to the partly periodic nature of the mixed message.

2.4. Discussion

In the first method [8], the Event Timer is reset every time the mixed message is queued for transmission. The implementation of the mixed message in method 2 [9] is similar to method 1 to some extent. The main difference is that the periodic transmission can be delayed until the expiry of the MDT in method 2. 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 the MDT

respectively. Therefore, the existing analyses hold intact. However, the periodic transmission is independent of the sporadic transmission in the third method [10]. The periodic timer is not reset with every sporadic transmission. The mixed message can be queued for transmission even if the MUT is not expired. The worst-case periodicity of the mixed message is neither bounded by the period nor by the MUT. Therefore, the existing analyses cannot be applied to the mixed messages in the third implementation method. Further, there is no free tool that is able to analyze mixed messages that are implemented using the third method. Our main goal is to develop a free tool that analyzes periodic, sporadic, and as well as mixed messages in CAN.

3. Queueing policies and buffer limitations in the CAN controllers

The different types of queueing polices implemented by the CAN device drivers and communications stacks, internal organization, and hardware limitations in the CAN controllers can have significant impact on the timing behavior of CAN messages. In this section, we discuss various queueing policies and buffer limitations in the CAN controllers.

3.1. Common queueing policies used in the CAN controllers

The most common queueing policies in the *nodes* connected to the CAN network are priority-based and FIFO-based policies. It should be noted that a node or an ECU contains a CAN controller. We overload the terms node, ECU and CAN controller throughout this paper.

3.1.1. Priority-ordered queues

CAN implements priority-based arbitration which means that each node selects the highest priority message from its transmit buffers while entering into the bus arbitrations. The highest priority message among the messages selected from each node wins the bus arbitration, i.e., the right to transmit on the bus. Thus the most natural queueing policy suited to CAN controllers is priority-based queueing.

In order to demonstrate the priority based queueing policy, consider the example of three nodes namely Node A, Node B and Node C that are connected to a single CAN network as shown in Figure 2. Assume that each node sends three messages over the network. Node A sends the messages m_1, m_3 and m_5. Node B sends the messages m_2, m_4 and m_9. Whereas, Node C sends the messages m_6, m_7 and m_8. The number in the subscript denotes the message priority. We assume that the smaller the value of the subscript, the higher the priority. Thus m_1 is the highest priority message, whereas m_9 is the lowest priority message in the system. Assume that all messages in each node are queued for transmission. In order to simplify the example, assume that the periods of all messages are very high compared to their corresponding transmission times. We also assume that there cannot be multiple instances of a message queued for transmission at the same time.

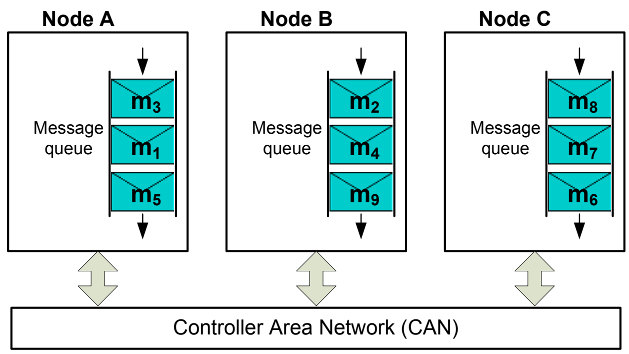

Figure 2: Example to demonstrate different queueing policies

Let the nodes implement priority ordered queues. Intuitively, each node will select the highest priority message from its queue to enter into the bus arbitration. In the first round, Nodes A, B, and C pick messages m_1, m_2 and m_6 respectively. m_1 wins the bus arbitration and is transmitted over the network as shown in Figure 3. In the second round, Nodes A, B, and C pick messages m_3, m_2 and m_6 respectively. This time, m_2 wins the bus arbitration and is transmitted over the network. Similar priority-based selection and arbitration occur during the rest of the rounds as shown in Figure 3.

3.1.2. FIFO queues

Due to simplicity of FIFO policy, some CAN controllers implement FIFO queues, e.g., Microchip PIC32MX, Infineon XC161CS, Renesas R32C/160 and XILINX LogiCORE IP AXI Controller [11, 12]. When the nodes implement FIFO queues, the oldest message in the transmit queue of each node competes for the bus with the oldest messages in the transmit queues in the rest of the nodes. However, the bus arbitration among these messages is done on priority basis. Consider again the example of the three nodes as shown

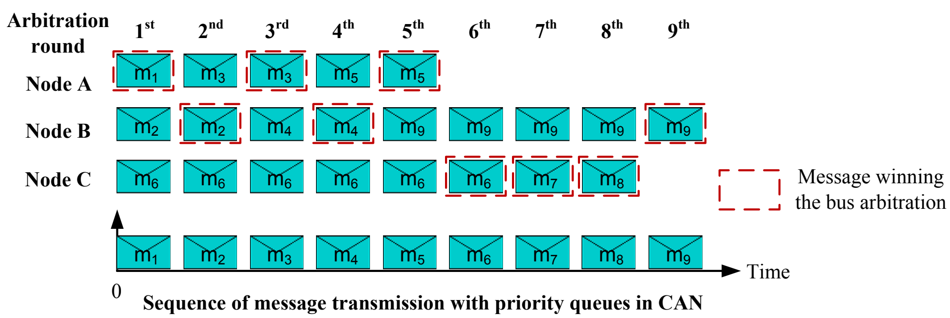

Figure 3: priority-based queues and CAN arbitration

in Figure 2. Assume that the nodes implement FIFO queues. Intuitively, each node will select the oldest message in its queue to enter into the bus arbitration. In the first round, Nodes A, B, and C pick messages m_5, m_9 and m_6 respectively. m_5 wins the bus arbitration due to its higher priority and is transmitted over the network as shown in Figure 4. In the second round, Nodes A, B, and C pick messages m_1, m_9 and m_6 respectively. This time, m_1 wins the bus arbitration and is transmitted over the network. Similar FIFO selection and priority-based arbitration occur during the rest of the rounds as shown in Figure 4.

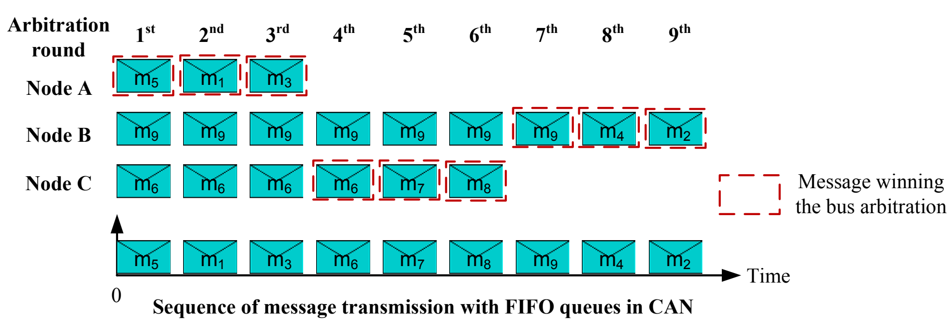

Figure 4: FIFO-based queues and CAN arbitration

When FIFO queues are used, the priorities of messages are often not respected in the transmit queue within a node, e.g., the lower priority message m_5 is transmitted before the highest priority message m_1 as shown in Figure 4. Moreover, priority inversions can occur due to which higher priority messages may have very large response times. This becomes evident by comparing the response time of m_2 in the systems with priority and FIFO queues as shown in Figures 3 and 4 respectively.

3.2. Buffer limitations in the CAN controllers

When there are fewer number of transmit buffers in the CAN controller compared to the number of messages sent by the ECU, the messages may be subjected to extra delay and jitter due to priority inversion. Examples of the CAN controllers that implement less than three transmit buffers are 8xC592, SJA1000 and 82C200 by Philips [11, 13, 14]. If a CAN controller has less than three transmit buffers and does not support transmission abort requests as in the case of Philips 82C200, a higher priority message released in the same controller may suffer from priority inversion [13, 15, 16]. That is, if all buffers in the CAN controller are occupied by lower priority messages, a higher priority message released in the same controller has to wait for one of the lower priority messages to transmit, thereby, vacating a space in the transmit buffer. During this waiting time, priority inversion occurs that adds an additional delay to the response time of the higher priority message.

The priority inversion can occur even if the controllers support transmission abort requests. Consider the case of two transmit buffers in every CAN controller. If a higher priority message becomes ready when both transmit buffers are occupied by the lower priority messages, the lowest priority message in the transmit buffer (that is not under transmission) is swapped with the higher priority message from the message queue. During the swapping process, it may be possible that the lower priority message from the second buffer finishes its transmission and the next arbitration period starts. At this point, both buffers may be empty while any other lower priority message from another node wins the arbitration and starts to transmit. This causes priority inversion for the higher priority message that is being swapped.

In the remaining part of this subsection, we consider the CAN controllers to implement limited number (at least three) of transmit buffers. First we consider the case where the CAN controllers support transmission abort requests, e.g., Atmel AT89C51CC03/AT90CAN32/64 and Microchip MPC2515 [11]. Second we consider the case in which the CAN controllers implement non-abortable transmit buffers, e.g., Philips 82C200 [13, 15, 16].

3.2.1. Additional delay and jitter due to priority inversion in the case of abortable transmit buffers

If the CAN controller supports transmission abort requests (and implements at least 3 transmit buffers) then the lowest priority message in the transmit buffer that is not undergoing transmission is swapped with the higher priority message from the message queue. During the swapping process, a lower priority message from the transmit buffer in any other controller may win the bus arbitration and contribute an extra delay to the response time of the higher priority message. The copying delay and the extra blocking delay during the swapping process should be taken into account while calculating the response time of the higher priority message.

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 shown in Figure 5. 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. m_6 cannot be preempted because CAN uses fixed priority non-preemptive scheduling. 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.

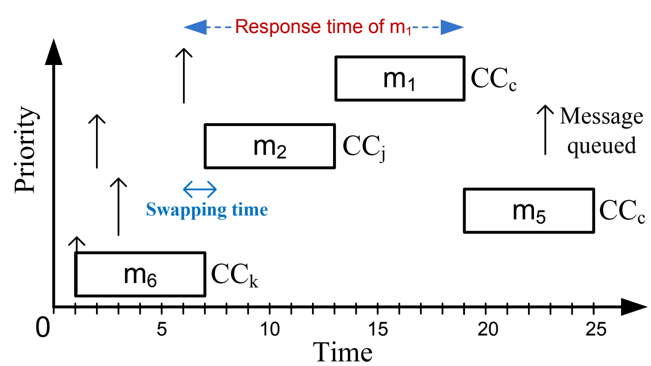

Figure 5: Demonstration of priority inversion in the case of abortable transmit buffers

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 5. During the swapping time a series of events may 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 for m_1 and adds an extra delay to its response time. In [12], 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 5.

Discussion on message copy time and delay

If the message copy time is smaller than or equal to the inter-frame space (i.e., time to transmit 3 bits on CAN bus), a lower priority message in the transmit buffer (that is not under transmission) can be swapped with a higher priority message in the message queue before the transmission of the next frame on the CAN bus [1]. Hence, there will be no priority inversion. This means that the message copy time must be, at least, $4{*}\tau_{bit}$ for the priority inversion to occur. Where τ_{bit} is the time required to transmit a single bit on CAN. For example, it is equal to 1 microsecond for the CAN bus speed of 1 Mbit/s. In Legacy systems, there may be slow controllers, i.e., the speed of the controllers can be slower than the maximum operating speed of the CAN bus (1 Mbit/s). Since the amount of data transmitted in a CAN message rages from 0 to 8 bytes, the transmission time of a message also varies accordingly. According to [17], the transmission time of a CAN message with standard frame format ranges from $55{*}\tau_{bit}$ to $135{*}\tau_{bit}$ for the amount of data contained in the message that ranges from 0 to 8 bytes respectively. Let us assume the message copy time to be equal to $4{*}\tau_{bit}$. Intuitively, the message copy time can range from 7.3% to 3% of transmission time of a message with 0 to 8 bytes of data respectively. Due to slow controllers that may be found in legacy systems, the message copy time can be greater than $4{*}\tau_{bit}$. Hence, the message copy time can be higher than 7.3% of its transmission time.

3.2.2. Additional delay and jitter due to priority inversion in the case of non-abortable transmit buffers

When CAN controllers do not support transmission abort requests, a higher priority message may suffer from priority inversion and this, in turn, may add extra delay to its response time [13]. Consider an example of three controllers CC_c, CC_j, CC_k connected to a single CAN network in Figure 6. 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 the transmit buffers of node CC_c is transmitted.

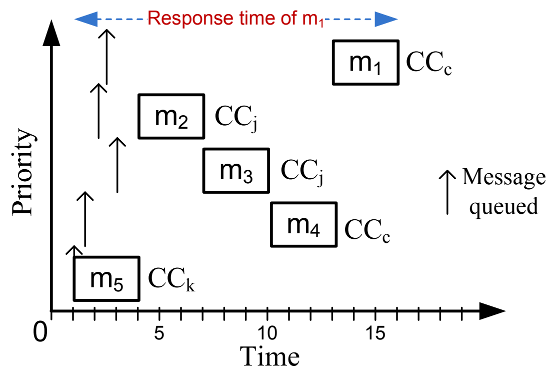

Figure 6: Demonstration of priority inversion in the case of non-abortable transmit buffers

Let m_4 be the highest priority message in the transmit buffers of node CC_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 for m_1 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. In this example, this additional delay is the sum of the worst-case transmission times of m_2, m_3 and m_4. This additional delay appears as additional jitter of m_1 as seen by the lower priority messages.

4. Related works

4.1. Related analyses

Tindell et al. [16] developed the schedulability analysis for CAN. It has been implemented in the analysis tools that are used in the automotive industry, e.g., Volcano Network Architect (VNA) [18]. Davis et al. [17] refuted, revisited and revised the seminal analysis of [16]. The revised analysis is implemented in the existing industrial tool suite Rubus-ICE [19, 20]. These analyses assume that each node selects the highest priority message, that is ready for transmission, from its transmit buffers when entering into the bus arbitration. It is noted in [11, 12, 13, 14, 15, 21, 22, 23, 24] that this assumption may become invalid in some cases due to various practical limitations such as controllers implementing FIFO and work-conserving queues, limited number of transmit buffers, copying delays in transmit buffers, transmit buffers supporting abort requests and protocol stack prohibiting transmission abort requests in some configurations as in the case of AUTOSAR [25].

In [11, 14, 24], Davis et al. extended the analysis of CAN with FIFO and work-conserving queues while supporting arbitrary deadlines of messages. In [22], Meschi et al. proved the priority inversion due to limited buffers can be avoided if the controller implements at least three transmit buffers. However, the analysis in [22] does not account the overhead of the copying delay. Khan et al. [12] integrated this extra delay with the analysis in [16, 17] for the case of abortable transmit buffers. In the case of CAN controllers implementing non-abortable transmit requests, RTA for CAN is extended in [13, 15]. But, none of the analysis discussed above supports messages that are scheduled with offsets. The worst-case RTA for CAN messages with offsets has been developed in several works [26, 27, 28, 29, 30].

However, all these analyses assume that the messages are queued for transmission either periodically or sporadically. They do not support mixed messages that are partly periodic and partly sporadic. Mubeen et al. [31] extended the seminal and revised analyses [16, 17] to support mixed messages in CAN where nodes implement priority queues. Mubeen et al. [32] further extended their analysis to support mixed messages in CAN where some nodes implement priority queues while others implement FIFO queues. In [33] and [34] we extended the analysis for mixed messages in CAN where the controllers implement abortable and non-abortable transmit buffers respectively. Mubeen et al. also extended the existing analysis for CAN to support periodic, sporadic and mixed messages that are scheduled with offsets [35, 36].

4.2. Related tools

VNA [18] is a communication design tool that supports RTA for CAN. It implements RTA of CAN developed by Tindell et al. [16].

Vector[2] is a tools provider for the development of networked electronic

[2]http://www.vector.com

systems. CANalyzer [37] supports the simulation, analysis and data logging for the systems that use CAN for network communication. CANoe [38] is a tool for the simulation of functional and extra-functional (e.g., timing) behavior of ECU networks. Network Designer CAN is another tool by Vector that is used to design the architecture and perform timing analysis of CAN.

SymTA/S [39] is a tool by Symtavision for model-based timing analysis and optimization. Among other analyses, it supports statistical, and worst- and best-case timing analysis for CAN.

RTaW-Sim [40] is a tool for the simulation and performance evaluation of the CAN network.

The Rubus-ICE is a commercial tool suite developed by Arcticus Systems[3] in close collaboration with Mälardalen University Sweden. It supports model- and component-based development of real-time embedded systems[41, 42]. Among other analyses, it supports RTA of CAN [16, 17] and RTA of CAN for mixed messages[31].

To the best of our knowledge, there is no freely-available tool that implements RTA of CAN for mixed messages. The main purpose of MPS-CAN analyzer is to support RTA of periodic, sporadic and mixed messages in CAN while taking into account different queueing policies and buffer limitations in the CAN controllers and device drivers.

4.3. Extended version

This paper extends our previous work [43] where we discussed the implementation of RTA for periodic, sporadic and mixed messages in CAN without considering hardware and software limitations in the CAN controllers and device drivers. In the extended version of the paper, we discuss the integration of these limitations with the response-time analysis for CAN and implementation of the analyses in the tool. Moreover, we conduct a detailed case study from the automotive domain. We also evaluate the implemented analysis.

5. Implemented analyses, layout and usage of the tool

5.1. Analyses implemented in the MPS-CAN analyzer

The analyses that we implemented in the MPS-CAN analyzer consist of RTA for CAN and its several extensions as shown in Figure 7. The figure

[3]http://www.arcticus-systems.com

also shows the relationship among the implemented analyses. We denote each extension of the RTA for CAN by the term "analysis profile".

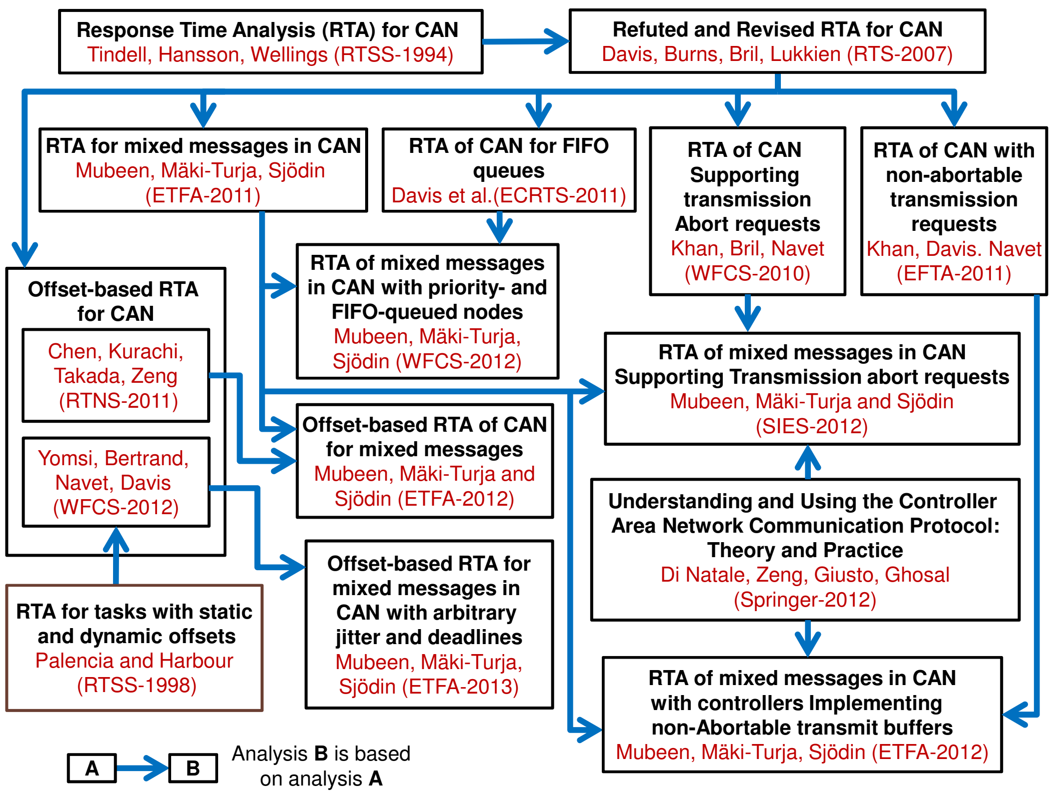

Figure 7: Graphical representation of the Response Time Analysis (RTA) and its extensions implemented in the MPS-CAN analyzer

5.2. Implementation and distribution

We developed an algorithm that integrates the analysis profiles that are shown in Figure 7. It also shows the high-level implementation of the analyses in the MPS-CAN analyzer as depicted in Algorithm 1. The MPS-CAN analyzer is implemented in the C language. The graphical user interface of the tool is developed using the Windows Application Programming Interface (WinAPI). Each analysis profile supported by the tool is implemented as a separate C file which is accessed using function calls in the main file. The Figure 8 shows the screen shot of the code where a number of functions corresponding to different analyses are shown. A new analysis can be easily added to the MPS-CAN analyzer by adding a similar function and corresponding source files (.c and .h) provided the new analysis complies with the input and output interfaces shown in the structures in Figure 9. Hence, the tool supports a simple and easy mechanism for further extensions and implementation of other related analyses in the future. The tool, user manual, and test cases can be downloaded at https://github.com/saadmubeen/MPS-CAN.

```
        analysis_type.analysis_property = PRIORITY;
        analysis_type.offset = OFFSET_UNAWARE;

        Compute_Rm_of_all_MSGs_Mixed_Prio();

        Compute_Rm_of_ ll_MSGs_Mixed_Abort();

        Compute_Rm_of_all_MSGs_Mixed_NonAbort();

        Compute_Rm_of_all_MSGs_Mixed_Prio_Offset();

        Compute_Rm_of_all_MSGs_Mixed_FIFO();
```

Figure 8: Screen shot from the code: functions corresponding to different analyses

```
typedef struct ECU_TYPE
{
    long int    ID;                  // Node ID
    buf_type    buffer_type;         // abortable, non-abortable, not aplicable--Added for CAN journal analysis
    long int    Kc;                  // Size of transmission buffer
    long int    max_Pm_Kc;           // Priority of highest priority message in Kc for the respective node
    long int    nr_of_msgs;          // Total number of messages belonging to the ECU
} ECU_type;

typedef struct message
{
    long int    ID;                  // Reference to corresponding msg ID in Network Specification
    long int    FRAME_TYPE;          // Standard(11-bit) or extended(29-bit)
    long int    TRANSMISSION_TYPE;   // PERIODIC, EVENT(e.g. All signals in frame are of event type), MIXED
    long int    Tm;                  // Message Transmission Period
    long int    MIN_UPDATE_TIME;     // Minimum Update Time for Event and Mixed Transmission of the msg,
    long int    Dm;                  // Message deadline i.e. from queuing the msg to delivery to the receiving CAN controller
    long int    DLC;                 // Data Length Code: Number of data bytes in a CAN Data Frame (0-8 bytes)
    long int    Jm;                  // Release Jitter of a message inherited as WCRT from the task producing it
    long int    Om;                  // Offset
    long int    CC;                  // Sender Node
    long int    Cm;                  // Transmission time
    int         Cm_computation;      // Calculated Cm
    long int    ECU_ID;              // The node to which this msg belongs
    long int    CTm;                 // Copy Time
    CTm_input   CTm_in;              // How CTm has been provided: DEFAULT (more than 3bits, decide later), PERCENTAGE, USER_DEFINED
    long int    Jm_hat_A;            // Total jitter
    long int    Bm_hat_A;            // Additional blocking
    long int    AJm_A;               // Addition jitter
    long int    Rm_P;                // Worst Case Response-Time of a Periodic copy of MIXED message
    long int    Rm_S;                // Worst Case Response-Time of a Sporadic copy of MIXED message
    long int    Rm;                  // Worst Case Response-Time of a message
    MSG_trace_type  MSG_trace;       // Contains the linking information of the message, i.e., who is the sender and who are the receivers

} MSGtype;
```

Figure 9: Screen shot from the code: structures for inputs and outputs

Algorithm 1 Algorithm for high-level implementation of the analyses

```
begin
RT_Prev ← 0                          ▷ Initialize all Response Times (RTs) to zero
READ_INPUT ()                        ▷ Bus speed, ECUs, and messages input
procedure CALCULATE_MESSAGE_RESPONSE_TIME ()
    if message_under_analysis ∈ ECU_with_prrioty_queue then
        if buffer_type == no_limitaton then
            RTA_OF_CAN_WITH_PRIORITY_QUEUES ()
        else if buffer_type == abort then
            RTA_OF_CAN_ABORTABLE_TRASNSMIT_BUFFERS ()
        else if buffer_type == non_abort then
            RTA_OF_CAN_NON-ABORTABLE_TRASNSMIT_BUFFERS ()
        end if
    else
        RTA_OF_CAN_WITH_FIFO_QUEUES ()
    end if
end procedure
for all Messages_in_the_system do
    Repeat ← TRUE
    while Repeat = TRUE do
        if messages_are_scheduled_with_offsets == FALSE then
            CALCULATE_MESSAGE_RESPONSE_TIME ()
        else
            CALCULATE_MESSAGE_RESPONSE_TIME_WITH_OFFSETS ()
        end if
        if RT > RT_Prev then
            RT_Prev ← RT
            Repeat ← TRUE
        else
            Repeat ← FALSE
        end if
        if RT ≥ Deadline then
            Repeat ← FALSE
        end if
    end while
end for
end
```

5.3. Tool layout, inputs and outputs

The Layout of the MPS-CAN analyzer is shown in Figure 10. There is a main window denoted by “MPS-CAN Analyzer” which serves as the user interface. The input section of the tool consists of the list boxes (“Message List”, “Node List”, “Network Speed” and “Number of Nodes”) and buttons. Whereas, the output section of the tool comprises of the list boxes namely “Output”, “Network Utilization”, and “Errors and Warnings”.

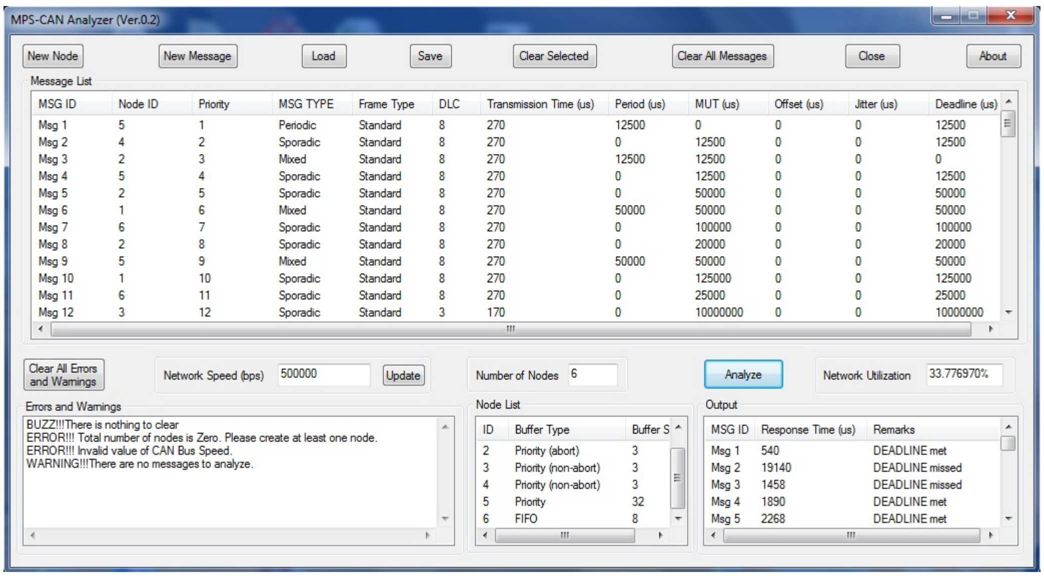

Figure 10: MPS-CAN analyzer layout, inputs and outputs

When the “New Node” button is clicked on the main window, a new window namely “New Node” opens up as shown in Figure 11. This window is used to create a new node. In this window the user can specify the node ID and the number of transmission buffers in the node. This window also allows implicit selection of the analysis profiles. If the selected type of transmit buffers is “Priority (no buffer limitations)”, the node is assumed to implement priority-based queueing policy. Furthermore, the node contains very high (but finite) number of transmit buffers compared to the number of messages that are sent by this node. In this case, the RTA for mixed, periodic, and sporadic messages without any buffer limitations is used to analyze all messages that are sent by this node.

If the selected type of transmit buffers is “Priority (abortable buffer)” or “Priority (non-abortable buffer)”, the node contains limited (at least three) number of transmit buffers which are of abortable or non-abortable type respectively. In both of these cases, the node is assumed to implement priority-based queueing policy. In these two cases, the RTA for mixed, periodic, and sporadic messages supporting abortable or non-abortable transmit buffers is used to analyze all messages that are sent by this node respectively. Similarly, if the selected type of transmit buffers is "FIFO", the node is assumed to implement FIFO-based queueing policy. In this case, the RTA for mixed, periodic, and sporadic messages in CAN with FIFO queues is used to analyze all messages that are sent by this node.

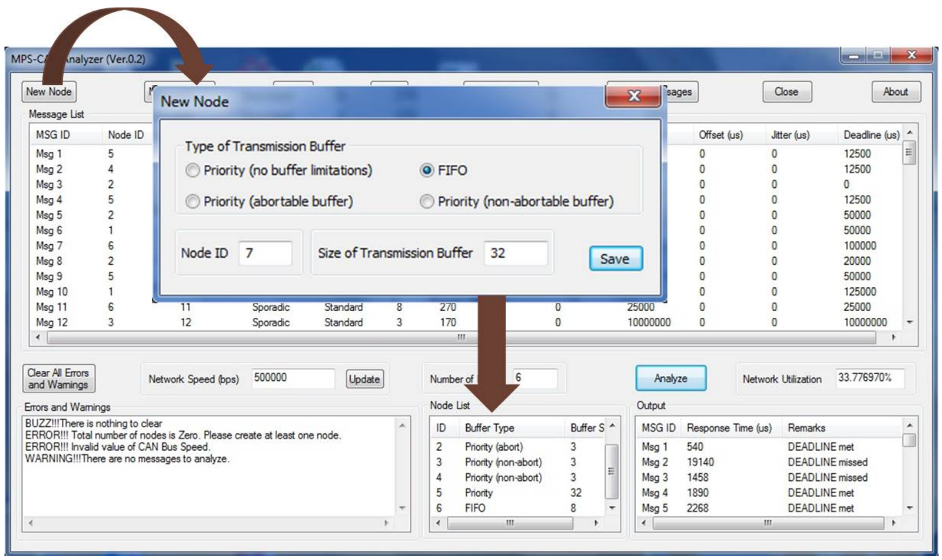

Figure 11: Creating a new node and implicitly selecting the RTA in the MPS-CAN analyzer

When the "New Message" button is clicked on the main window, a new window namely "New Message" pops up as shown in Figure 12. This window is used to create a new message. In this window, message attributes are provided as input. For a mixed message, both period and minimum update time are specified. Whereas for a periodic or sporadic message, only period or minimum update time is specified respectively. The transmission type of a message can be selected from periodic, sporadic, or mixed. There are two options for specifying transmission type of a message. First option is based on specifying Data Length Code (DLC), i.e., the number of data bytes present in the CAN message. The second option allows to specify user-defined transmission time. This option may be used for analyzing simplified test cases

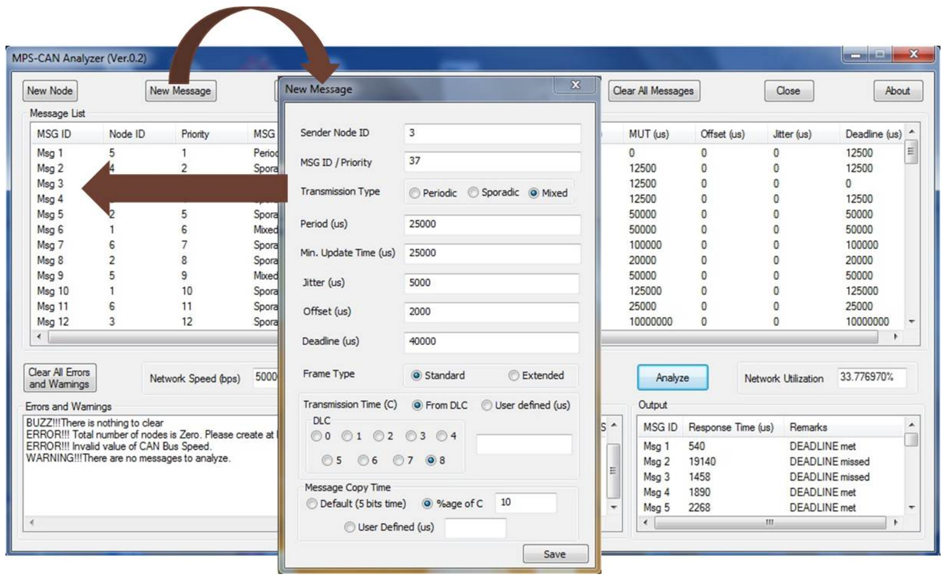

Figure 12: Creating a new message in the MPS-CAN analyzer

that are more suitable for research-oriented work. There are several options to select and specify "Message Copy Time" which is the time required to copy a message from the transmit buffer to the message queue or vice versa. If the message offset is specified, then the messages are analyzed using the offset-based RTA for mixed, periodic, and sporadic messages in CAN.

In the main window, the network speed in bits per second (bps) can be specified. Moreover, there are buttons provided to clear, save and load messages. Any message set can be analyzed by clicking the "Analyze" button. If errors and warnings occur during the run of the analyzer, they are displayed in the "Errors and Warnings" list box. Figure 10 shows some errors and warnings that may occur when the analyzer is run. The "Output" list box displays the calculated response times of the messages. It also displays whether a message meets its deadline or not (provided the deadline is specified by the user). The percentage network utilization is also calculated and displayed in the "Network Utilization" list box.

6. Case study and evaluation

In order to show the usability of the MPS-CAN analyzer, we conduct an automotive-application case study. Basically, we adapt the case study of the experimental vehicle that is discussed and analyzed in [21].

6.1. Experimental setup

The system model in the original experimental vehicle consists of 6 identical ECUs (identical in terms of buffer limitations) that are connected to a single CAN network. There are 81 periodic CAN messages in the system. We adapt this system in such a way that it becomes heterogeneous in terms of different queueing policies and buffer limitations in the ECUs. However, the number of ECUs and messages remains unchanged. That is, the modified experimental vehicle contains six ECUs out of which two use priority-based queueing policy and each of them implements 3 abortable transmit buffers; two use priority-based queueing policy and each of them implements 3 non-abortable transmit buffers; one implements FIFO queue with 8 buffers; and the remaining ECU uses priority-based queueing policy and has no buffer limitations which means that it implements very large but finite number of transmit buffers (32 buffers). The 81 messages are equally assigned different transmission types. This means, there are 27 periodic, 27 sporadic, and 27 mixed messages in the system.

All the attributes of these messages are tabulated in Figure 13. The attributes of each message are identified as follows. The priority, sender ECU ID, type of transmit buffers implemented by the sender ECU, transmission type, number of data bytes in the message, transmission period, minimum update time, deadline, and calculated worst-case response time are represented by *Prio*, *ECU_ID*, *ECU_Type*, ξ, DLC, T, MUT, D, and R respectively. We assume, the smaller the value of the *Prio* parameter of a message, the higher its priority. Thus, the message with priority 1 is the highest priority message, whereas the message with priority 81 is the lowest priority message in the system under analysis. We assume that the copy time of each message is more than the time required to transmit 4 bits on the CAN bus. For simplicity, the copy time of each message is selected to be 10% of its transmission time. All timing parameters are in microseconds. The selected speed for CAN is 500 Kbit/s.

The MPS-CAN analyzer treats each message differently depending upon its transmission type; and the type of queueing policy and buffer limitations in the sender ECU. The worst-case response times of all messages calculated by the MPS-CAN analyzer are listed in Figure 13. The network utilization calculated by the MPS-CAN analyzer for this message set is equal to 33.776970%. The tool takes less than 2 seconds to analyze the case study

Prio	ECU ID	ECU Type	ξ	DLC(byte)	T (us)	MUT (us)	D (us)	R (us)	Prio	ECU ID	ECU Type	ξ	DLC(byte)	T (us)	MUT (us)	D (us)	R (us)
1	5	Prio-No-Limit	P	8	12500	0	12500	540	42	2	Abort	P	8	100000	0	100000	16748
2	4	Non-Abort	S	8	0	12500	25000	19140	43	4	Non-Abort	S	8	0	100000	100000	22110
3	2	Abort	M	8	12500	12500	12500	1458	44	6	FIFO	P	8	100000	0	100000	32610
4	5	Prio-No-Limit	S	8	0	12500	12500	1890	45	5	Prio-No-Limit	S	8	0	50000	50000	17450
5	2	Abort	S	8	0	50000	50000	2268	46	4	Non-Abort	P	8	50000	0	50000	22380
6	1	Abort	M	8	50000	50000	50000	2538	47	1	Abort	S	8	0	50000	50000	18368
7	6	FIFO	S	8	0	100000	100000	32610	48	4	Non-Abort	M	8	50000	50000	50000	22650
8	2	Abort	S	8	0	20000	20000	3348	49	1	Abort	S	8	0	1000000	1000000	19448
9	5	Prio-No-Limit	M	8	50000	50000	50000	3510	50	3	Non-Abort	P	8	1000000	0	1000000	29670
10	1	Abort	S	8	0	125000	125000	4158	51	4	Non-Abort	S	8	0	1000000	1000000	23190
11	6	FIFO	S	8	0	25000	35000	32610	52	6	FIFO	P	8	1000000	0	1000000	32610
12	3	Non-Abort	S	3	0	10000000	10000000	26700	53	3	Non-Abort	M	8	128000	128000	128000	29940
13	6	FIFO	M	8	100000	100000	100000	32730	54	2	Abort	S	8	0	128000	128000	22418
14	4	Non-Abort	P	8	100000	0	100000	20760	55	1	Abort	P	8	128000	0	128000	22688
15	6	FIFO	M	8	100000	100000	100000	32730	56	4	Non-Abort	M	8	1000000	1000000	1000000	23460
16	6	FIFO	M	8	100000	100000	100000	32730	57	4	Non-Abort	S	8	0	250000	250000	24000
17	5	Prio-No-Limit	S	8	0	100000	100000	7190	58	3	Non-Abort	M	3	250000	250000	250000	30380
18	5	Prio-No-Limit	P	8	1000000	0	1000000	7460	59	4	Non-Abort	M	8	500000	500000	500000	24000
19	4	Non-Abort	S	8	0	1000000	1000000	21030	60	2	Abort	M	8	500000	500000	500000	24648
20	1	Abort	P	8	1000000	0	1000000	8108	61	5	Prio-No-Limit	M	7	500000	500000	500000	25060
21	5	Prio-No-Limit	P	8	1000000	0	1000000	8270	62	1	Abort	M	8	500000	500000	500000	26714
22	1	Abort	M	8	500000	500000	500000	8648	63	1	Abort	S	2	0	500000	500000	27110
23	1	Abort	P	8	500000	0	500000	9188	64	1	Abort	M	8	1000000	1000000	1000000	27404
24	3	Non-Abort	S	8	0	500000	500000	26970	65	2	Abort	P	8	1000000	0	1000000	28808
25	4	Non-Abort	P	8	500000	0	500000	21300	66	2	Abort	M	8	1000000	1000000	1000000	29078
26	2	Abort	P	8	100000	0	100000	9998	67	2	Abort	P	8	1000000	0	1000000	29564
27	3	Non-Abort	S	8	0	100000	100000	27240	68	3	Non-Abort	P	8	1000000	0	1000000	31090
28	1	Abort	P	8	100000	0	100000	10538	69	6	FIFO	P	6	1000000	0	1000000	32610
29	3	Non-Abort	S	8	0	1000000	1000000	27510	70	5	Prio-No-Limit	S	8	0	2000000	2000000	30280
30	5	Prio-No-Limit	M	8	1000000	1000000	1000000	10970	71	6	FIFO	S	8	0	2000000	2000000	32610
31	5	Prio-No-Limit	S	8	0	1000000	1000000	11510	72	3	Non-Abort	P	8	2000000	0	2000000	31090
32	2	Abort	M	8	20000	20000	30000	11888	73	3	Non-Abort	M	8	2000000	2000000	2000000	31360
33	1	Abort	S	8	0	50000	50000	12428	74	4	Non-Abort	M	8	2000000	2000000	2000000	32170
34	5	Prio-No-Limit	M	8	500000	500000	500000	12590	75	2	Abort	S	8	0	2000000	2000000	32764
35	5	Prio-No-Limit	P	8	20000	0	50000	14210	76	6	FIFO	P	8	2000000	0	2000000	32610
36	4	Non-Abort	P	8	500000	0	500000	21570	77	2	Abort	M	8	2000000	2000000	2000000	33304
37	5	Prio-No-Limit	P	8	20000	0	50000	14750	78	6	FIFO	M	2	2000000	2000000	2000000	32610
38	6	FIFO	S	8	0	200000	200000	32610	79	4	Non-Abort	M	1	50000	50000	100000	33830
39	3	Non-Abort	P	8	20000	0	50000	27780	80	6	FIFO	M	2	1000000	1000000	1000000	32610
40	1	Abort	P	8	200000	0	200000	16208	81	3	Non-Abort	M	2	2000000	2000000	2000000	33410
41	3	Non-Abort	P	8	1000000	0	1000000	28590									

Figure 13: Attributes and calculated response times of periodic, sporadic and mixed messages in the automotive case study

on a laptop with dual core 2.4 GHz processor, 2 GB RAM and Windows (OS). By comparing the calculated response time with the corresponding deadline of each message in the table, it is obvious that all messages meet their deadlines. Hence, the heterogeneous system is schedulable.

6.2. Comparison of various response-time analyses

In order to compare the response times calculated from different analyses in the MPS-CAN analyzer, we perform four more tests on four different sets of ECUs. There are identical ECUs in each set. The same message set is analyzed in all tests. In the first test, each ECU uses priority-based queueing policy and implements large but finite number of transmit buffers (32 in this case). In this test we use the analysis for mixed, periodic, and sporadic messages in CAN with priority queues and no buffer limitations. In the second test, each ECU uses priority-based queueing policy and implements

3 abortable transmit buffers. In this test, we use the analysis for mixed, periodic, and sporadic messages in CAN with abortable transmit buffers. In the third test, each ECU uses priority-based queueing policy and implements 3 transmit buffers which are of non-abortable type. In this test we use the analysis for mixed, periodic, and sporadic messages in CAN with non-abortable transmit buffers available. Whereas, in the fourth test, each ECU uses FIFO-based queueing policy and implements 8 transmit buffers. In this test, the same message set is analyzed using the analysis for mixed, periodic, and sporadic messages in CAN with FIFO queues.

The response times of all messages in these four cases along with the response times of messages in the heterogeneous system are shown by the bar graphs in Figure 14. The results indicate that the message response times are the best (smallest) in the first test. This is because the corresponding analysis assumes the ideal behavior of the CAN controllers, i.e., no buffer limitations, and hence, no extra delays due to priority inversion. The second best response times are obtained in the second test. The response times in this test are higher than the response times in the first test due to the copying delay and extra delay because of the priority inversion discussed in the Section 3.2.1. The third best response times are obtained in the third test. However, these response times are considerably large compared to the response times in the first and second tests. This is because of the extra delay due to priority inversion discussed in the Section 3.2.2. Due to priority inversion, some higher priority messages have larger response times compared to the lower priority messages. For example, the response time of message with priority 2 is higher than the response time of the message with priority 10. On the average, the response times of the messages in the heterogeneous system are comparable to the response times of the messages in the third test. Finally, the response times of the messages are the worst (largest) in the fourth test. The response times in this case are significantly large compared to the first two tests because of large delays due to priority inversion within the FIFO queues as discussed in the Section 4.

6.3. Discussion

In order to get short response times of CAN messages, those ECUs should be selected which use priority-based queueing policy and implement much higher number of transmit buffers compared to the number of messages sent by them. However, practical systems use ECUs with limited number of transmit buffers. If ECUs with very large number of transmit buffers are not

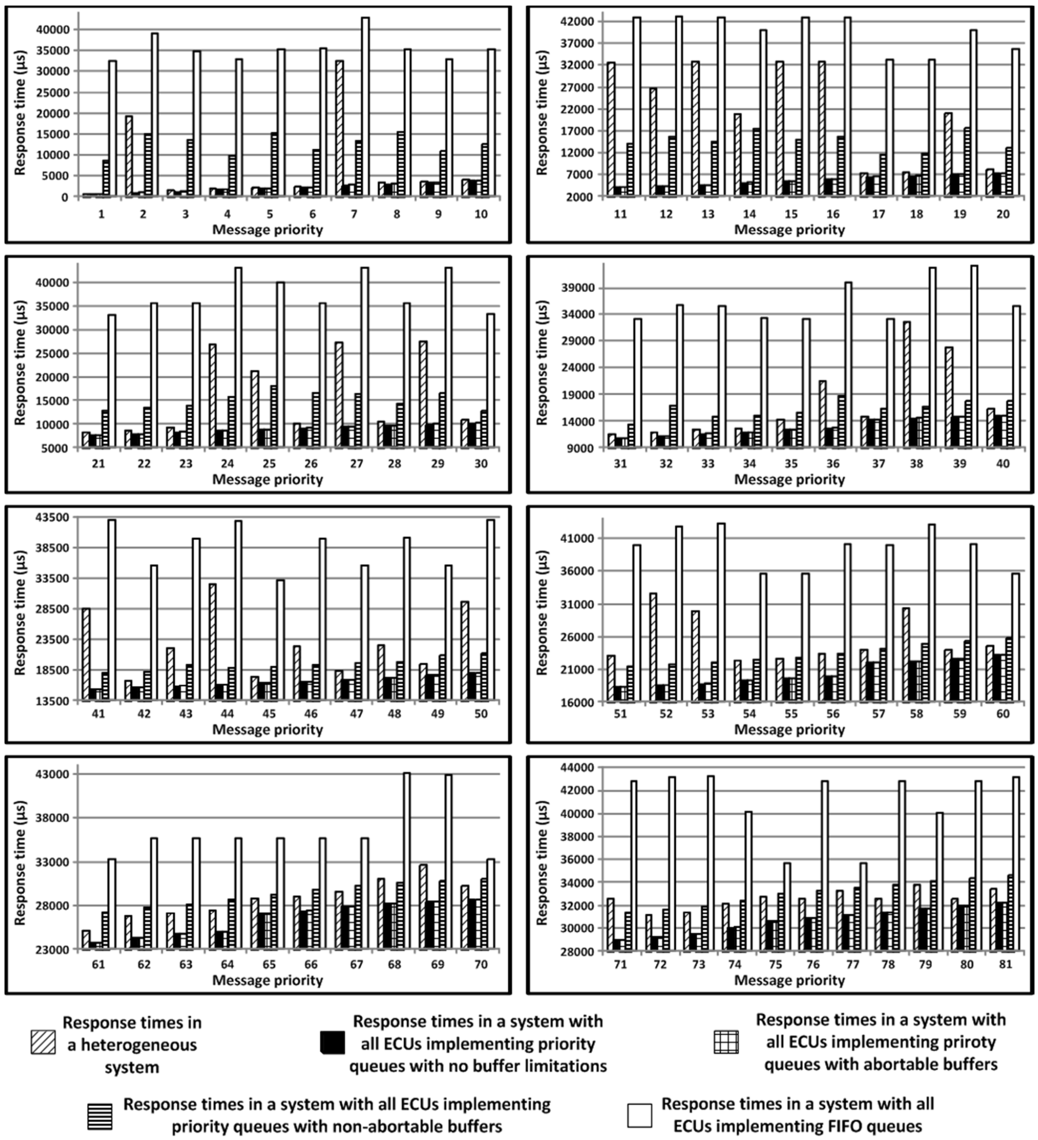

Figure 14: Comparison of message response-times with respect to different types of buffer limitations in the ECUs

available then the ECUs with abortable transmit buffers should be preferred over the ECUs that implement non-abortable transmit buffers. Although FIFO policy is easy to implement and simple to use as compared to the priority queueing policy, the messages can have very large worst-case response times in the case of ECUs implementing FIFO queues. The ECUs which implement priority-based queueing policy should be preferred over the ECUs which implement FIFO queues especially in high utilization systems.

Moreover, it is important to use the right RTA that correctly matches the queueing policies; buffer limitation in the CAN controllers; and transmission type of messages used in the higher-level protocols. If the practical limitations and constraints are not considered in the RTA, the calculated response times can be optimistic. The MPS-CAN analyzer considers these limitations and constraints while analyzing the CAN messages. It treats each message differently based on its transmission type, and queueing policy and buffer limitations in the CAN controller of its sender ECU.

7. Conclusion

We introduced a new tool MPS-CAN analyzer to support Response Time Analysis (RTA) of periodic, sporadic and mixed messages in the Controller Area Network (CAN). The existing RTA tools for CAN analyze only periodic and sporadic messages. They do not support the analysis of mixed messages which are partly periodic and partly sporadic. These messages are implemented by several higher-level protocols for CAN that are used in the automotive industry today.

The MPS-CAN analyzer implements various extensions of the RTA for CAN while taking into account mixed messages, messages scheduled with offsets, messages with arbitrary jitter and deadlines, various queueing policies (e.g., priority- or FIFO-based), and limitations of transmit buffers in the CAN controllers (e.g., abortable or non-abortable). With the implementation of these analyses, the MPS-CAN analyzer is able to analyze network communications in heterogeneous systems which may consist of different types of ECU's supplied by different Tier 1 suppliers.

We also showed the usability of the MPS-CAN analyzer by conducting the case study of a heterogeneous automotive application where ECUs use different queueing policies and have different buffer limitations, i.e., some have a very large number of transmit buffers, whereas, some have limited number of transmit buffers with some supporting transmission abort requests while others don't. In this application, we considered a large message set consisting of periodic, sporadic, and mixed messages. By evaluating the case study, we showed that 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.

The structural organization of the MPS-CAN analyzer provides ease for further extensions and implementations of other related analyses. Since, this tool is freely available, we believe, it may prove helpful in the research-oriented projects that require the analysis of CAN-based systems.

Acknowledgement

This work is supported by the Swedish Research Council (VR) within the projects SynthSoft and TiPCES, the Swedish Knowledge Foundation (KKS) within the projects FEMMVA and EEMDEF, and the Strategic Research Foundation (SSF) with the centre PROGRESS. The authors would like to thank the industrial partners Arcticus Systems, BAE Systems Hägglunds and Volvo Construction Equipment (VCE), Sweden.

References

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

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

[3] Automotive networks. CAN in Automation (CiA), 2011. http://www.can-cia.org/index.php?id=416.

[4] 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.

[5] 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.

[6] 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.

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

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

[9] AUTOSAR Techincal Overview, Version 2.2.2., Release 3.1, The AUTOSAR Consortium, Aug., 2008. Http://autosar.org.

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

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

[17] 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.

[18] Volcano Network Architect. Mentor Graphics, http://www.mentor.com/products/vnd/communication-management/vna.

[19] Rubus-ICE: Integrated component Development Environment, 2013. http://www.arcticus-systems.com.

[20] 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, ISSN: 1361-1384, 10 (2013).

[21] 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.

[22] 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.

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

[24] R. I. Davis, S. Kollmann, V. Pollex, F. Slomka, Controller Area Network (CAN) schedulability analysis with FIFO queues, in: 23rd Euromicro Conference on Real-Time Systems, July, 2011.

[25] Transmit cancellation in AUTOSAR Specification of CAN Driver, Release 4.0, Rev 3, Ver. 4.0. Nov., 2012. http://www.autosar.org/download/R4.0/AUTOSAR_SWS_CANDriver.pdf.

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

[27] 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.

[28] L. Du, G. Xu, Worst case response time analysis for CAN messages with offsets, in: IEEE International Conference on Vehicular Electronics and Safety (ICVES), Nov., 2009, pp. 41 –45.

[29] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, R. Ernst, System level performance analysis - the SymTA/S approach, Computers and Digital Techniques, 152 (2005) 148–166.

[30] 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.

[31] 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.

[32] 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.

[33] 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), June, 2012.

[34] 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.

[35] 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.

[36] 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.

[37] CANalyzer (2013). http://www.vector.com/vi_canalyzer_en.html.

[38] CANoe, accessed in Oct., 2013. http://www.vector.com/portal/medien/cmc/info/CANoe_ProductInfor-mation_EN.pdf.

[39] A. Hamann, R. Henia, R. Racu, M. Jersak, K. Richter, R. Ernst, Symta/s - symbolic timing analysis for systems (2004).

[40] RTaW-Sim (2013). http://www.realtimeatwork.com/software/rtaw-sim.

[41] K. Hänninen et.al., The Rubus Component Model for Resource Constrained Real-Time Systems, in: 3rd IEEE International Symposium on Industrial Embedded Systems.

[42] S. Mubeen, J. Mäki-Turja, M. Sjödin, Communications-Oriented Development of Component- Based Vehicular Distributed Real-Time Embedded Systems, Journal of Systems Architecture (2013).

[43] S. Mubeen, J. Mäki-Turja, M. Sjödin, Many-in-one Response-Time Analyzer for Controller Area Network, in: 4th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS).