A Real-Time CAN-CAN Gateway with Tight Latency Analysis and Targeted Priority Assignment

Guoqi Xie[1,2], Haijie Gong[1,2], Yunbo Han[3], Samarjit Chakraborty[4], Wanli Chang[5,*]
[1]College of Computer Science and Electronic Engineering, Hunan University, China
[2]Key Laboratory for Embedded and Network Computing of Hunan Province, China
[3]Future Network Lab, CSIG, Tencent, China
[4]Department of Computer Science, University of North Carolina at Chapel Hill, USA
[5]Department of Computer Science, University of York, UK
{xgqman@hnu.edu.cn, haijie@hnu.edu.cn, yunbohan@tencent.com, samarjit@cs.unc.edu, wanli.chang@york.ac.uk}

***Abstract*—There is a demand in the automotive industry to connect two CAN-based subsystems. The commercial CAN-CAN gateway supports basic message forwarding with no real-time behavior. To address this issue, a new gateway architecture is described, on which we present a novel worst-case latency analysis. Specifically, we bound the arrival of the messages at the gateway, which is then used by the Pointer Reachability Exploration (PRE) to derive the interfering message jobs. Our analysis computes a safe gateway latency tighter than the conventional one applied in CAN. Furthermore, we propose a Targeted Priority Assignment (TPA) algorithm that targets at the priorities assigned at the CAN bus and runs a reordering at the gateway to enhance the schedulability. TPA performs better than DMPO (Deadline Monotonic Priority Ordering), while OPA (Audsley's Optimal Priority Assignment) cannot be applied in this context. Evaluation over real-life and scalable CAN message sets is conducted. The reported analysis and priority assignment algorithm are developed for dynamic use to improve the acceptance ratio and can also be deployed statically to provide timing guarantees. This work can be easily extended to support multiple CAN subsystems.**

I. Introduction

The Controller Area Network (CAN) is the most widely used communication protocol in automobiles due to its simple, efficient, and stable broadcast communication mechanism [1]. There has been a practical demand to connect two CAN-based subsystems via a gateway [2]–[4]. For example, through data exchange at the gateway, the engine control module in the powertrain CAN subsystem obtains the input signal from the instrument unit in the body CAN subsystem, which displays the information from the powertrain CAN subsystem [5].

In the commercial CAN-CAN gateways being used (e.g., CG-ARM7 [6] and J1939 [7]), each CAN transceiver is equipped with a receiving queue (also called input queue or Rx-Buffer) and a transmission queue (also called output queue or Tx-Buffer), implemented with fixed priority [8], first-in first-out (FIFO) [9], or a mixture of both [10], [11]. The receiving queue temporarily stores the messages sent from the source-end CAN subsystem to the gateway, while the transmission queue temporarily stores the messages sent from the gateway to the destination-end CAN subsystem. At present, these gateways only support basic message forwarding, where the worst-case latency analysis is difficult. There are two main reasons for this. First, messages may be dropped due to the limited queue length. Second, the message transmission in one CAN subsystem is affected by the other one.

Main contributions: In order to address this problem, we describe a new gateway architecture, which trades hardware resources for isolation and predictability, facilitating the worst-case latency analysis. We present a novel worst-case latency analysis for the gateway. Specifically, the arrival of the messages at the gateway (including how the jobs of one message arrive and how the earliest jobs of all messages arrive) is bounded and used by the Pointer Reachability Exploration (PRE) to derive the interfering message jobs. Our analysis is proven to be safe and is able to achieve a tighter bound than the conventional non-preemptive analysis applied in CAN (hereafter named Davis' timing analysis) [8]. Furthermore, we propose a Targeted Priority Assignment (TPA) algorithm that targets at the priorities assigned at the CAN bus and runs a reordering at the gateway to enhance the schedulability. TPA tries to make unschedulable messages schedulable with limited priority reordering, and gives up (assigning the lowest priority) if a message inevitably compromises the schedulability of the entire system. TPA performs better than DMPO (Deadline Monotonic Priority Ordering) [12], while OPA (Audsley's Optimal Priority Assignment) [13] cannot be used in this context, as it is terminated when facing unschedulability. Evaluation over real-life and scalable CAN message sets is conducted. The reported analysis and priority assignment algorithm are developed for dynamic use to improve the acceptance ratio (i.e., the number of messages meeting deadlines divided by the total number of messages) and can also be deployed statically to provide timing guarantees. This work can be easily extended to support multiple CAN subsystems if needed.

II. Related Work

Sommer et al. [3] studied the optimization of the gateway processing time, receiving queue size, and transmission queue size of the CAN-CAN gateway by proposing the round robin scheduler. Lee et al. [14] investigated the traffic-balancing

*The corresponding author is Wanli Chang.

algorithm for the CAN-CAN gateway to predict the traffic of each CAN subsystem, thereby increasing the network capacity of the CAN subsystems and reducing end-to-end response time. Sojka et al. [4] studied the reasonable gateway configurations for the CAN-CAN gateway by automating the measurements and comparing various results. However, the results obtained from measurement can only represent partial cases and cannot reflect all the cases of the queue. The methods proposed by the above works (i.e., [3], [4], [14]) can not generate a safe bound, thereby lacking the ability of timing prediction.

Based on the Davis' timing analysis [8], Davis et al [10] continued to study the response time analysis for messages in a CAN bus connecting the gateway, which is configured with FIFO queues and priority queues together. Azketa et al. [5] studied the end-to-end response time analysis for a multi-packet CAN message (i.e., a message consists of multiple packets) in two CAN buses interconnected by the CAN-CAN gateway; the method is implemented by considering the pipelining of the packets in the gateway. Xie et al. [15] proposed the end-to-end response time analysis through examining the different actual arrival orders in detail, thereby obtaining a safe and tight bound; however, this work ignores the message-processing latency in the gateway and assumes it to be zero.

III. Gateway Architecture and Models

A. Gateway Architecture

As emphasised in Section I, current commercial CAN-CAN gateways only support basic message forwarding and ignore the worst-case latency analysis due to the limited queue length and the mutual influence between two subsystems. To facilitate worst-case latency analysis, we present a new gateway architecture different from the existing commercial gateways.

The presented CAN-CAN gateway architecture has two CAN subsystems (i.e., CAN_1 and CAN_2) with the same bandwidth of 500 KBit/s, as shown in Fig. 1. Each subsystem consists of two buses named $e2e\&e2g$ and $g2e$ buses. The $e2e\&e2g$ bus is responsible for transmitting messages released from one Electronic Control Unit (ECU) to other ECUs (or the gateway) in the same subsystem, whereas the $g2e$ bus is responsible for receiving the messages sent from the gateway and transmitted to ECUs. Although adopting the above dual CAN bus solution could increase the hardware cost, it has the advantage of reducing the mutual influence of messages between two subsystems, thereby reducing the complexity of worst-case latency analysis; in other words, we trade hardware resources for isolation and predictability, facilitating worst-case latency analysis. As far as we know, some dual CAN bus solutions have been used in some studies to address individual particular requirements [16]–[18].

There are two message awaiting priority queues (i.e., $awaiting\_queue\_12$ and $awaiting\_queue\_21$) in the presented gateway, where $awaiting\_queue\_12$ temporarily stores messages from CAN_1 to CAN_2, and $awaiting\_queue\_21$ temporarily stores messages from CAN_2 to CAN_1. Note the awaiting priority queues are buffers created in memory and can be dynamically adjusted in size. The queue creation cost is negligible, especially considering that the CAN message size is small. Conventionally, the fixed-size pre-fabricated buffers in CAN controllers are used for message queuing.

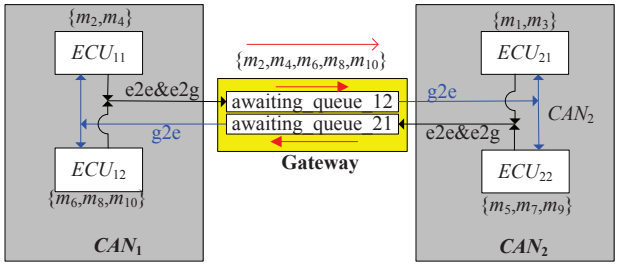

Fig. 1: CAN-CAN gateway and in-vehicle networks.

In Fig. 1, there are two ECUs mounted on each bus, and each ECU lists the messages that need to be sent. For example, ECU_{11} in CAN_1 bus can send two messages (i.e., m_2 and m_4). This study assumes that each message in each ECU is stored in a priority queue; from a safe and conservative timing analysis, each message that needs to be sent has no released jitter and offset, so that the messages in the same ECU can be queued simultaneously.

In summarization, the presented gateway architecture has the following changes compared with current commercial gateways: 1) we adopt the dual CAN bus solution (i.e., each subsystem has two CAN buses) to isolate the mutual influence between two subsystem, thereby reducing the complexity of worst-case latency analysis; 2) we create two awaiting priority queues in memory of the gateway to store the messages that need to be forwarded, and each CAN transceiver no longer has receiving and transmission queues. Through the above changes, the gateway architecture trades hardware resources for isolation and predictability, facilitating the worst-case latency analysis. When more CAN subsystems are connected to the gateway, each CAN subsystem still adopts the dual CAN bus solution and the latency analysis still holds, so that the gateway is scalable; note that the number of queues will increase and the contention needs to be resolved on the $g2e$ bus, which is supported by the conventional CAN analysis.

B. Message Model and Classification

Let $M = \{m_1, m_2, ..., m_{|M|}\}$ represent a message set, where $|M|$ is the message set size. Let m_i=(P_i, CAN_i^{source}, CAN_i^{dest}, C_i, T_i, D_i) represent the ith periodic message in the message set and i is the unique identifier of a message; P_i means the priority of message m_i; the smaller the i, the higher the priority. CAN_i^{source} and CAN_i^{dest} represent the source-end and destination-end subsystems of m_i, respectively. C_i represents the Worst-case Transmission Time (WCTT) of m_i; T_i represents the period (all messages are released strictly in individual ECUs according to the given period) of m_i; and D_i represents the end-to-end deadline of m_i.

Definition 1: **Non-gateway message:** A non-gateway message is the message whose source-end subsystem and destination-end subsystem are the same).

Definition 2: **Gateway message:** A gateway message is the message that is transmitted from one source-end subsystem to another destination-end subsystem.

Table I shows a motivating example (i.e., the message set has 10 messages) used in this study. In this example, m_1, m_3, m_5, m_7, and m_9 are non-gateway messages, while m_2, m_4, m_6, m_8, and m_{10} are gateway messages.

TABLE I: Motivating example (i.e., the message set has 10 messages) in this study.

m_i	P_i	CAN_i^{source}	CAN_i^{dest}	C_i	T_i	D_i	R_i^{source}	R_i^{dest}	D_i^{GW}
m_1	1	CAN_2	CAN_2	230 μs	1200 μs	1200 μs	500 μs	-	-
$\boldsymbol{m_2}$	2	$\boldsymbol{CAN_1}$	$\boldsymbol{CAN_2}$	**210 μs**	**1000 μs**	**1000 μs**	**480 μs**	**210 μs**	**310 μs**
m_3	3	CAN_2	CAN_2	270 μs	1600 μs	1600 μs	770 μs	-	-
$\boldsymbol{m_4}$	4	$\boldsymbol{CAN_1}$	$\boldsymbol{CAN_2}$	**170 μs**	**1800 μs**	**1800 μs**	**650 μs**	**170 μs**	**980 μs**
m_5	5	CAN_2	CAN_2	190 μs	1700 μs	1700 μs	900 μs	-	-
$\boldsymbol{m_6}$	6	$\boldsymbol{CAN_1}$	$\boldsymbol{CAN_2}$	**210 μs**	**1700 μs**	**1700 μs**	**860 μs**	**210 μs**	**630 μs**
m_7	7	CAN_2	CAN_2	150 μs	2000 μs	2000 μs	1050 μs	-	-
$\boldsymbol{m_8}$	8	$\boldsymbol{CAN_1}$	$\boldsymbol{CAN_2}$	**270 μs**	**3000 μs**	**3000 μs**	**1130 μs**	**270 μs**	**1600 μs**
m_9	9	CAN_2	CAN_2	210 μs	3000 μs	3000 μs	1260 μs	-	-
$\boldsymbol{m_{10}}$	10	$\boldsymbol{CAN_1}$	$\boldsymbol{CAN_2}$	**210 μs**	**3000 μs**	**3000 μs**	**1490 μs**	**210 μs**	**1300 μs**

C. WCRTs of Non-Gateway Messages

(1) For a non-gateway message m_i, its end-to-end Worst-Case Response Time (WCRT) is its source-end WCRT, namely,

$$R_i^{\text{e2e}} = R_i^{\text{source}}.$$

The details of obtaining R_i^{source} can refer to the Davis' timing analysis [8] and are as follows.

Let $hp_{\text{source}}(m_i)$ represent the source-end high-priority messages of m_i. According to the Davis' timing analysis in [8], if m_i is simultaneously released with its high-priority messages in $hp_{\text{source}}(m_i)$, then the maximum interfering latency w_i^{source} is

$$w_i^{\text{source}}(n+1) = B_i + \sum_{\forall m_j \in hp_{\text{source}}(m_i)} \left\lceil \frac{w_i^{\text{source}}(n) + \tau_{\text{data}}}{T_j} \right\rceil C_j. \tag{1}$$

τ_{data} is the time to transmit one bit (i.e., bit time); for a CAN bus with the bandwidth of 500 Kbits/s, its bit time is 2 μs. B_i is the blocking time and is calculated by

$$B_i = \max(^{\max\ (C_k)}_{\forall m_k \in lp_{\text{source}}(m_i)}, C_i), \tag{2}$$

where $lp_{\text{source}}(m_i)$ represents the source-end low-priority messages of m_i.

In Eq. (1), iteration starts with a suitable initial value like $w_i^{\text{source}}(0) = C_i$. The iteration process continues until one of the following two situations occurs [8].

(1) $w_i^{\text{source}}(n+1)$ + $C_i > D_i$, where m_i is unschedulable.
(2) $w_i^{\text{source}}(n+1)$ = $w_i^{\text{source}}(n)$ and $w_i^{\text{source}}(n+1) + C_i \leqslant D_i$, where m_i is schedulable.

In the end, R_i^{source} of m_i is calculated by

$$R_i^{\text{source}} = w_i^{\text{source}}(n+1) + C_i. \tag{3}$$

Table I has listed the R_i^{source} values of non-gateway messages (i.e., m_1, m_3, m_5, m_7, and m_9) through the Davis' timing analysis [8]. From Table I, we can see that five non-gateway messages (i.e., m_1, m_3, m_5, m_7, and m_9) meet individual deadlines, because the WCRTs of these messages are less than individual deadlines, that is,

$$R_i^{\text{source}} \leqslant D_i.$$

D. In-Gateway Worst-Case Latency

Definition 3: **In-Gateway worst-case latency:** The in-gateway worst-case latency of gateway message m_i is defined as the maximum awaiting time of m_i among its all possible awaiting times in the gateway.

For a gateway message m_i, its end-to-end WCRT is the sum of its source-end WCRT, in-gateway worst-case latency, and destination-end WCRT, namely,

$$R_i^{\text{e2e}} = R_i^{\text{source}} + L_i^{\text{GW}} + R_i^{\text{dest}}.$$

Note that there is a fact that the message copy speed inside the gateway is much faster than the message transmission speed in CAN bus. In our gateway prototype platform, the Cortex-M4 processor is used and its clock frequency is 168 MHz, such that the message copy time needs merely dozens of μs; however, the message transmission time for the CAN bus with the bandwidth of 500 KBit/s is approximately 250 μs [5]. Therefore, the copy time of the message in the gateway is ignored and the waiting time is considered in this paper.

(1) When m_i is transmitted in its source-end, the calculation of R_i^{source} has been explained in Section III.C. The R_i^{source} values of five gateway messages are shown in Table I.

(2) When m_i is transmitted in its destination-end, the value of R_i^{dest} is its WCET C_i. The reason is that m_i occupies the $g2e$ bus without interference from the destination-end messages. That is,

$$R_i^{\text{dest}} = C_i.$$

The R_i^{dest} values of five gateway messages have been obtained, as shown in Table I.

Considering that R_i^{source} and R_i^{dest} are easy to obtain, the main concern is to obtain the in-gateway worst-case latency (i.e., L_i^{GW}). L_i^{GW} is actually the maximum awaiting time in the specified awaiting priority queue inside the gateway.

E. In-Gateway Deadline

For the five gateway messages (i.e., m_2, m_4, m_6, m_8, and m_{10}), we are not sure yet whether they have met the end-to-end deadline, because we have not determined yet the in-gateway worst-case latencies of these messages.

Definition 4: **In-Gateway Deadline:** The gateway message m_i can only meet its end-to-end deadline if the following condition is met

$$R_i^{\text{e2e}} = R_i^{\text{source}} + L_i^{\text{GW}} + R_i^{\text{dest}} \leqslant D_i.$$

Therefore, m_i can meet its end-to-end deadline when L_i^{GW} meets the following condition:

$$L_i^{\text{GW}} \leqslant D_i^{\text{e2e}} - R_i^{\text{source}} - R_i^{\text{dest}}. \tag{4}$$

The right part of Eq. (4) is defined as the in-gateway deadline of message m_i (i.e., D_i^{GW}), namely,

$$D_i^{\text{GW}} = D_i^{\text{e2e}} - R_i^{\text{source}} - R_i^{\text{dest}}. \quad (5)$$

To locate the presented gateway in this paper, we focus on the in-gateway worst-case latency through modular separation, thereby transferring the end-to-end deadline to the in-gateway deadline. Through Eq. (5), the D_i^{GW} values of five gateway messages (i.e., m_2, m_4, m_6, m_8, and m_{10}) have been obtained, as shown in Table I.

IV. Bounding Arrival of Messages

To obtain safe and tight worst-case latency analysis for the CAN-CAN gateway, we should understand the transmission details inside the gateway from a realistic perspective as much as possible. In this section, the arrival of the messages at the gateway (including how the jobs of one message arrive and how the earliest jobs of all messages arrive) is bounded through Definitions 7 and 9.

A. Minimum In-Gateway Inter-Arrival Time

The gateway messages in the same awaiting priority queue are arranged in descending order of priority (i.e., the higher the priority of the message, the sooner the message is out of the awaiting priority queue). In the motivating example, m_2, m_4, m_6, m_8, and m_{10} are in the same awaiting priority queue $awaiting\_queue\_12$. When analyzing the in-gateway worst-case latency of a gateway message, source-end high-priority gateway messages can interfere with the current analyzed message m_i (in this paper, m_i is called the current analyze message, and m_j is a high-priority gateway message relative to m_i). For instance, when we analyze the in-gateway worst-case latency of m_{10}, it will be interfered by m_2, m_4, m_6, m_8, which are from CAN_1 subsystem.

Let $m_{j,1}$, $m_{j,2}$, and $m_{j,3}$ represent the 1st, 2nd, and 3rd message jobs (a job is an instance of a message) of gateway message m_j, respectively.

Definition 5: **In-gateway interval time:** The in-gateway interval time means the interval time between the two adjacent in-gateway arrival jobs $m_{j,x}$ and $m_{j,x+1}$, and it is denoted by $I_{j,x,x+1}$.

Definition 6: **Minimum in-gateway inter-arrival time:** The minimum in-gateway inter-arrival time of gateway message m_j means the minimum in-gateway interval between any two adjacent in-gateway arrival jobs of m_j, and it is denoted by $T_j^{\min}$.

Lemma 1: If $m_{j,1}$ experiences source-end WCRT of R_j^{source} in its source-end subsystem, and $m_{j,2}$ experiences WCET of C_j in its source-end subsystem, then $T_j^{\min}$ is equal to $I_{j,1,2}$ [15]:

$$T_j^{\min} = I_{j,1,2} = T_j - R_j^{\text{source}} + C_j. \quad (6)$$

For example, we know that the source-end WCRT of m_8 (m_8 is a high-priority message relative to m_{10}) is 1130 μs (i.e., R_8^{source} = 1130 μs) from Table I, then the minimum in-gateway inter-arrival time $T_8^{\min}$ is

$$T_8^{\min} = I_{8,1,2} = 3000 - 1130 + 270 = 2140\ \mu\text{s}. \quad (7)$$

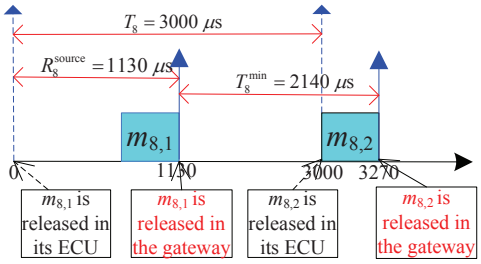

Fig. 2: Minimum in-gateway inter-arrival time of m_8 and in-gateway interval time between $m_{8,1}$ and $m_{8,2}$ in the gateway.

Fig. 2 shows the schematic diagram of obtaining the minimum in-gateway inter-arrival time of m_8: 1) at instant of 0 μs, $m_{8,1}$ is released in its ECU; 2) at instant of 1,130 μs, $m_{8,1}$ arrives at the gateway (i.e., $m_{8,1}$ is transmission finished with its WCRT in its source-end CAN subsystem); 3) at instant of 3,000 μs, $m_{8,2}$ is released in its ECU; and 4) at instant of 3,270 μs (3,000 + 270 = 3,270), $m_{8,2}$ arrives at the gateway (i.e., $m_{8,2}$ is transmission finished with its WCTT in its source-end CAN subsystem). Finally, the minimum in-gateway inter-arrival time of m_8 is $T_8^{\min}$ = 2140 μs, as shown in Fig. 2.

B. Pessimistic Worst-Case Latency Analysis

Referring to the Davis' timing analysis [8] in Section III.C, we can calculate the in-gateway worst-case latency for each gateway message based on the minimum in-gateway inter-arrival time obtained by Eq. (6). The details are below.

Let $hp_{\text{source-GW}}(m_i)$ represent the source-end high-priority gateway messages of m_i. According to the Davis' timing analysis in [8], if m_i simultaneously arrives at the gateway with messages in $hp_{\text{source-GW}}(m_i)$, then its in-gateway worst-case latency L_i^{GW} is

$$L_i^{\text{GW}}(n+1) = {B_i}^{\text{GW}} + \sum_{\forall m_j \in hp_{\text{source-GW}}(m_i)} \left\lceil \frac{L_i^{\text{GW}}(n) + \tau_{\text{data}}}{T_j^{\min}} \right\rceil C_j, \quad (8)$$

where the period of the each source-end high-priority gateway message is denoted by the minimum in-gateway inter-arrival time. B_i^{GW} is the in-gateway blocking time, which occurs when a gateway message just starts transmitting in the $g2e$ bus of the destination-end CAN system; therefore, B_i^{GW} is calculated by

$${B_i}^{\text{GW}} = \max_{h \in p_{\text{source-GW}}(i)} C_h, \quad (9)$$

where $p_{\text{source}}(m_i)$ represents the source-end all gateway messages of m_i.

It can be seen from Eqs. (8) and (9) that m_i's worst-case latency depends on the occupancy of the $g2e$ bus of the destination-end CAN subsystem. In other words, if we take the gateway and $g2e$ bus as a whole, then the overall WCRT is

$$R_i^{\text{GW-dest}} = L_i^{\text{GW}} + R_i^{\text{dest}} = L_i^{\text{GW}} + C_i.$$

However, we focus on the in-gateway worst-case latency through modular separation as pointed earlier.

The worst-case latency obtained by the Davis' timing analysis is a safe bound. However, there is a practical consideration:

although the released time of each message is periodic at the ECU, but its actual start time is not periodic at the source-end CAN subsystem, where interface and blocking exist; therefore, the actual finish time of each message is not periodic at the source-end CAN subsystem, such that the in-gateway arrival time of each message is not periodic.

In summarization, if the minimum in-gateway inter-arrival time $T_j^{\min}$ is considered as the in-gateway period, then the worst-case latency obtained by Eq. (8) is safe but quite pessimistic.

C. Tightest Message Arrival Pattern

Theorem 1: If the in-gateway interval time between $m_{j,1}$ and $m_{j,2}$ is the minimum in-gateway inter-arrival time of m_j, namely,

$$I_{j,1,2} = T_j^{\min},$$

then the minimum in-gateway interval time between subsequent two adjacent jobs is T_j, namely,

$$I_{j,x,x+1}^{\min} = T_j \quad (x \geqslant 2).$$

Proof. According to the content described for the establishment of Eq. (6), the minimum in-gateway inter-arrival time of m_j appears on the premises that: 1) $m_{j,1}$ experiences source-end WCRT (R_j^{source}) in its source-end subsystem; 2) $m_{j,2}$ experiences WCET (C_j) in its source-end subsystem. (i.e., $m_{j,2}$ begins its transmission in the $e2e\&e2g$ bus without any interference and blocking). In other words, once $m_{j,2}$ is released in its ECU, it will immediately start to be transmitted in the $e2e\&e2g$ bus (i.e., the released time of $m_{j,2}$ in its ECU is equal to its actual start transmission time on $e2e\&e2g$ bus). To minimize the in-gateway interval time between $m_{j,2}$ and $m_{j,3}$, $m_{j,3}$ must also be in a state of being transmitted once released (i.e., the minimum in-gateway interval time between $m_{j,2}$ and $m_{j,3}$ is their period of T_j); otherwise, any possible interference and blocking to $m_{j,3}$ will cause its actual start transmission time to be delayed. By analogy, all subsequent jobs ($m_{j,4}$, $m_{j,5}$ $m_{j,5}$,...) must also be the same state of being transmitted once released as $m_{j,3}$. Therefore, the minimum in-gateway interval time between subsequent two adjacent jobs is T_j. ■

Fig. 3 shows the schematic diagram of obtaining the minimum in-gateway interval time between $m_{8,2}$ and $m_{8,3}$: 1) at instant of 1,130 μs, $m_{8,1}$ arrives at the gateway; 2) at instant of 3,270 μs (3,000 + 270 = 3,270), $m_{8,2}$ arrives at the gateway; and 3) at instant of 6,270 μs (3,270 + 3,000 = 6,270), $m_{8,3}$ arrives at the gateway. Finally, the minimum in-gateway interval time between $m_{8,2}$ and $m_{8,3}$ is 3000 μs, as shown in Fig. 3.

Definition 7: **Tightest message arrival pattern:** If the in-gateway interval time between $m_{j,1}$ and $m_{j,2}$ is $T_j^{\min}$ and the in-gateway interval time between subsequent two adjacent jobs is T_j, then such message arrival pattern is called the tightest message arrival pattern, which is denoted by

$$Pattern_j^{\text{tight}} = (0,\ T_j^{\min},\ T_j^{\min} + T_j,\ T_j^{\min} + 2 \times T_j). \quad (10)$$

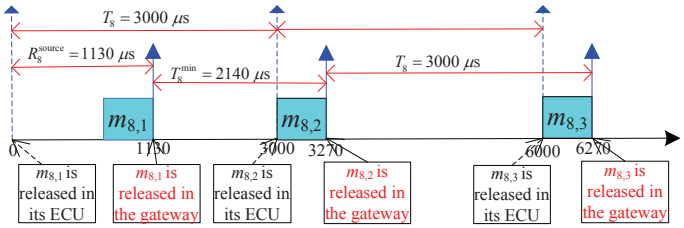

Fig. 3: Minimum in-gateway interval time between $m_{8,2}$ and $m_{8,3}$ in the gateway.

Definition 7 bounds how the jobs of one message arrive. Note that due to the bus competition among multiple messages, the tightest message arrival patterns of some messages are not necessarily actual existing patterns, but may be virtual patterns; therefore, the tightest message arrival pattern is still conservative.

Theorem 2: The tightest message arrival pattern $Pattern_j^{\text{tight}} = (0,\ T_j^{\min},\ T_j^{\min} + T_j,\ T_j^{\min} + 2 \times T_j, ...)$ can ensure that the number of in-gateway arrival jobs of message m_j in any time interval is the maximum.

Proof. The in-gateway interval time between $m_{j,1}$ and $m_{j,2}$ under the tightest message arrival pattern $Pattern_j^{\text{tight}}$ is $T_j^{\min}$, whereas that under other actual existing patterns is larger than $T_j^{\min}$. Assume that the given time interval is T^{interval}, and it has the following multiple cases.

Case 1. $\boldsymbol{T^{\text{interval}} < T_j^{\min}}$. First, there is only one in-gateway arrival job of m_j in time interval T^{interval} under the tightest message arrival pattern. Second, since T^{interval} is less than $T_j^{\min}$, there is at most one in-gateway arrival job of m_j in T^{interval} under other actual existing patterns. Therefore, under the tightest message arrival pattern, m_j has the maximum number of in-gateway arrival jobs.

Case 2. $\boldsymbol{T_j^{\min} \leqslant T^{\text{interval}} < T_j^{\min} + T_j}$. First, there are two in-gateway arrival jobs of m_j in time interval T^{interval} under the tightest message arrival pattern $Pattern_j^{\text{tight}}$. Second, we use the counter-evidence method and assume that there are three in-gateway arrival jobs of message m_j in T^{interval} in another actual existing pattern. As the in-gateway interval time between $m_{j,x}$ and $m_{j,x+1}$ in other actual existing pattern is larger than $T_j^{\min}$, two in-gateway jobs can arrival in interval $I_{j,x,x+1}$; however, the in-gateway interval time between $m_{j,x+1}$ and $m_{j,x+2}$ must be less than T_j because $m_{j,x+1}$ is not in the state of being transmitted once released, such that no job can arrival in interval $I_{j,x+1,x+2}$. Therefore, under other actual existing patterns, m_j has at most two in-gateway arrival jobs.

Case 3. $\boldsymbol{T_j^{\min} + T_j \leqslant T^{\text{interval}} < T_j^{\min} + 2 \times T_j}$. First, there are three in-gateway arrival jobs of message m_j in time interval T^{interval} under the tightest message arrival pattern $Pattern_j^{\text{tight}}$. Second, we use the counter-evidence method and assume that there are four in-gateway arrival jobs of message m_j in T^{interval} in another actual existing pattern. Similar to the proof in Case 2, m_j has at most three in-gateway arrival jobs under other actual existing patterns.

Default. The other cases can use the same proof as Cases 2 and 3. ■

Note that Theorem 1 corresponds to the tightest message arrival pattern, as explained in Theorem 2.

Theorem 3: If high-priority gateway messages arrive at the gateway under the individual tightest message arrival patterns, and the first in-gateway jobs of all the high-priority gateway messages simultaneously arrival at the gateway, then the worst-case latency for the low-priority analysed message is generated.

Proof. According to Theorem 2, the tightest message arrival pattern of each high-priority gateway message can ensure that the number of in-gateway arrival jobs of each message in any time interval is the maximum compared with other actual existing patterns of this message. Then, according to the principle of accumulation, the tightest message arrival patterns of all high-priority gateway messages can ensure that the number of in-gateway arrival jobs of all these messages in any time interval is the maximum compared with other actual existing patterns of these messages. Therefore, the worst-case latency of the low-priority analyzed message is the time interval, in which high-priority messages cause successive interference to the analyzed message. ■

D. Motivating Example

Based on the already obtained source-end WCRTs (i.e., R_i^{source}) of gateway messages in Table I, we can easily obtain their minimum in-gateway inter-arrival time (i.e., T_j^{min}) values of five gateway messages, namely, T_2^{min} = 730 μs, T_4^{min} = 1320 μs, T_6^{min} = 1050 μs, T_8^{min} = 2140 μs, and T_{10}^{min} = 1720 μs.

Then, according to Eq. (10), we can obtain the tightest message arrival patterns of four messages as follows (for simplicity, we only list the first 4 in-gateway arrival instants for each message).

$$\begin{cases} Pattern_2^{\text{tight}} = \{0\ \mu\text{s},\ 730\ \mu\text{s},\ 1730\ \mu\text{s},\ 2730\ \mu\text{s}\} \\ Pattern_4^{\text{tight}} = \{0\ \mu\text{s},\ 1320\ \mu\text{s},\ 3120\ \mu\text{s},\ 4920\ \mu\text{s}\} \\ Pattern_6^{\text{tight}} = \{0\ \mu\text{s},\ 1050\ \mu\text{s},\ 2750\ \mu\text{s},\ 4450\ \mu\text{s}\} \\ Pattern_8^{\text{tight}} = \{0\ \mu\text{s},\ 2140\ \mu\text{s},\ 5140\ \mu\text{s},\ 8140\ \mu\text{s}\}. \end{cases}$$

According to Theorem 3, when m_{10} and m_2, m_4, m_6, m_8 simultaneously arrive at the gateway, m_{10} will generate the in-gateway worst-case latency. However, the in-gateway worst-case latency occurring in the above scenario is safe but pessimistic. The reason is that it is impossible for m_{10} to simultaneously arrive at the gateway as any message in m_2, m_4, m_6, and m_8 due to the serial transmission in the $e2e\&e2g$ CAN bus of the source-end subsystem.

Fortunately, since m_2, m_4, m_6, m_8 and m_{10} can be simultaneously released in individual source-end ECUs, we can obtain individual source-end WCRTs of the above messages. That is, $m_{i,1}$ (including $m_{2,1}$, $m_{4,1}$, $m_{6,1}$, and $m_{8,1}$) experiences source-end WCRT (R_i^{source}) in its source-end subsystem, such that $m_{i,2}$ (including $m_{2,2}$, $m_{4,2}$, $m_{6,2}$, and $m_{8,2}$) would experience WCET (i.e., C_i) in its source-end subsystem.

E. Earliest In-Gateway Arrival Sequence

Definition 8: **Earliest in-gateway arrival instant:** The earliest in-gateway arrival instant of high-priority job $m_{j,1}$ is the instant relative to the in-gateway arrival instant of the analyzed message m_i, and is calculated by

$$RI_j^i(1) = C_i + \sum_{m_k \in hp_{\text{source-GW}}(m_j)} C_k, \tag{11}$$

where $hp_{\text{source-GW}}(m_i)$ represents the source-end high-priority gateway messages of m_i.

Definition 8 indicates that the earliest in-gateway arrival instant is relative to the in-gateway arrival instant of the analyzed message. In the motivating example, we can get the earliest in-gateway arrival instant of $m_{j,1}$ (including $m_{2,1}$, $m_{4,1}$, $m_{6,1}$, and $m_{8,1}$) through Eq. (11). For instance, when m_{10} (the analyzed message) arrives at the gateway at instant of 0, the earliest in-gateway arrival instants of the first jobs $m_{2,1}$, $m_{4,1}$, $m_{6,1}$, and $m_{8,1}$ are 210 μs, 420 μs, 590 μs, and 800 μs, respectively, as shown in Fig. 4.

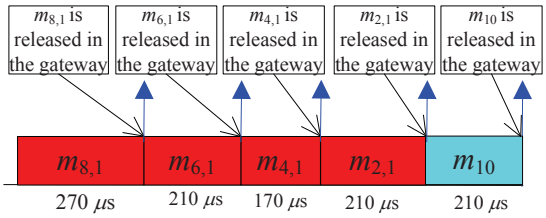

Fig. 4: Earliest in-gateway arrival instants of high-priority gateway messages when m_{10} arrivals at the gateway at instant of 0.

Note that due to the bus competition among multiple messages, the earliest in-gateway arrival instants of some messages do not necessarily actually exist. For instance, $m_{2,2}$ may be transmitted earlier than $m_{4,1}$ (i.e., $m_{4,1}$ may be continuously interfered by $m_{2,1}$ and $m_{2,2}$); therefore, the earliest in-gateway arrival instant is conservative.

Combining Eqs. (10) and (11), we can obtain the earliest in-gateway arrival instants of m_j's multiple jobs shown in Eq. (12) (for simplicity, we only list first 4 arrival instants of each message).

$$RI_j^i = \begin{cases} RI_j^i(1) = C_i + \sum\limits_{m_k \in hp_{\text{source-GW}}(m_j)} C_k, \\ RI_j^i(2) = C_i + \sum\limits_{m_k \in hp_{\text{source-GW}}(m_j)} C_k + T_j^{\min}, \\ RI_j^i(3) = C_i + \sum\limits_{m_k \in hp_{\text{source-GW}}(m_j)} C_k + T_j^{\min} + T_j, \\ RI_j^i(4) = C_i + \sum\limits_{m_k \in hp_{\text{source-GW}}(m_j)} C_k + T_j^{\min} + 2 \times T_j. \end{cases} \tag{12}$$

On the basis of Eq. (12), when m_{10} arrives at the gateway at instant 0, the earliest in-gateway arrival instants of messages $m_{2,1}$, $m_{4,1}$, $m_{6,1}$, and $m_{8,1}$ are listed in Eq. (13).

$$\begin{cases} RI_2^{10} = \{210\ \mu\text{s},\ 940\ \mu\text{s},\ 1940\ \mu\text{s},\ 2940\ \mu\text{s}\} \\ RI_4^{10} = \{420\ \mu\text{s},\ 1740\ \mu\text{s},\ 3540\ \mu\text{s},\ 5340\ \mu\text{s}\} \\ RI_6^{10} = \{590\ \mu\text{s},\ 1640\ \mu\text{s},\ 3340\ \mu\text{s},\ 5040\ \mu\text{s}\} \\ RI_8^{10} = \{800\ \mu\text{s},\ 2940\ \mu\text{s},\ 5940\ \mu\text{s},\ 8940\ \mu\text{s}\}. \end{cases} \tag{13}$$

Definition 9: **Earliest in-gateway arrival sequence:** The earliest in-gateway arrival sequence for the analyzed messages

consists of earliest in-gateway arrival instant of the 1st job of each high-priority message according to the ascending order of priority value.

Definition 9 bounds how the earliest jobs of all messages arrive. Similar to the earliest in-gateway arrival instant, the earliest in-gateway arrival sequence may not be real but conservative. Based on the fact that m_2, m_4, m_6, and m_8 are released simultaneously in the ECU of the source-end CAN subsystem, the earliest in-gateway arrival sequence for m_{10} is ($m_{2,1}$, $m_{4,1}$, $m_{6,1}$, $m_{8,1}$), as shown in Fig. 4.

Theorem 4: If high-priority gateway messages arrive at the gateway with the individual tightest message arrival patterns and the earliest in-gateway arrival sequence, then the worst-case latency for the low-priority analyzed message is generated.

Proof. First, according to Eq. (9), the blocking to m_i is $B_i^{\text{GW}} = \max_{h \in p_{\text{source-GW}}(i)} C_h$, which is larger than or equal to any WCTT of high-priority messages; therefore, all the jobs in the earliest in-gateway arrival sequence can cause inference to m_i. For instance, sequence ($m_{2,1}$, $m_{4,1}$, $m_{6,1}$, $m_{8,1}$) can cause successive inference to m_{10}. Second, according to Theorem 2, the tightest message arrival pattern of each message can ensure that the number of in-gateway arrival jobs of message in any time interval is the maximum compared with other actual existing patterns of this message. Therefore, the individual tightest message arrival patterns and the earliest in-gateway arrival sequence can generate worst-case latency for the low-priority analyzed message, and this worst-case latency is larger than or equal to awaiting times of all other actual existing patterns. ■

Note that the worst-case latency obtained through Theorem 4 is not necessarily a real worst-case latency, only if both individual tightest message arrival patterns and the earliest in-gateway arrival sequence are real. Regardless of whether the obtained worst-case latency actually exists, it is greater than or equal to all actual awaiting times.

V. Pointer Reachability Exploration

After the arrival of the messages at the gateway having been bounded in the previous section, it is used by PRE to derive the interfering message jobs and corresponding worst-case latency to the analyzed message in this section.

A. The PRE Algorithm

We propose a new method called PRE to calculate the in-gateway worst-case latency for a gateway message. The algorithm description is shown in Algorithm 1.

For the current analyzed message, each high-priority message has an exploratory pointer to its first job. The idea of PRE is to iteratively move all pointers forward to individual next jobs until the interference from all high-priority messages to the analyzed message is unreachable. We explain the PRE algorithm with an example.

(1) Lines 1–4 aim at initializing the job pointer (i.e., $pointer_j$) and interference flag (i.e., $iflag_j$) for each source-end high-priority gateway message m_j ($m_j \in hp_{\text{source-GW}}(m_i)$). $pointer_j$ always points to the latest in-gateway arrival job of m_j that has caused inference with m_i, whereas $iflag_j$ continuously determines whether the jobs in m_j can still interfere with m_i. For instance, when analyzing m_{10}, we have $pointer_2$ = 1, $pointer_4$ = 1, $pointer_6$ = 1, and $pointer_8$ = 1; $iflag_2$ = false, $iflag_4$ = false, $iflag_6$ = false, and $iflag_8$ = false.

Algorithm 1 The PRE Algorithm

```
1: for (m_j ∈ hp_source-GW(m_i)) do
2:     pointer_j ← 1;
3:     iflag_j ← false;
4: end for
5: B_i^GW ← max_{h ∈ p_source-GW(i)} C_h;
6: L_i^GW ← B_i^GW;
7: while (true) do
8:     for (m_j ∈ hp_source-GW(m_i)) do
9:         if (RI_j^i(pointer_j) ⩽ L_i^GW) then
10:            L_i^GW ← L_i^GW + C_j;
11:            pointer_j++;
12:            iflag_j ← true;
13:        else
14:            iflag_j ← false;
15:        end if
16:    end for
17:    iflag_count ← 0;
18:    for (m_j ∈ hp_source-GW(m_i)) do
19:        if (iflag_j == true) then
20:            iflag_false_count++;
21:        end if
22:    end for
23:    if (iflag_false_count == 0) then
24:        return L_i^GW;
25:    end if
26: end while
```

(2) In Line 5, we obtain the in-gateway blocking time of m_i. When the analyzed message m_i arrives at the gateway at instant of 0, the $g2e$ bus of the destination-end subsystem is just occupied and transmitted by another message m_h ($m_h \in sp_{\text{source-GW}}(m_i)$), which is any message from the same source-end subsystem as m_i. Therefore, the in-gateway blocking time of m_i is

$$B_i^{\text{GW}} = \max_{m_h \in sp_{\text{source-GW}}(m_i)} C_h.$$

Line 6 sets the initial awaiting time (i.e., L_i^{GW}) of m_i as B_i^{GW}. For instance, the in-gateway blocking time and the initial awaiting time of m_{10} is 270 μs.

(3) Lines 7 - 26 are the core of the algorithm. In the **while** loop, we have two **for** loops in series.

Step 1: In the 1st **for** loop (Lines 8 - 16), each source-end high-priority gateway message m_j ($m_j \in hp_{\text{source-GW}}(m_i)$) is judged whether its $pointer_j$th arrival instant (i.e., $RI_j^i(pointer_j)$) pointed by $pointer_j$ is within current interfering latency (i.e., L_i^{GW}). If the result is true, it indicates that the job pointed by $pointer_j$ can cause interference to m_i, thereby increasing the w_i^{source} value by C_j; and $pointer_j$ points to the next job (i.e., $pointer_j$ ++), and the interference flag (i.e., $iflag_j$) is set to true; otherwise, we merely set $iflag_j$ to false.

Step 2: The 2nd **for** loop (Lines 18 - 22) counts the number of source-end high-priority messages that interfere with m_i in the 1st **for** loop, and the number is recorded by $iflag\_false\_count$.

Step 3: In Lines 23 - 25, if $iflag\_false\_count$ is equal to 0 (i.e., all source-end high-priority messages would not cause interference with m_i), then we set the in-gateway worst-case latency as w_i^{source} + C_i, and the algorithm returns; otherwise, the **while** loop continues.

B. Motivating Example through PRE

For instance, when analyzing m_{10}, the job pointer (i.e., $pointer_j$) and interference flag (i.e., $iflag_j$) values of each source-end high-priority gateway message will change, as shown in Fig. 5.

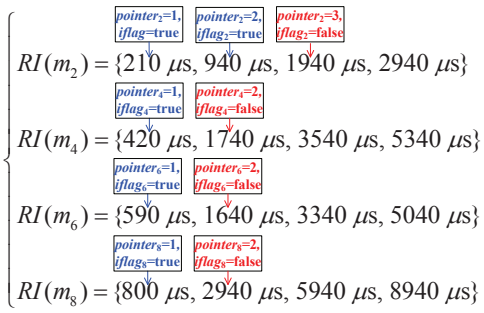

Fig. 5: Job pointer (i.e., $pointer_j$) and interference flag (i.e., $iflag_j$) values change for each source-end high-priority gateway message.

In Fig. 5, when $pointer_2$ = 3, $pointer_4$ = 2, $pointer_6$ = 2, and $pointer_8$ = 2, we have $iflag_2$ = false, $iflag_4$ = false, $iflag_6$ = false, and $iflag_8$ = false. Therefore, $iflag\_false\_count$ is equal to 0 in this case, and the current awaiting time of m_{10} is gradually increased by adding 210 μs (C_2), 170 μs (C_4), 210 μs (C_6), 270 μs (C_8), and 210 μs (C_2). That is, w_{10}^{source} is increased to

$$\begin{aligned} L_{10}^{\text{GW}} &= B_{10} + C_2 + C_4 + C_6 + C_8 + C_2 \\ &= 270 + 210 + 170 + 210 + 270 + 210 \\ &= 1340\ \mu\text{s}. \end{aligned}$$

As shown in Table II, the in-gateway deadline of m_{10} is 1300 μs (calculated by Eq. (5)). Therefore, the in-gateway worst-case latency of m_{10} exceeds its in-gateway deadline, such that m_{10} may violate its end-to-end deadline during transmission.

TABLE II: In-gateway worst-case latencies of messages in the motivating example through PRE and TPA.

m_i	D_i^{GW}	PRE (Algorithm 1)			TPA (Algorithm 2)		
		Priority	L_i^{GW}	schedulable ?	Priority	L_i^{GW}	schedulable ?
m_2	310 μs	2	270 μs	yes	2	270 μs	yes
m_4	980 μs	4	480 μs	yes	6	690 μs	yes
m_6	630 μs	6	650 μs	no	4	480 μs	yes
m_8	1600 μs	8	860 μs	yes	10	1280 μs	yes
m_{10}	1300 μs	10	1340 μs	no	8	860 μs	yes

Let $M_{\text{source-GW}}(CAN_x)$ represent all source-end gateway messages sent from subsystem CAN_x, and let $M_{\text{source-GW}}^{\text{schedulable}}(CAN_x)$ represent the source-end schedulable gateway messages sent from subsystem CAN_x. Then, the acceptance ratio of $M_{\text{source-GW}}(CAN_1)$ messages (including m_2, m_4, m_6, m_8, and m_{10}) through PRE is

$$AR(M_{\text{source-GW}}(CAN_1)) = \frac{|M_{\text{source-GW}}^{\text{schedulable}}(CAN_1)|}{|M_{\text{source-GW}}(CAN_1)|} = \frac{3}{5} = 0.6.$$

VI. Targeted Priority Assignment

Considering that the worst-case latencies of gateway messages obtained by PRE may exceed their individual in-gateway deadlines, priority re-assignments of messages are proposed to enhance the schedulability in this section.

A. The TPA Algorithm

The Optimal Priority Assignment (OPA) method proposed by Audsley [13] can address the problem of the priority re-assignments of messages. However, OPA cannot be applied to the schedulability enhancement for the gateway. First, OPA is an arbitrary priority assignment, which means that the priority of each message can be arbitrarily changed as long as it is schedulable. Second, OPA is terminated when facing unschedulability.

Since PRE has determined which messages are unschedulable, we propose the TPA algorithm that targets at the priorities assigned at the CAN bus and runs a reordering at the gateway to enhance the schedulability. The algorithm description of TPA is shown in Algorithm 2.

Algorithm 2 The TPA Algorithm

```
1: remaining_message_set ← M_source-GW(CAN_x);
2: remaining_priority_set ← M_source-GW(CAN_x).priority();
3: schedulable_message_set ← NULL;
4: unschedulable_message_set ← NULL;
5: Sort the messages in remaining_message_set according to their the descending order of identifiers;
6: while (remaining_priority_set is not NULL) do
7:     priority ← remaining_priority_set.get();
8:     m_i ← remaining_message_set.get();
9:     if (priority == i&&L_i^GW ⩽ D_i^GW) then
10:        remaining_priority_set.remove(priority);
11:        remaining_message_set.remove(m_i);
12:        schedulable_message_set.add(m_i);
13:        continue;
14:    else
15:        Boolean schedulable = false;
16:        for (m_j ∈ remaining_message_set && i! = j) do
17:            Assume that m_j has the lowest priority in the remaining_message_set;
18:            Calculate L_j^GW through PRE (Algorithm 1);
19:            if (L_j^GW ⩽ D_j^GW) then
20:                schedulable = true;
21:                Assign priority to m_j;
22:                remaining_priority_set.remove(priority);
23:                remaining_message_set.remove(m_j);
24:                schedulable_message_set.add(m_j);
25:                break;
26:            end if
27:        end for
28:        if (schedulable == false) then
29:            Give up the real-time guarantee for m_i by setting the lowest priority to m_i;
30:            remaining_priority_set.remove(priority);
31:            remaining_message_set.remove(m_i);
32:            unschedulable_message_set.add(m_i);
33:        end if
34:    end if
35: end while
```

TPA tries to make unschedulable messages schedulable with limited priority reordering, and gives up (assigning the

lowest priority) if a message inevitably compromises the schedulability of the entire system. The details are below.

(1) Lines 1 - 4 provide four sets, including 1) $remaining\_message\_set$ contains messages that are not determined through TPA; 2) $remaining\_priority\_set$ contains priorities that are not determined through TPA; 3) $schedulable\_message\_set$ contains messages that have already been determined to be schedulable through TPA; and 4) $unschedulable\_message\_set$ contains messages that have already been determined to be unschedulable by TPA. In the initial state, $remaining\_message\_set$ is equal to $M_{\text{source-GW}}(CAN_x)$, and the priorities in $remaining\_priority\_set$ corresponds to the messages in $remaining\_message\_set$. Line 5 sorts the messages in $remaining\_message\_set$ according to the descending order of identifiers. For instance, considering the gateway messages in $M_{\text{source-GW}}(CAN_1)$ is (m_2, m_4, m_6, m_8, m_{10}), we have $remaining\_message\_set$ = (m_{10}, m_8, m_6, m_4, m_2) and $remaining\_priority\_set$ = (10, 8, 6, 4, 2).

(2) Lines 6 - 34 loop all priorities in $remaining\_priority\_set$. For instance, priories 10, 8, 6, 4, and 2 in $remaining\_priority\_set$ are accessed once, as shown in the first column of Table III.

Step (1): Lines 7 and 8 get a $priority$ and a message m_i from $remaining\_priority\_set$ and $remaining\_message\_set$, respectively. Considering the motivating example, TPA first get priority 10 and m_{10} from $remaining\_priority\_set$ and $remaining\_message\_set$, respectively.

Step (2): If $priority == i$ and m_i's in-gateway worst-case latency is within its in-gateway deadline (i.e., $L_i^{\text{GW}} \leqslant D_i^{\text{GW}}$), then priority re-assignment is not needed and TPA just removes m_i from $remaining\_message\_set$ and adds it to the $schedulable\_message\_set$ (Line 9 - 14).

Step (3): If Step (2) is not met, then TPA tries to find a new message m_j from $remaining\_message\_set$, and assumes m_j to have the lowest priority in $remaining\_message\_set$ until m_j's in-gateway worst-case latency is within its in-gateway deadline (i.e., $L_j^{\text{GW}} \leqslant D_j^{\text{GW}}$); correspondingly, TPA removes m_j from $remaining\_message\_set$ and adds it to the $schedulable\_message\_set$ (Lines 16 - 27). Considering the motivating example, priority 10 cannot be assigned to m_{10}, because L_{10}^{GW} exceeds D_{10}^{GW} shown in Table II; therefore, priority 10 is re-assigned to m_8 because $L_8^{\text{GW}} \leqslant D_8^{\text{GW}}$ (i.e., 860 μs $<$ 1,300 μs), as shown in Table III. In short, we choose the lowest-priority message in $remaining\_message\_set$. We hope this lowest-priority message satisfies its deadline if the remaining lowest priority is assigned to it. If the deadline is not satisfied, we look for another low-priority message that can take this remaining lowest priority.

Step (4): If Step (3) does not find the new message m_j, then TPA gives up the real-time guarantee for m_i by setting the lowest priority to m_i; correspondingly, TPA removes m_i from $remaining\_message\_set$ and adds it to the $unschedulable\_message\_set$ (Lines 28 - 33).

TABLE III: Process of improving acceptance ratio through TPA for the motivating example.

Assigned priority	Remaining messages	Corresponding message	In-gateway worst-case latency	In-gateway deadline
10	m_{10}, m_8, m_6, m_4, m_2	m_8	1,280 μs	1,660 μs
8	m_{10}, m_6, m_4, m_2	m_{10}	860 μs	1,300 μs
6	m_6, m_4, m_2	m_4	690 μs	980 μs
4	m_6, m_2	m_6	480 μs	630μs
2	m_2	m_2	270 μs	310 μs

B. Motivating Example through TPA

After the priority 10 is assigned to m_8, the motivating example is continued in terms of Table III.

(1) We consider the priority 8. TPA tries to find a new message from $remaining\_message\_set$ = (m_{10}, m_6, m_4, m_2). Given that m_{10}'s in-gateway worst-case latency is within its in-gateway deadline, the priority 8 is assigned to m_{10}, and the $remaining\_message\_set$ is updated to = (m_6, m_4, m_2).

(2) We further consider the priority 6. TPA tries to find a new message from $remaining\_message\_set$ = (m_6, m_4, m_2). Given that m_4's in-gateway worst-case latency is within its in-gateway deadline, the priority 6 is assigned to m_4, and the $remaining\_message\_set$ is updated to = (m_6, m_2).

(3) We further consider the priority 4. TPA tries to find a new message from $remaining\_message\_set$ = (m_6, m_2). Given that m_6's in-gateway worst-case latency is within its in-gateway deadline, the priority 4 is assigned to m_6, and the $remaining\_message\_set$ is updated to = (m_2).

(4) We further consider the priority 2, which is equal to the identifier of m_2 and m_2's in-gateway worst-case latency is within its in-gateway deadline, priority re-assignment is not needed.

Finally, all the five messages in $seq_{\text{source-GW}}(CAN_1)$ = (m_2, m_6, m_4, m_{10}, m_8) have met their individual in-gateway deadlines through TPA. Therefore, the acceptance ratio of $M_{\text{source-GW}}(CAN_1)$ messages (including m_2, m_4, m_6, m_8, and m_{10}) through TPA (Algorithm 2) is

$$AR(M_{\text{source-GW}}(CAN_1)) = \frac{|M_{\text{source-GW}}^{\text{schedulable}}(CAN_1)|}{|M_{\text{source-GW}}(CAN_1)|} = \frac{5}{5} = 1.0.$$

For the motivating example, the detailed result comparison between PRE and TPA is shown in Table II.

VII. Performance Evaluation

A. Real-Life CAN Message Set with 64 Messages

In this section, we adopt a real-life CAN message set with 64 messages from an automotive manufacturer. The priority, WCET, and deadline values of each message in the real-life CAN message set are shown in Table IV. These 64 original messages are collected from a separate subsystem and are not gateway messages. For the purposes of this study, we treat these 64 messages as gateway messages, which are all sent from the CAN_1 subsystem to the CAN_2 subsystem.

(1) In-gateway deadlines of gateway messages.

Through the introduction of Section III.C, we can easily obtain the source-end WCRT (R_i^{source}) and destination-end

TABLE IV: Real-life CAN message set with 64 messages.

m_i	P_i	C_i	T_i (D_i)	m_i	P_i	C_i	T_i (D_i)
m_1	1	230 μs	10,000 μs	m_{33}	33	270 μs	100,000 μs
m_2	2	210 μs	10,000 μs	m_{34}	34	230 μs	100,000 μs
m_3	3	250 μs	10,000 μs	m_{35}	35	190 μs	200,000 μs
m_4	4	170 μs	10,000 μs	m_{36}	36	210 μs	1,000,000 μs
m_5	5	250 μs	10,000 μs	m_{37}	37	250 μs	12,000 μs
m_6	6	190 μs	25,000 μs	m_{38}	38	150 μs	100,000 μs
m_7	7	270 μs	100,000 μs	m_{39}	39	210 μs	1,000,000 μs
m_8	8	270 μs	100,000 μs	m_{40}	40	150 μs	15,000 μs
m_9	9	270 μs	100,000 μs	m_{41}	41	270 μs	15,000 μs
m_{10}	10	250 μs	100,000 μs	m_{42}	42	150 μs	14,000 μs
m_{11}	11	210 μs	1,000,000 μs	m_{43}	43	150 μs	20,000 μs
m_{12}	12	270 μs	100000 μs	m_{44}	44	150 μs	20,000 μs
m_{13}	13	270 μs	100,000 μs	m_{45}	45	210 μs	20,000 μs
m_{14}	14	270 μs	200,000 μs	m_{46}	46	270 μs	50,000 μs
m_{15}	15	210 μs	1,000,000 μs	m_{47}	47	270 μs	50,000 μs
m_{16}	16	270 μs	10,000 μs	m_{48}	48	270 μs	100,000 μs
m_{17}	17	250 μs	10,000 μs	m_{49}	49	190 μs	100,000 μs
m_{18}	18	270 μs	100,000 μs	m_{50}	50	190 μs	100,000 μs
m_{19}	19	270 μs	100,000 μs	m_{51}	51	210 μs	1,000,000 μs
m_{20}	20	270 μs	100,000 μs	m_{52}	52	150 μs	25,000 μs
m_{21}	21	170 μs	100,000 μs	m_{53}	53	190 μs	100,000 μs
m_{22}	22	210 μs	1,000,000 μs	m_{54}	54	210 μs	1,000,000 μs
m_{23}	23	270 μs	10,000 μs	m_{55}	55	150 μs	25,000 μs
m_{24}	24	170 μs	100,000 μs	m_{56}	56	210 μs	31,000 μs
m_{25}	25	270 μs	100,000 μs	m_{57}	57	170 μs	32,000 μs
m_{26}	26	270 μs	100,000 μs	m_{58}	58	210 μs	33,000 μs
m_{27}	27	210 μs	100,000 μs	m_{59}	59	190 μs	33,000 μs
m_{28}	28	210 μs	1,000,000 μs	m_{60}	60	210 μs	33,000 μs
m_{29}	29	270 μs	100,000 μs	m_{61}	61	270 μs	34,000 μs
m_{30}	30	270 μs	100,000 μs	m_{62}	62	250 μs	34,000 μs
m_{31}	31	270 μs	100,000 μs	m_{63}	63	210 μs	36,000 μs
m_{32}	32	210 μs	1,000,000 μs	m_{64}	64	170 μs	36,000 μs

WCRT (R_i^{dest}) of each message, thereby getting its in-gateway deadline (D_i^{GW}) based on Eq. (5). The results are shown in Table V.

(2) In-gateway worst-case latency analysis through PRE.

Through the Davis' timing analysis (Davis for short) [8] and PRE, we can obtain the worst-case latency of each message, as shown in Table V. By comparing the in-gateway deadline and worst-case latency of each message, we find that 19 messages do not meet their individual in-gateway deadlines (i.e., unschedulable) through the Davis' timing analysis, such that the acceptance ratio is 19/64= 70.31%. Compared with the Davis' timing analysis, 10 messages (including m_{23}, m_{37}, m_{40}, m_{41}, m_{42}, m_{43}, m_{44}, m_{45}, m_{52}, and m_{55}) are unschedulable through PRE, such that the acceptance ratio is increased to 54/64= 84.38%.

The fundamental reasons why the in-gateway worst-case latencies obtained through PRE are much tighter than the Davis' timing analysis are: 1) PRE obtains the earliest in-gateway arrival instants of each gateway message as much as possible based on the actual situation inside the gateway; 2) PRE does not treat the minimum in-gateway inter-arrival time of each message as its in-gateway period.

(3) Acceptance ratio improvement through TPA.

To improve the acceptance ratio, we attempt to re-assign the priorities of these messages by adopting DMPO (Deadline Monotonic Priority Ordering) [12] and TPA, respectively. DMPO assigns higher priorities to messages with shorter in-gateway deadlines in this paper. The in-gateway deadlines of 64 messages have been obtained and are shown in Table V.

The priorities of 46 messages are re-assigned through TPA, and the priorities of messages m_{56} - m_{64} and m_1 - m_9 are not changed; the reason is that the in-gateway worst-case latencies of these messages satisfy the condition that doe not require priority re-assignment (i.e., Line 9 - 13 of Algorithm 1). This phenomenon reflects the targeted advantage of TPA. As a comparison, the priorities of all 64 messages are re-assigned through DMPO. Finally, 64 messages are all schedulable through DMPO or TPA, such that the acceptance rate is increased from 84.375% (through PRE) to 100%.

(4) Gateway prototype platform.

To illustrate the practical applicability of PRE and TPA in the actual platform, we implement the gateway prototype platform (Fig. 6), which uses STM32F407VET6 [19] provided by STMicroelectronics as the motherboard of both ECUs and the gateway. In the gateway prototype platform, PRE and TPA are developed for dynamic use to improve the acceptance ratio and can also be deployed statically to provide timing guarantees. In Fig. 6, ECU1 and ECU2 are message senders in the source-end CAN subsystem, and ECU3 is the message receiver in the destination-end subsystem. The collected data are uploaded to the host computer through the serial port conversion module.

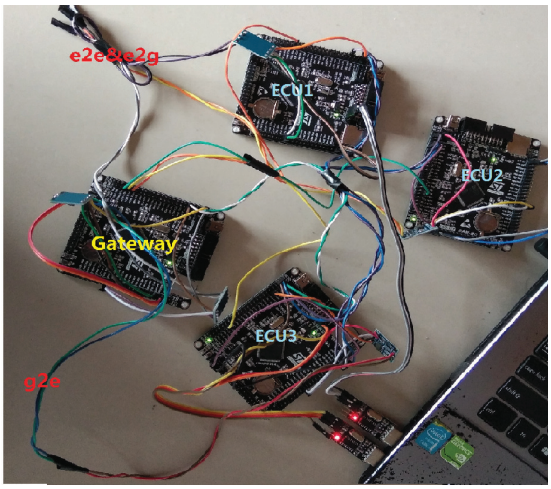

Fig. 6: Gateway prototype platform.

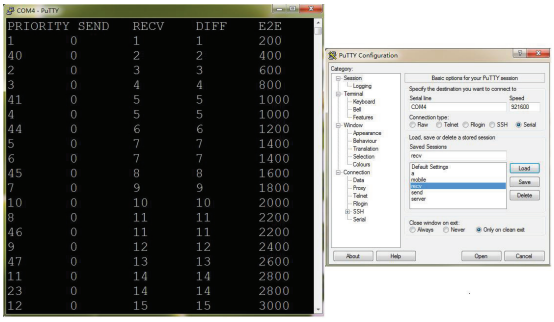

Fig. 7: Message information from the receiving ECU in the gateway prototype platform.

TABLE V: In-gateway deadlines, worst-case latencies, and acceptance ratios of 64 messages.

m_i	P_i	D_i^{GW}	Davis [8]		PRE		m_i	P_i	D_i^{GW}	Davis [8]		PRE	
			L_i^{GW}	Schedulable ?	L_i^{GW}	Schedulable ?				L_i^{GW}	Schedulable ?	L_i^{GW}	Schedulable ?
m_1	1	9,270 μs	270 μs	yes	270 μs	yes	m_{33}	33	91,470μs	7,990 μs	yes	7,990 μs	yes
m_2	2	9,080 μs	500 μs	yes	500 μs	yes	m_{34}	34	91,280 μs	8,260 μs	yes	500 μs	yes
m_3	3	8,790 μs	710 μs	yes	710 μs	yes	m_{35}	35	191,130 μs	8,490 μs	yes	8,490 μs	yes
m_4	4	8,700 μs	960 μs	yes	960 μs	yes	m_{36}	36	990,900 μs	8,680 μs	yes	8,680 μs	yes
m_5	5	8,370 μs	1,130 μs	yes	1,130 μs	yes	m_{37}	37	2,610 μs	8,890 μs	**no**	8,890 μs	**no**
m_6	6	23,240 μs	1,380 μs	yes	1,380 μs	yes	m_{38}	38	90,560 μs	9,140 μs	yes	9,140 μs	yes
m_7	7	97,890 μs	1,570 μs	yes	9,140 μs	yes	m_{39}	39	990,290 μs	9,290 μs	yes	9,290 μs	yes
m_8	8	97,620 μs	1,840 μs	yes	1,840 μs	yes	m_{40}	40	5,200 μs	9,500 μs	**no**	9,500 μs	**no**
m_9	9	97,350 μs	2,110 μs	yes	2,110 μs	yes	m_{41}	41	4,810 μs	9,650 μs	**no**	9,650 μs	**no**
m_{10}	10	97,120 μs	2,380 μs	yes	2,380 μs	yes	m_{42}	42	3,780 μs	11,820 μs	**no**	11,820 μs	**no**
m_{11}	11	996,950 μs	2,630 μs	yes	2,630 μs	yes	m_{43}	43	7,730 μs	12,220 μs	**no**	12,220 μs	**no**
m_{12}	12	96,620 μs	2,840 μs	yes	2,840 μs	yes	m_{44}	44	7,330 μs	12,370 μs	**no**	12,370 μs	**no**
m_{13}	13	96,350 μs	3,110 μs	yes	3,110 μs	yes	m_{45}	45	7,060 μs	12,520 μs	**no**	12,520 μs	**no**
m_{14}	14	196,080 μs	3,380 μs	yes	3,380 μs	yes	m_{46}	46	36,730μs	12,730 μs	yes	12,730 μs	yes
m_{15}	15	995,930 μs	3,650 μs	yes	3,650 μs	yes	m_{47}	47	36,460 μs	13,000 μs	yes	13,000 μs	yes
m_{16}	16	5,600 μs	3,860 μs	yes	3,860 μs	yes	m_{48}	48	86,190 μs	13,270 μs	yes	13,270 μs	yes
m_{17}	17	5,370 μs	4,130 μs	yes	13,270 μs	yes	m_{49}	49	86,080 μs	13,540 μs	yes	13,540 μs	yes
m_{18}	18	95,080 μs	4,380 μs	yes	4,380 μs	yes	m_{50}	50	85,890 μs	13,730 μs	yes	13,730 μs	yes
m_{19}	19	94,810 μs	4,650 μs	yes	4,650 μs	yes	m_{51}	51	985,660 μs	13,920 μs	yes	13,920 μs	yes
m_{20}	20	94,540 μs	4,920 μs	yes	4,920 μs	yes	m_{52}	52	10,420 μs	14,280 μs	**no**	14,280 μs	**no**
m_{21}	21	94,470 μs	5,190 μs	yes	5,190 μs	yes	m_{53}	53	85,190 μs	14,700 μs	yes	14,430 μs	yes
m_{22}	22	994,220 μs	5,360 μs	yes	5,360 μs	yes	m_{54}	54	984,960 μs	14,890 μs	yes	14,620 μs	yes
m_{23}	23	3,890 μs	5,570 μs	**no**	5,570 μs	**no**	m_{55}	55	9,870 μs	16,290 μs	**no**	14,830 μs	**no**
m_{24}	24	93,820 μs	5,840 μs	yes	5,840 μs	yes	m_{56}	56	15,600 μs	16,440 μs	**no**	15,400 μs	yes
m_{25}	25	93,450 μs	6,010 μs	yes	6,010 μs	yes	m_{57}	57	16,050 μs	16,650μs	**no**	15,610 μs	yes
m_{26}	26	93,180 μs	6,280μs	yes	6,280μs	yes	m_{58}	58	16,800 μs	16,820 μs	**no**	15,780 μs	yes
m_{27}	27	93,030 μs	6,550 μs	yes	15,780 μs	yes	m_{59}	59	16,630 μs	17,030 μs	**no**	15,990 μs	yes
m_{28}	28	992,820 μs	6,760 μs	yes	6,760 μs	yes	m_{60}	60	16,400 μs	17,220 μs	**no**	16,180 μs	yes
m_{29}	29	92,490 μs	6,970 μs	yes	6,970 μs	yes	m_{61}	61	17,070 μs	17,430 μs	**no**	16,390 μs	yes
m_{30}	30	92,220 μs	7,240 μs	yes	7,240 μs	yes	m_{62}	62	16,860μs	17,700 μs	**no**	16,660 μs	yes
m_{31}	31	91,950 μs	7,510 μs	yes	16,660 μs	yes	m_{63}	63	18,730 μs	20,240 μs	**no**	16,910 μs	yes
m_{32}	32	991,800 μs	7,780 μs	yes	7,780 μs	yes	m_{64}	64	18,640 μs	20,870 μs	**no**	17,120 μs	yes

Fig. 7 shows the message information from the receiving ECU in the gateway prototype platform. The time unit of each timer is 200 μs. The timers of the sending ECUs and the receiving ECU are incremented at the same frequency and synchronized after the system starts. The values of 1st column are the new priorities of messages; note that the ID of each message cannot be changed but its priority may be modified according to TPA when the message arrives at the gateway. The values of the 5th column are end-to-end response time values (unit: 200 μs) of messages. By observing the data, we find that there are no message distortions in the two sending ECUs of the message set. That is, each message job sent by ECU1 and ECU2 can ensure that the message is transferred safely to ECU3. In other words, the message set is actually schedulable in the gateway.

B. Scalable CAN Message Sets with 96 and 128 Messages

In practice, 64 messages for the inter-domain communication (across the gateway) are quite large. We further consider larger-scale CAN message sets in this work to show scalability. We directly copy top 32 and all 64 messages, respectively, in Table IV to form two new message sets. The only change in the new message set is the priority (ID) of each message starting from 65 and ending at 96 as well as starting from 65 and ending at 128. Finally, we form two scalable large-scale message sets with 96 and 128 messages, respectively.

(1) In-gateway worst-case latency analysis through PRE. Table VI shows the acceptance ratios through Davis' timing analysis and PRE on different scales. For the message set with 96 messages, there are 61 messages do not satisfy their individual in-gateway deadlines through the Davis' timing analysis, such that the acceptance ratio is merely (96 - 61)/96 $\approx$ 35.46%. The number of messages that misses deadlines is reduced to 6 through PRE; and the acceptance ratio is increased to (96 - 28)/96 $\approx$ 70.83%, which exceeds Davis' timing analysis by at least 30%. For the message set with 128 messages, the acceptance ratio through PRE still exceeds Davis' timing analysis by 30%.

TABLE VI: Acceptance ratios through four methods in different scales.

Scale	Davis [8]	PRE	DBMP [12]	TPA
64 messages	70.31%	84.38%	100%	100%
96 messages	36.46%	70.83%	68.75%	93.88%
128 messages	35.16%	65.63%	62.5%	78.13%

(2) Acceptance ratio improvement through TPA. Table VI shows that TPA can improve the acceptance ratio by 23% (message set with 96 messages) and 13% (message set with 128 messages) on the basis of PRE. However, DBMP does not achieve improvement in acceptance ratio, but a slight decrease in acceptance ratio. This unexpected result indicates that DBMP affects on the growth of the size of the message set negatively. According to the analysis in [20], DMPO is

optimal under the idealized message model, including that the message trigger mechanism must be strictly periodic, the deadline must be less than or equal to the period, and the message scheduling paradigm must be preemptive, etc. However, the trigger mechanism is not periodic, the deadline is arbitrary, and the scheduling paradigm is non-preemptive in the in-gateway; therefore, TPA performs better than DMPO in acceptance ratio improvement for the real-time gateway.

VIII. Conclusion

This paper develops a real-time CAN-CAN gateway with tighter worst-case latency analysis. First, our presented gateway architecture facilitates real-time analysis, which is different from the existing commercial gateways. Second, we obtain the safe and tight in-gateway worst-case latencies for messages through PRE. Third, we improve the acceptance ratio of message sets through TPA. Finally, the gateway prototype platform has been implemented using STM32F407VET6 provided by STMicroelectronics; we adopt the real-life and scalable CAN message sets to analyze the in-gateway worst-case latency and further improve the in-gateway acceptance ratio. The gateway can be easily extended to support multiple CAN subsystems if needed, because the awaiting priority queues in the gateway architecture are created in memory and can be dynamically adjusted in size.

References

[1] M. Di Natale, H. Zeng, P. Giusto, and A. Ghosal, *Understanding and using the Controller Area Network communication protocol: theory and practice*. Springer Science & Business Media, Jan. 2012.

[2] J. Taube, F. Hartwich, and H. Beikirch, "Comparison of can gateway modules for automotive and industrial control applications," in *Proceedings of the 10th international CAN Conference*, 2005.

[3] J. Sommer and R. Blind, "Optimized resource dimensioning in an embedded can-can gateway," in *Proceedings of the International Symposium on Industrial Embedded Systems (SIES'07)*. IEEE, 2007, pp. 55–62.

[4] M. Sojka, P. Píša, O. Špinka, and Z. Hanzálek, "Measurement automation and result processing in timing analysis of a linux-based can-to-can gateway," in *Proceedings of the IEEE 6th International Conference on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS'11)*, vol. 2. IEEE, Sep. 2011, pp. 963–968.

[5] E. Azketa, J. J. Gutiérrez, J. C. Palencia, M. G. Harbour, L. Almeida, and M. Marcos, "Schedulability analysis of multi-packet messages in segmented can," in *Proceedings of the IEEE 17th Conference on Emerging Technologies and Factory Automation (ETFA'12)*. IEEE, 2012, pp. 1–8.

[6] T. Wunsche, "Can/can-gateway cg-arm7/rmd user manual," Feb. 2018. [Online]. Available: https://www.traquair.com/pdfs/ems/manuals/cg-arm7-rmd_dr2_5.pdf

[7] "J1939 can gateway," 2016. [Online]. Available: https://www.electronicdesigninc.com/products/j1939-devices/j1939-can-gateway

[8] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien, "Controller area network (can) schedulability analysis: Refuted, revisited and revised," *Real-Time Systems*, vol. 35, no. 3, pp. 239–272, Apr. 2007.

[9] Y. Chen, R. Kurachi, G. Zeng, and H. Takada, "The worst-case response time analysis for fifo-based offset assigned can messages," *Journal of Information Processing*, vol. 20, no. 2, pp. 451–462, May 2012.

[10] R. I. Davis, S. Kollmann, V. Pollex, and F. Slomka, "Schedulability analysis for controller area network (can) with fifo queues priority queues and gateways," *Real-Time Systems*, vol. 49, no. 1, pp. 73–116, Jan. 2013.

[11] S. Mubeen, J. Makiturja, and M. Sjodin, "Extending worst case response-time analysis for mixed messages in controller area network with priority and fifo queues," *IEEE Access*, vol. 2, pp. 365–380, Apr. 2014.

[12] J. Y. T. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," *Performance Evaluation*, vol. 2, no. 4, pp. 237–250, Dec. 1982.

[13] N. C. Audsley, "On priority assignment in fixed priority scheduling," *Information Processing Letters*, vol. 79, no. 1, pp. 39–44, May 2001.

[14] S. Lee, D. Lee, M. Kim, and K. Lee, "Traffic-balancing algorithm for can systems with dual communication channels to enhance the network capacity," *International Journal of Automotive Technology*, vol. 11, no. 4, pp. 525–531, Aug. 2010.

[15] G. Xie, G. Zeng, R. Kurachi, H. Takada, Z. Li, R. Li, and K. Li, "Wcrt analysis of can messages in gateway-integrated in-vehicle networks," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 11, pp. 9623–9637, Nov. 2017.

[16] J. Rufino, "Redundant can architectures for dependable communication," *Technical Report CSTC Technical Report RT-98-02*, 1998.

[17] H. Hilmer, H. Kochs, and E. Dittmar, "A can-based architecture for highly reliable communication systems," in *Proc. Fifth CAN Conf*, 1998, pp. 6–10.

[18] P. Drazdil, "Amis-42700/amis-42770-redundant bus connection," SAE Technical Paper, Tech. Rep., 2005. [Online]. Available: https://www.onsemi.com/pub/Collateral/AND8348-D.PDF

[19] "Stm32f407ve," 2020. [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32f407ve.html

[20] S. Zhao, W. Chang, R. Wei, W. Liu, N. Guan, A. Burns, and A. Wellings, "Priority assignment on partitioned multiprocessor systems with shared resources," *IEEE Transactions on Computers*, DOI: 10.1109/TC.2020.3000051, June 2020.