EACAN: Reliable and Resource-Efficient CAN Communications

TAEJU PARK and KANG G. SHIN, University of Michigan, Ann Arbor

Worst-case-based timing verification for the controller area network (CAN) has been the bottleneck to efficient use of its bandwidth. Especially, this inefficiency comes from the worst-case transmission error rate (WCTER) when transmission errors are accounted for. To alleviate this inefficiency, we propose a runtime adaptation scheme, *error-adaptive* CAN (EACAN). EACAN observes the behavior of transmission errors at runtime, and reconfigures the message period based on the observation to meet the timing-failure requirement. We experimentally evaluate the bandwidth utilization of both EACAN- and WCTER-based verification, showing that the former improves the bandwidth utilization by 14% over the latter.

CCS Concepts: • **Computer systems organization** → **Embedded systems**; • **Networks** → Network reliability;

Additional Key Words and Phrases: Controller area network, mixed-criticality, in-vehicle network

ACM Reference format:
Taeju Park and Kang G. Shin. 2019. EACAN: Reliable and Resource-Efficient CAN Communications. *ACM Trans. Embed. Comput. Syst.* 18, 1, Article 8 (February 2019), 23 pages.
https://doi.org/10.1145/3301309

1 INTRODUCTION

More and more functions, such as advanced driving assistance system (ADAS), are being introduced to improve the driver's safety and comfort, and to reduce maintenance cost. The introduction of these new functions rapidly increases the bandwidth demand for in-vehicle communications [30], especially in the controller area network (CAN), which is the *de facto* standard of in-vehicle networks. To meet this increasing bandwidth demand, both the CAN data rate and the number of CAN buses within a vehicle have been increased [25], thus raising in-vehicle communication costs. So, achieving high efficiency of CAN bandwidth utilization has become important for cost-effective in-vehicle communications.

Timing verification for CAN is key in ensuring safety during the early design phases of a vehicle [22]. The timing verification used in COTS tools [31, 35] relies on the schedulability analysis based on the worst-case response time (WCRT) [9]. In particular, a probabilistic schedulability analysis based on the worst-case transmission error rate (WCTER) [2, 5, 6] is employed when a temporal requirement has to be verified while accounting for transmission errors. However, the worst-case-based timing verification for CAN results in severe under-utilization of bandwidth, because the

The work was supported in part by the US Office of Naval Research under Grants N00014-15-1-2163 and N00014-18-1-2141.
Authors' addresses: T. Park and K. G. Shin, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109-2121, USA; email: {taeju, kgshin}@umich.edu.

1539-9087/2019/02-ART8 $15.00
https://doi.org/10.1145/3301309

worst case requires too conservative a safety margin [27]. Besides, the under-utilization of CAN bandwidth will exacerbate even more as WCTER is expected to increase in future. For example, the rate of bit errors induced by electromagnetic interference (EMI), a major cause of bit errors in CAN [29], has been continuously increasing due to the changes in the external environment (5G networks using millimeter wave [15]) and internal vehicle systems (hybrid electrical vehicles and electrical vehicles [12], on-line electrical vehicles [8]).

To alleviate this problem, we propose a runtime adaptation, called *error-adaptive* CAN (EACAN). Instead of using WCTER, EACAN observes the behavior of transmission errors at runtime. Based on the observed behavior of recent past transmission errors, EACAN reconfigures the periods of low-criticality messages to guarantee the reliability (timing-failure) requirement to be met. As a result, we can remove the assumption used in the existing probabilistic schedulability analyses that the system is always exposed to the WCTER. There are two challenges in designing EACAN: determination of (1) when to adjust the message period to meet the given reliability requirement and to maximize the bandwidth usage and (2) how to make a quick adjustment of the message period.

To address the first challenge, EACAN measures the runtime transmission error rate (TER) based on the observed behavior of recent past transmission errors. Because the probability of deadline misses depends on the TER, EACAN determines *system criticality level* using the runtime TER. The thus-determined system criticality level adaptively controls the periods of given messages. To address the second challenge, we employ pre-defined thresholds in EACAN to make a quick decision on the system criticality level at runtime. The pre-defined thresholds are directly compared against the runtime TER instead of computing the probability of a deadline miss, which is computationally expensive. We formulate an optimization problem to find the thresholds that maximize the utilization of CAN bandwidth. We also provide a fast heuristic algorithm that yields a near-optimal solution. According to our evaluation result, EACAN improves bandwidth utilization by 14% over WCTER-based analyses without violating the reliability requirement.

The rest of this article is organized as follows. Section 2 discusses the existing timing analysis of CAN and an example of CAN messages with mixed-criticality. Section 3 states our target system, error, and mixed-criticality CAN message models, followed by the problem statement in Section 4. Section 5 details EACAN by describing how to measure the runtime TER, determine the system criticality level, and compute pre-defined thresholds and the overhead of changing system criticality level. We evaluate EACAN in comparison with the WCTER-based schedulability test in Section 6. Finally, we conclude the article in Section 7.

2 BACKGROUND

2.1 Message Transmission on CAN Bus

When an ECU transmits a CAN message on a CAN bus, the message is broadcast to all the ECUs connected to the CAN bus. Since multiple ECUs can try to transmit messages on the CAN bus at the same time, a decentralized message ordering mechanism is used in the CAN protocol according to the value of identifier (ID) field of CAN message. When multiple ECUs transmit CAN messages at the same time, the message with the lowest ID value is selected to be transmitted under the CAN protocol. For the transmitted message on the bus, each ECU decides to accept the message by comparing the value of ID of the message with the IDs registered in its receive filter.

2.2 Timing Analysis of CAN

Many applications that use CAN are time-critical, and hence it is important to determine, at design time, whether or not a CAN message can be delivered before its deadline. To meet this requirement, researchers have analyzed the worst-case response time (WCRT) of each CAN message.

The first timing analysis of a CAN message was done by Tindell et al. [33, 34]. They analyzed the WCRT, R_i, of a CAN message, m_i, by decomposing the response time into three components. The first component is release jitter (J_i), the maximum time necessary to queue the message in a transmission buffer (TxObject) of the CAN controller. The second component is queuing delay (w_i), which is the waiting time of the message in the TxObject before its transmission. The third component is the transmission time (C_i) on the CAN bus. Since the release jitter and the transmission time are both determined *a priori* by the message's priority, data length, the size of transmission buffer, and the CAN speed, Tindell et al. focused on analysis of the message's queuing delay. To derive the worst-case queuing delay of a message, they analyzed the message's critical instant. They recursively derived the worst-case queuing delay of a message as

$$w_i^{n+1} = B_i + \sum_{\forall k \in hp(i)} \left\lceil \frac{w_i^n + J_k + \tau}{T_k} \right\rceil C_k, \tag{1}$$

$$R_i = J_i + w_i + C_i, \tag{2}$$

where T_i is the period of the message, B_i is the blocking time by a lower-priority message, hp(i) is a set of the messages whose priority is higher than the message (m_i), and τ is a bit time.

However, this analysis has severe flaw, and hence Davis et al. [9] used a fine-grained approach to correct the nontrivial error in Equation (2). They computed the response time of every instance of the message and chose the maximum response time as the message's WCRT. The queuing delay of the qth instance of the message ($w_i(q)$) is defined as (q starts from 0)

$$w_i^{n+1}(q) = B_i + qC_i + \sum_{\forall k \in hp(i)} \left\lceil \frac{w_i^n(q) + J_k + \tau}{T_k} \right\rceil C_k, \tag{3}$$

and WCRT of the message is defined as

$$R_i(q) = J_i + w_i(q) - qT_i + C_i, \tag{4}$$

$$R_i = \max_q R_i(q). \tag{5}$$

Since the qth instance of the message is released at qT_i, qT_i is subtracted from the completion time of the qth instance to calculate the response time as shown in Equation (4).

These WCRT analyses have been extended to address various practical issues, such as the limited size of TxObject [18], FIFO queuing in the device driver [10], non-abortable TxObject [20], and non-negligible time for copying a message into TxObject [19]. In particular, several studies [2, 6] focused on the impact of transmission errors on the response time. They derived an equation to compute the worst-case queuing delay of a message with Z transmission errors:

$$w_{i|Z}^{n+1}(q) = B_i + qC_i + E_{i|Z} + \sum_{\forall k \in hp(i)} \left\lceil \frac{w_{i|Z}^n(q) + J_k + \tau}{T_k} \right\rceil C_k, \tag{6}$$

where $E_{i|Z}$ is the error recovery time (time for transmitting an error frame and time for retransmitting the message) for Z errors. Similar to Equations (4) and (5), WCRT with Z transmission errors is defined as

$$R_{i|Z}(q) = J_i + w_{i|Z}(q) - qT_i + C_i, \tag{7}$$

$$R_{i|Z} = \max_q \quad R_{i|Z}(q). \tag{8}$$

Since there is no way to predict the exact number of transmission errors that will occur in future, every CAN message is intrinsically unschedulable. Any schedulability test cannot guarantee the timing requirements to be met deterministically. Thus, previous studies [2, 6] have focused on probabilistic schedulability analyses, which compute the probability of deadline misses for a given set of CAN messages. To compute the probability of CAN message deadline misses, they first compute the probability of WCRT of the message that experiences Z transmission errors ($p(R_{i|Z})$) as

$$p(R_{i|Z}) = p(Z, R_{i|Z}) - \sum_{j=0}^{Z-1} p(R_{i|j}) p(Z-j, R_{i|Z} - R_{i|j}), \tag{9}$$

by assuming that the distribution of transmission errors follows a Poisson distribution with given TER (λ):

$$p(Z, R_{i|Z}) = \frac{e^{-\lambda R_{i|Z}} (\lambda R_{i|Z})^Z}{Z!}. \tag{10}$$

They then compute the probability of a message deadline miss by adding all the probabilities $p(R_{i|Z})$ such that $R_{i|Z} \leq D_i$ (D_i is the relative deadline of message i):

$$p_i(DM) = 1 - \sum_{\forall Z | R_{i|Z} \leq D_i} p(R_{i|Z}), \tag{11}$$

where DM stands for "deadline miss."

If the probability of missing the deadline of a given message is less than a pre-specified value, then the probabilistic schedulability test regards the message as schedulable.

2.3 Mixed-Criticality In-Vehicle Communications

To meet non-functional design goals, such as cost and weight, different criticality functions are forced to share a common hardware resource. In-vehicle communication follows the same practice. In a vehicle, multiple electronic control units (ECUs) share the same physical link (e.g., CAN bus or Ethernet link) to communicate with each other. The messages transmitted by the ECUs can be used by high- or low-criticality functions. Thus, the system designer can classify in-vehicle messages into multiple criticality levels according to their corresponding functions.[1]

Due to this resource sharing, low-criticality messages have to sacrifice their performance for high-criticality messages in an abnormal situation (e.g., TER exceeds the permitted error rate at runtime) to achieve a fail-safe operation.

3 SYSTEM MODEL AND ASSUMPTIONS

3.1 Overall Architecture

We consider a system composed of a single CAN bus and multiple devices/ECUs that share the CAN bus as shown in Figure 1. Applications running on each ECU initiate CAN messages periodically. The initiated messages are then copied into a TxObject. A message in the TxObject is broadcast over the CAN bus if the value in the ID value of the message is lower (higher priority) than that of any other queued messages.

We propose an error-adaptive CAN (EACAN), which is composed of master and slave components. The master component (mEACAN) is deployed on a monitoring ECU that has more computing power (higher performance CPU, larger memory size), like a vehicle domain controller [14]. The slave component (sEACAN) is deployed in all ECUs, except for the monitoring ECUs,

[1]The criticality level of a function can be determined according to the standard ISO26262.

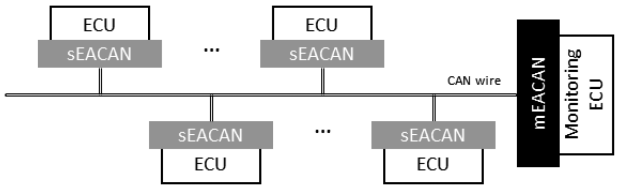

Fig. 1. Overall system architecture.

as shown in Figure 1. Whenever a transmission error occurs, mEACAN computes the runtime TER and determines the *system criticality level* (γ_{sys}), which starts from the lowest level, based on the runtime TER. If transmission errors occur more frequently than usual, then mEACAN raises the criticality level and broadcasts a special message to notify the raised criticality level to the sEACAN. Otherwise, the system criticality level stays at the low level.

3.2 Error Model

The electrical signal on the CAN bus can be temporarily distorted by EMI [29]. This distortion will, in turn, induce bit errors during the transmission of a CAN message. To cope with these transient errors, the CAN protocol comprises robust error detection mechanisms such as transmitter-based-monitoring, bit stuffing, cyclic redundancy check (CRC), and message format check [3]. The CAN protocol can detect the following transmission errors:

- *Bit error*: the value on the bus is not the same as that the transmitter sent (except during an arbitration phase);
- *Stuff error*: 6 same consecutive bits on the bus;
- *Form error*: invalid value shown in value-fixed bits (e.g., CRC delimiter, ACK delimiter, etc.);
- *ACK error*: no dominant value found in the ACK slot;
- *CRC error*: the received CRC is not the same as the computed value.

Upon detection of a transmission error, (1) an error frame is generated by the device that detected the error, (2) devices discard the erroneous message, and (3) the transmitter of the erroneous message automatically retransmits the message.

In this article, we only consider the detected transmission errors because the response time is increased only as a result of their detection. Even though multiple bit errors can occur within a single message transmission, we err on the side of safety by making a conservative assumption that every single bit error causes one transmission error for timing assurance. Thus, the bit error rate (BER) is the same as the transmission error rate. Moreover, we assume that every transmission error is detected by the underlying robust error detection mechanism. As in previous studies [2, 6], we assume that the distribution of bit errors follows a Poisson process and the WCTER, λ_{max}, is given, e.g., λ_{max} is determined and then specified during the design phase of a vehicle based on the knowledge of the worst environment/condition in which the vehicle must operate [26].

3.3 Mixed-Criticality CAN Message Model

According to ISO26262 [17], vehicular functions can be classified into multiple criticality levels, e.g., Automotive Safety Integrity Level (ASIL), and each function has a different reliability requirement according to its criticality level. For example, ISO26262 [17] specifies the requirement of failure rate caused by random hardware faults as shown in Table 1. Transmission errors can be regarded as random hardware faults, and also message deadline misses as timing failures. The timing failure of a CAN message, in turn, causes the execution failure of the associated functions

Table 1. Failure-rate Requirements Due to Random Hardware Faults in ISO26262

ASIL Level	Reliability Requirement
D	10^{-8}/h
C	10^{-7}/h
B	10^{-7}/h
A	10^{-6}/h

because the correct execution of the functions relies on the correct and timely delivery of input data. Thus, to meet the reliability requirement of a function, we must consider timely delivery of the corresponding CAN messages.

We propose a new mixed-criticality CAN message model based on the model in Reference [7]. In the model proposed in Reference [7], CAN messages have their own criticality levels and (multiple) periods, and the message period is altered when the system is in an abnormal state to ensure the timely delivery of high-criticality messages.

Our model contains an additional parameter—the probabilistic requirement of deadline misses. This requirement for a CAN message is derived from the reliability requirement of its associated function. In our model, low-criticality messages are transmitted less frequently at a higher system criticality level than at a lower system criticality level. As a result, the system criticality level has a direct impact on the CAN bandwidth utilization. Thus, the higher the system criticality level, the lower the bandwidth utilization of CAN.

We assume that the message parameters, such as periods, deadline, data length, and criticality, are defined *a priori* by the application programmers or vehicle system designers. Also, we assume that the given parameters satisfy the functional requirement (e.g., control system stability) of the corresponding functions. A mixed-criticality CAN message is defined as $m_i = \{\chi_i, \vec{T_i}, J_i, L_i, \vec{D_i}, \vec{\epsilon_i}\}$ where

- $\chi_i \in \{1, \ldots, L\}$: criticality; Criticality is mapped to an ASIL, e.g. for a 2-level system, criticality 1(2) is mapped to ASIL A(D);
- $\vec{T_i}$: periods (function of the system criticality level), $T_i(1) = \cdots = T_i(\chi_i) \leq \cdots \leq T_i(L)$;
- J_i: release jitter;
- L_i: data length. Transmission time (C_i) of the message is proportional to the data length;
- $\vec{D_i}$: relative deadline (function of the system criticality level). Assume $D_i(l) \leq T_i(l)$;
- $\vec{\epsilon_i}$: requirement of probability of deadline miss (function of the system criticality level).

In practice, tasks running on ECUs, or CAN messages can be time- or event-triggered, e.g., a user input or a specific vehicle condition [21]. However, it is difficult to predict the initiation of event-triggered messages at runtime, the event-triggered messages are regarded as sporadic messages with the minimum inter-arrival time in the timing verification process. The minimum inter-arrival time is treated as the period in our message model.

In addition, in our model, the period (or the minimum inter-arrival time) of CAN messages are altered according to the system criticality level. However, because the performance of applications (usually control tasks) running on ECUs is affected greatly by their periods [32], the periods adaptation according to the system criticality level could degrade the app functionality. Thus, the system designer should carefully determine the allowable (elastic) range of period and adapt the period within the allowable range.

3.3.1 Deriving the Requirement of Probability of Deadline Misses. We derive the requirement of probability of deadline miss of each message from the given reliability requirement in Table 1. Suppose reliability requirement of a message (corresponding function) is $RR(\chi_i)$, and its period is $T_i(l)$ at the system criticality level l. Also, suppose the probability of deadline miss of the message is $p_i(DM|\gamma_{sys} = l)$ at the system criticality level l.

If the message is transmitted $\frac{1}{p_i(DM|\gamma_{sys}=l)}$ times, then there will be one timing failure in average. Because the message is transmitted $\frac{1\text{h}}{T_i(l)}$ times in 1h, $\frac{1\text{h}}{T_i(l)} \times p_i(DM|\gamma_{sys} = l)$ timing failures occur on average in 1h. To meet the reliability requirement, $\frac{1\text{h}}{T_i(l)} \times p_i(DM|\gamma_{sys} = l) \leq RR(\chi_i)$. Then, we can derive

$$\frac{1\text{h}}{T_i(l)} \times p_i(DM|\gamma_{sys} = l) \leq RR(\chi_i) \times 1\text{h} \Rightarrow \frac{1}{T_i(l)} \times p_i(DM|\gamma_{sys} = l) \leq RR(\chi_i).$$

Thus, we can define the requirement of probability of message deadline misses at system criticality level l as

$$\epsilon_i(l) = RR(\chi_i) \times T_i(l).$$

Definition 3.1 (Mixed-Criticality CAN Message Set Probabilistic Schedulability). For a given mixed-criticality CAN message set, if $\forall l\, p_i(DM|\gamma_{sys} = l) \leq \epsilon_i(l)$ holds where $\chi_i \geq l$, then the given mixed-criticality message set is schedulable.

4 PROBLEM STATEMENT

Timing verification for CAN communications is key in ensuring vehicle safety during the early design phases of a vehicle. However, the WCRT-based pessimistic timing verification for CAN has been the bottleneck to its bandwidth usage efficiency. The bandwidth under-utilization due to the WCTER-based probabilistic schedulability analysis is expected to become even worse in future because EMI-induced bit errors are continuously increasing. To alleviate this problem, we propose EACAN with the following goals:

- **G1:** Ensure $p_i(DM|\gamma_{sys} = l) \leq \epsilon_i(l)$ where $\chi_i \geq l$ if $l \neq L$ where L is the highest system criticality level;
- **G2:** Maximize the bandwidth usage for a given mixed-criticality message set.

Even though EACAN achieves **G1**, ensuring $P_i(DM|\gamma_{sys} = L) \leq \epsilon_i(L)$ for the highest criticality messages is still needed offline. Thus, we propose a probabilistic schedulability test that fully exploits the characteristics of EACAN.

5 ERROR-ADAPTIVE CAN (EACAN)

5.1 Overview

5.1.1 Basic Idea. Our basic idea is to adapt the periods of low-criticality messages to the behavior of recent past transmission errors. To achieve that, mEACAN observes the behavior of recent past transmission errors, and measure runtime TER. If the runtime TER exceeds the pre-defined threshold that is embedded in EACAN, then EACAN changes the system criticality level and thus adaptively controls the periods to guarantee the satisfaction of the requirement of probability of message deadline misses. The challenges in realizing this idea are to determine when to reconfigure the system (when to change the system criticality level) and how to make such a decision and reconfigure the system quickly.

5.1.2 Workflow of EACAN. The workflow of EACAN is illustrated in Figure 2. Whenever a transmission error occurs, an interrupt is generated to handle it. The interrupt-handling routine

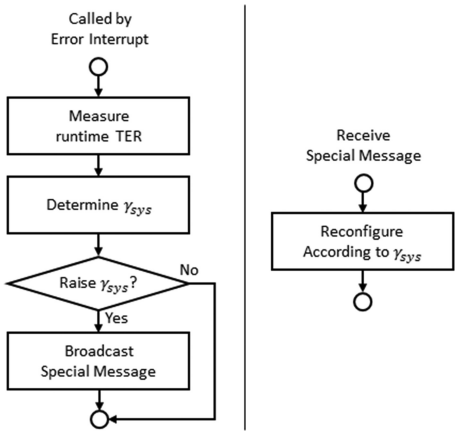

Fig. 2. Flow chart of (Left) mEACAN (Right) sEACAN.

calls the functions of mEACAN. First, mEACAN computes the runtime TER based on the behavior of recent past transmission errors. It then determines the system criticality level (γ_{sys}) for use in the immediate future. If the determined system criticality level is higher than the current system criticality, then mEACAN broadcasts a special CAN message to notify the change of system criticality level to other ECUs (sEACANs). Upon receiving this special CAN message, sEACANs reconfigure their message set according to the system criticality level. To guarantee all or no node to receive the special CAN message, every ECU connected to the CAN bus should accept the special message by registering the ID of the special CAN message.[2]

Also, when the CAN bus becomes idle, mEACAN re-initializes the runtime TER to 0 and the system criticality level to the lowest level (see Section 5.2.3).

5.2 Runtime TER

To measure the runtime TER, mEACAN needs to know, at runtime, when the transmission errors occurred. Fortunately, mEACAN can easily obtain this information because the commercial CAN controller [24] generates an interrupt to handle each transmission error.

5.2.1 Requirement of Runtime TER. As can be seen from Equation (9), computing the probability of deadline misses requires the transmission error rate. We will use *runtime TER* instead of WCTER to compute the probability of deadline misses. To achieve **G1** (Requirement), the runtime TER must be larger than the TER that a message actually experiences.

5.2.2 Definition of Runtime TER. Let us consider the CAN bandwidth usage during $[t_s,\ t_e)$. Suppose a transmission error occurs at time t_c such that $t_s \leq t_c < t_e$. Then, mEACAN computes the runtime TER (λ_{run}) and determines the system criticality level at time t_c as described in the workflow.

At time t_c, we want to know whether or not the probability of missing a message's deadline will be lower than its requirement during $[t_c,\ t_e)$. However, the behavior of transmission errors in $[t_c,\ t_e)$ is unpredictable at time t_c. That is, it is impossible to know the TER that a message actually experiences, and is thus difficult to determine the value of TER at runtime to meet the runtime TER requirement.

[2]The value, 0x1, is used as the ID of the special message in our experiments.

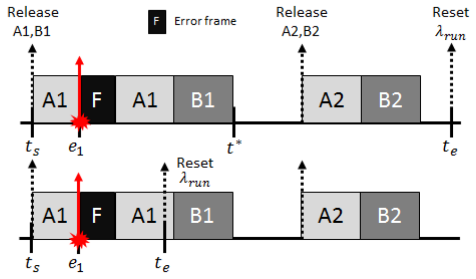

Fig. 3. (Top) Time interval of interest $[t_s, t_e)$ is unnecessarily long. (Bottom) Time interval of interest is $[t_s,\ t_e)$ is too short.

To overcome this difficulty, we assume that inter-arrival times of transmission errors in the near future $[t_c,\ t_e)$ are greater than the minimum inter-arrival time ($\check{\xi}_{[t_s,t_c]}$) of transmission errors occurred in the recent past $[t_s,\ t_c]$. Under this assumption, we can use the inverse of the minimum inter-arrival time as the value of runtime TER because the TER that a message actually experiences must be lower than the inverse.

$$\lambda_{run} = \frac{1}{\check{\xi}_{[t_s,t_c]}}, \tag{12}$$

$$\check{\xi}_{[t_s,t_c]} = \min_i (e_i - e_{i-1}), \tag{13}$$

where e_i is the arrival time of the ith transmission error in $[t_s, t_e)$, $e_0 = t_s$ and $i \in \mathbb{N}^+$.

However, the above assumption may not hold at runtime. For example, at a certain time (t_f such that $t_c < t_f < t_e$), a new transmission error can yield a smaller inter-arrival time than the inverse of the runtime TER, computed at time t_c. At that time (t_f), the probability of missing message deadlines computed with the runtime TER becomes useless. Thus, the probability of missing message deadlines must be re-computed with the inverse of the new minimum inter-arrival time to meet our first goal (**G1**). So, whenever a new transmission error occurs, EACAN updates the runtime TER, accounting for the effect of the new transmission error.

Our upper bound—the inverse of the minimum inter-arrival times—could be larger than the true upper bound (λ_{max}) due to short-time burst errors. We increase the system criticality level to the highest level L to cope with this urgent case.

5.2.3 Deciding on the Time Interval of Interest. When computing the runtime TER, we must carefully determine the time interval of interest $[t_s, t_e)$. If the interval $[t_s, t_e)$ is too long, then a transmission error may negatively and unnecessarily affect the probability of deadline misses. For example, as shown in Figure 3 (Top), the transmission error occurring at time e_1 affects the probability for the messages $A2$ and $B2$ unnecessarily even though the transmission error does not influence the response time of the messages. This is because the increased runtime TER at time e_1 does not decrease until the end of time interval of interest t_e.

However, if $[t_s, t_e)$ is too short, EACAN may fail to achieve **G1**. For example, as shown in Figure 3 (Bottom), the runtime TER is re-initialized to 0 at t_e because it reaches the end of time interval of interest. However, the transmission error occurred at time e_1 increases the response time of the message $B1$. It means that EACAN ignores the impact of the transmission error, and thus EACAN cannot ensure the probability of deadline miss of $B1$ to be smaller than its requirement.

To configure $[t_s, t_e)$ properly, we define t_s as the starting point of a busy period and t_e as the closest bus idle instant after e_z, where e_z is the arrival time of the latest transmission error after t_s. For example, as shown in Figure 3 (Top), the time t^* becomes t_e because t^* is the closest bus idle

instant after e_1, which is the arrival time of the latest transmission error after t_s. In other words, a time interval of interest $[t_s, t_e)$ is the same as a busy period. By design, there will not be any message whose response time is lengthened by the latest transmission error after t_e, and thus the defined time interval of interest $[t_s, t_e)$ is not too short. Also, $[t_s, t_e)$ is not too long because the closest bus idle instant after e_z is the minimum possible value for t_e. If t_e is smaller than the closest bus idle time after e_z, then a message may be delayed by the latest transmission error after t_e. Thus, $[t_s, t_e)$ becomes too short. At time t_e, the runtime TER and the system criticality level are reset to their initial values.

The pseudocode for computing runtime TER is provided in Algorithm 1. mEACAN executes Algorithm 1 whenever a transmission error occurs. As stated in Line 1, we can compute the minimum inter-arrival time of transmission errors in $[t_s, t_c]$ by comparing the previous minimum inter-arrival time of transmission errors and inter-arrival time of two most recent transmission errors. Thus, we can measure the runtime TER with a small computation time overhead. We omit the description of how to reset the system criticality level and the runtime TER, since it is trivial.

5.3 Deciding on System Criticality Level

The periods of CAN messages depend on the system criticality level, and thus changing the system criticality level significantly affects the bandwidth utilization of CAN. Optimizing the instant of changing the system criticality level is, therefore, important to achieve **G**2.

We first seek a condition to decide on the system criticality level (γ_{sys}) and then derive a TER threshold from the condition. The TER threshold is directly compared against the runtime TER to determine the system criticality level quickly at runtime.

5.3.1 Decision Based on the Probability of Deadline Misses. During the mission, the system criticality level must satisfy the following condition to achieve **G1**:

$$\chi_i \geq l \wedge p_i(DM|\gamma_{sys} = l, \lambda = \lambda_{run}) > \epsilon_i(l) \Rightarrow \gamma_{sys}^{[t_c, t_e)} > l,$$

where $\gamma_{sys}^{[t_c, t_e)}$ is the system criticality level in the future time interval $[t_c, t_e)$.

If the probability of deadline miss of message (m_i) at the system criticality level l is greater than its requirement, then the system criticality level should be greater than l in the future time interval $[t_c, t_e)$. Otherwise, the requirement will not be met in the future time interval. However,

ALGORITHM 1: Computing runtime TER

Input: $\xi_{[t_s,t_c]}$: previous minimum inter-arrival time of errors
e_{prev}: arrival time of the previous error
e_{cur}: arrival time of current error
Output: λ_{run}: the updated runtime TER

1 **if** $\xi_{[t_s,t_c]} = 0$ OR $\xi_{[t_s,t_c]} > e_{cur} - e_{prev}$ **then**
2 $\quad \xi_{[t_s,t_c]} = e_{cur} - e_{prev}$;
3 $\quad \lambda_{run} = \frac{1}{\xi_{[t_s,t_c]}}$;
4 **end**
5 **if** $\lambda_{run} > \lambda_{max}$ **then**
6 $\quad \lambda_{run} = \lambda_{max}$;
7 **end**
8 **return** λ_{run};

computing the probability of deadline misses for all the messages and all the system criticality levels incurs a significant computation overhead, thus making it impractical.

5.3.2 Decision Based on the Runtime TER. Instead of computing the probability of deadline miss at runtime, we define a proxy task: we compare the runtime TER against the pre-defined thresholds. Since the probability of deadline miss relies on the TER as stated in Equations (9), (10), and (11), the system criticality level has to be higher than l in the future time interval if the runtime TER exceeds the embedded threshold (θ_i^l) such that $p_i(DM|\gamma_{sys} = l, \lambda = \theta_i^l) = \epsilon_i(l)$. Thus, we can derive the following condition that the system criticality level must satisfy:

$$\exists i, \chi_i \geq l \wedge \theta_i^l < \lambda_{run} \Rightarrow \gamma_{sys}^{[t_c, te)} > l.$$

To find this threshold, we formulate the optimization problem as

$$\begin{aligned} \theta_i^l = \quad & \underset{\theta}{\text{argmax}} \quad p_i(DM|\gamma_{sys} = l; \theta), \\ & \text{subject to} \quad p_i(DM|\gamma_{sys} = l; \theta) \leq \epsilon_i(l). \end{aligned} \tag{14}$$

Since the objective of optimization is the largest possible argmax, the system criticality level can stay at the lower level as long as possible. As a result, EACAN maximizes the bandwidth utilization of CAN.

5.4 Solving the Optimization Problem

Described below is how to solve the proposed optimization problem. We first address how to compute the optimization objective function and then present an efficient heuristic algorithm that yields a near-optimal solution.

5.4.1 Computing the Objective Function. We compute the objective function, $p_i(DM|\gamma_{sys} = l)$, using the following three steps.

Step 1: Compute $R_{i|Z}^l$, the upper bound of response time of message m_i with Z transmission errors at the system criticality level l. We can easily derive the worst-case queuing delay for message m_i with Z transmission errors at system criticality level l using Equation (6), because only the period and the deadline depend on the system criticality level:

$$\begin{aligned} w_{i|Z}^{n+1}(q, l) = & B_i + qC_i + E_{i|Z} \\ & + \sum_{\forall k \in hp(i)} \left\lceil \frac{w_{i|Z}^n(q, l) + J_k + \tau}{T_k(l)} \right\rceil C_k \\ & + O_{chg}, \end{aligned} \tag{15}$$

where O_{chg} is the time overhead of changing the system criticality level, which will be detailed later. $R_{i|Z}^l$ can then be defined as

$$R_{i|Z}^l(q) = J_i + w_{i|Z}(q, l) - qT_i(l) + C_i, \tag{16}$$

$$R_{i|Z}^l = \max_q R_{i|Z}^l(q). \tag{17}$$

Step 2: Compute $p_i(R_{i|Z}^l)$, the probability of $R_{i|Z}^l$. We derive $p_i(R_{i|Z}^l)$ from Equation (9) by replacing $R_{i|Z}$ with $R_{i|Z}^l$:

$$p\left(R_{i|Z}^l\right) = p\left(Z, R_{i|Z}^l\right) - \sum_{j=0}^{Z-1} p\left(R_{i|j}^l\right) p\left(Z - j, R_{i|Z}^l - R_{i|j}^l\right). \tag{18}$$

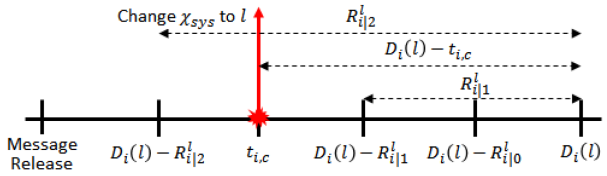

Fig. 4. The probability of missing a message's deadline depends on the remaining execution/transmission time after changing the system criticality level.

Step 3: Compute $p_i(DM|\gamma_{sys} = l)$. Suppose the system criticality level is raised from a lower level to l at time $t_{i,c}$, where $t_{i,c}$ is the time between the release of m_i and the current time t_c. Also, suppose message m_i is released before the system criticality level is raised. Then, the remaining (execution) time of the message is $D_i(l) - t_{i,c}$ after increasing the system criticality level. We want to show that $p_i(DM|\gamma_{sys} = l)$ depends on the remaining time. For example, if the remaining time satisfies the inequality $R^l_{i|1} \le D_i(l) - t_{i,c} \le R^l_{i|2}$, as illustrated in Figure 4 (according to Reference [6], the response time with two errors is always larger than that with 1 error), then two or more errors are not allowed after $t_{i,c}$ to meet its deadline. Thus, the probability of missing the message deadline at system criticality level l is $1 - p(R^l_{i|0}) - p(R^l_{i|1})$. Likewise, if the remaining time satisfies the inequality $R^l_{i|2} \le D_i(l) - t_{i,c} \le R^l_{i|3}$, then the probability of missing the message deadline at criticality level l is $1 - p(R^l_{i|0}) - p(R^l_{i|1}) - p(R^l_{i|2})$. Hence, we can define the probability of missing the message deadline at system criticality level l as

$$p_i(DM|\gamma_{sys} = l) = 1 - \sum_{Z=0}^{Z_m} p\left(R^l_{i|Z}\right), \tag{19}$$

where Z_m is an integer such that $R^l_{i|Z_m} \le D_i(l) - t_{i,c} \le R^l_{i|Z_m+1}$. If a message is released after raising the system criticality level to l, then $t_{i,c}$ is 0.

5.4.2 Finding a Near-Optimal Solution. Since the objective function of the optimization problem is computed recursively, it is difficult to solve the problem directly. So, we find an alternative, near-optimal solution by using a binary search. Since the objective function is an increasing function of λ, the optimal solution (θ^o) exists between a lower bound ($\theta^{l,LB}_i$) and an upper bound ($\theta^{l,LB}_i$) such that

$$p_i\left(DM|\gamma_{sys} = l, \lambda = \theta^{l,LB}_i\right) \le \epsilon_i(l),$$
$$p_i\left(DM|\gamma_{sys} = l, \lambda = \theta^{l,UB}_i\right) > \epsilon_i(l).$$

Also, we can easily find lower and upper bound candidates, e.g., selecting $0/ms$ and a large number (e.g., 1000/ms), respectively. We can thus gradually approach the optimal solution from these bounds using a binary search. Even though we cannot guarantee the finding of the optimal solution, we can find a near-optimal solution (θ^{no}) such that $\theta^o - \theta^{no} < \delta$ where δ is a suitably small number.

As mentioned before, because $p_i(DM|\gamma_{sys} = l)$ depends on the remaining execution time at $\gamma_{sys} = l$, the thresholds θ^l_i also depend on the remaining time. Thus, we need to find all $\theta^l_{i|k}$ thresholds for message i, criticality l and the remaining time $D_i(l) - t_{i,c}$ such that $R^l_{i|k} \le D_i(l) - t_{i,c} \le R^l_{i|k+1}$. The pre-defined thresholds $\theta^l_{i|k}$ are computed offline and saved in a three-dimensional array (Θ), which is then embedded in mEACAN. So, there is no need to consider the runtime overhead for solving the optimization problem. Note that space complexity of Θ is $O(N * L * K_{max})$ (see Section 5.6 for detailed information of N, L, and K_{max}).

ALGORITHM 2: Determining System Criticality Level

Input: N: the number of messages
L: the number of criticality levels
K_{max}: the maximum number of errors within a lifetime of a message
Θ: an array contains thresholds of TER
R: an array contains the upper bound of response times
λ_{run}: the runtime TER over $[t_s, t_e]$
γ_{sys}: current system criticality level
t_c: current time
Output: $\chi_{sys}^{[t_c,t_e]}$: determined system criticality level

1 $\chi_{sys}^{[t_c,t_e]} \leftarrow \gamma_{sys}$;
2 MsgFlag [] $\leftarrow$ false;
3 **for** $i \leftarrow 1$ **to** N **do**
4 **if** $\chi_i < \gamma_{sys}$ **then**
5 Continue;
6 **end**
7 **if** $T_i(\gamma_{sys}) < (t_c - base_i^{\gamma_{sys}})$ **then**
8 $t_{i,c} \leftarrow 0$;
9 **end**
10 **else**
11 $t_{i,c} \leftarrow (t_c - base_i^{\gamma_{sys}}) \bmod T_i(\gamma_{sys})$;
12 **end**
13 **for** $l \leftarrow \chi_{sys}^{[t_c,t_e]}$ **to** L **do**
14 **for** $k \leftarrow 1$ **to** K_{max} **do**
15 **if** MsgFlag $[i]$ = *false* **then**
16 **if** $R_{i|k}^l \le D_i(l) - t_{i,c} \le R_{i|k+1}^l$ **then**
17 **if** $\theta_{i|k}^l > \lambda_{run}$ *and* $\theta_{i|k}^{l+1} \le \lambda_{run}$ **then**
18 $\chi_{sys}^{[t_c,t_e]} \leftarrow l + 1$;
19 MsgFlag $[i] \leftarrow$ true;
20 **end**
21 **end**
22 **end**
23 **end**
24 **end**
25 **end**
26 **if** $\gamma_{sys} < \chi_{sys}^{[t_c,t_e]}$ **then**
27 **for** $i \leftarrow 1$ **to** N **do**
28 $base_i^{\chi_{sys}^{[t_c,t_e]}} \leftarrow t_c + (T_i(\chi_{sys}^{[t_c,t_e]}) - t_{i,c})$
29 **end**
30 **end**
31 **return** $\chi_{sys}^{[t_c,t_e]}$

5.5 Runtime Decision on System Criticality Level

Algorithm 2 describes how to determine the system criticality level at runtime for the future time interval. In line 1, the system criticality level for the future time interval is initialized with the current system criticality level to maximize CAN bandwidth utilization. Algorithm 2 then tries

to find the lowest possible system criticality level by comparing the runtime TER with the predefined thresholds. As stated in lines 17 and 18, if the runtime TER is larger than $\theta^{l}_{i|k}$ and smaller than $\theta^{l+1}_{i|k}$, the system criticality level for the future time interval will be switched to $l+1$.

The lines between 7 and 12 state how to compute $t_{i,c}$ and the lines between 26 and 30 states how to compute $base^{l}_{i}$, which is needed to compute $t_{i,c}$. $base^{l}_{i}$ is the first release time of m_i after changing the system criticality level to l. Thus, the value of $base^{l}_{i}$ is assigned only when the system criticality level is raised to l as stated in line 24.

If the determined system criticality level is larger than the current system criticality level, then mEACAN broadcasts a special CAN message to change the system criticality level. Then, sEACANs reconfigure their message set according to the determined system criticality level.

5.6 EACAN Schedulability Analysis

The inequality $p_i(DM|\gamma_{sys} = l) \leq \epsilon_i(l)$ holds where $l < L$ (the highest level), because EACAN automatically raises the system criticality level if it doesn't hold. But, we still need timing verification for the system criticality level L. To analyze the schedulability at the highest level L, we need to analyze the worst-case response time of the highest-criticality messages when the messages are delivered at the highest level L.

When the delivery of a highest criticality message is completed at the highest level L, in terms of response time, the worst case occurs when the system criticality level is changed directly from 1 to L, and the message stays at the lowest level ($\gamma_{sys} = 1$) as long as possible. This is because messages' periods are the smallest at the lowest level, and thus the interference by higher-priority messages is the greatest at the lowest level.

We first analyze the worst-case busy period of a highest-criticality message when its transmission is completed at the highest level L. We assume that the system criticality level is changed from 1 to L at time t_{chg}. Because the periods of higher priority messages are changed after t_{chg}, the last term (the interference by higher-priority messages) in Equation (6) should be separated out, and also the overhead of changing the criticality level should be accounted for as

$$w^{n+1}_{i|Z,[1.L]}(q) = B_i + qC_i + E_{i|Z} + O_{chg} + I_{BC}(t_{chg}) + I_{AC}\left(w^{n}_{i|Z,[1.L]}(q) - t_{chg}\right), \tag{20}$$

where $w^{n+1}_{i|Z,[1.L]}(q)$ is the busy period of the qth instance of message i with Z transmission errors when the system criticality level is changed directly from 1 to L at t_{chg} and the message transmission is completed at the highest level L. I_{BC} is the interference by higher-priority messages before changing the system criticality level, and I_{AC} is the interference by higher-priority messages after changing the system criticality level. We can easily compute I_{BC} and I_{AC} as

$$I_{BC}(t_{chg}) = \sum_{\forall k \in hp(i)} \left\lceil \frac{t_{chg} + J_k + \tau}{T_k(1)} \right\rceil C_k, \tag{21}$$

$$I_{AC}\left(w^{n}_{i|Z,[1.L]}(q) - t_{chg}\right) = \sum_{\forall k \in hp(i)} \left(\left\lceil \frac{w^{n}_{i|Z,[1.L]}(q) - t_{chg} + J_k + \tau}{T_k(L)} \right\rceil - 1 \right) C_k. \tag{22}$$

In the equation of I_{AC}, there is "-1" term, because due to the ceiling function, one frame instance can be counted twice in both I_{BC} and I_{AC}. For example, suppose that the period of a higher-priority frame is 3 at the system criticality level 1 and 6 at the system criticality level L. Also, suppose that $t_{chg} = 10$ and transmission of the frame instance starts at time 20. Then, the higher-priority frame is counted 4 times by the ceiling function in Equation (21), and also counted 2 times in Equation (22). However, the higher-priority frame is released only 5 times within 20 time units (0, 3, 6, 9, 15). Thus, this double counting should be figured out. Note that when the remainder of division

in the ceiling function is 0, we do not apply the "−1" term, because there is no double counting. For example, suppose that the period of a higher priority frame is 2 at the system criticality level 1, and 5 at the system criticality level L. Then, the frame is counted 5 times by the ceiling function in Equation (21), and also counted 2 times in Equation (22). In this case, the frame is released 7 times (0, 2, 4, 6, 8, 13, 18), which is the same as the number we counted (5 from I_{BC}, 2 from I_{AC}).

Because when to change the system criticality level to L is unknown beforehand, it is challenging to bound the maximum possible duration (the value of t_{chg}) at the lowest level 1. To deal with this difficulty, we utilize the property of EACAN: the system criticality level changes to the highest level L if $\lambda_{run} \geq \lambda_{max}$.

Lemma 5.1. $\lambda_{run} \geq \frac{Z}{w_{i|Z}(q,1)}$, *where* $w_{i|Z}(q,1)$ *is the busy period of message* i *with* Z *transmission errors when the system only stays at the lowest level* 1.

Proof. Suppose n transmission errors occur during time duration t. λ_{run} is then minimized when all the intervals between any two consecutive errors are the same. So, $\lambda_{run} = \frac{n}{t}$. $w_{i|Z}(q,1)$ is the busy period for the qth instance of message i, and $w_{i|Z}(q,1)$ inherently includes Z transmission errors. In such a case, the minimum possible value of λ_{run} is $\frac{Z}{w_{i|Z}(q,1)}$. Thus, $\lambda_{run} \geq \frac{Z}{w_{i|Z}(q,1)}$. □

By Lemma 5.1 and the property of EACAN, we can derive the following proposition:

$$\frac{Z}{w_{i|Z}(q,1)} \geq \lambda_{max} \Rightarrow \gamma_{sys}^{[w_{i|Z}(q,1),t_e)} = L.$$

From this proposition, we know that the system criticality level changes to L before $w_{i|Z}(q,1)$ such that $\frac{Z}{w_{i|Z}(q,1)} \geq \lambda_{max}$. To find such Z, we need to solve the following optimization problem:

$$Z_c = \underset{Z}{\operatorname{argmin}} \left[\frac{Z}{w_{i|Z}(q,1)} \geq \lambda_{max} \right]. \tag{23}$$

Since Equation (15) can be used to compute $w_{i|Z}(q,1)$, the optimization problem can be solved easily, and $w_{i|Z_c}(q,1)$ can be used as t_{chg} for the computation of Equation (20). With the result of Equation (20), we can compute the worst-case response time of the highest-criticality messages when its transmission is completed at the highest level L as

$$R_{i|Z,[1,L]}(q) = J_i + w_{i|Z,[1,L]}(q) - qT_i(L) + C_i, \tag{24}$$

$$R_{i|Z,[1,L]} = \max_q \quad R_{i|Z,[1,L]}(q). \tag{25}$$

Since the periods of highest-criticality messages are the same for all system criticality levels, we can use $qT_i(L)$ as the release time of the qth instance.

Also, the probability of missing message deadlines can be computed and the probabilistic schedulability can be tested for the system criticality level L as

$$p\left(R_{i|Z,[1,L]}\right) = p\left(Z, R_{i|Z,[1,L]}\right) - \sum_{j=0}^{Z-1} p\left(R_{i|j,[1,L]}\right) p\left(Z-j, R_{i|Z,[1,L]} - R_{i|j,[1,L]}\right), \tag{26}$$

$$p_i(DM) = 1 - \sum_{\forall Z | R_{i|Z,[1,L]} \leq D_i} p\left(R_{i|Z,[1,L]}\right). \tag{27}$$

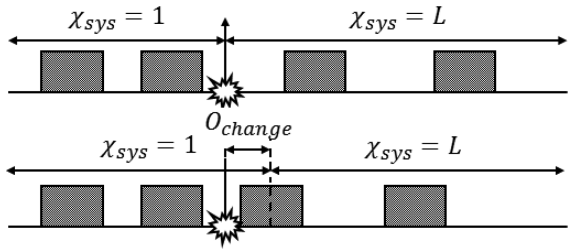

Fig. 5. (Top) Ideal change of system criticality level. (Bottom) Changing system criticality level with overhead considered.

5.7 Analysis of Overhead of Changing γ_{sys}

Suppose a transmission error occurs at time t_c and the system criticality level is raised due to the transmission error. Ideally, we expect that the raise of system criticality level is synchronized with the arrival of the transmission error as shown in Figure 5 (Top). However, this ideal synchronization is infeasible, because a reconfiguration based on the determined system criticality level requires time (O_{change}). The system criticality level is raised from a lower level to a higher level after consuming that amount of time as shown in Figure 5 (Bottom). Thus, the period change is also delayed by that amount of time. This is the reason for considering the time overhead in the analysis of the worst-case queuing delay in Equation (16).

The time overhead of changing the system criticality level consists of two parts: the time for determining the system criticality level (computation time) and the time for notifying the new system criticality level to sEACAN (communication time). The period adjustment on sEACAN also requires a small amount of time, and it is simple to change the period, and hence its analysis is omitted.

Computation Time Analysis. We analyze the time complexity of Algorithms 1 and 2 (in the Appendix) as they represent the core functions of determining the system criticality level. Algorithm 2 contains only several comparisons and statements, and its time complexity is $O(1)$. In Algorithm 1, Lines 6–13 are repeated at most $N * L * K_{max}$ times, and hence its time complexity is $O(N * L * K_{max})$.

The number (N) of messages is typically less than 100 on a single CAN bus in contemporary vehicles and the criticality is typically divided into four or five levels. Also, Ferreira et al. [13] reported that less than 1,000 bit errors per hour take place in a CAN bus 2m away from a welding machine. Thus, there might be a few transmission errors within a few seconds (a typical maximum lifetime of a message). Formally, we can obtain

$$K_{max} = \underset{n}{\text{argmax}}\ \exists i\ p_i(n, D_i(L) | \lambda = \lambda_{max}) \leq RR(m_i).$$

Communication Time Analysis. If mEACAN decides to raise the system criticality level, then it broadcasts a special CAN message to notify the change of criticality level to other ECUs (sEACANs). We use a unique identifier for the special CAN message and assign it the lowest ID value (e.g., 0x1) or the highest priority. Therefore, the special CAN message can only be blocked by one lower-priority message already being transmitted on the bus. Also, the special CAN message only contains the information of criticality level. Since there are typically four or five criticality levels, we can assume that the special CAN message embeds at most 1-byte data. As a result, the worst-case communication time of this message is the transmission time of a CAN message with 8-byte data (a lower-priority message) plus the transmission time of a CAN message with 1-byte data (the special CAN message).

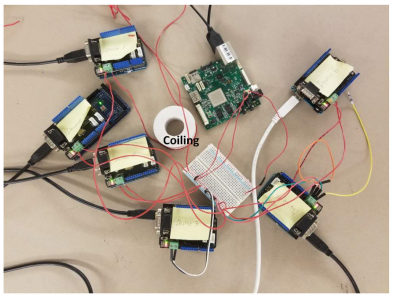

Fig. 6. Experimental platform.

6 EVALUATION

We have evaluated EACAN in comparison with existing WCTER-based approaches [2, 6]. Our evaluation focuses on measuring the utilization of CAN bandwidth. To show the realism of EACAN, we have conducted simple experiments on our testing platform. To show the utility of EACAN for a wide range of scenarios, we have also conducted simulations while varying mutable parameters, such as λ_{max}, maximum criticality (L), and utilization of a given message set.

6.1 Experimentation

6.1.1 Experimental Platform. We have built an experimental platform to evaluate EACAN as shown in Figure 6. The experimental platform consists of one imx6 sabrelite[3] and six Arduino [1] boards with MCP2515 CAN controller [24]. The boards are all connected through a CAN bus. We use the imx6 sabrelite board as a domain controller, and thus mEACAN is deployed on the imx6 sabrelite board and sEACAN is deployed on the Arduino boards.

By winding the CAN wire as shown in Figure 6, we can control the strength of EMI. Thus, we can control the transmission error rate by increasing and decreasing the number of coil turns.

6.1.2 Benchmark and Configuration. We slightly modified the SAE benchmark [16] as shown in Table 2, and then used it as the set of CAN messages in our experiments. We modified the SAE benchmark to include two criticality levels. To introduce the criticality characteristics, we assign criticality-level 2 to the messages whose period is 5ms, and assign criticality-level 1 to the others. This is because safety-critical (e.g., control) messages usually have short periods. At criticality-level 2, the modified SAE benchmark is identical to the original SAE benchmark.

The CAN bus speed was set to 250Kbps, a widely-used speed of current in-vehicle network communications, such as body and powertrain control networks. For the binary search, we set $\delta = 10^{-15}$. We control the number of coil turns to set $\lambda_{max} = 10^{-3}$/ms, since the TER measured in an aggressive environment (near a welding machine) is about 10^{-4}/ms [13]. The TER for 1h experiments was observed to be about. 8.0×10^{-4}/ms $\pm$ 1.0×10^{-4}/ms. We also performed experiments with $\lambda_{max} = 10^{-2}$/ms.

In practice, it is difficult to know whether the CAN bus is in idle or active state without sensing voltage of CAN bus. Thus, we identified the idle state of CAN bus when the monitoring ECU does not receive any message for 700μs, since the transmission time for an 8-byte message is 544μs over a 250Kbps CAN.

[3]Imx6 sabrelite board has 1GHz CPU and 1GB Memory size. On the board, LinuxRK [28] is used as the operating system for the board.

Table 2. Modified SAE Benchmark

ID	χ_i	L_i	J_i	$T_i(1)$	$T_i(2)$	$D_i(1)$	$D_i(2)$
1	1 (ASIL A)	1	0.1	25	50	2.5	5
2	2 (ASIL D)	2	0.1	5	5	5	5
3	2 (ASIL D)	1	0.1	5	5	5	5
4	2 (ASIL D)	2	0.1	5	5	5	5
5	2 (ASIL D)	1	0.1	5	5	5	5
6	2 (ASIL D)	2	0.1	5	5	5	5
7	1 (ASIL A)	6	0.2	5	10	5	10
8	1 (ASIL A)	1	0.2	5	10	5	10
9	1 (ASIL A)	2	0.2	5	10	5	10
10	1 (ASIL A)	3	0.2	5	10	5	10
11	1 (ASIL A)	1	0.2	25	50	10	20
12	1 (ASIL A)	4	0.3	50	100	50	100
13	1 (ASIL A)	1	0.3	50	100	50	100
14	1 (ASIL A)	1	0.2	50	100	50	100
15	1 (ASIL A)	3	0.4	500	1,000	500	1,000
16	1 (ASIL A)	1	0.3	500	1,000	500	1,000
17	1 (ASIL A)	1	0.3	500	1,000	500	1,000

Table 3. Experimental Results ($\lambda_{max} = 10^{-3}$/ms)

Method	Schedulability Test	Utilization	L1 Time	L2 Time	Deadline Miss
EACAN	Pass	57.94%	3,599,822,751μs	177,249μs	0
WCTER-Based ($\gamma_{sys} = 2$)	Pass	44.01%	0μs	360,000,000μs	0
WCTER-Based ($\gamma_{sys} = 1$)	Fail	57.95%	360,000,000μs	0μs	0

6.1.3 Experimental Results. First, we measured time needed to execute Algorithms 1 and 2. On the imx6 sabrelite board, at the maximum 34μs was required to execute both Algorithms 1 and 2 for the given modified SAE benchmark.[4] We also measured the size of the arrays R and θ in Algorithm 1. For the given modified SAE benchmark, the amount of memory required to embed the arrays R and θ was 1,288 bytes.

By applying the measured overhead to Equation (20), we performed the EACAN schedulability test for the given modified SAE benchmark. The results are shown in Tables 3 and 4. The modified SAE benchmark passes the EACAN schedulability test where $\lambda_{max} = 10^{-3}$/ms and $\lambda_{max} = 10^{-2}$/ms. Thus, the given mixed-criticality CAN message set can operate without violating its requirement using EACAN. Also, we performed the WCTER-based schedulability test. The modified SAE benchmark at system criticality-level 2 passes the WCTER-based schedulability tests [2, 6] where $\lambda_{max} = 10^{-3}$/ms and $\lambda_{max} = 10^{-2}$/ms. However, the modified SAE benchmark at system criticality-level 1 failed the WCTER-based schedulability tests where $\lambda_{max} = 10^{-3}$/ms and $\lambda_{max} = 10^{-3}$/ms. That is, the system can only operate acceptably at criticality-level 2.

We measured the utilization of CAN bandwidth by transmitting messages in the modified SAE benchmark for 1h. The utilization of CAN bandwidth is calculated by active (transmission) time

[4]When we measured the overheads on the Arduino board, it was 780μ.

Table 4. Experimental Results ($\lambda_{max} = 10^{-2}$/ms)

Method	Schedulability Test	Utilization	L1 Time	L2 Time	Deadline Miss
EACAN	Pass	57.89%	3,582,859,215μs	17,140,785μs	0
WCTER-Based ($\gamma_{sys} = 2$)	Pass	44.01%	0μs	360,000,000μs	0
WCTER-Based ($\gamma_{sys} = 1$)	Fail	57.95%	360,000,000μs	0μs	0

during which the CAN messages are transmitted. To assess the actual utilization (akin to goodput), we subtracted the error frame transmission time and the time for transmitting corrupted messages from the entire active time. We only count the successfully transmitted messages. Due to the clock drift in Arduino boards, the number of initiated CAN messages is very slightly different from the theoretic number. Thus, the measured utilization was also different from the theoretically computed utilization.

In our experiments, with EACAN, the system operates at both criticality levels 1 and 2, but only operates at system criticality level 2 without EACAN, because the given message set fails WCTER-based schedulability tests at level 1.

Tables 3 and 4 summarize our experimental results. Inthe case of $\lambda_{max} = 10^{-3}$/ms, the bandwidth utilization is 57.94% with EACAN, which is very close to the maximum achievable utilization (57.95%) (the system operates only at the system criticality-level 1). However, with the WCTER-based analysis, the system can achieve only 44.01% bandwidth utilization, because the system is limited to operate at system criticality-level 1. For the case of $\lambda_{max} = 10^{-2}$/ms, the bandwidth utilization is 57.89% with EACAN. The presence of more errors changes the system criticality level to 2 more times than the case of $\lambda_{max} = 10^{-3}$.

In addition, we measured the number of deadline misses during the experimentation. In every case considered, we did not see any deadline miss, because significant burst transmission errors are required to cause a deadline miss, but the probability of occurrence of significant burst errors is very low. Despite the low probability of deadline misses, we could not utilize the bandwidth efficiently with the existing WCTER-based schedulability test, but we could with EACAN. In summary, EACAN made a 14% improvement of bandwidth utilization over the existing schemes using a modified SAE benchmark.

6.2 Simulation

6.2.1 Benchmark and Configuration. We generated 1,800 mixed frame sets by using NETCARBENCH (powertrain configuration[5]) [4] to evaluate the usefulness of EACAN for various cases. Since the latest version of NETCARBENCH does not support mixed criticality, we slightly modified the benchmark program to support it. We also slightly modified the powertrain configuration to generate 5ms-period messages to consider the recent ADAS[6]-related messages, which require very short latency. Since critical messages usually have relatively short periods, we assign criticality to each CAN message based on its period as shown in Table 5.

The change of period according to the system criticality level is described in Table 6. As stated before, the period of messages gradually increases according to the system criticality level if $\gamma_{sys} > \chi_i$. For example, in this simulation, the period of ASIL A messages at system criticality level 2 is 1.5x higher than that at the system criticality level 1. Likewise, the period of ASIL A messages

[5]In the configuration, the distribution of payload size, message period, jitter, and so on, are defined.
[6]Advanced Driver-Assistance System—adaptive cruise control, collision detection, and so on.

Table 5. Criticality Assignment Based on the Message Period

Period	2-Level System	3-Level System	4-Level System
5ms	70%-(ASIL) D 30%-A	50%-D, 30%-C 20%-A	70%-D, 10%-C 10%-B, 10%-A
10ms			
20ms		25%-D, 50%-C 25%-A	10%-D, 70%-C 10%-B, 10%-A
50ms			
100ms	30%-D, 70%-A		10%-D, 10%-C 70%-B, 10%-A
200ms			
1s		20%-D, 30%-C 50%-A	10%-D, 10%-C 10%-B, 70%-A
2s			

Table 6. Period and Deadline Changes According to the System Criticality Level

Criticality	2-Level System	3-Level System	4-Level System
ASIL A	$3T_i(1) = T_i(2)$	$1.5T_i(1) = T_i(2)$ $3T_i(1) = T_i(3)$	$1.5T_i(1) = T_i(2)$ $2T_i(1) = T_i(3)$ $3T_i(1) = T_i(4)$
ASIL B	—	—	$T_i(1) = T_i(2)$ $1.5T_i(2) = T_i(3)$ $2T_i(2) = T_i(4)$
ASIL C	—	$T_i(1) = T_i(2)$ $2T_i(2) = T_i(3)$	$T_i(1) = T_i(2)$ $T_i(2) = T_i(3)$ $1.5T_i(3) = T_i(4)$
ASIL D	$T_i(1) = T_i(2)$	$T_i(1) = T_i(2)$ $T_i(2) = T_i(3)$	$T_i(1) = T_i(2)$ $T_i(2) = T_i(3)$ $T_i(3) = T_i(4)$

at the system criticality level 3 is 2x higher than that at the system criticality level 1, and the period of ASIL A messages at the system criticality level 4 is 3x higher than that at the system criticality level 1. Consequently, $T_i(1) <= T_i(2) <= T_i(3) <= T_i(4)$ as we modeled in Section 3.3. We arbitrarily determine the amount of period stretch in this simulation.

We applied the Robust Priority Assignment [11], which is proven to maximize the number of tolerable transmission errors, to the generated frame set to assign priority to each message.

6.2.2 Simulation Results. We measured the (coverage) rate of passing the EACAN schedulability test and that of passing the WCTER-based test for the generated message sets. The results are summarized in Table 7. If a given message set passes the EACAN schedulabilty test, then we count the given message set verifiable with EACAN. Also, if a given message set passes the WCTER-based test at the system criticality level 1, we count the given message set verifiable with WCTER.

Regardless of the utilization[7] of given message sets and the applied λ_{max}, the coverage of EACAN schedulability test is shown to be higher than that of WCTER-based test. This is because the EACAN schedulability test considers the system criticality-level transition and the level change provides more chances to pass the test. The average improvement of coverage for a total of 1,800 generated sets is 7%, and the maximum improvement is 17% where $\lambda_{max} = 10^{-3}$/ms and the utilization of the given message set is in the range of 70–80%.

[7]Note that the utilization is measured using the periods at the system criticality level 1.

Table 7. Coverage of EACAN Schedulability Test and WCTER-based Schedulability Test

		$\lambda_{max} = 10^{-3}$/ms				$\lambda_{max} = 10^{-2}$/ms			
		Total	50%–60%	60%–70%	70%–80%	Total	50%–60%	60%–70%	70%–80%
EACAN	Pass	703	288	246	169	374	211	116	47
	Fail	197	12	54	131	526	89	184	253
	Coverage	78.1%	96.0%	82.0%	56.3%	41.5%	70.3%	38.6%	15.6%
WCTER Based	Pass	638	287	235	118	313	198	95	20
	Fail	262	13	65	182	587	102	205	280
	Coverage	70.8%	95.6%	78.3%	39.3%	34.7%	66.0%	31.6%	6.6%
Coverage Difference		7.2%	0.3%	3.6%	17%	6.7%	4.3%	7%	9%

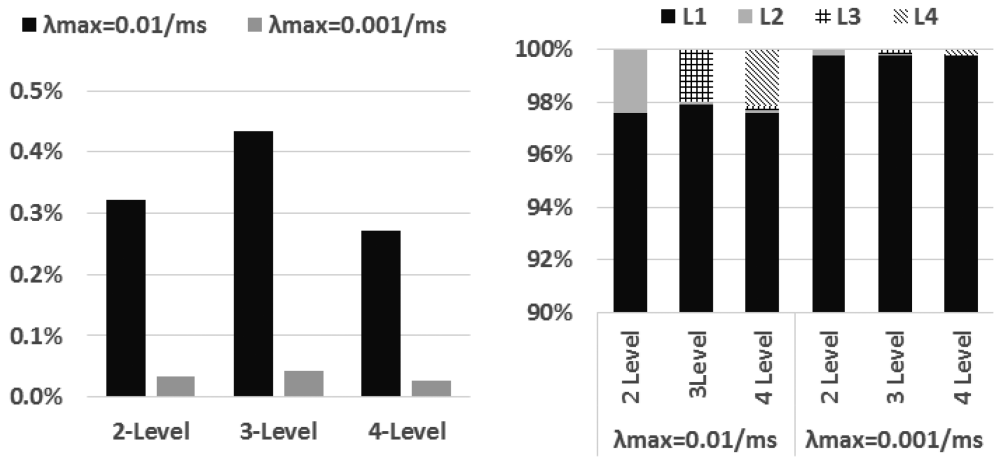

Fig. 7. (Left) Bandwidth utilization gap between EACAN operation and L1-Only operation. (Right) System criticality-level distribution for a 1h CAN operation.

We also measured the bandwidth utilization of EACAN by simulating a 1-hour CAN operation for the generated message sets. We generated transmission errors using a Poisson process. For example, if $\lambda_{max} = 10^{-3}$/ms, then we generate 3,600 transmission errors, following a Poisson distribution for the 1h operation. Figure 7 (Right) shows the system criticality-level distribution over the 1h operation. The system is shown to stay at the system criticality-level 1 for over 97% of its operation. This is because the length of a busy period is usually not large, and thus system criticality-level returns quickly to the base level ($\gamma_{sys} = 1$). The figure also shows that the proportion of $\gamma_{sys} = 1$ increases if the λ_{max} decreases. The large portion of staying at the lowest level makes the bandwidth utilization gap between EACAN operation and L1-only operation (the best achievable utilization) very small as shown in Figure 7 (Left). This result is very similar to our experimental result.

In our simulation, deadline miss occurs in only two test cases of 70%–80% utilization and $\lambda_{max} = 10^{-2}$/ms when the system only operates at the system criticality level 1 (WCTER-Based). With EACAN, there is no deadline miss for every test cases. This is because we generate the errors using Poisson distribution in the simulations, and thus burst error arrival, which could lead to deadline miss, rarely showed up in the simulations like the experimental result.

From the evaluation and simulation results, we see that network utilization and schedulability can be improved by using EACAN. That is, more CAN messages can be transmitted on a single

CAN bus, and thus system designers can reduce the number of CAN buses required to implement the target system. From the automotive perspective, the implementation cost, space, and weight for CAN bus wire can be saved.

7 CONCLUSION

In this article, we have developed EACAN, which adaptively controls the period of messages according to the recent past transmission errors to achieve reliable and resource-efficient CAN communications. Also, we have analyzed probabilistic schedulability for a given CAN frame set with EACAN operation. Our experimental results show that EACAN makes a 14% improvement of bandwidth utilization for the modified SAE benchmark. Besides, our simulation results show that 7% more CAN message sets can be verified for their schedulability with EACAN schedulability test, on average.

Even though of use CAN may decline in future, our methodology can be easily applied to Controller Area Network with Flexible Data-rate (CAN-FD), which is an emerging substitute of CAN, by just replacing CAN timing analysis with the state-of-art CAN-FD timing analysis [23].

REFERENCES

[1] Arduino. [n.d.]. Retrieved from https://www.arduino.cc/.

[2] P. Axer, M. Sebastian, and R. Ernst. 2012. Probabilistic response time bound for CAN messages with arbitrary deadlines. In *Proceedings of the Design, Automation Test in Europe Conference Exhibition (DATE'12)*. 1114–1117. DOI: https://doi.org/10.1109/DATE.2012.6176662

[3] H. Aysan, A. Thekkilakattil, R. Dobrin, and S. Punnekkat. 2010. Efficient fault tolerant scheduling on controller area network (CAN). In *Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation (ETFA'10)*. 1–8. DOI: https://doi.org/10.1109/ETFA.2010.5641318

[4] Christelle Braun, Lionel Havet, and Nicolas Navet. 2007. NETCARBENCH: A benchmark for techniques and tools used in the design of automotive communication systems. In *Proceedings of the 7th IFAC International Conference on Fieldbuses & Networks in Industrial & Embedded Systems (FeT'07)*. Toulouse, France, 321–328. Retrieved from https://hal.inria.fr/inria-00188629.

[5] I. Broster, A. Burns, and G. Rodriguez-Navas. 2002. Probabilistic analysis of CAN with faults. In *Proceedings of the Real-Time Systems Symposium*. 269–278. DOI: https://doi.org/10.1109/REAL.2002.1181581

[6] I. Broster, A. Burns, and G. Rodriguez-Navas. 2004. Comparing real-time communication under electromagnetic interference. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS'04)*. 45–52. DOI: https://doi.org/10.1109/EMRTS.2004.1310997

[7] A. Burns and R. I. Davis. 2013. Mixed criticality on controller area network. In *Proceedings of the 25th Euromicro Conference on Real-Time Systems*. 125–134. DOI: https://doi.org/10.1109/ECRTS.2013.23

[8] Y. Chun, S. Park, J. Kim, H. Kim, K. Hwang, J. Kim, and S. Ahn. 2012. System and electromagnetic compatibility of resonance coupling wireless power transfer in on-line electric vehicle. In *Proceedings of the International Symposium on Antennas and Propagation (ISAP'12)*. 158–161.

[9] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien. 2007. Controller area network (CAN) schedulability analysis: Refuted, revisited and revised. *Real-Time Syst.* 35, 3 (Apr. 2007), 239–272. DOI: https://doi.org/10.1007/s11241-007-9012-7

[10] R. I. Davis, S. Kollmann, V. Pollex, and F. Slomka. 2011. Controller area network (CAN) Schedulability analysis with FIFO queues. In *Proceedings of the 23rd Euromicro Conference on Real-Time Systems*. 45–56. DOI: https://doi.org/10.1109/ECRTS.2011.13

[11] Robert I. Davis and Alan Burns. 2009. Robust priority assignment for messages on controller area network (CAN). *Real-Time Syst.* 41, 2 (Feb. 2009), 152–180. DOI: https://doi.org/10.1007/s11241-008-9065-2

[12] Reinhard Felgenhauser. 2011. Electromagnetic interference (EMI) in E-vehicles. Retrieved from www.automotive-eetimes.com/content/electromagnetic-interference-emi-e-vehicles.

[13] J. Ferreira, A. Oliveira, P. Fonseca, and J. Fonseca. 2004. An experiment to assess bit error rate in CAN. In *Proceedings of the 3rd International Workshop of Real-Time Networks (RTN'04)*. 15–18.

[14] Freescale. [n.d.]. Future Advances in Body Electronics. Retrieved from http://www.nxp.com/assets/documents/data/en/white-papers/BODYDELECTRWP.pdf.

[15] A. R. Guraliuc, M. Zhadobov, R. Sauleau, L. Marnat, and L. Dussopt. 2015. Millimeter-wave electromagnetic field exposure from mobile terminals. In *Proceedings of the European Conference on Networks and Communications (EuCNC'15)*. 82–85. DOI: https://doi.org/10.1109/EuCNC.2015.7194045

[16] SAE International. 2000. Class C application requirement considerations. *SAE Technical Report J2056/1* (Feb. 2000).
[17] ISO/TC22. 2011. *ISO26262: Road Vehicles—Functional Safety.* Technical Report. International Organization for Standardization.
[18] U. Keskin. 2013. Evaluating message transmission times in controller area network (CAN) without buffer preemption revisited. In *Proceedings of the IEEE 78th Vehicular Technology Conference (VTC'13)*. 1–5. DOI: https://doi.org/10.1109/VTCFall.2013.6692198
[19] D. A. Khan, R. J. Bril, and N. Navet. 2010. Integrating hardware limitations in CAN schedulability analysis. In *Proceedings of the 8th IEEE International Workshop on Factory Communication Systems (WFCS'10)*. 207–210. DOI: https://doi.org/10.1109/WFCS.2010.5548604
[20] D. A. Khan, R. I. Davis, and N. Navet. 2011. Schedulability analysis of CAN with non-abortable transmission requests. In *Proceedings of the IEEE 16th Conference on Emerging Technologies Factory Automation (ETFA'11)*. 1–8. DOI: https://doi.org/10.1109/ETFA.2011.6058998
[21] J. Kim, K. Lakshmanan, and R. Rajkumar. 2012. Rhythmic tasks: A new task model with continually varying periods for cyber-physical systems. In *Proceedings of the IEEE/ACM Third International Conference on Cyber-Physical Systems*. 55–64. DOI: https://doi.org/10.1109/ICCPS.2012.14
[22] Daniel KAd'stner, Marek Jersak, Christian Ferdinand, Peter Gliwa, and Reinhold Heckmann. 2011. An integrated timing analysis methodology for real-time systems. In *SAE Technical Paper*. DOI: https://doi.org/10.4271/2011-01-0444
[23] R. Lange, A. C. Bonatto, F. Vasques, and R. S. de Oliveira. 2016. Timing analysis of hybrid FlexRay, CAN-FD and CAN vehicular networks. In *Proceedings of the 42nd Annual Conference of the IEEE Industrial Electronics Society (IECON'16)*. 4725–4730. DOI: https://doi.org/10.1109/IECON.2016.7793791
[24] Microchip 2012. *Stand-Alone CAN Controller with SPI Interface*. Microchip. Rev. G.
[25] N. Navet and H. Perrault. 2012. CAN in automotive applications: A look forward. In *Proceedings of the 13th International CAN Conference*.
[26] N. Navet, Y.-Q. Song, and F. Simonot. 2000. Worst-case deadline failure probability in real-time applications distributed over controller area network. *J. Syst. Archit.* 46, 7 (Apr. 2000), 607–617. DOI: https://doi.org/10.1016/S1383-7621(99)00016-8
[27] Nicolas Navet, Schehnaz Louvart, Jose Villanueva, Sergio Campoy-Martinez, and Jorn Migge. 2014. Timing verification of automotive communication architectures using quantile estimation. In *Proceedings of the Conference on Embedded Real Time Software and Systems*.
[28] Shuichi Oikawa and Ragunathan Rajkumar. 1998. Linux/RK: A portable resource kernel in Linux. In *Proceedings of the 19th IEEE Real-Time Systems Sumposium*.
[29] F. Ren, Y. R. Zheng, M. Zawodniok, and J. Sarangapani. 2007. Effects of electromagnetic interference on control area network performance. In *Proceedings of the IEEE Region 5 Technical Conference*. 199–204. DOI: https://doi.org/10.1109/TPSD.2007.4380381
[30] M. Schreiner, H. Mahmoud, M. Huber, S. Koc, and J. Waldmann. 2013. CAN FD from an OEM point of view. In *Proceedings of the 14th International CAN Conference*.
[31] Symtavision. [n.d.]. Retrieved from https://www.symtavision.com/.
[32] L. Tan, C. Du, and Y. Dong. 2015. Control-performance-driven period and deadline selection for cyber-physical systems. In *Proceedings of the 10th Asian Control Conference (ASCC'15)*. 1–6. DOI: https://doi.org/10.1109/ASCC.2015.7244832
[33] K. W. Tindell and A. Burns. 1994. *Guaranteed Message Latencies for Distributed Safety-Critical Hard Real-Time Control Networks*. Technical Report YCS 229. University of York, Department of Computer Science.
[34] K. W. Tindell, H. Hansson, and A. J. Wellings. 1994. Analysing real-time communications: Controller area network (CAN). In *Proceedings of the Real-Time Systems Symposium*. 259–263. DOI: https://doi.org/10.1109/REAL.1994.342710
[35] Volcano. [n.d.]. Retrieved from https://www.mentor.com/products/vnd.

Received January 2018; revised September 2018; accepted December 2018