Optimization of In-Vehicle Network Design

by

Taeju Park

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in the University of Michigan
2021

Doctoral Committee:

Professor Kang G. Shin, Chair
Professor Ella M. Atkins
Research Professor Emeritus Peter Honeyman
Associate Professor Alanson Sample

Taeju Park
taeju@umich.edu
ORCID iD: 0000-0002-1439-2709

To my wife Minjeong and my family Minsu, Myungok and Jieun

ACKNOWLEDGEMENTS

First and foremost, I thank my parents and my wife for their eternal love and support. Without their unswerving dedication, my PhD journey would have been more arduous and difficult.

I sincerely and greatly thank my advisor, Professor Kang G. Shin, for his support, patience, advice, and guidance. During my doctoral study, he has always guided me in the right direction and encouraged me to aim high. With his insightful guidance and extraordinary patience, I could grow up to be what I am today. Also, I would like to thank Professors Ella M. Atkins, Peter Honeyman, and Alanson Sample for serving on my dissertation committee and giving me valuable feedback on this thesis.

Being a member of Real-Time Computing Laboratory (RTCL) was a great fortune to me. RTCL members are always enthusiastic of solving real problems, and their attitude and passion motivated me greatly. I would like to thank all RTCL members; Huan Feng, Kassem Fawaz, Eugene Kim, Yu-Chih Tung, Kyong Tak Cho, Youngmoon Lee, Arun Ganesan, Dongyao Chen, Timothy Trippel, Juncheng Gu, Chun-Yu Chen, Mert D. Pese, Duc Bui, Hsun-Wei Cho, Youssef Tobah, Noah Curran, Wei-Lun Huang, Kyusuk Han, Liang He, Hoon Sung Chwa, Xiufeng Xie, Hamed Yousefi, Suining He, Haichuan Ding, and Jinkyu Lee. I would also like to thank my collaborators; Soheil Samnii and Prachi Joshi at General Motors and Jiarui Lyu.

I am also grateful to my friends in Ann Arbor – Kibok Lee, Sunghyun Park, Heewoo Kim, Dongyoung Yoon, Yoontae Kang and Amaryllis Rodríguez Mojica – for their care and encouragements.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Automakers keep adding new functions to their products to attract more customers. Since such newly-introduced functions usually require communication with other electronic control units (ECUs) to acquire & deliver sensor (e.g., speedometer, radar, etc.) data, the amount of in-vehicle network traffic keeps rising. To deal with this ever-increasing trend, automakers have re-designed in-vehicle network architecture and adopted high-bandwidth protocols such as controller area network with flexible data-rate (CAN-FD), switched-Ethernet, etc. However, since the complexity and cost related to in-vehicle networks increases with this change, optimizing the in-vehicle network to minimize the cost becomes a major challenge to the automakers.

To tackle such a challenge, we propose a suite of design optimization methods for modern in-vehicle network architectures. First, we present **PAMT**, an optimal priority-assignment algorithm for a single mixed CAN and CAN-FD bus. By clustering messages based on their type, PAMT minimizes the timing overhead for mode transitions. Second, we propose **EACAN** to relax the pessimistic assumptions used in the formal verification for CAN communication. Third, we identify configurable parameters for standardized frame preemption of Ethernet Time-Sensitive Networking (TSN) and present **DOFP**, a genetic algorithm based optimization for the frame preemption. Fourth, we propose **OPMB**, an optimal priority assignment algorithm for multi CAN/CAN-FD buses with a central gateway. Finally, we propose **PRMB** which finds a schedulable priority assignment and generates routing tables to use signal-based routing at the central gateway while meeting the timing requirements of in-vehicle data.

CHAPTER 1

Introduction

A modern vehicle is typically equipped with dozens of small computers [71], called *electronic control units* (ECUs), and applications running thereon control almost everything (even including invisible things) in the vehicle by exchanging sensor data and control commands with each other through an established in-vehicle network. Since automakers keep adding new functions to their products for safe, convenient, and fun driving as illustrated in Fig. 1.1, the amount of in-vehicle network traffic keeps rising. One such example is Advanced Driver Assistant Systems (ADAS) that require large amounts of high-resolution image data.

To deal with this ever-increasing trend, the automakers (e.g., GM, Ford, Toyota, Dailmer-Benz, Hyundai, etc.) have re-designed in-vehicle network architecture over the past several decades as described in Fig. 1.2. In the early days of automotive electronics, in-vehicle applications were confined to a single, stand-alone ECU. Today, applications are distributed over several ECUs that must constantly exchange data. Until the beginning of 90s, data was exchanged through point-to-point links between ECUs. However, the point-to-point

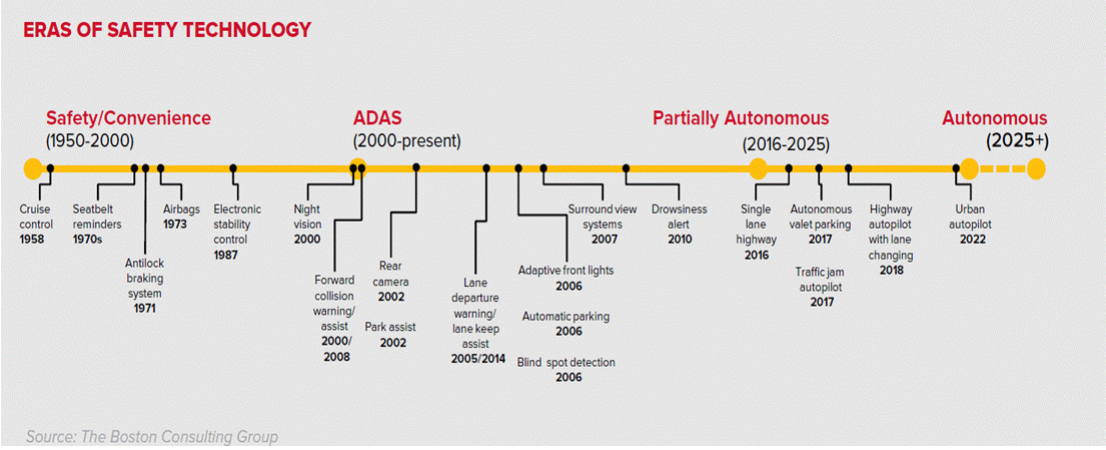


Figure 1.1: History of application deployment on a vehicle (Source: Boston Consulting Group)

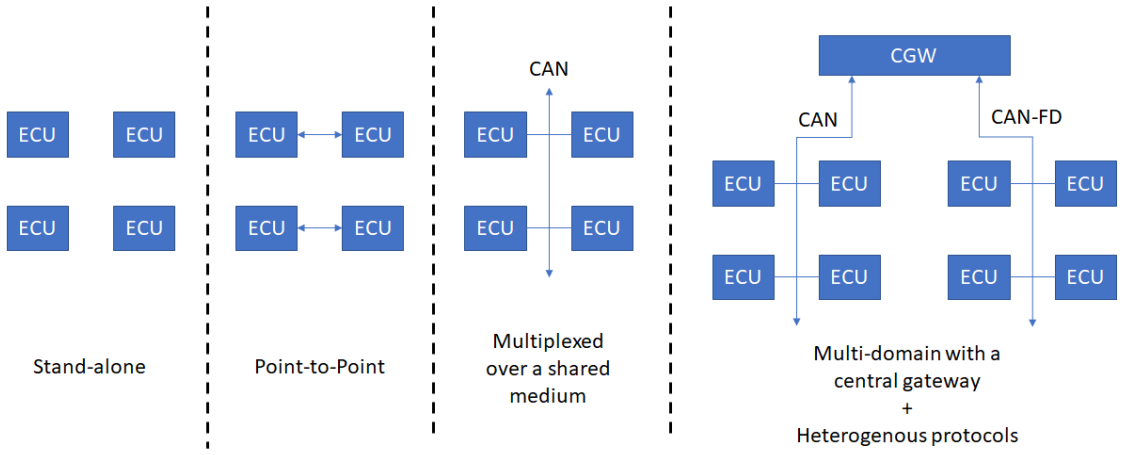


Figure 1.2: Evolution of in-vehicle network architecture

architecture lacked scalability, due to the rapidly increasing number of ECUs within a vehicle, which led to the use of multiplexed communication over a shared medium. Since then, the Controller Area Network (CAN) has been widely used. Even though this bus architecture has been successful, the bandwidth requirements of ECU applications quickly exceeded the bandwidth limit of a single bus. Moreover, not all ECU functions require the same communication performance either. Thus today, modern vehicles deploy heterogeneous buses interconnected via a central gateway, e.g., both low-bandwidth protocols, such as CAN, and high-bandwidth protocols, such as CAN-FD, and Ethernet time-sensitive network (TSN) are used together.

Even though re-designing the in-vehicle architecture has been effective in dealing with the increasing network traffic, the cost of in-vehicle network is also rising. For example, (1) more buses require more wires and connectors within a vehicle, (2) high-bandwidth buses require expensive materials, (3) and the more complex the bus architecture (i.e., number of bus protocols and ECUs), the more human capital and time are needed to design the vehicle. Thus, optimizing in-vehicle networks to minimize cost has become a major focus of automakers today.

1.1 Design challenges

The process of vehicle system design is shown in Fig. 1.3. In the planning phase, system designers define the functionality of a target system. In the design phase, they analyze the requirements of the system for the planned functions, and design prototypes for the target system. The designed prototypes then go into the phase of validation and verification (V&V). In this phase, test engineers check if every function works correctly and also check

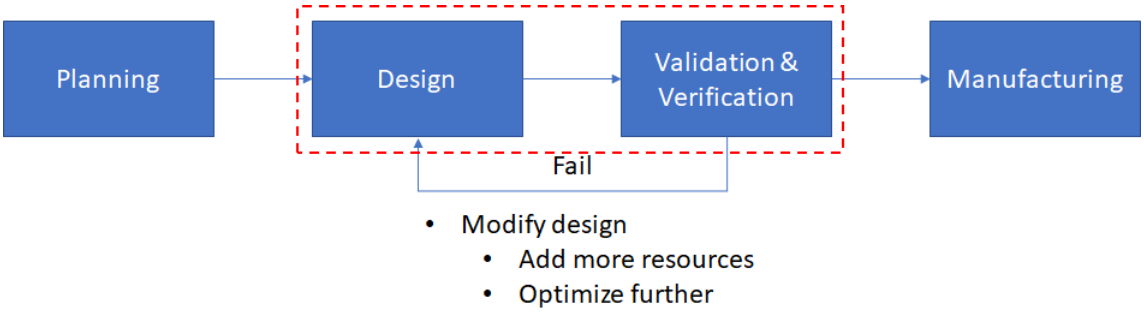


Figure 1.3: The process of vehicle system design

whether every requirement is met or not. If a prototype passes the V&V phase, it goes to the manufacturing phase. Otherwise, system designers have to modify them by adding more resources or by optimizing the tested prototypes further to pass the V&V phase. Thus, finding an optimal network configuration in the design phase and relaxing test conditions while guaranteeing the satisfaction of system requirements in the V&V phase are important for in-vehicle network optimization. Unfortunately, these are challenging.

First, there are too many configurable parameters that the system designers should consider in the design phase. For example, they have to determine not only network architecture, network hardware, and network protocols but also scheduling of in-vehicle traffic, signal packing, etc. Moreover, finding an optimal configuration for a single configurable parameter is a difficult problem. For example, frame packing for CAN-FD [18] and synthesizing gate control list for TSN [76] is NP problems.

Second, formal verification is widely used in the V&V phase to verify whether a prototype design satisfies the system requirements or not. At design and verification time, there are many unknownn (e.g., the number of transmission errors that will happen at runtime). So, the pessimistic case (or the worst case) is assumed in the formal verification. However, the pessimistic case rarely happens at runtime, and thus the pessimism causes severe under-utilization of given resources and lowers the acceptance ratio in the V&V phase.

1.2 State-of-art optimization techniques

Various ideas have been proposed for the design of cost-optimized in-vehicle networks. The design phase ideas focus on how to find an optimal or a near-optimal network configuration from a large design space. The V&V phase ideas focus on finding tight conditions that can guarantee the satisfaction of the given requirements. The following subsections summarize the proposed ideas related to the formal timing verification and configurations of in-vehicle

network architectures.

1.2.1 Timing verification

Because sensor data or control commands generated in a vehicle have valid times, each in-vehicle message should be delivered within a pre-defined time. For example, delivering a brake control command generated by a cruise control system 10 seconds late could cause a catastrophic accident. Thus, guaranteeing the worst-case end-to-end latency of a message to be less than its deadline is an essential requirement. For timing verification, various ways of analyzing/computing the worst-case end-to-end latency have been proposed in both academia and industry.

Tindell *et al.* [102, 104] proposed the first timing analysis for CAN(-FD) protocol based on the well-known *busy period* analysis [59]. However, that analysis has flaws in finding the critical instant, which was corrected by Davis *et al.* [82]. Based on this analysis, various practical issues have been addressed such as the limited size of Tx buffer [51] in the CAN controller, FIFO queuing in the device driver [26], non-abortable Tx buffer [53], and non-negligible time for copying a message into Tx buffer [52]. In particular, several studies [21, 12] focus on the impact of transmission errors on the worst-case latency on CAN.

For Ethernet TSN, the worst-case end-to-end delay of network traffic is usually computed based on either network calculus [109, 108] or compositional performance analysis [101, 100]. These analyses focus on the effects of the various standardized features of TSN on the worst-case end-to-end delay. For example, [101] analyzes the worst-case latency when the time-aware shaper (802.1Qbv) or the peristaltic shaper (802.1Qch) is used. [100] analyzes the impact of frame preemption (802.1Qbu) on the worst-case delay.

1.2.2 Network configuration

In the timing analyses introduced above, the worst-case end-to-end latency of a message depends on how the given networked-system is configured. For example, the priority of a message determines its waiting time at Tx buffer. Thus, it is essential to find a network configuration that makes every message meet its timing requirement. Also, if there are multiple configurations that can satisfy the given requirements, it is important to select the one that minimizes cost. Various ways of finding an optimal/near-optimal configuration have also been proposed.

For CAN(-FD) protocol, [18] proposes heuristic algorithms to pack signals into CAN-FD messages to minimize bus utilization. [50] solves the frame packing problem for the system consisting of multiple CAN-FD buses with a central gateway. [49] further improves

the bus utilization by formulating and solving the signal offset assignment problem. Also, researchers have studied how to assign priorities to the packed frames to find a schedulable priority assignment. Audsley's priority assignment [7] is proven optimal for a single CAN/CAN-FD bus if there is no priority inversion problem. However, priority inversions can happen in practice [26, 51, 52]. [89, 28, 79] try to maximize extensibility for future changes of the message set in addition to finding a schedulable priority assignment. [50] proposes an optimal priority-assignment algorithm for the multi-bus system with a central gateway where the gateway does not change ID of messages.

For Ethernet TSN, [24, 76, 40] focus on the scheduling of network traffic on TSN. In particular, they focus on synthesizing gate control list (GCL). [58, 39, 74] studied the routing of network traffic on TSNs. Especially, they focus on determining static routes for time-triggered traffic [74] and audio-video-bridge (AVB)[58] traffic because the predictable and deterministic operation is important for a real-time and safety-critical system. Topology planning for TSN [6, 39] and traffic type assignment [38] problem are also studied there.

1.3 Thesis Statement and Contributions

The growing complexity of the in-vehicle network makes it challenging to design a cost-optimized network. To tackle this challenge, researchers have explored various ways of designing an in-vehicle network to use the given resource efficiently and reduce production and manufacturing costs by finding optimal/near-optimal network configurations and by developing verification methods for prototype designs while guaranteeing the satisfaction of requirements. However, many optimization problems for the in-vehicle network still remain unsolved, and pessimistic assumptions are applied in the design verification. Thus, there still remains ample room to design a cost-optimized in-vehicle network.

Thesis Statement: *Design minimum-cost in-vehicle networks by optimizing their configuration and removing/reducing pessimism in their verification*

This thesis proposes a set of design optimization methodologies for in-vehicle networks. It designs algorithms and frameworks to optimize the parameters of communication protocols used in modern vehicles. Moreover, it relaxes a pessimistic assumption in timing verification based on run-time system reconfiguration schemes.

1.3.1 PAMT

Due to the differences of their frame format, whenever CAN controllers receive a message with the CAN-FD frame format, they recognize it as an erroneous frame and generate an error frame. In other words, high-speed communication between CAN-FD nodes is impossible due to the error frame when CAN nodes and CAN-FD nodes share the same bus. Two approaches [3, 67] have been proposed to resolve the problem. One is adding a hardware filter in front of the CAN controller to remove CAN-FD frames. The other uses the silent mode (known as listen only mode) in commercial CAN controllers that do not transmit any messages.

Resolving the problem by using the silent mode saves money because hardware filters are not needed. However, CAN controllers must first switch their operation mode from the *normal* mode to the *silent* mode before transmitting a CAN-FD frame, and must then switch their operational mode back to normal mode before transmitting a CAN frame at run-time. The transitions between the two modes are not free: they require non-negligible timing overheads that are a big burden for time-critical automotive systems. This overhead can cause failure of timing verification for a significant number of message sets.

To mitigate the timing overhead, we propose a novel message scheduling algorithm, called PAMT. Because the mode change is only needed when transmitting different types of messages successively, clustering the same type of messages to the maximum extent possible can minimize the required number of mode transitions. To achieve the type-based clustering, we apply priority promotion to messages which are scheduled using NP-EDF. We prove that PAMT is an optimal priority assignment algorithm for the bus with both CAN and CAN-FD nodes. Also, our simulation results show that PAMT effectively reduces the required number of mode transitions, and thus PAMT can schedule 17-–18% more mixed CAN and CAN-FD frame sets than existing optimal priority-assignment algorithms for CAN.

1.3.2 EACAN

At the design time of an automotive system, the timing verification relies on the worst-case scenario. However, the worst-case scenario seldom happens at run-time. This inefficient timing verification is also applied to estimate the latency of a CAN message on a bus. In particular, the worst-case transmission error rate measured in a very pessimistic environment is used in the timing verification to guarantee safe operation in a harsh environment. Thus, the worst-case-based timing verification results in inefficient use of networking resources.

To alleviate this problem, we propose a run-time adaptation methodology, called *error-adaptive CAN* (EACAN). Instead of using the worst-case transmission error rate in the timing verification, timing verification with EACAN assumes a zero transmission error rate at design time. Then, EACAN observes the runtime behavior of transmission error. If the transmission error rate measured at run-time exceeds the pre-defined threshold, EACAN reconfigures the periods of low-criticality messages to guarantee the reliability requirement continuously. According to our experimental results, EACAN improves bandwidth utilization by 14% over the timing verification with the worst-case transmission error rate without violating the reliability requirement.

1.3.3 DOFP

Ethenet TSN is gaining popularity for backbone and domain networks which require high bandwidth in the next-generation automotive communication architectures. In particular, in-vehicle messages with various payload sizes and criticality levels would share a backbone network. Thus, (1) small-size packets wait for the completion of transmitting a large-size packet and (2) high-critical messages wait for the completion of transmitting a low-critical message. To avoid these unfortunate situations, the frame preemption (802.1Qbu and 802.3br) feature is standardized by the IEEE TSN Task Group. However, to gain benefits from the frame preemption, careful shaping of frame preemption at design time is required.

In this piece of work, we identify configurable parameters to shape the frame preemption at design time and present a genetic algorithm-based optimization framework to find optimal parameter configurations. Our framework exploits the worst-case preemption-aware timing analysis presented in Section 2.3. We show the usefulness of our framework through two case studies; improving reliability and extensibility of networks. In the case studies of effective and efficient use of the framework, we also propose an initialization algorithm for each goal. Our simulation results have demonstrated that the proposed initialization algorithms outperforms the existing assignment algorithms (AOPA, RPA) used in general real-time systems and an intuitive approach (FPCP) in the reliability and extensibility.

1.3.4 OPMB

Automakers keep introducing new functions to vehicles to entice customers, thus increasing the size/number of in-vehicle buses. As a result, adopting multi-buses with a central gateway is becoming a norm in current and future vehicles. Since adding buses increases

production cost, knowing whether or not to add a bus at design time to meet the given requirements is essential to design a cost-optimized in-vehicle network. However, due to the lack of an optimal priority-assignment algorithm for multi-bus systems, it is difficult to determine whether additional buses are needed.

To address this difficulty, we propose an optimal priority-assignment algorithm, called OPMB, for multiple CAN/CAN-FD buses with a central gateway. OPMB builds on backtracking, and is thus of exponential time complexity. To reduce the execution time effectively for industry-size problems, we identify several theory-proven search-space reduction conditions. Our in-depth simulation has demonstrated that OPMB outperforms the state-of-art priority-assignment algorithms for multi-bus systems, and is suitable for high-speed systems which represent future automotive systems. Also, OPMB is shown to be feasible for most realistic automotive message sets.

1.3.5 PRMB

This architecture, consisting of multiple network buses connected via a central gateway, is widely used in modern vehicles. Additionally, the AUTomotive Open System ARchitecture (AUTOSAR) had standardized a software framework for the gateway as part of communication services. The AUTOSAR gateway supports multiple routing methods such as PDU-direct routing and signal-based routing. With signal-based routing, the gateway can extract signals from incoming messages and can generate new messages by assembling some of the extracted signals. Thus, signal-based routing can reduce network loads by avoiding forwarding unnecessary signals to their destinations, and by reducing the total number of messages on buses. However, applying signal-based routing while guaranteeing timely delivery of given signals is non-trivial. When the gateway uses signal-based routing, the priority assignment problem and routing table synthesis problem should be considered together to guarantee the timely delivery of given signals.

To gain benefits of signal-based routing, we propose an algorithm which performs Priority assignment and Routing table synthesis for Multiple CAN/CAN-FD Buses with a central gateway, called PRMB. For the priority assignment in PRMB, we extend OPMB, the state-of-art optimal priority-assignment algorithm, to consider signal-based routing. When the extended OPMB cannot find a schedulable priority assignment for a given message set, PRMB tries to merge two messages into one message to reduce network load, and retries to find a schedulable priority assignment. Internally, when PRMB applies a merge, PRMB keeps updating routing tables to reflect the merge. Our simulation results show that PRMB outperforms the state-of-art priority assignment algorithms in schedulability coverage.

1.4 Outline

Chapter 2 introduces the basics of CAN, CAN-FD, and TSN as background knowledge. Chapter 3 discusses the priority-assignment problem for a bus shared by both CAN and CAN-FD ECUs. Chapter 4 presents relaxation of the worst-case transmission error rate in formal verification. Chapter 5 presents a genetic-algorithm-based framework for synthesizing configurable parameters for the standardized frame preemption on Ethernet TSN. Chapter 6 considers the priority-assignment problem for multi CAN/CAN-FD buses with a central gateway. Chapter 7 presents a way of priority assignment and routing table synthesis to use signal-based routing at the central gateway while guaranteeing the timely delivery of the given signals. Finally, Chapter 8 concludes this thesis and discusses future directions.

CHAPTER 2

Background

2.1 Controller Area Network (CAN)

CAN is regarded as *de facto* standard of in-vehicle network since its official release in the mid 80s. It covers from physical layer to data link layer (in the perspective of OSI model) to allow ECUs connected to a shared physical wire (bus) to communicate each other via broadcast without a centralized controller. The maximum speed of a CAN bus is 1Mbps with a maximum cable length of about 40 meters according to the standard (ISO 11898-2), but usually 500Kbps and 250Kbps bus speeds are used in automotive systems.

2.1.1 CAN frame format

A CAN protocol supports two different frame formats; (a) base and (b) extended. Because the difference between the base format and the extended format is only the length of identifier field (11-bit for base and 29-bit for extended), we cover the base format here in detail. The frame format of CAN protocol is illustrated in Fig. 2.1, and consists of start-of-frame, arbitration field, control field, data field, cyclic redundancy check, acknowledgement and end-of-frame.

A frame starts from a dominant (0) bit, called start-of-frame, which indicates the beginning of transmission of a frame. The following unique 11-bits in the arbitration field are the

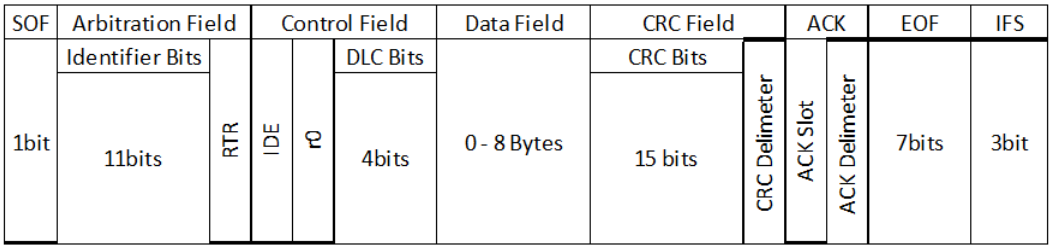


Figure 2.1: CAN frame format (base)

identifier of the frame, which represents not only the what data it carries but also priority of the frame. The remaining 1-bit (RTR bit) in the arbitration field indicates a type of the frame. If RTR bit is dominant, the frame is data frame, which carries data from a transmitter to receiver. If RTR bit is recessive (1), the frame is remote frame, which requests a transmission of a data frame with the same identifier. In the next control field, there are identifier extension (IDE) bit and data length code (DLC). IDE bit indicates frame format. If IDE bit is dominant then the frame has the base frame format else the frame has the extended frame format. The 4-bit DLC represents the length of data field in bytes (from 0–8 bytes).

In addition to the data frame and remote frame, there are two more types of CAN frame; error frame and overload frame. Error frame represents the detection of transmission errors by sending at least 6 consecutive dominant bits. Note that every bit stream of more than 5 same bits is considered as error in the CAN protocol. Overload frame is used to shape an extra delay between successive data or remote frame.

2.1.2 Bus Arbitration

The CAN protocol schedules messages based on the value of identifier (ID) field of messages in a decentralized way. When CAN bus is idle, multiple ECUs can try to transmit messages at the same time. Because CAN physical layer is designed to operate as a wired-AND gate, ECUs which send recessive bit can see the dominant bit on the bus during the transmission of ID bits. When an ECU loses bus arbitration by the dominant bit, the ECU gives up transmitting its message. As a result, the message with the lowest ID value is selected to be transmitted under the CAN protocol. The ECUs that lost the arbitration will try retransmission of their messages after completing the transmission of the message which won the current round of arbitration. Thus, by assigning ID properly, the system designer can control the transmission order of messages. In other words, priority assignment to a CAN message means an ID assignment.

2.1.3 Timing analysis of a CAN message

Many applications which 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 delivery/response time (WCRT) of each CAN message.

The first timing analysis of a CAN message was done by Tindell *et al.* [102, 104]. 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. The transmission time of a single CAN message can be easily analyzed [82] by considering the bit-stuffing rule according to the following equation where p is the payload size in bytes, t_{nom} is the transmission time for a single bit and g is 34 for the base format or 54 for the extended format:

$$C_i = \left(g + 8p + 13 + \lfloor \frac{g + 8p - 1}{4} \rfloor \right) t_{nom}. \tag{2.1}$$

Since the release jitter is determined *a priori* by considering the performance of microprocessor, Tindell *et al.* focused on the analysis of a 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:

$$w_i^{n+1} = B_i + \sum_{\forall k \in hp(i)} \left\lceil \frac{w_i^n + J_k + \tau}{T_k} \right\rceil C_k \tag{2.2}$$

$$R_i = J_i + w_i + C_i, \tag{2.3}$$

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 that of message (m_i), and τ is a bit time.

However, this analysis has a severe flaw, and hence Davis *et al.* [82] used a fine-grained approach to correct the flaw in Eq. (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 q-th instance of the message ($w_i(q)$) is defined as: (q starts from 0).

$$w_i^{n+1}(q) = B_i + qC_i + \sum_{\forall k \in hp(i)} \left\lceil \frac{w_i^n(q) + J_k + \tau}{T_k} \right\rceil C_k \tag{2.4}$$

and WCRT of the message is defined as:

$$R_i(q) = J_i + w_i(q) - qT_i + C_i \tag{2.5}$$

$$R_i = \max_q R_i(q). \tag{2.6}$$

Since the q-th instance of the message is released at qT_i, qT_i is subtracted from the com-

pletion time of the q-th instance to calculate the response time as shown in Eq. (4).

These WCRT analyses have been extended to address various practical issues, such as the limited size of TxObject [51], FIFO queuing in the device driver [26], non-abortable TxObject [53], and non-negligible time for copying a message into TxObject [52]. In particular, several studies [21, 12] 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:

$$w_{i|Z}^{n+1}(q) = B_i + qC_i + E_{i|Z} + \sum_{\forall k \in hp(i)} \left\lceil \frac{w_{i|Z}^{n}(q) + J_k + \tau}{T_k} \right\rceil C_k, \tag{2.7}$$

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

$$R_{i|Z}(q) = J_i + w_{i|Z}(q) - qT_i + C_i \tag{2.8}$$

$$R_{i|Z} = \max_{q} \quad R_{i|Z}(q). \tag{2.9}$$

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 [21, 12] 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:

$$p(R_{i|Z}) = p(Z, R_{i|Z}) - \sum_{j=0}^{Z-1} p(R_{i|j}) p(Z - j, R_{i|Z} - R_{i|j}) \tag{2.10}$$

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

$$p(Z, R_{i|Z}) = \frac{e^{-\lambda R_{i|Z}} (\lambda R_{i|Z})^Z}{Z!}. \tag{2.11}$$

They then compute the probability of a message deadline miss by adding all the probabili-

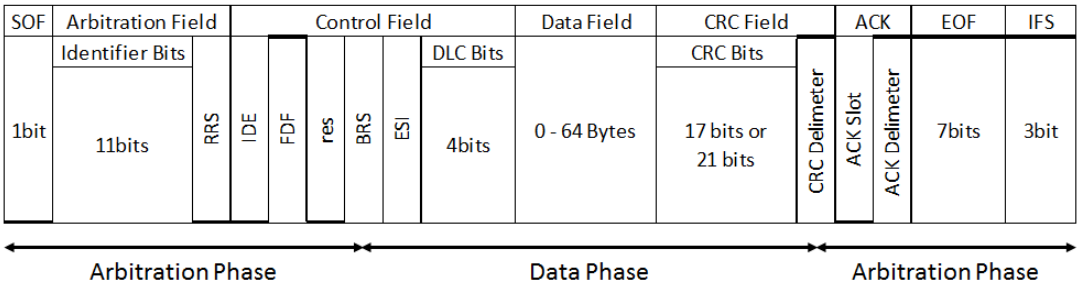


Figure 2.2: CAN-FD frame format (base)

ties $p(R_{i|Z})$ such that $R_{i|Z} \leq D_i$ (D_i is the relative deadline of message i):

$$p_i(DM) = 1 - \sum_{\forall Z | R_{i|Z} \leq D_i} p(R_{i|Z}), \tag{2.12}$$

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.

2.2 Controller area network with flexible data rate (CAN-FD)

CAN is simple and cheap but can support only up to 8 bytes of data field and 1 Mbps bandwidth, making it difficult to achieve secure transmission by appending a secure key and deal with the increasing amount of in-vehicle traffic. Thus, Robert Bosch, the inventor of CAN, proposed CAN-FD to overcome the drawbacks of CAN in 2012. CAN-FD can support up to 64 bytes of data field and 12 Mbps bandwidth while having the same architecture and physical layer backward compatibility with the original CAN. Thus, CAN-FD became attractive to many OEMs and suppliers, unlike Flexray [47] which was also proposed as a substitute for CAN and has higher bandwidth (10Mbps).

2.2.1 CAN-FD frame format

The CAN-FD frame format is similar to the CAN frame format as shown in Fig. 2.2. Both frames consist of start-of-frame, arbitration field, control field, data field, cyclic redundancy check, acknowledgment, and end-of-frame. However, there are several differences between them in the arbitration field, control field, data field, and CRC field.

Data Frame Type	DLC	Payload Size (in Bytes)	CRC Bits
CAN & CAN-FD	0000	0	CAN:15 CAN-FD:17
	0001	1	
	0010	2	
	0011	3	
	0100	4	
	0101	5	
	0110	6	
	0111	7	
CAN	1xxx	8	
CAN-FD	1000	8	17
	1001	12	
	1010	16	
	1011	20	21
	1100	24	
	1101	32	
	1110	48	
	1111	64	

Table 2.1: Supported Payload Size

The RTR bit in the arbitration field of the CAN frame format, which indicates whether the frame is data frame (dominant bit) or remote frame (recessive bit), is replaced with a reserved bit, and the following three control bits are newly introduced.

- Flexible Data-rate Format (FDF) bit indicates whether the frame is encoded as CAN frame (dominant bit) or CAN-FD frame (recessive bit).
- Bit Rate Switch (BRS) bit indicates whether the bit time changes during the payload transmission (recessive bit) or not (dominant bit).
- Error Status Indicator (ESI) bit indicates whether the transmitter which sends the frame is in error active state (dominant bit) or error passive state (recessive bit).

The supported payload sizes in CAN frame and CAN-FD frame are listed in Table 2.1. CAN-FD frame supports up to 64 bytes payload size. Since the increase of the maximum payload size requires more redundancy bits to check the correctness in transmitted data, the number of CRC bits also increases. Note that, for the CRC field, a static number of stuff bits is used. For example, if 17-bits CRC is used, 4 stuff bits are added. If 21-bits CRC is used, 5 stuff bits are added.

2.2.2 Switching Bit Rate

Since only one ECU can access the shared CAN bus during payload transmission (other ECUs already lost bus arbitration), synchronization between ECUs is not required for the payload transmission. Hence, CAN-FD increases the maximum payload size and transmission speed by boosting up the bit rate when it transmits payload bits.

CAN-FD frame is separated into two phases, arbitration phase and data phase, as shown in Figure 2.2. The interval between BRS bit and CRC delimiter bit is defined as the data phase. The other intervals are defined as the arbitration phase. The purposes of this separation are to support improved transmission rate and to keep the key features of CAN such as non-destructive arbitration. Thus, the CAN-FD protocol defines two bit-times, nominal bit-time (t_{nom}) and data bit-time (t_{data}). These bit-times are configured by considering properties of the given CAN-FD network (e.g., the number of ECUs, the length of wire, limitations of CAN-FD transceivers, etc.). Since the CAN-FD protocol does not allow slowing down the bit rate in the middle of transmission of a frame, t_{data} must be smaller than, or equal to t_{nom}.

The bit-time for the arbitration phase is always set to t_{nom}. However, the bit-time for the data phase depends on the value of the BRS bit. If the BRS bit of a CAN-FD frame is set to recessive bit, then the bit time is switched from t_{nom} to t_{data} to boost the transmission rate. Otherwise, t_{nom} holds for the data phase.

2.2.3 Timing analysis of CAN-FD message

The timing analysis of CAN (Eq. 2.6) is applicable to analyze the worst-case response time of CAN-FD messages. However, Eq. 2.1 has to be revised to consider differences between CAN frame format and CAN-FD frame format. Bordoloi *et al.* [18] provided the revised equation for CAN-FD as follows where p is the payload size in bytes:

$$C_i = 32t_{nom} + \left(28 + 5\lceil\frac{p-16}{64} + 10p\rceil\right) t_{data}. \tag{2.13}$$

2.3 Ethernet Time-Sensitive Networking (TSN)

The IEEE 802.1 Time-Sensitive Networking (TSN) Task Group[1] developed a set of standards to enhance the real-time and dependability properties of IEEE 802 networks. Example standards include credit-based shaping (IEEE Std 802.1Qav-2009), time synchro-

[1] formerly known as the Audio/Video Bridging (AVB) Task Group

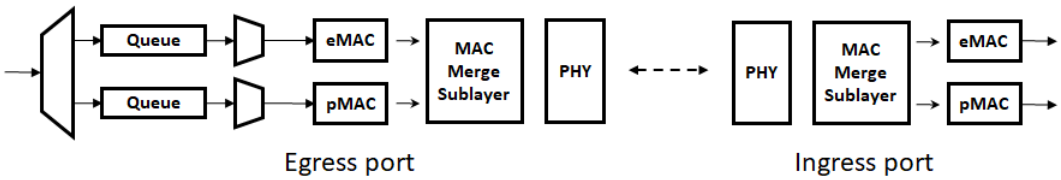


Figure 2.3: Frame preemption supporting switched-Ethernet port

nization (802.1AS-2011), time-triggered communication (802.1Qbv-2015), and frame preemption (802.1Qbu-2016 and 802.3br-2016). The enhancements help automotive OEMs adopt the switched-Ethernet for next-generation vehicles. In particular, the OEMs consider the switched-Ethernet for some domain networks (ADAS, infotainment) and the in-vehicle backbone network.

Chapter 5 of this dissertation focuses on design optimization of the standardized frame preemption, and thus we provide an overview of Ethernet TSN with the frame preemption.

2.3.1 Frame preemption

IEEE 802.1Qbu and IEEE 802.3br describe how a frame preemption takes place at egress and ingress ports. To enable frame preemption, an egress port and the corresponding ingress port have to support the frame preemption.

An egress port can have up to 8 queues to serve multiple frame classes, and each queue is mapped to either express MAC (eMAC) interface or preemptable MAC (pMAC) interface as shown in Fig. 2.3. If a message passes through the eMAC, the message is called an *express frame*. If a message passes through the pMAC, the message is called a *preemptable frame*. An express frame cannot be preempted by any other frames, but a preemptable frame can be preempted by any express frame. Note that preemptable frames cannot preempt each other. Hence, only one level of frame preemption is allowed. The formats of both express and preemptable frames are shown in Fig. 2.4. In addition to the MAC interfaces, IEEE 802.3br introduces a key component, *MAC merge sublayer*, that performs both transmit and receive processing for express and preemptable frames to implement the frame preemption function.

In the transmit processing, the MAC merge sublayer (1) replaces the value of start of mPacket delimiter (SMD) with either SMD-E, SMD-Sx or SMD-Cx[2] to indicate whether the current frame is either express or preemptable frame. For example, if a frame is deliv-

[2]The values of SMD-E, SMD-Sx and SMD-Cx are: SMD-E(0xD5), SMD-S0(0xE6), SMD-S1(0x4C), SMD-S2(0x7F), SMD-S3(0xB3), SMD-C0(0x61), SMD-C1(0x52), SMD-C2(0x9E), SMD-C3(0x2A). Note that the 2-bit frame count is encoded in SMD-Sx and SMD-Cx.

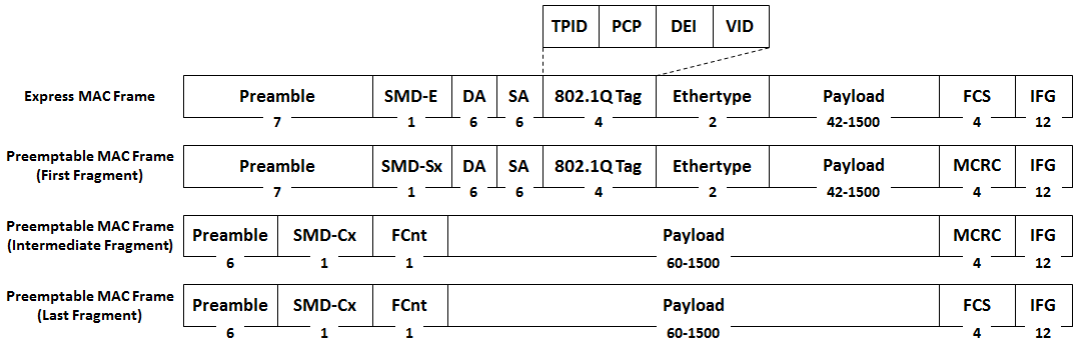


Figure 2.4: Format of preemptable and express frames

ered via the eMAC interface, the value of SMD is replaced with SMD-E. It also (2) generates cyclic redundancy code (CRC) for each frame and fragment, (3) preempts a preemptable frame to intersperse an express frame between fragments of the preemptable frame, and (4) resumes the transmission of the remainder of the preempted frame. To inform the order of fragments to a receiver, the fragment count (FCnt) is appended after SMD-Cx.

In the receive processing, the MAC merge sublayer checks the type of received frames by inspecting their SMD values. If SMD = SMD-E, the frame is directly delivered to eMAC. If SMD = SMD-Sx, it checks whether the current frame is the last fragment or not by comparing MCRC and the last four octets of the current frame. If the current frame is not the last fragment (matching the MCRC), the MAC merge sublayer sends the data in the current frame to pMAC and waits for continuing fragments. If SMD = SMD-Cx, the MAC merge sublayer checks frame count, FCnt, and the last four octets. If there is any violation in these checked values, it informs pMAC of the error state. The reason for checking the last four octets is the same as for SMD-Sx.

2.3.2 Timing analysis of messages on Ethernet TSN with frame preemption

Thiele *et al.* [100] proposed a worst-case latency analysis for Ethernet TSN with the frame preemption. The analysis leverages the compositional performance analysis (CPA) framework, and models egress ports as the major resources. Below we briefly introduce this analysis.

Transmission Time Analysis: At an egress port, a frame consumes service time for transmission, and the service time depends on its payload size. The maximum service time for a frame of traffic flow i is modeled as follows where L_i is the maximum payload size

of traffic flow i and r_{TX} is the link speed:

$$C_i^+ = \frac{42bytes + max\{42bytes, L_i\}}{r_{TX}}. \tag{2.14}$$

Because an egress port can transmit only one frame at a time, the frames queued at the port compete with each other to receive the service time. So, a frame needs to wait for the completion of other frames' transmission in a queue. The queuing delay (waiting time in a queue) of the frame depends on its priority and the type of queue where the frame is queued.

Queuing delay analysis for preemptable frame: A (preemptable) frame can be preempted by express frames, and thus can be fragmented. Thus, the queuing delay of a preemptable frame consists of (1) blocking time by a lower-priority frame, (2) blocking time by same-priority frames, (3) blocking time by higher-priority frames, and (4) preemption overhead.

If a lower-priority frame A is being transmitted when a preemptable frame B is queued, B has to wait for the completion of A's transmission regardless of the type of A. Thus, the worst-case blocking time by a lower-priority frame for a preemptable frame is:

$$I_i^{LPB,P} = \max_{j \in lp(i)} \left\{ C_j^+ \right\} \tag{2.15}$$

Because of the first-in-first-out (FIFO) policy, a preemptable frame needs to wait for the completion of transmission of frames that are already in the same queue. Moreover, because of preemption, we should analyze the blocking time by same-priority frames from the perspective of the last fragment of the preemptable frame. Suppose the q-th frame of traffic i is queued at time a_i^q. Then, the last fragment of the frame should wait for the completion of transmission of (a) $q-1$ frames of traffic i, (b) continuous fragments of the frame and (c) $\eta_j^+(a_i^q)$ frames of traffic j where $\eta_j^+(t)$ is the number of frame of traffic j queued before time t. Because the worst case happens when the last fragment has the minimum size (84 bytes), the worst-case blocking time by same-priority frames is:

$$I_i^{SPB,P}(q, a_i^q) = (q-1)C_i^+ + C_i^+ - \frac{84bytes}{r_{TX}} + \sum_{j \in sp(i)} \left\{ \eta_j^+(a_i^q) C_j^+ \right\}. \tag{2.16}$$

Suppose the transmission of the last fragment of a preemptable frame begins at time t. Then, higher-priority frames queued before time t must be transmitted before the transmission of the last fragment. Thus, the worst-case blocking time by higher-priority frames is:

$$I_i^{HPB}(t) = \sum_{j \in hp(i)} \left\{ \eta_j^+(t) C_j^+ \right\}. \tag{2.17}$$

Because the minimum frame size is 84 bytes, the first fragment of a preemptable frame contains at least 42-bytes payload and the other fragments of the preemptable frame contain at least 60 bytes payload. Thus, a preemptable frame of traffic i can experience up to the following number of preemptions:

$$F_i^+ = \left\lfloor \frac{L_i - 42bytes}{60bytes} \right\rfloor. \tag{2.18}$$

Suppose the q-th frame of traffic i is blocked by a lower-priority preemptable frame and the lower-priority preemptable frame experiences F_j^+ preemptions. Then, the maximum number of frame preemptions that occurs during the transmission of the lower-priority frame is:

$$N_i^{LP} = \max_{j \in lp^P(i)} \left\{ F_j^+ \right\} \tag{2.19}$$

We also need to analyze the maximum number of frame preemptions for same-priority preemptable frames and higher-priority preemptable frames. Because these analyses are similar to the blocking time analysis, the maximum number of frame preemptions during the transmission of same-priority preemptable frames.

Suppose the q-th frame of traffic i is blocked by same-priority preemptable frames and a same-priority preemptable frame experiences F_j^+ preemptions. Then, the maximum number of frame preemptions that occurs during the transmission of the same-priority frames is:

$$N_i^{SP}(q, a_i^q) = aF_i^+ - 1 + \sum_{j \in sp(i)} \left\{ \eta_j^+(a_i^q) F_j^+ \right\} \tag{2.20}$$

and the maximum number of frame preemptions that occurs during the transmission of higher-priority preemptable frames is:

$$N_i^{HP}(t) = \sum_{j \in hp^P(i)} \left\{ \eta_j^+(t) F_j^+ \right\}. \tag{2.21}$$

The summation of N_i^{LP}, $N_i^{SP}(q, a_i^q)$ and $N_i^{HP}(t)$ bounds the number of frame preemptions which increase the queuing delay of the q-th frame. However, the number of frame preemptions cannot exceed the number of express frames which are queued before the transmission of the last fragment of the q-th frame. So, if the transmission of the last fragment of the q-th frame begins at time t, then the number of frame preemptions cannot exceed:

$$\sum_{j \in hp^E(i)} \left\{ \eta_j^+(t) \right\}. \tag{2.22}$$

Consequently, the maximum number of frame preemptions happens before the transmis-

sion of the last fragment of the q-th frame of traffic i is:

$$N_i(t,q,a_i^q) = \min\left\{ \sum_{j \in hp^E(i)} \left\{ \eta_j^+(t) \right\}, N_i^{LP} + N_i^{SP}(q,a_i^q) + N_i^{HP}(t) \right\}. \tag{2.23}$$

Because additional overhead per frame preemption is 24 bytes (preamble, SMD-Cx, FCnt, MCRC, and IFG), the preemption overhead for the q-th frame of traffic i is:

$$I_i^{PO}(t,q,a_i^q) = \frac{24bytes}{r_{TX}} N_i(t,q,a_i^q) \tag{2.24}$$

and the queuing delay of a preemptable frame is:

$$w_i^P(q,a_i^q) = I_i^{LPB,P} + I_i^{SPB,P}(q,a_i^q) + I_i^{HPB}(w_i^P(q,a_i^q)) + I_i^{PO}(w_i^P(q,a_i^q),q,a_i^q). \tag{2.25}$$

Queuing delay analysis for an express frame. Queuing delay analysis for an express frame. Because an express frame cannot be preempted, it cannot be fragmented. So, the queuing delay of the express frame consists of (1) blocking time by a lower-priority frame, (2) blocking time by same-priority frames, and (3) blocking time by higher-priority frames.

To analyze the blocking time by a lower-priority frame, we need to consider two cases: the lower-priority frame is (a) an express frame or (b) a preemptable frame. If the type of the lower-priority frame is preemptable, the express frame can preempt the lower-priority frame. Otherwise, the express frame has to wait until the transmission of the lower-priority frame is completed. Note that 143 bytes are the longest frame size which cannot be fragmented. Thus, the worst-case blocking time by a lower-priority frame for an express frame is:

$$I_i^{LPB,E} = \max\left\{ \max_{j \in lp^E(i)} \left\{ C_j^+ \right\}, \min\left\{ \max_{j \in lp^P(i)} \left\{ C_j^+ \right\}, \frac{143bytes}{r_{TX}} \right\} \right\}. \tag{2.26}$$

Unlike the preemptable frame case (Eq. 2.15), we analyze blocking time by same-priority frames from the perspective of the p-th frame (not the last fragment of the p-th frame). Thus, the worst-case blocking time by same-priority frames is:

$$I_i^{SPB,E}(q,a_i^q) = (q-1)C_i^+ + \sum_{j \in sp(i)} \left\{ \eta_j^+(a_i^q) C_j^+ \right\}. \tag{2.27}$$

The worst-case blocking time by a higher-priority frame for the express frames is the same as that for preemptable frames. Consequently, the queuing delay of an express frame is:

$$w_i^E(q,a_i^q) = I_i^{LPB,E} + I_i^{SPB,E}(q,a_i^q) + I_i^{HPB}(w_i^E(q,a_i^q)). \tag{2.28}$$

End-to-end frame latency. From the queuing delay, we can easily derive the worst-case response time of the q-th frame of frame i at egress port j. Because the queuing delay depends on the queuing instant of the frame (a_i^q), all possible instants should be considered. Let A_i^q be the set of possible instants.[3] Then, the worst-case response time of the q-th frame of frame i at egress port j is:

$$R_i^j(q) = \begin{cases} \max\limits_{a_i^q \in A_i^q} \Big\{ w_i^E(q, a_i^q) + C_i^+ - a_i^q \Big\}, & \text{if express at } j \\ \max\limits_{a_i^q \in A_i^q} \Big\{ w_i^P(q, a_i^q) + \frac{84bytes}{r_{TX}} - a_i^q \Big\}, & \text{if preemtable at } j \end{cases} \tag{2.29}$$

the worst-case response time of frames of traffic i at egress port j is:

$$R_i^j = \max_q \Big\{ R_i^j(q) \Big\}. \tag{2.30}$$

So, the worst-case E2E latency of frames of traffic i is:

$$R_i^+ = \sum_{j \in \rho_i} \Big\{ R_i^j \Big\}. \tag{2.31}$$

[3] See [100] for details of finding possible instants.

CHAPTER 3

PAMT: Optimal Priority Assignment for Scheduling Mixed CAN and CAN-FD Frames

3.1 Introduction

Controller Area Network (CAN) [86] is the *de facto* standard of current in-vehicle networks because of its robustness, wide deployment, low resource requirement, and real-time support. However, the advent of new functions to improve the driver's safety and comfort will make CAN unlikely to meet in-vehicle communication requirements in the near future [91]. To overcome the shortcomings of CAN, a new protocol, *Controller Area Network with Flexible Data-rate* (CAN-FD), has recently been proposed [87]. CAN-FD not only overcomes the drawbacks of CAN but also allows use of existing/legacy CAN infrastructures — e.g., Electronic Control Units (ECUs) developed with CAN controllers and transceivers, CAN wires, etc. — thanks to its physical-layer compatibility with CAN. As a result, CAN-FD has been attracting significant attention as the most promising substitute of CAN [93].

Even though CAN-FD-based ECUs can share the same communication bus with CAN-based ECUs, legacy CAN controllers, which do not support CAN-FD frame format, cause a significant problem [67]. Whenever CAN controllers receive a CAN-FD frame, they generate an error frame because the CAN-FD frame is recognized as an erroneous frame due to the difference between CAN and CAN-FD frame formats [87]. As a result, CAN-FD-based ECUs discard the CAN-FD frame upon receiving an error frame, in accordance with the CAN protocol [86]. So, CAN-FD-based ECUs cannot communicate with each other via CAN-FD frames, making it infeasible to realize the advantages of CAN-FD, such as relatively high bandwidth and large payload size.

Recently, hardware [3, 60] and software [67] solutions have been proposed to address this problem. The hardware solutions [3, 60] use an additional hardware component (e.g, NXP FD Shield) which filters out CAN-FD frames before reaching the CAN controllers.

However, they are more expensive and more difficult to deploy, than the software solutions. The software solution [67] is cheaper than the hardware solutions, but relies on the silent mode of CAN controller to prevent the CAN controller from generating error frames. Thus, all CAN controllers must switch their mode from normal mode to silent mode at run-time before transmitting CAN-FD frames. Also, the CAN controllers have to return to normal mode after transmitting the CAN-FD frames to resume reception of CAN frames. These mode transitions incur non-negligible time overheads and hence negatively impact the schedulability of mixed CAN and CAN-FD frame sets significantly. Thus, the existing optimal priority-assignment algorithms for CAN [7, 111] cannot find a *schedulable* priority order for the mixed frame sets even when a schedulable priority assignment exists for the mixed frame sets.

To remedy the above problem, we propose a new priority-assignment algorithm, called *Priority Assignment with Mode Transition* (PAMT), which minimizes the negative impact of the silent mode-based solution on the schedulability of a given set of mixed CAN and CAN-FD frames. PAMT reduces the required number of mode transitions for the given set of mixed frames via *type-based clustering* which groups frame instances based on their type. We prove that PAMT is an *optimal* priority assignment algorithm for mixed frame sets. We also conduct extensive simulations to evaluate the effectiveness of PAMT for mixed frame sets by comparing it with the existing optimal priority-assignment algorithms for CAN. Our simulation results show that PAMT effectively reduces the required number of mode transitions, and thus PAMT can schedule 17–18% more mixed CAN and CAN-FD frame sets than existing optimal priority-assignment algorithms for CAN. This chapter makes the following main contributions:

- Identify and analyze the negative impact of the software (silent-mode-based) solution on the schedulability of mixed CAN and CAN-FD frame sets;
- Propose a new priority assignment algorithm, PAMT, to minimize the negative impact of using silent mode on the schedulability of mixed frame sets, and prove that PAMT is optimal priority assignment for mixed frame sets; and
- Demonstrate via extensive simulations that PAMT effectively reduces the required number of mode transitions and outperforms the existing optimal priority-assignment algorithms for mixed frame sets.

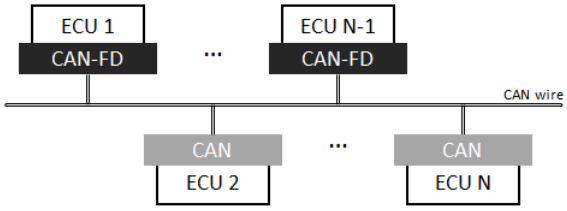


Figure 3.1: Network Model

3.2 Mixed CAN and CAN-FD System Model

3.2.1 Network Model

We consider a system consisting of multiple ECUs connected via one shared CAN bus as illustrated in Fig. 3.1. Some of the ECUs are equipped with CAN-FD controllers and transceivers, while the others are equipped with CAN controllers and transceivers. For convenience, we will use the term 'CAN-FD node' ('CAN node') to represent an ECU equipped with a CAN-FD (CAN) controller and a CAN-FD (CAN) transceiver.

We say the system is in *CAN mode* when the CAN controllers are in normal mode because only CAN frames are transmitted on the bus in this mode. Also, we say the system is in *FD mode* when the CAN controllers are in silent mode. To avoid transmitting CAN-FD frames in CAN mode, CAN-FD nodes do not queue the CAN-FD frames in CAN mode. Because our approach sends a special message to trigger mode transition of CAN controllers, CAN-FD nodes can easily know the current system mode.

Some may expect this mixed CAN and CAN-FD network architecture to be short-lived and used only during the transition from CAN to CAN-FD, but this architecture would last for a long time, because (1) a vehicle platform, once developed, is utilized for a long time[1]; (2) the implementation cost of a CAN node is cheaper than that of a CAN-FD node and the automotive industry seldom uses anything more than absolutely needed to save cost; and (3) automotive manufacturers tend to use already-verified modules/systems to meet the mandatory requirements, such as safety.

3.2.2 Mixed Frame Model

We adopt the widely-used CAN frame model in [26, 53] with an additional parameter indicating whether a frame is CAN or CAN-FD frame. Thus, a frame is defined as $F_i = \{T_i, D_i, J_i, FD_i, C_i\}$ where T_i is the period of F_i, D_i the relative deadline of F_i; J_i the

[1]According to a report by the Center for Automotive Research, a developed platform lasts over 5 years on average (http://www.cargroup.org/automotive-product-development-cycles-and-the-need-for-balance-with-the-regulatory-environment/)

release jitter of F_i; FD_i the type of F_i, $FD_i \in \{0, 1\}$ – if $FD_i = 0$ then F_i is a CAN frame else F_i is a CAN-FD frame; C_i is the transmission time of F_i and depends on the data length of F_i and FD_i.

3.2.3 Mixed Frame-Instance Model

F_i^j is an instance of frame F_i, and hence inherits the properties of F_i such as frame type and transmission time. Thus, F_i^j is defined as $F_i^j = \{A_i^j, D_i^j, J_i, FD_i, C_i\}$ where A_i^j is the release time of F_i^j and D_i^j is its deadline such that $D_i^j = A_i^j + D_i$.

We will assign priorities to frame instances in an mixed frame instance set I:

$$I = \{F_i^j | \forall i, j \; A_i^j < HP\}$$

where $HP = LCM\{T_1, \ldots, T_n\}$ is the planning cycle (the least common multiple of periods) of a given mixed frame set.

We use f_i to represent a frame instance of priority i; f_i has a higher priority than f_j if $i < j$, i.e., the lower the number, the higher the priority.

3.3 Scheduling Mixed CAN and CAN-FD

3.3.1 Why Problem?

According to the CAN-FD specification [87], the format of CAN-FD data frame is partially different from that of CAN data frame to support higher bandwidth and larger payload size as shown in Fig. 2.1 and Fig. 2.2. Due to this frame format difference, CAN nodes always recognize CAN-FD frames as erroneous frames (CRC error) and generates an error frame whenever they receive a CAN-FD frame. Because of this incorrect error detection, an 'innocent' CAN-FD frame will be retransmitted by the sender and the retransmitted CAN-FD frame will again be detected as an erroneous frame as illustrated in Fig. 3.2 (Left). This makes the communication between CAN-FD nodes via CAN-FD frames impossible, thus losing all the advantages of CAN-FD. Both hardware [3, 60] and software (silent mode-based) [67] solutions to this problem have been proposed recently.

3.3.2 Hardware Solution

The hardware solutions [3, 60] require an additional hardware component which filters out the CAN-FD frames by inspecting FDF (FD Format) bit of all incoming frames in front of

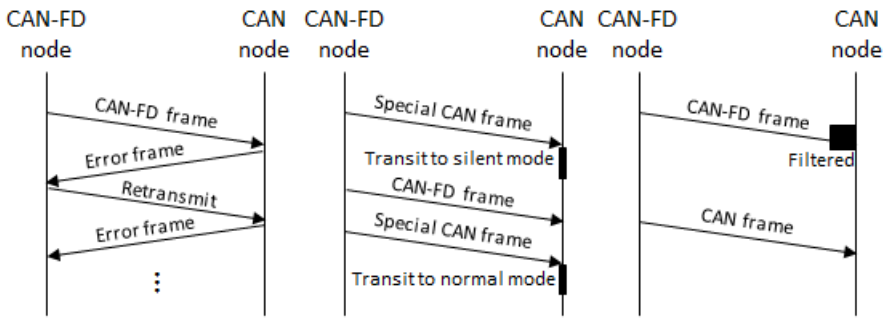


Figure 3.2: (Left) Problem in scheduling mixed CAN and CAN-FD frames; software (Mid) and hardware (Right) solutions

the CAN controller as shown in Fig. 3.2 (Right). Thus, CAN-FD nodes can communicate with each other using CAN-FD frames without any software change. However, the specialized hardware has to be attached to all the CAN nodes, increasing implementation and deployment costs. A single hardware component might be inexpensive, but the cost for its mass production could be significant. Note that more than 100 million new vehicles are built and sold each year [1].

3.3.3 Software Solution

The software solution [67] relies on the silent mode of the legacy CAN controllers. Before transmitting a CAN-FD frame, an ECU has to transmit a special CAN frame (trigger frame) that triggers a mode transition. All CAN nodes must transit from normal mode to silent mode upon receiving a trigger frame. A CAN-FD node then begins the transmission of CAN-FD frames as shown in Fig. 3.2 (Mid). Since all the CAN nodes are in silent mode, CAN-FD nodes can communicate with each other using CAN-FD frames. A CAN-FD node must thereafter send another trigger frame to wake up the CAN nodes from silent mode to normal mode as shown in Fig. 3.2 (Mid). Although the software solution can resolve the problem without any additional hardware, it incurs non-negligible delays for mode transitions.

3.3.3.1 Analysis of Mode-Transition Delay

The time overhead of the software solution for a mode transition consists of two parts; *transmission time* of a trigger frame and *processing time* of a mode transition.

The transmission time of a trigger frame depends on the CAN bus speed as well as its payload size. We must thus define and use the format of a trigger frame to compute its transmission time. So, a trigger frame is differentiated from a normal data frame by

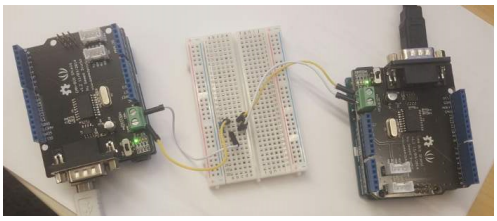


Figure 3.3: Experimental platform

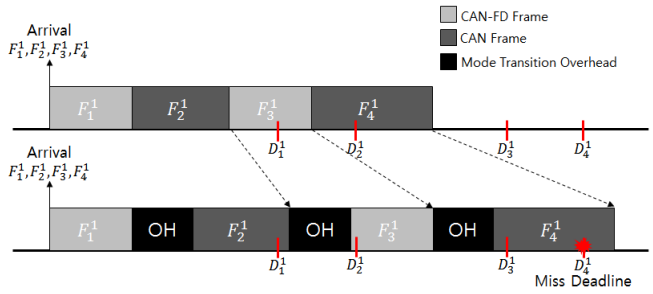


Figure 3.4: Due to the time overhead, the delivery/completion time of a given frame increases and a frame misses its deadline

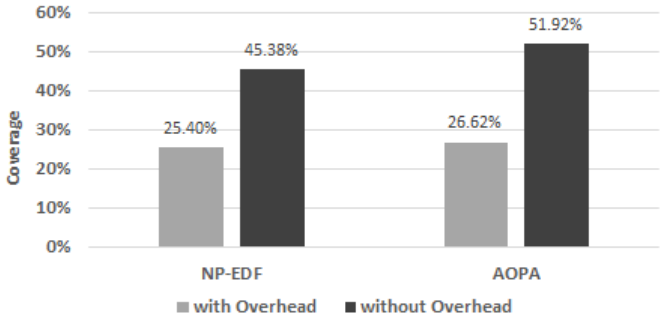


Figure 3.5: The coverage of existing optimal priority assignment with or without the mode-transition overhead for given mixed frame sets.

assigning it a unique ID (of 11 bits) and its payload size is set to 0. With this setting, if the CAN bus speed is 500Kbps then the transmission time of the trigger frame is 112μs.

The processing time of a mode transition depends on the computing power of a CAN node. However, since a mode transition is very simple (writing a value to a register in the CAN controller and reading the changed value in the register), there will be only a small processing time variation. We have measured the processing time on our experimental platform as shown in Fig. 3.3. We use Arduino [5] and MCP2515 CAN controller [64] to build a CAN node. The processing time was about 84μs on average, and hence a mode transition takes about 200μs in total (500Kbps CAN bus speed).

3.3.3.2 Negative Impact of Time Overhead

Fig. 3.4 illustrates the negative impact of the time overhead of a mode transition; the delivery/completion times of F_2^1, F_3^1, and F_4^1 increase, missing the deadline of F_4^1. That is, the time overhead degrades the schedulability of a given mixed frame set. According to our simulation results (in Section 7), more than 20% of given mixed frame sets become unschedulable due to the overhead with the existing frame-level optimal priority assignment (AOPA [7]) and frame-instance-level optimal priority assignment (NP-EDF [48]) as shown in Fig. 3.5.

3.4 Problem Statement

Placing CAN and CAN-FD nodes on the same network to utilize the established CAN infrastructure is problematic as we discussed earlier, and thus hardware and software solutions have been proposed to address the problem. The software solution is more attractive than the hardware solution for a cost reason, but it degrades the schedulability of mixed CAN and CAN-FD frame sets due to its reliance on the mode transitions of the existing CAN controllers. As a result, the schedulability or coverage of the existing optimal priority assignment degrades significantly, i.e., the software solution with the existing optimal priority assignment fails to schedule many mixed frame sets while meeting all of their frame deadlines.

In order to enable the software solution to schedule more mixed frame sets, we propose a new priority-assignment algorithm, called *Priority Assignment with Mode Transition* (PAMT), that minimizes the schedulability degradation by minimizing the mode-transition overhead.

3.5 Priority Assignment with Mode Transitions

We present PAMT for a given mixed frame instance set, which minimizes the coverage loss of the software solution. We first introduce the basic idea of PAMT and then provide its details.

3.5.1 Basic Idea of PAMT

Non-Preemptive Earliest Deadline First (NP-EDF) based priority assignment and *type-based clustering* are the key of PAMT.

Frame	T_i	D_i	J_i	FD_i	C_i
F_1	5ms	0.75ms	0ms	0	272μs
F_2	5ms	1ms	0ms	1	320μs
F_3	5ms	1.5ms	0ms	0	272μs
F_4	5ms	1.75ms	0ms	1	400μs

Table 3.1: An example frame set

PAMT assigns priorities to the frame instances in a given mixed frame set based on NP-EDF, because NP-EDF is known to be optimal for work-conserving system like CAN [48] if there were no mode transition overhead. However, the software solution may incur a high mode-transition overhead, and hence PAMT performs type-based clustering of frame instances to reduce the mode-transition overhead. For example, suppose that priorities are assigned to the frame instances of a given mixed frame in Table 3.1. PAMT first assigns priorities to the frame instances based on NP-EDF, but this incurs 3 mode transitions, causing F_4^1 to miss its deadline as shown in Fig. 3.6 (Top). To reduce mode transitions and to make the given mixed frame set schedulable, PAMT performs type-based clustering. As illustrated in Fig. 3.6 (Bottom), after the type-based clustering, the same-type frame instances are clustered and the number of mode transitions is reduced to 1. As a result, there are no deadline misses with the clustered priority ordering, making the given mixed frame set schedulable.

3.5.2 PAMT Algorithm

Even though the type-based clustering is a natural way to reduce mode transition overheads, it is challenging to group frame instances so as to minimize the degradation of schedulability, because the type-based clustering can increase the delivery/completion times of frame instances (e.g., F_2^1 in Fig. 3.6 (Bottom)), which can cause unexpected deadline misses.

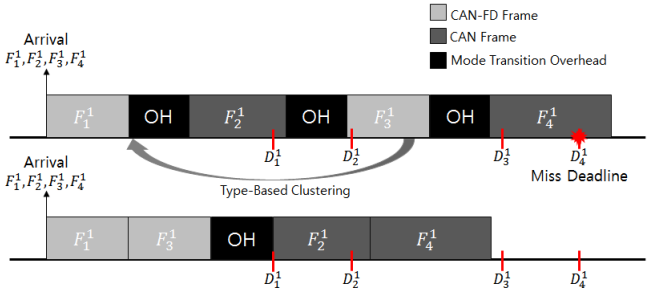


Figure 3.6: (Top) Assign priorities to frame instances based on NP-EDF; (Bottom) reducing mode-transition overheads via type-based clustering

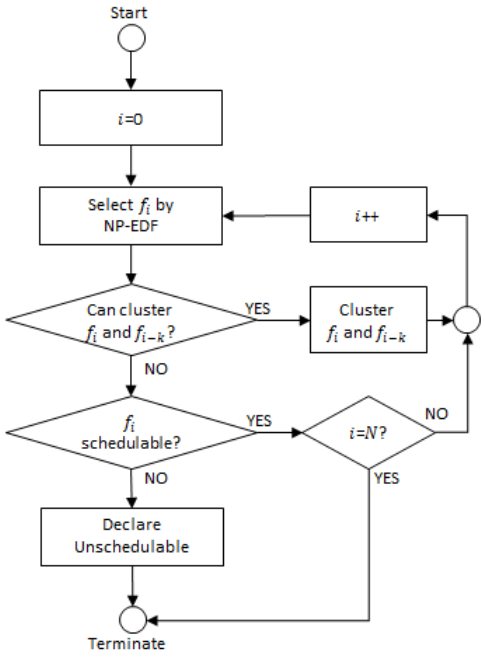


Figure 3.7: Flowchart of PAMT

Next, we will first give an overview of PAMT algorithm that meets the above challenge, and then give a detailed account of each part of the algorithm.

3.5.2.1 Algorithm Overview

Fig. 3.7 shows how PAMT operates on a given mixed frame instance set I. PAMT selects a frame instance according to NP-EDF and assigns priority i to the frame instance as the first step. After selecting the frame instance (f_i), PAMT checks several conditions to cluster f_i and f_{i-k}, where f_{i-k} is the nearest frame instance whose type is the same as the type of f_i. If all conditions are met, PAMT performs the type-based clustering. Otherwise, PAMT does not perform the type-based clustering but just checks whether f_i meets its deadline or not. If f_i meets its deadline, PAMT selects another frame instance to assign priority $i + 1$ according to NP-EDF. Otherwise, PAMT declares the given mixed frame instance set I unschedulable and terminates the process. This process will be repeated until all the frame instances in I are assigned priorities. The pseudo-codes of PAMT implementation are stated in Algorithm 1 and 2. Next, we will detail how to implement each procedure.

3.5.2.2 Select f_i according to NP-EDF

We need an off-line assignment of priorities to the frame instances in a mixed frame set I even though NP-EDF is an on-line scheduling algorithm. So, we simulate a planning

Algorithm 1: PAMT

Input : I: Mixed frame instance set
o_{mode}: Mode transition overhead
Output: I_{pas}: Priority-assigned frame instance set

```
1  init(); // Initialize virtual time, competing set and priority assigned set
2  N ← |I|;
3  while N ≠ |I_pas| do
4      isMigrated ← false;
5      sort(I); // By arrival time — ascending order
6      for k ← 0 to |I| − 1 do
7          if I[k].arrival_time ≤ t_v then
8              I_cs[|I_cs|] ← I[k]; I[k] ← null; isMigrated ← true;
9          end
10     end
11     if isMigrated = false and |I_cs| = 0 then
12         t_v ← I[0].arrival_time;
13         continue;
14     end
15     sort(I_cs); // By deadline — ascending order
16     idx_sel ← 0; f_sel ← I_cs[0]; // f_sel: selected frame instance by NP-EDF
17     for k ← 1 to |I_cs| − 1 do
18         if I_cs[k].deadline = f_sel.deadline and I_pas[|I_pas| − 1].type = I_cs[k].type
             then
19             f_sel ← I_cs[k]; idx_sel ← k;
20         end
21     end
22     I_pas[|I_pas|] ← f_sel; I_cs[idx_sel] ← null;
23     if Cluster(I_pas) = true then
24         t_v ← t_v + f_sel.transmission_time;
25     end
26     else
27         t_v ← t_v + f_sel.transmission_time;
28         if I_pas[|I_pas| − 1].type ≠ I_pas[|I_pas| − 2].type then
29             t_v ← t_v + o_mode;
30         end
31         I_pas[|I_pas| − 1].completion_time ← t_v;
32         if I_pas[|I_pas| − 1].completion_time > I_pas[|I_pas| − 1].deadline then
33             return null; // Declare unschedulable
34         end
35     end
36 end
37 return I_pas;
```

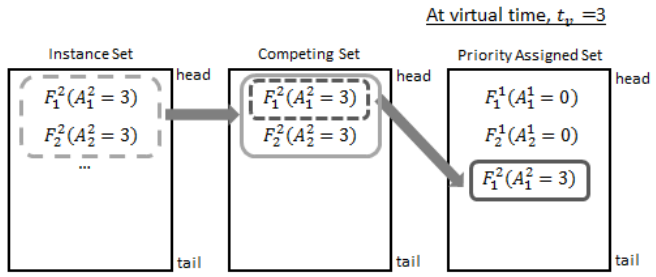


Figure 3.8: Assign priority to a frame instance based on NP-EDF

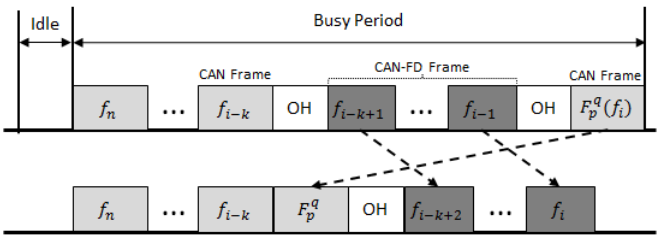


Figure 3.9: Type-based clustering. Promoting the priority of f_i to $i-k+1$ to reduce the mode-transition overhead

cycle or hyper-period (HP) of I to assign priorities to the frame instances in I. Usually, the periods of in-vehicle CAN frames are harmonic [79], and thus an HP is not too long to simulate. According to our measurements through on-board diagnostic (OBD) port, HP is 6s for 2015 Chevrolet Trax LT AWD.

To simulate an HP, we manage a virtual time t_v and 3 data structures: *mixed-frame instance set* (I), *competing set*, *priority-assigned set*. This simulation requires I to be sorted in ascending order of frame instances' arrival times and the virtual time to be initialized with 0. Described below is how the HP is simulated.

At time t_v, if the arrival times of frame instances in I are earlier than, or equal to t_v, then the frame instances are migrated from I to the competing set as shown in Fig. 3.8. After the migration, PAMT sorts the competing set in ascending order by the deadline, and then selects the frame instance with the earliest deadline from the competing set to assign priority i. If two or more frame instances have the same earliest deadline, PAMT selects one of them that has the same type as f_{i-1} to avoid a mode transition between f_{i-1} and f_i. The chosen frame instance is then moved to the tail of the priority-assigned set as shown in Fig. 3.8. Note that the index of a priority-assigned set indicates the priority of a frame instance.

3.5.2.3 Cluster f_i and f_{i-k}

Suppose PAMT selects the frame instance (F_p^q) to assign priority i and f_{i-1} has a different type from F_p^q. Let f_{i-k} be the nearest frame instance which has the same type as F_p^q (f_i), and both f_{i-k} and F_p^q belong to the same busy period as shown in Fig. 3.9. Clearly, promoting the priority of F_p^q to $i-k+1$ reduces the number of mode transitions since it eliminates the mode transition between F_p^q and f_{i-1}. However, this priority promotion is not always possible. All of the following conditions must be met for the priority promotion:

C1: $f_{i-1}.type \neq f_i.type$

C2: After promoting the priority of f_i to $i-k+1$, $d_p \geq e_p, \forall p \in \{i-k+1, \ldots, i\}$

C3: $a_i \leq a_{i-k+1}$ or $a_i \leq e_{i-k}$

where a_i is the arrival time of f_i, d_i is the deadline of f_i, and e_i is the completion time of f_i.

C1 is obvious, and hence its discussion is omitted. If we promote the priority of f_i to $i-k+1$, then it will delay the completion of frame instances between f_i and f_{i-k}. If any of these delayed frame instances violates its deadline, then the priority promotion is not allowed. That is, the delayed frame instances must finish before their deadlines and **C2** must hold.

The last condition **C3** comes from the unique characteristic of CAN scheduling, non-preemptive work-conserving scheduling. Suppose $a_i > a_{i-k+1}$ and $a_i > e_{i-k}$ as shown in Fig. 3.10. We expect F_p^q to be scheduled right after the transmission of f_{i-k} by promoting the priority of F_p^q to $i-k+1$ (dotted line). However, f_{i-k+2} is scheduled before transmission of $F_p^q(f_{i-k+1})$ (solid line), since there is no ready frame instance at e_{i-k} and f_{i-k+2} arrives before F_p^q arrives. So, clustering f_i and f_{i-k} is impossible even though the priority of F_p^q is promoted to $i-k+1$. The priority promotion in this case could rather increase the number of mode-transitions as shown in Fig. 3.10.

After executing the above procedures, the virtual time is updated to the completion time of the lowest-priority frame instance. If there is no frame instance to be moved from the mixed frame instance set to the competing set and if the competing set is empty, then the arrival time of the frame instance at the head of the instance set is assigned as the next virtual time. For example, if the arrival time of the frame instance at the head of the instance set is A_p^q, then the virtual time becomes A_p^q.

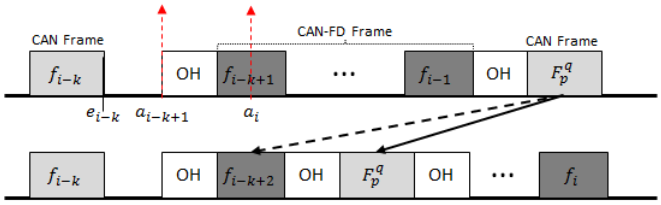


Figure 3.10: Violation of **C3**. ($a_i > a_{i-k+1}$ and $a_i > e_{i-k}$)

3.5.3 Optimality of PAMT

We now prove that PAMT is the optimal priority assignment for a given mixed frame instance set. That is, if PAMT cannot schedule a given mixed frame instance set, then no other priority assignment algorithm can find a schedulable priority order for the mixed frame instance set.

Lemma 1. *Let I_A be the set of frame instances in a busy period A. PAMT minimizes the number of mode transitions for I_A if no deadline miss is allowed.*

Proof: Let S_k ($S_k \subset I_A$) be the set of chosen frame instances whose cardinality is k. We will show that PAMT minimizes the number of mode transitions for S_k regardless of k if no deadline miss is allowed. This way, we can prove Lemma 1 because S_k is the same as I_A if $k = |I_A|$.

1. (Initial, $k = 1$). Since there is only one frame instance, there is no mode transition.

2. (Suppose this holds for $k = i - 1$). Assume that PAMT minimizes the number of mode transitions for S_{i-1} without any deadline miss.

3. (Show this holds for $k = i$). Let g be the i^{th} frame instance chosen by PAMT. To avoid any additional mode transition, we need to schedule g right after the transmission of a frame instance whose type is the same as that of g. Since PAMT selects frame instances according to NP-EDF, we only consider placing g right after f_{i-k} which is the latest same-type frame instance in S_{i-1}. Let's analyze the following three cases.

Case 1. Suppose the arrival time of g is earlier than, or equal to e_{i-k} and placing g right after f_{i-k} does not cause any deadline miss. Then, we need to show that PAMT moves g to right after f_{i-k}. We know that **C**3 ($a_i \leq e_{i-k}$) and **C**2 holds by supposition. If **C**1 is not met, then $f_{i-k} = f_{i-1}$. So, g is already placed at right after f_{i-k}. Otherwise, all three conditions are met and PAMT moves g to right after f_{i-k} by performing type-based clustering. Thus, g doesn't incur any additional mode transition, and hence PAMT minimizes the mode transitions.

Case 2. Suppose the arrival time of g is larger than e_{i-k}. For this case, we need to show that additional mode transitions incurred by g are unavoidable and only one additional

mode transition is incurred by g under PAMT. Since all the frame instances scheduled after f_{i-k} have different types from g, the type of the frame instance scheduled right before g is different from that of g. Thus, the additional mode transition before transmitting g is unavoidable. In this case, PAMT does not perform type-based clustering due to the violation of **C**3. Instead, PAMT schedules g last, i.e., there is no frame instance after g and the number of mode transitions incurred by g is 1. So, PAMT minimizes the number of mode transitions.

Case 3. Suppose the arrival time of g is earlier than, or equal to e_{i-k} and placing g right after f_{i-k} causes at least one deadline miss. In this case, we need to show that the additional mode transitions incurred by g are unavoidable and only one additional mode transition is incurred by g under PAMT. Since placing g right after f_{i-k} causes at least one deadline miss, g must be scheduled after f_{i-k+1} to avoid any deadline miss. However, the frames instances (from f_{i-k+1} to f_{i-1}) have different types from g. Thus, the additional mode transition before transmitting g is unavoidable. In this case, PAMT does not perform type-based clustering due to the violation of **C**2. Instead, PAMT schedules g last, i.e., there is no frame instance after g and the number of mode transitions incurred by g is 1. Thus, PAMT minimizes the number of mode transitions. □

Theorem 1. *PAMT is the optimal priority-assignment algorithm for a mixed frame instance set I.*

Proof: We prove this theorem by induction. Let K be the number of priority-assigned frame instances and let g_i be the i^{th} frame instance chosen by PAMT.

1. (Initial, $K = 1$). Since there is only one frame instance, every priority-assignment algorithm is optimal.

2. (Suppose this holds for $K = i - 1$). Assume that PAMT is the optimal for $I_{i-1} = \{g_1, \ldots, g_{i-1}\}$.

3. (Show this holds for $k = i$). We will show that PAMT is optimal for the mixed frame instance set $I_i = \{g_1, \ldots, g_i\}$ by proving that there is no schedulable priority order for I_i if PAMT declares I_i unschedulable. Let's consider the following two cases.

Case 1. The type of g_i is the same as that of f_{i-1}. In this case, PAMT schedules g_i last (after f_{i-1}) because **C1** is not met. If g_i meets its deadline, PAMT makes I_i schedulable. However, if g_i misses its deadline, we need to show that there is no schedulable priority order for I_i. Suppose g_i misses its deadline. There are two ways to make g_i schedulable: (1) reduce the number of mode transitions during a busy period in which g_i resides; (2) schedule g_i earlier than last. By Lemma 1, PAMT minimizes the number of mode transitions in a busy period, so there is no way to reduce the number of mode transitions. Thus,

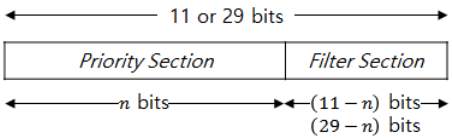


Figure 3.11: Separation of CAN ID into priority and filter sections

we only need to consider (2). Since PAMT selects a frame instance according to NP-EDF, the frame instances, which are transmitted after the arrival of g_i, have earlier deadlines than g_i. This means that scheduling g_i at any possible instant causes at least one deadline miss. Thus, there is no schedulable priority order for I_i, and hence PAMT is optimal.

Case 2. The type of g_i is different from that of f_{i-1}, satisfying **C1**. Thus, if both **C2** and **C3** are met, PAMT schedules g_i right after f_{i-k} which is the nearest same-type frame instance in I_{i-1} and every frame instance in I_i meets its deadline (**C2**), making PAMT optimal. If either **C2** or **C3** is not met, PAMT schedules g_i last. If g_i meets its deadline last, every frame instance meets its deadline, thus making PAMT optimal. If g_i misses its deadline, we need to show that there is no schedulable priority order for I_i. As in **Case 1**, there are two ways to make g_i schedulable and we only need to consider the second case by Lemma 1. Also, like **Case 1**, scheduling g_i at any possible instant causes at least one deadline miss because the frame instances transmitted after the arrival of g_i have earlier deadline than g_i. Thus, there is no schedulable priority order for I_i, hence making PAMT optimal. □

3.6 Practical Issues

3.6.1 Assigning ID to frame instances

A CAN controller does not, in practice, accept all incoming CAN frames to reduce the processing load of the host ECU [80]. The CAN controller filters the incoming CAN frames by comparing the IDs of incoming frames with the registered values in its receive filter. However, the number of receive filters is limited in a commercial CAN controller (e.g., only 6 filters in MCP2515 [64]). The limited number of receive filters makes difficult to implement PAMT because PAMT assigns priorities to frames instances, not frames. For example, when a frame F_1 is instantiated three times in a planning cycle, these three different instances have different priorities/IDs. Then, to receive all the instances F_1, an ECU has to register all of the three IDs in its receive filters.

To resolve this problem, we separate a CAN ID into *priority* and *filter* sections, as shown in Fig. 3.11. The priority section is only used to distinguish the priority of CAN frames. Thus, the bits in the priority section are set as '*don't care bits*' in the mask reg-

isters[2]. Since the priority section is forwarded to the filter section, it can be used in the ID arbitration process. That is, the lower the number in the priority section, the higher the priority in CAN frame scheduling. We can set the priority determined by PAMT in the priority section directly. The filter section is only used for filtering the incoming CAN frames. We give a unique value to each frame (not frame instance) and put the value in the filter section. For example, we give a value of 1 to F_1 and put the value of 1 to the filter section of the instances of F_1. As a result, an ECU can receive all the instances of F_1 by using only one (not multiple) receive filter(s).

Let n be the number of bits in the priority section with which all possible priorities must be covered. PAMT assigns a unique priority to each frame instance in a given mixed-frame set I. We also reserve whole odd numbers for a special CAN frame to trigger a mode transition. A frame instance can only have an even-numbered priority. For example, the priority of f_{i-1} is 0x10 and that of f_i is 0x12. 0x11 is reserved for the trigger frame (see Section 7.2). Hence, 2^{n-1} has to be larger than the number of frame instances in I and we select the minimum n that satisfies this requirement. Also, 2^{11-n} or 2^{29-n} has to be larger than the number of frames in a given mixed-frame set because we need to assign a unique value to each frame.

3.6.2 Triggering a Mode Transition

Triggering a mode transition precisely at the specified time is important because a late/early mode transition can cause severe problems. For example, a CAN node will generate an error frame when the node receives a CAN-FD frame due to a late transition (from normal to silent). Also, a CAN frame instance, which is sent by a CAN-FD node, may not be delivered to a CAN node due to the late transition (from silent to normal). To transmit a trigger frame at a precise time, we mark frame instances after which a trigger frame must be transmitted. For example, if f_{i-1} is a CAN frame and f_i is a CAN-FD frame, then we mark f_{i-1}. Since frame instances in the priority-assigned set are sorted in their transmission order, we can easily determine which frame instances should be marked.

At runtime, if an ECU queues a marked frame instance in its transmission buffer (TxObject), it also queues a trigger frame in TxObject. To transmit the trigger frame right after the marked frame instance, the value in the priority section of the trigger frame is larger by 1 than that of the marked frame instance. For example, as shown in Fig. 3.12, the value in the priority section of a marked frame instance (f_{i-1}) is 0x10 and that of the corresponding trigger frame is 0x11. This way, the ID of a trigger frame can be larger than that of any

[2]When the CAN controller performs bitwise comparisons, it ignores several bits which are set to the *'don't-care bits'* in a mask register.

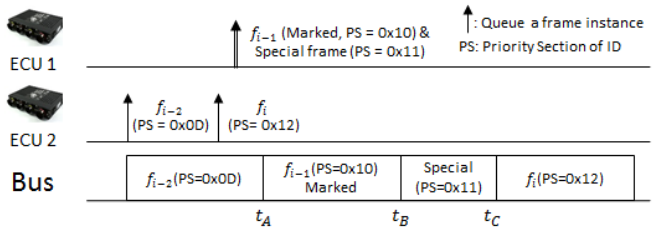


Figure 3.12: Insert a special CAN frame

other frame instances queued in TxObject. Thus, the trigger frame can be transmitted right after the transmission of the corresponding marked frame instance by winning the CAN bus arbitration.

3.6.3 Transient Error

Rare transient errors (bit error) can occur on CAN bus due to electromagnetic interference (EMI). Since each transient error is handled by generating error frames and retransmitting the unsuccessful CAN frame, the error causes additional delays to the delivery/response time of CAN frames. That is, the frame instances scheduled under PAMT may miss their deadlines due to the transient errors. To account the transient errors, we compute the maximum possible delay to f_i caused by the transient error (α_i^{tc}) and reflect the delay into the deadline.

$$\alpha_i^{tc} = \eta(\lambda, R, d_i) \times E \tag{3.1}$$

where λ is the maximum transient error rate determined during the design phase of a vehicle according to the knowledge of the worst environment in which the vehicle operates [72], R is the reliability requirement of a vehicle system, $\eta(\lambda, R, d_i)$ is the maximum number of transient errors possible within d_i, and E is the error recovery time [21]. Here, we compute $N = \eta(\lambda, R, d_i)$ under the assumption that the process of transient errors is Poisson, as commonly used for CAN transient errors [21, 12]:

$$\begin{aligned} N = \quad & \underset{Z_m}{\operatorname{argmin}} \quad 1 - \sum_{Z=0}^{Z_m} p(Z, d_i) \\ & \text{subject to} \quad 1 - \sum_{Z=0}^{Z_m} p(Z, d_i) \leq R. \end{aligned} \tag{3.2}$$

where $p(Z, d_i)$ is the probability of Z errors within d_i.

$$p(Z,t) = \frac{e^{-\lambda t}(\lambda t)^Z}{Z!}. \tag{3.3}$$

3.6.4 Unsynchronized Clock

Since there is no global clock on CAN, CAN frames are triggered according to the local time clock of each ECU. However, the local clocks are not synchronized with each other, and thus there is a clock drift/skew between ECUs. Unfortunately, this clock drift may alter the scheduling order of messages at runtime. For example, if the maximum drift on an ECU is δ and f_i and f_j $(i < j)$ are sent by different ECUs and $a_i + \delta > a_j - \delta$, f_j could be transmitted before f_i.

Because the runtime change in scheduling order can incur an additional delay to the delivery/response time of CAN frames, the frame instances scheduled under PAMT may miss their deadlines. Thus, as in the previous subsection, we compute the maximum possible delay to f_i caused by the unsynchronized clocks (α_i^{uc}) and account for the delay in the deadline:

Here we assume that a software-based synchronization protocol [88] for CAN is applied, and thus the maximum drift (δ) is limited and the arrival time range of f_i is also limited; $a_i \in \left[a_i - \delta, a_i + \delta\right]$. According to the re-synchronization interval condition in [88], we can infer that the maximum drift can be limited by $20\mu s$ with 5s re-synchronization interval, and the $20\mu s$ is much smaller than the transmission time of a CAN frame. So, we assume that $\delta < \min_{i \in F} C_i$ and $\delta < E$.

3.6.4.1 Finding maximum possible delay by unsynchronized clocks

Suppose $I_{i,rev} = \{f_j | i < j \ \& \ a_i + \delta \geq a_j - \delta\}$. Then, the frame instances in $I_{i,rev}$ can be transmitted before transmitting f_i due to the clock drift at runtime unlike the deterministic scheduling order by PAMT.

$I_{i,rev,diff}$ is the set of frame instances whose type is different from the type of f_i in $I_{i,rev}$, and $I_{i,rev,same}$ is the set of frame instances whose type is the same as that of f_i in $I_{i,rev}$.

Lemma 2. *f_i is always transmitted before f_j if $f_j \in I_{i,rev,diff}$.*

proof: Suppose the type of f_i is different from that of f_j. Then, there is at least one marked frame instance f_k $(i \leq k < j)$ to trigger a mode transition. If $a_k + \delta > a_j - \delta$, f_j can be queued before queuing f_k. However, the mode is not changed yet at the arrival of f_j.

Thus, f_j cannot be transmitted before f_k because a FD frame cannot be queued in the CAN mode and a CAN frame cannot be transmitted in the FD mode. If f_i arrives earlier than f_k, f_i is transmitted before f_j. If f_k arrives earlier than f_i (in the case of $f_i + \delta > f_k - \delta$), f_k can be transmitted before f_i. Since $\delta < \min_{i \in F} C_i$, f_i is guaranteed to arrive during the transmission of f_k. Thus, f_i is transmitted right after the transmission of f_k. Hence, f_i is transmitted before f_j.

Lemma 3. *The maximum delay to f_i caused by the reversed order between f_i and $f_j \in I_{i,rev,same}$ is $\max_{f_j \in I_{i,rev,same}} c_j + \delta$ where c_j is the transmission time of f_j.*

proof: Suppose the type of f_i is the same as that of f_j, and f_j is transmitted before f_i at runtime due to the unsynchronized clocks. Because f_i is guaranteed to arrive during the transmission of f_j, f_i is always transmitted right after the transmission of f_j. The worst-case scenario which contributes the maximum delay to f_i is that f_j arrives at $a_i + \delta - \epsilon$ and f_i arrives at $a_i + \delta$ where ϵ is a very small. In the worst case, the delay to f_i is $c_j + \delta$. Thus, the maximum delay to f_i caused by the reversed order between f_i and $f_j \in I_{i,rev,same}$ is $\max_{f_j \in I_{i,rev,same}} c_j + \delta$.

Collorary 1. *α_i^{uc} is $\max_{f_j \in I_{i,rev,same}} c_j + \delta$*

proof: By Lemma 2, if $f_j \in I_{i,rev,diff}$, f_i is always transmitted before f_j. In such a case, the maximum delay of f_i due to the unsynchronized clocks is δ. Since $\delta \leq \max_{f_j \in I_{i,rev,same}} c_j + \delta$ (by Lemma 3), α_i^{uc} is $\max_{f_j \in I_{i,rev,same}} c_j + \delta$.

3.6.5 Sporadic Frames

As described in Section 3.C, frames arrive periodically. However, in practice, some frames can be triggered by asynchronous events or vehicle conditions, and arrivals of such messages can be represented with a sporadic frame model (with minimum inter-arrival times). Since our approach is designed with a periodic model, sporadic frames should be converted to periodic frames by using their minimum inter-arrival time as the period. Suppose, for example, transmitting a message which includes brake pedal pressure is triggered by an event that a vehicle driver is pressing the brake pedal. This message should be converted to a periodic message. So, a null message (when the driver does not press the brake pedal) or a message that contains the brake pedal pressure (when the driver presses the brake pedal) should be sent periodically.

3.7 Evaluation

We have conducted extensive simulations to evaluate PAMT in comparison with NP-EDF, optimal frame-instance level priority assignment, and Audsley's Optimal Priority Assignment (AOPA), the well-known optimal frame-level priority assignment. We focus on the schedulability degradation of each priority assignment algorithm by measuring its coverage. We also measure the coverage of the optimal priority assignments when we use the hardware-based solution by ignoring the mode-transition overhead (labeled with AOPA* and NP-EDF*). The coverage of the hardware-based solution is the best achievable because AOPA and NP-EDF are proven to be optimal for CAN scheduling. We also evaluate PAMT-R, which accounts for transient errors (Section 6.C) and unsynchronized clocks (Section 6.D) by using a virtual deadline $d_{i,r} = d_i - \alpha_i^{tc} - \alpha_i^{uc}$ instead of d_i. To compute $d_{i,r}$, we set $\lambda = 0.01/s$, $R = 2.6 * 10^{-}9/s$ (SIL in IEC-61508 [2]) and $\delta = 20\mu s$.

3.7.1 Simulation Setup

3.7.1.1 The Benchmark for Simulations

We use NETCARBENCH [19] (powertrain configuration), which is a widely-used CAN benchmark. Since its latest version does not yet support the CAN-FD frame, we slightly modified NETCARBENCH to support CAN-FD. If the payload of a generated frame is larger than 8, then the type of the frame is CAN-FD. Otherwise, we assign the frame type randomly. For our simulation, we generated 10,000 mixed frame sets from NETCARBENCH. As a result of frame type assignment, the CAN-FD frame ratio is in the range of [0.361, 0.79], and about 60% of the simulated mixed frame sets have 40-60% CAN-FD frame ratio. We assume that the jitter of each frame is 0 and the transmission of all the frames begins at time 0.

3.7.1.2 Simulation Configuration

500Kbps is used as the bit-rate for the arbitration phase and 2Mbps for the data phase, because 500Kbps is commonly used for the powertrain network [30] and up to 2Mbps is supported by the current commercial CAN-FD transceiver which satisfies the automotive OEM's EMC requirement [44]. Also, we use 200μs as the mode-transition overhead according to our experimental measurement.

In addition, we use 11-bit IDs to simulate AOPA and AOPA* because there are typically about 100 different frames[3] for a single in-vehicle CAN bus [79]. and 11-bit ID suffices for

[3]We observed 96 different messages in 2015 Chevorlet Trax LT AWD through the On Board Diagnostic

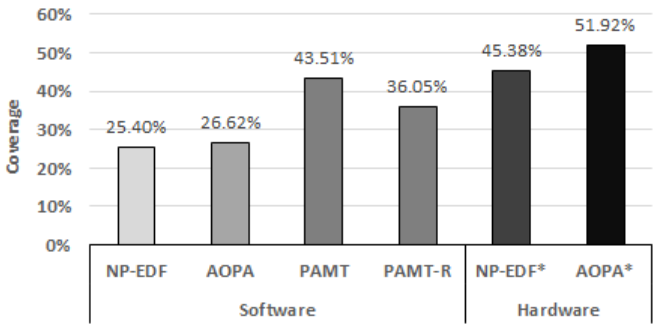


Figure 3.13: Coverage of each priority assignment algorithm for the generated frame sets.

that number. However, we use 29-bit IDs to simulate PAMT, PAMT-R, NP-EDF, and NP-EDF* because these frame-instance-level priority assignment algorithms require multiple IDs for a frame and 11 bits are not enough.

3.7.2 Results and Analysis

Coverage: is defined as the percentage of given frame sets whose *schedulable* priority order is found by each priority assignment algorithm. We evaluate the coverage of each priority assignment algorithm for the generated mixed frame sets. Fig. 3.13 plots the simulation results. PAMT is shown to outperform the existing optimal priority assignments when the software solution is used. PAMT can find a schedulable priority order for 17–18% more mixed frame sets than the existing optimal priority assignments when we use the software solution because PAMT effectively reduces the negative impact of mode transitions on schedulability by performing type-based clustering. PAMT-R has about 7.5% less coverage than PAMT because each frame has the reduced deadline to account transient error and unsynchronized clock. But, PAMT-R still has 10% larger coverage than existing optimal priority assignments. Thus, for 10% additional mixed frame sets, we can use the economic software solution with our approach. Also, the software solution is shown to have lower coverage than the hardware solution because the latter is not affected by the negative impact of mode transitions on schedulability. The coverage difference between PAMT and AOPA* (the maximum achievable coverage using 11-bit ID) is about 8%. Also, the coverage difference between PAMT and NP-EDF* (the maximum achievable coverage using 29-bit ID) is about 2%. Because 29-bit ID is common in trucks [90], PAMT is now more useful for trucks, although this may change in future. Interestingly, the coverage of NP-EDF is lower than the coverage of AOPA due to its use of 29-bit ID.

(OBD) port.

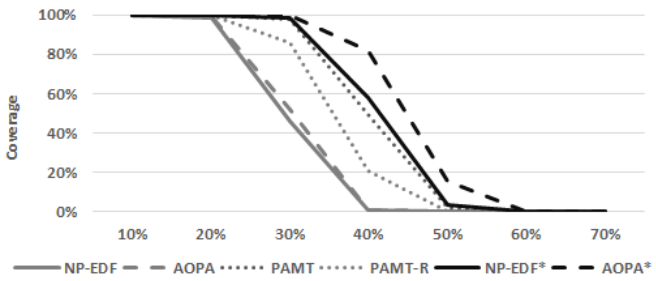


Figure 3.14: Coverage of each priority assignment algorithm while varying utilization.

Varying utilization: Fig. 3.14 shows the coverage of each priority assignment algorithm while varying the utilization of the generated frame sets. PAMT outperforms the existing priority assignment algorithms regardless of the utilization. PAMT can cover over 97% of mixed frame sets if the utilization of the mixed frame sets is in the range of 30–40%. The number is 52% higher than NP-EDF and 46% higher than AOPA. Fig. 3.15 shows the number of mode transitions incurred by each priority assignment algorithm. The number of mode transitions incurred by PAMT is much smaller than those incurred by AOPA and NP-EDF and the gap in the number of mode transitions between PAMT and the others becomes larger as the utilization of mixed frame sets increases. For example, the number of mode transitions incurred by AOPA is 1.6x larger than that by PAMT in the 10–20% range and the gap becomes 2.5x in the 60–70% range. The coverage improvement of PAMT over existing optimal priority assignments comes from the reduced number of mode transitions.

3.8 Related Work

Priority assignment affects greatly the schedulability of a given CAN frame set [28]. Representative frame-level fixed priority assignment algorithms are Deadline minus Jitter Monotonic Priority Order (DJMPO) [111] and AOPA [7]. AOPA is proven optimal [82] if there were no priority inversion which may occur in practice [26, 52, 51]. Since frame-level fixed priority is less efficient than frame-instance-level fixed priority in utilization, use of the frame-instance-level fixed priority has been proposed [69, 110]. The representative frame-instance-level fixed priority assignment algorithm is NP-EDF, which is proven to be optimal among work-conserving scheduling algorithms for periodic tasks [48].

As in a typical electronic system, signals on CAN are interfered with by EMI [84], which may induce bit errors by distorting the signals. Since error recovery delays the delivery of CAN data frames, it impacts the schedulability of a given CAN frame set. Thus, Davis *et al.* [27] proposed a robust priority assignment algorithm which not only is

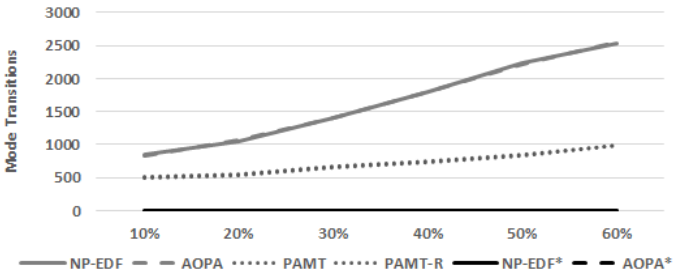


Figure 3.15: The number of mode transitions required by each priority-assignment algorithm in a planning cycle (X-axis: utilization)

optimal but also maximizes the number of successive tolerable transmission errors.

Since assigned IDs of the existing frame sets usually do not change even though new frames are introduced for new functions (e.g., updating an ECU), researchers focused on the backward compatibility of priority assignment. Schmidt [89] proposed a robust priority-assignment algorithm for a frame set when some frame IDs are fixed. Davis *et al.* [28] showed the existence of flaws in [89] when there is not an enough gap between fixed IDs and proposed a correction of robust priority assignment. Davis *et al.* [29] also consider optimal priority assignment under mixed use of FIFO queues and priority queues.

Prior work on priority assignment for CAN focused on the single-type frame set, and thus is agnostic of mode-transition overhead which must only be accounted for mixed frame sets. Therefore, the priority order determined by existing priority-assignment algorithms may incur many mode transitions.

3.9 Conclusion

Utilizing the silent mode of the existing CAN controller is a simple solution to solve the CAN and CAN-FD coexistence problem. However, it is non-trivial to minimize the negative impact of mode-transition overhead of the silent mode. To minimize the negative impact, we have proposed a new priority-assignment algorithm, called PAMT. PAMT minimizes the negative impact of mode transitions by clustering the frame instances based on their type, and is shown to be the optimal priority assignment for a mixed frame set. Also, our extensive simulation results show that PAMT outperforms existing priority-assignment algorithms in minimizing the negative impact of mode-transition overhead.

Algorithm 2: Cluster

Input : I_{pas}: Priority-assigned frame instance set

$i \leftarrow |I_{pas}| - 1$;
// Check **C1**
if $I_{pas}[i].type = I_{pas}[i-1].type$ **then**
  return false; // Already clustered
end
$isSameTypeExist \leftarrow false$;
$k \leftarrow 0$;
for $j \leftarrow 2$ **to** $|I_{pas}| + 1$ **do**
  if $I_{pas}[i].type = I_{pas}[i-j].type$ **then**
    $isSameTypeExist \leftarrow true$;
    $k \leftarrow j$;
    break;
  end
end
if *isSameTypeExist = false* **then**
  return false;
end
// Check **C3**
if $I_{pas}[i].arrival\_time > I_{pas}[i-k+1].arrival\_time$ **and** $I_{pas}[i].arrival\_time > I_{pas}[i-k].completion\_time$ **then**
  return false;
end
// Check **C2**
for $p \leftarrow k-1$ **to** 0 **do**
  if $I_{pas}[i-p].completion\_time + I_{pas}[i].transmission\_time > I_{pas}[i-p].deadline$ **then**
    return false;
  end
end
$temp \leftarrow I_{pas}[i]$;
for $p \leftarrow 0$ **to** $k-1$ **do**
  $I_{pas}[i-p] \leftarrow I_{pas}[i-p-1]$;
  $I_{pas}[i-p].completion\_time \leftarrow I_{pas}[i-p].completion\_time + temp.transmission\_time$;
end
$I_{pas}[i-k+1] \leftarrow temp$;
$I_{pas}[i-k+1].completion\_time \leftarrow I_{pas}[i-k].completion\_time + temp.transmission\_time$;
return true;

CHAPTER 4

EACAN: Reliable and Resource-Efficient CAN Communications

4.1 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 [92], 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 [70], 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 [56]. The timing verification used in COTS tools [97, 105] relies on the schedulability analysis based on the worst-case response time (WCRT) [82]. In particular, a probabilistic schedulability analysis based on the worst-case transmission error rate (WCTER) [12, 20, 21] 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 worst case requires too conservative a safety margin [73]. 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 [84], has been continuously increasing due to the changes in the external environment (5G networks using millimeter wave [42]) and internal vehicle systems (hybrid electrical vehicles & electrical vehicles [32], on-line electrical vehicles [23]).

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.

4.2 System Model and Assumptions

4.2.1 Overall Architecture

We consider a system composed of a single CAN bus and multiple devices/ECUs which share the CAN bus as shown in Fig. 4.1. 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 which has more computing power (higher performance CPU, larger memory size) like a vehicle domain controller [35]. The slave component (sEACAN) is deployed in all ECUs, except for the monitoring ECUs, as shown in Fig. 4.1. 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

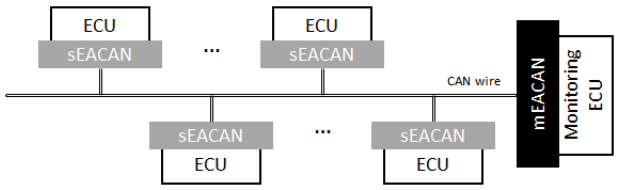


Figure 4.1: Overall system architecture

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.

4.2.2 Error Model

The electrical signal on the CAN bus can be temporarily distorted by EMI [84]. 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 [14]. 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 chapter, 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

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

Table 4.1: Failure-rate requirements due to random hardware faults in ISO26262

detection mechanism. As in previous studies [21, 12], 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 [72].

4.2.3 Mixed-Criticality CAN Message Model

In a vehicle, multiple electronic control units (ECUs) share a physical link (e.g., CAN bus) to communicate with each other. The messages on the shared bus can be used by high- or low-criticality applications. 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 applications have to sacrifice their performance for high-criticality applications in an abnormal situation (e.g., TER exceeds the permitted error rate at runtime) to achieve a fail-safe operation.

According to ISO26262 [46], 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 [46] specifies the requirement of failure rate caused by random hardware faults as shown in Table 4.1. 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 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 [22]. In the model proposed in [22], 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

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

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), $T_i(1) = \ldots = T_i(\chi_i) \leq \ldots \leq T_i(L)$;
- 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 [54]. 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 [99], 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.

4.2.3.1 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 4.1. 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 $\frac{1hr}{T_i(l)}$ times in 1 hour, $\frac{1hr}{T_i(l)} \times p_i(DM|\gamma_{sys} = l)$ timing failures occur on average in 1-hour. To meet the reliability requirement, $\frac{1hr}{T_i(l)} \times p_i(DM|\gamma_{sys} = l) \leq RR(\chi_i)$. Then, we can derive:

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

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

$$\epsilon_i(l) = RR(\chi_i) \times T_i(l) \tag{4.1}$$

Definition 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.*

4.3 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 which fully exploits the characteristics of EACAN.

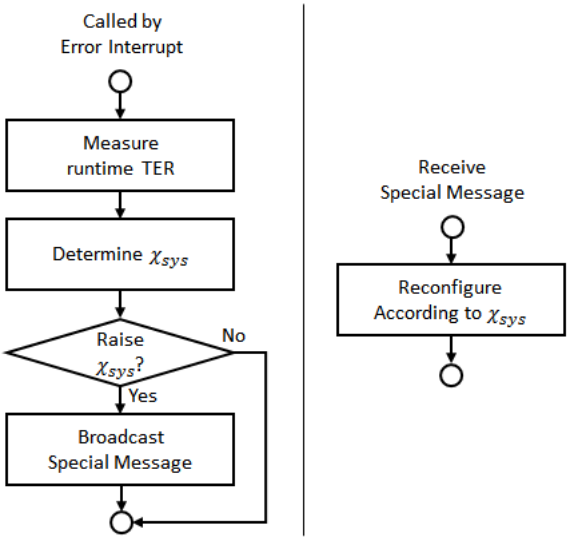


Figure 4.2: Flow chart of (Left) mEACAN (Right) sEACAN

4.4 Error-Adaptive CAN (EACAN)

4.4.1 Overview

4.4.1.1 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, 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.

4.4.1.2 Workflow of EACAN

The workflow of EACAN is illustrated in Fig. 4.2. Whenever a transmission error occurs, an interrupt is generated to handle it. The interrupt-handling routine 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, mEACAN broadcasts a special CAN message to notify the change of system criticality level to other ECUs (sEACANs). Upon receiving this special CAN mes-

sage, 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).

4.4.2 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 [64] generates an interrupt to handle each transmission error.

4.4.2.1 Requirement of Runtime TER

As can be seen from Eq. (2.10), 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.

4.4.2.2 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 in order to meet the runtime TER requirement.

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 ($\xi_{[t_s,t_c]}$) 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.

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

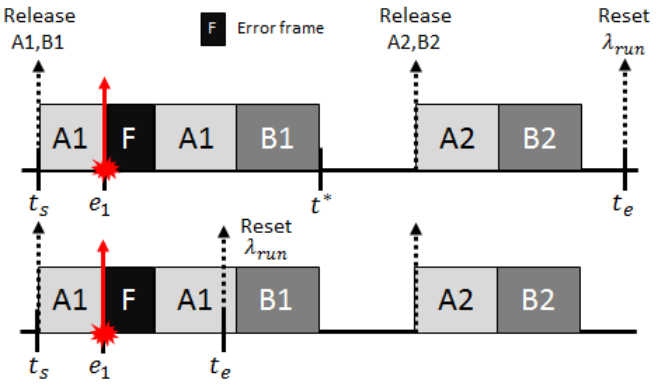


Figure 4.3: (Top) Time interval of interest $[t_s, t_e)$ is unnecessarily long. (Bottom) Time interval of interest is $[t_s,\ t_e)$ is too short

$$\lambda_{run} = \frac{1}{\xi_{[t_s,t_c]}} \tag{4.2}$$

$$\xi_{[t_s,t_c]} = \min_i \left(e_i - e_{i-1}\right) \tag{4.3}$$

where e_i is the arrival time of the i-th 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 (**G**1). 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.

4.4.2.3 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 Fig. 4.3 (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 re-

sponse 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.

On the other hand, if $[t_s, t_e)$ is too short, EACAN may fail to achieve **G**1. For example, as shown in Fig. 4.3 (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 Fig. 4.3 (Up), 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 pseudo code for computing runtime TER is provided in Algorithm 3. mEACAN executes Algorithm 3 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

Algorithm 3: 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 ξ_[ts,tc] = 0 OR ξ_[ts,tc] > e_cur − e_prev then
2 |   ξ_[ts,tc] = e_cur − e_prev;
3 |   λ_run = 1/ξ_[ts,tc];
4 end
5 if λ_run > λ_max then
6 |   λ_run = λ_max;
7 end
8 return λ_run;
```

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.

4.4.3 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 in order to determine the system criticality level quickly at runtime.

4.4.3.1 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, 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, 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.

4.4.3.2 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 Eqs. (2.10), (2.11), and (2.12), 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:

$$\begin{aligned} \theta_i^l = \quad & \underset{\theta}{\text{argmax}} \quad p_i(DM|\gamma_{sys} = l; \theta) \\ & \text{subject to} \quad p_i(DM|\gamma_{sys} = l; \theta) \leq \epsilon_i(l). \end{aligned} \tag{4.4}$$

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.

4.4.4 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 which yields a near-optimal solution.

4.4.4.1 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 Eq. (2.7) because only the period and the deadline depend on the system criticality level:

$$w_{i|Z}^{n+1}(q,l) = B_i + qC_i + E_{i|Z} + \sum_{\forall k \in hp(i)} \left\lceil \frac{w_{i|Z}^n(q,l) + J_k + \tau}{T_k(l)} \right\rceil C_k + O_{chg}, \tag{4.5}$$

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:

$$R_{i|Z}^l(q) = J_i + w_{i|Z}(q,l) - qT_i(l) + C_i \tag{4.6}$$

$$R_{i|Z}^l = \max_q R_{i|Z}^l(q) \tag{4.7}$$

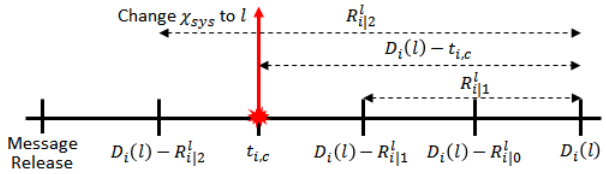


Figure 4.4: The probability of missing a message's deadline depends on the remaining execution/transmission time after changing the system criticality level

Step 2: Compute $p_i(R^l_{i|Z})$, the probability of $R^l_{i|Z}$. We derive $p_i(R^l_{i|Z})$ from Eq. (2.10) by replacing $R_{i|Z}$ with $R^l_{i|Z}$.

$$p(R^l_{i|Z}) = p(Z, R^l_{i|Z}) - \sum_{j=0}^{Z-1} p(R^l_{i|j}) p(Z-j, R^l_{i|Z} - R^l_{i|j}) \tag{4.8}$$

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 $R^l_{i|1} \leq D_i(l) - t_{i,c} \leq R^l_{i|2}$, as illustrated in Fig. 4.4 (according to [21], the response time with 2 errors is always larger than that with 1 error), then 2 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 $R^l_{i|2} \leq D_i(l) - t_{i,c} \leq R^l_{i|3}$, 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:

$$p_i(DM|\gamma_{sys} = l) = 1 - \sum_{Z=0}^{Z_m} p(R^l_{i|Z}) \tag{4.9}$$

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

4.4.4.2 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 ($\theta_i^{l,LB}$) and an upper bound ($\theta_i^{l,LB}$) such that

$$p_i(DM|\gamma_{sys} = l, \lambda = \theta_i^{l,LB}) \leq \epsilon_i(l)$$

$$p_i(DM|\gamma_{sys} = l, \lambda = \theta_i^{l,UB}) > \epsilon_i(l).$$

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 θ^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_i^l also depend on the remaining time. Thus, we need to find all $\theta_{i|k}^l$ thresholds for message i, criticality l and the remaining time $D_i(l) - t_{i,c}$ such that $R_{i|k}^l \leq D_i(l) - t_{i,c} \leq R_{i|k+1}^l$. The pre-defined thresholds $\theta_{i|k}^l$ are computed offline and saved in a 3-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}).

4.4.5 Runtime Decision on System Criticality Level

Algorithm 4 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 4 then tries to find the lowest possible system criticality level by comparing the runtime TER with the pre-defined thresholds. As stated in lines 17 and 18, if the runtime TER is larger than $\theta_{i|k}^l$ and smaller than $\theta_{i|k}^{l+1}$, 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 $base_i^l$ which is needed to compute $t_{i,c}$. $base_i^l$ is the first release time of m_i after changing the system criticality level to l. Thus, the value of $base_i^l$ 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, mEACAN broadcasts a special CAN message to change the system criticality level. Then, sEACANs reconfigure their message set according to the determined system critical-

Algorithm 4: 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

$\chi_{sys}^{[t_c,t_e]} \leftarrow \gamma_{sys}$;
MsgFlag [] $\leftarrow$ false;
for $i \leftarrow 1$ **to** N **do**
  if $\chi_i < \gamma_{sys}$ **then**
    Continue;
  end
  if $T_i(\gamma_{sys}) < (t_c - base_i^{\gamma_{sys}})$ **then**
    $t_{i,c} \leftarrow 0$;
  end
  else
    $t_{i,c} \leftarrow (t_c - base_i^{\gamma_{sys}}) \bmod T_i(\gamma_{sys})$;
  end
  for $l \leftarrow \chi_{sys}^{[t_c,t_e]}$ **to** L **do**
    for $k \leftarrow 1$ **to** K_{max} **do**
      if MsgFlag $[i]$ = *false* **then**
        if $R_{i|k}^l \leq D_i(l) - t_{i,c} \leq R_{i|k+1}^l$ *and* $\theta_{i|k}^l > \lambda_{run}$ *and* $\theta_{i|k}^{l+1} \leq \lambda_{run}$ **then**
          $\chi_{sys}^{[t_c,t_e]} \leftarrow l + 1$; MsgFlag $[i] \leftarrow$ true;
        end
      end
    end
  end
end
if $\gamma_{sys} < \chi_{sys}^{[t_c,t_e]}$ **then**
  for $i \leftarrow 1$ **to** N **do**
    $base_i^{\chi_{sys}^{[t_c,t_e]}} \leftarrow t_c + (T_i(\chi_{sys}^{[t_c,t_e]}) - t_{i,c})$
  end
end
return $\chi_{sys}^{[t_c,t_e]}$

ity level.

4.4.6 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 Eq. (2.7) should be separated out, and also the overhead of changing the criticality level should be accounted for as:

$$w^{n+1}_{i|Z,[1.L]}(q) = B_i + qC_i + E_{i|Z} + O_{chg} + I_{BC}(t_{chg}) + I_{AC}(w^n_{i|Z,[1.L]}(q) - t_{chg}) \quad (4.10)$$

where $w^{n+1}_{i|Z,[1.L]}(q)$ is the busy period of the q-th 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:

$$I_{BC}(t_{chg}) = \sum_{\forall k \in hp(i)} \left\lceil \frac{t_{chg} + J_k + \tau}{T_k(1)} \right\rceil C_k \quad (4.11)$$

$$I_{AC}(w^n_{i|Z,[1.L]}(q) - t_{chg}) = \sum_{\forall k \in hp(i)} \left(\left\lceil \frac{w^n_{i|Z,[1.L]}(q) - t_{chg} + J_k + \tau}{T_k(L)} \right\rceil - 1 \right) C_k. \quad (4.12)$$

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 Eq. (4.11), and also counted 2 times in Eq. (4.12). 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 Eq. (4.11), and also counted 2 times in Eq. (4.12). 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 4. *$\lambda_{run} \geq \frac{Z}{w_{i|Z}(q,1)}$ 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 q-th 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 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.

$$Z_c = \underset{Z}{\operatorname{argmin}} \left[\frac{Z}{w_{i|Z}(q,1)} \geq \lambda_{max} \right] \tag{4.13}$$

Since Eq. (4.5) 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 Eq. (4.10). With the result of Eq. (4.10), we can compute the worst-case response time of the highest-criticality

messages when its transmission is completed at the highest level L as:

$$R_{i|Z,[1,L]}(q) = J_i + w_{i|Z,[1,L]}(q) - qT_i(L) + C_i \tag{4.14}$$

$$R_{i|Z,[1,L]} = \max_q \quad R_{i|Z,[1,L]}(q). \tag{4.15}$$

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 q-th 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:

$$p(R_{i|Z,[1,L]}) = p(Z, R_{i|Z,[1,L]}) - \sum_{j=0}^{Z-1} p(R_{i|j,[1,L]}) p(Z-j, R_{i|Z,[1,L]} - R_{i|j,[1,L]}) \tag{4.16}$$

$$p_i(DM) = 1 - \sum_{\forall Z | R_{i|Z,[1,L]} \leq D_i} p(R_{i|Z,[1,L]}) \tag{4.17}$$

4.4.7 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 Fig. 4.5 (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 Fig. 4.5 (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 Eq. (4.6).

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 3 and Algorithm 4 as they represent the core functions of determining the system criticality level. Algorithm 4 contains only several comparisons and statements, and its time complexity is $O(1)$. In Algorithm 4, Lines 6–13 are repeated at most $N * L * K_{max}$ times, and hence its

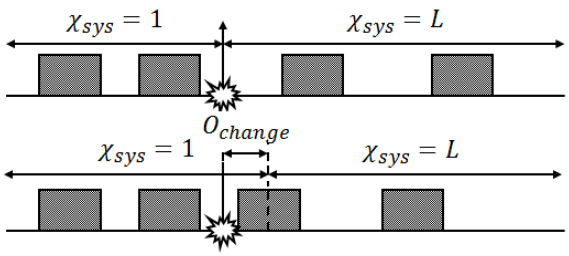


Figure 4.5: (Top) Ideal change of system criticality level. (Bottom) Changing system criticality level with overhead considered.

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 4 or 5 levels. Also, Ferreira *et al.* [33] reported that less than 1000 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, 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 4 or 5 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).

4.5 Evaluation

We have evaluated EACAN in comparison with existing WCTER-based approaches [12, 21]. 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

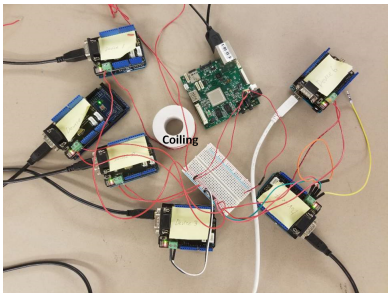


Figure 4.6: Experimental platform

utilization of a given message set.

4.5.1 Experimentation

4.5.1.1 Experimental Platform

We have built an experimental platform to evaluate EACAN as shown in Fig. 4.6. The experimental platform consists of one imx6 sabrelite[3] and six Arduino [5] boards with MCP2515 CAN controller [64]. 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 Fig. 4.6, we can control the strength of EMI. Thus, we can control the transmission error rate by increasing and decreasing the number of coil turns.

4.5.1.2 Benchmark and Configuration

We slightly modified the SAE benchmark [45] as shown in Table 4.2, 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 $\lambda_{max} =$

[3]Imx6 sabrelite board has 1GHz CPU and 1GB Memory size. On the board, LinuxRK[75] is used as operating system for the board.

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	1000	500	1000
16	1 (ASIL A)	1	0.3	500	1000	500	1000
17	1 (ASIL A)	1	0.3	500	1000	500	1000

Table 4.2: Modified SAE benchmark

Method	Schedulability Test	Utilization	L1 Time	L2 Time	Deadline Miss
EACAN	Pass	57.94%	3599822751μs	177249μs	0
WCTER-Based ($\gamma_{sys} = 2$)	Pass	44.01%	0μs	360000000μs	0
WCTER-Based ($\gamma_{sys} = 1$)	Fail	57.95%	360000000μs	0μs	0

Table 4.3: Experimental Results ($\lambda_{max} = 10^{-3}/ms$)

$10^{-3}/ms$, since the TER measured in an aggressive environment (near a welding machine) is about $10^{-4}/ms$ [33]. The TER for 1-hour experiments was observed to be about. $8.0 \times 10^{-4}/ms \pm 1.0 \times 10^{-4}/ms$. 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.

Method	Schedulability Test	Utilization	L1 Time	L2 Time	Deadline Miss
EACAN	Pass	57.89%	3582859215μs	17140785μs	0
WCTER-Based ($\gamma_{sys} = 2$)	Pass	44.01%	0μs	360000000μs	0
WCTER-Based ($\gamma_{sys} = 1$)	Fail	57.95%	360000000μs	0μs	0

Table 4.4: Experimental results ($\lambda_{max} = 10^{-2}/ms$)

4.5.1.3 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 the 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 1288 bytes.

By applying the measured overhead to Eq. (4.10), we performed the EACAN schedulability test for the given modified SAE benchmark. The results are shown in Table 4.3 and Table 4.4. 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 [12, 21] 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 an hour. The utilization of CAN bandwidth is calculated by active (transmission) time 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

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

set fails WCTER-based schedulability tests at level 1.

Tables 4.3 and 4.4 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.

4.5.2 Simulation

4.5.2.1 Benchmark and Configuration.

We generated 1800 mixed frame sets by using NETCARBENCH (powertrain configuration[5]) [19] 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

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			10%-D, 70%-C 10%-B, 10%-A
50ms		25%-D, 50%-C 25%-A	
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 4.5: Criticality assignment based on the message period

[5]In the configuration, the distribution of payload size, message period, jitter and etc. are defined.

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	$T_i(1) = T_i(2)$	$T_i(1) = T_i(2)$ $T_i(2) = T_i(3)$	$T_i(1) = T_i(2)$ $T_i(2) = T_i(3)$ $T_i(3) = T_i(4)$

Table 4.6: Period and deadline changes according to the system criticality level

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 4.5.

The change of period according to the system criticality level is described in Table 4.6. 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 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 (RPA) [27], which is proven to maximize the number of tolerable transmission errors, to the generated frame set to assign priority to each message.

		$\lambda_{max} = 10^{-3}/ms$			
		Total	50%-60%	60%-70%	70%-80%
EACAN	Pass	703	288	246	169
	Fail	197	12	54	131
	Coverage	78.1%	96.0%	82.0%	56.3%
WCTER Based	Pass	638	287	235	118
	Fail	262	13	65	182
	Coverage	70.8%	95.6%	78.3%	39.3%
Coverage Difference		7.2%	0.3%	3.6%	17%

Table 4.7: Coverage of EACAN and WCTER-based schedulability test ($\lambda_{max} = 10^{-3}/ms$)

		$\lambda_{max} = 10^{-2}/ms$			
		Total	50%-60%	60%-70%	70%-80%
EACAN	Pass	374	211	116	47
	Fail	526	89	184	253
	Coverage	41.5%	70.3%	38.6%	15.6%
WCTER Based	Pass	313	198	95	20
	Fail	587	102	205	280
	Coverage	34.7%	66.0%	31.6%	6.6%
Coverage Difference		6.7%	4.3%	7%	9%

Table 4.8: Coverage of EACAN and WCTER-based schedulability test ($\lambda_{max} = 10^{-2}/ms$)

4.5.2.2 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 4.7 and Table 4.8. If a given message set passes the EACAN schedulabilty test, 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 1800 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

[6]Advanced Driver-Assistance System — adaptive cruise control, collision detection,d etc.

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

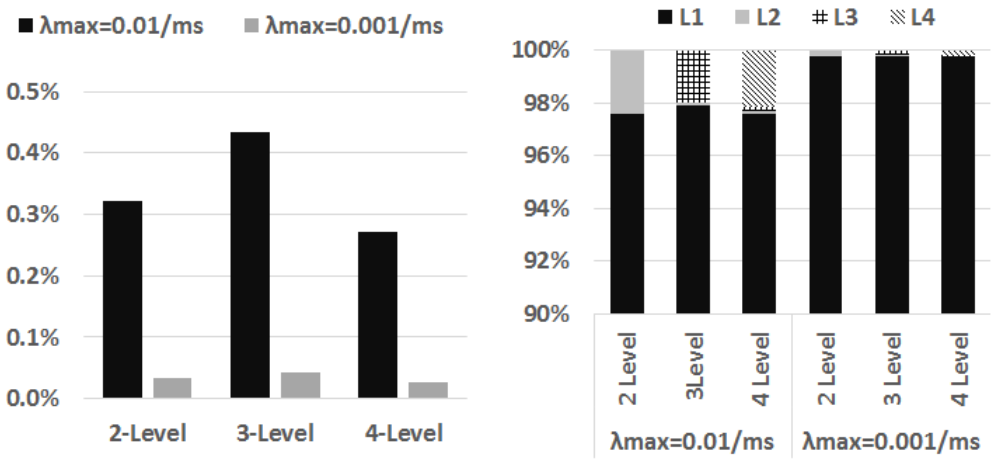


Figure 4.7: (Left) Bandwidth utilization gap between EACAN operation and L1-Only operation. (Right) System criticality-level distribution for a 1-hour CAN operation.

70–80%.

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 3600 transmission errors, following a Poisson distribution for the 1-hour operation. Fig. 4.7 (Right) shows the system criticality-level distribution over the 1-hour 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 Fig. 4.7 (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 that we generate the errors using poisson distribution in the simulations, and thus burst error arrival which could lead deadline miss was 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.

4.6 Conclusion

In this chapter, 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 [57].

CHAPTER 5

DOFP: Design Optimization of Frame Preemption in Real-Time Switched Ethernet

5.1 Introduction

Switched Ethernet is increasingly used in various real-time embedded and cyber-physical systems due to the increasing bandwidth requirements in industrial automation, avionics, and automotive electronics. The IEEE 802.1 Time-Sensitive Networking (TSN) Task Group[1] developed a set of standards to enhance the real-time and dependability properties of switched Ethernet. Examples of these standards include credit-based shaping (IEEE Std 802.1Qav-2009), time synchronization (802.1AS-2011), time-triggered communication (802.1Qbv-2015), and frame preemption (802.1Qbu-2016 and 802.3br-2016). a

The successful deployment of real-time applications in Ethernet networks with these new technologies relies on design-time synthesis and optimization to determine network configurations. Several researchers have recently developed methods and techniques for the optimization of real-time Ethernet app design. A large body of related work deals with the synthesis of time-triggered Ethernet schedules for hard real-time applications [17, 95, 43]. This has recently been extended to synthesize gate control lists (IEEE Std 802.1Qbv for time-driven scheduling) [76], as well as towards robustness to link failures [11] and mixed-criticality applications [96, 98]. Researchers have also addressed routing synthesis [58, 63] and traffic class assignment [37].

While most past research focused on synthesis of communication schedules and priorities, frame preemption on Ethernet has received very little attention. Thiele and Ernst [100] proposed a worst-case timing analysis framework for networks employing frame preemption. To the best of our knowledge, there has not been any design method for real-time, preemptable Ethernet.

[1] formerly known as the Audio/Video Bridging (AVB) Task Group

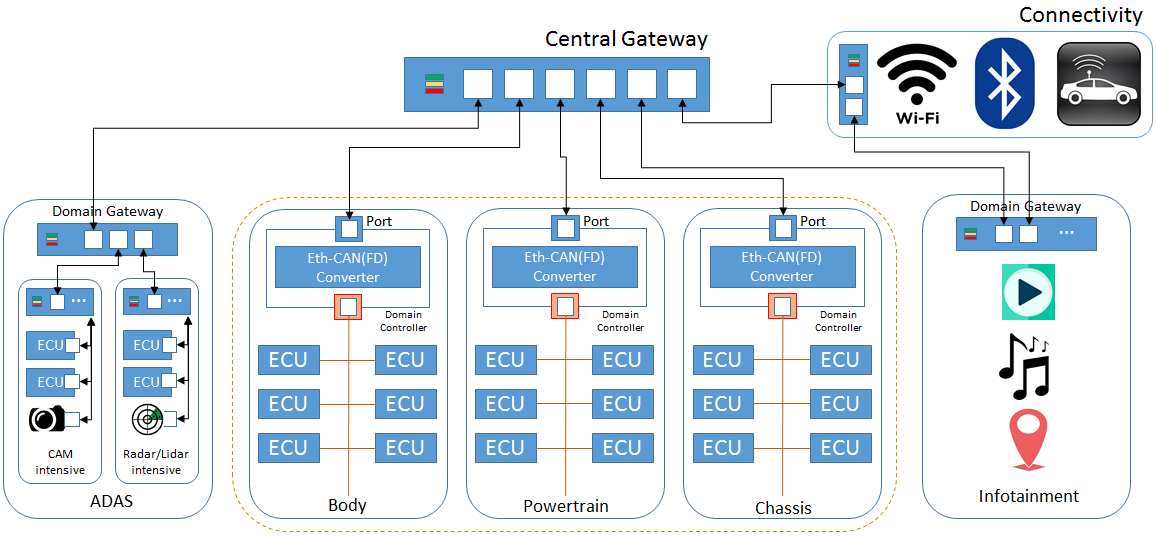


Figure 5.1: A promising in-vehicle architecture

Our contributions: We address the synthesis problem for frame preemption on Ethernet according to the recently developed 802.1Qbu-2016 and 802.3br-2016 standards. Specifically, we study the problem of assigning priorities, and hence queue allocation, of each real-time data flow, as well as, for each queue, assigning whether the flow is transmitted in the preemptable MAC (pMAC) or express MAC (eMAC) interface of the corresponding egress port. We present a genetic algorithm-based optimization framework that exploits a worst-case preemption-aware timing analysis, and use it to improve reliability and extensibility of networks as case studies. For the effective and efficient use of the framework, we propose an initialization algorithm for each goal. We conduct experimental evaluations on networks of different sizes and complexities, including a real-life automotive system.

5.2 System Description and Model

5.2.1 System Architecture

A promising in-vehicle architecture is provided in Fig. 5.1. Electronic Control Units (ECUs) are grouped by their functions, forming a *functional domain*. Each domain may use a different network protocol to optimize the implementation cost while satisfying its requirements. For example, body, powertrain and chassis domains use controller area networks (CANs) or CAN flexible data-rate (CAN-FD), while ADAS, infotainment and connectivity domains use switched-Ethernet as their internal network. Moreover, the switched-Ethernet may be used as the backbone-network. A domain whose internal network is

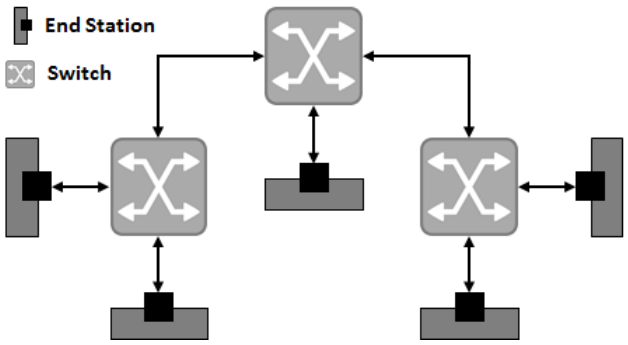


Figure 5.2: Abstracted system architecture

CAN(FD) must have an Ethernet-CAN(FD) converter, and the domain controller of the domain would assume the role of the converter internally. Domain controllers or gateways are physically connected to a central gateway to make connection between domains, and the central gateway receives and forwards inter-domain messages.

5.2.2 Networked System Model

The architecture shown in Fig. 5.1 can be abstracted as a tree-topology switched-Ethernet system as shown in Fig. 5.2. End-stations represent either domain controllers or ECUs equipped with a Ethernet port, and switches represent gateways. An end-station has only one physical Ethernet port, and the port is physically linked to a corresponding port in a switch. A switch has one or multiple ports, and each port is physically linked to a corresponding port of either an end-station or a switch. The physical links are full-duplex.

5.2.2.1 Frame Preemption Supportive Ethernet Port

We assume that every Ethernet port supports frame preemption based on IEEE 802.1Qbu and IEEE 802.3br. Each egress port has at most 8 queues to serve different classes of frame, and the queues are mapped to either express MAC (eMAC) interface or preemptable MAC (pMAC) interface as shown in Fig. 5.3. We call a queue mapped to eMAC interface an *express queue*, and call that mapped to pMAC interface *preemptable queue*.

Even though higher-priority (lower-priority) queues can be mapped to pMAC (eMAC) interface, we assume that every express frames have higher priority than preemptable frames because a lower priority frame's preemption of higher priority frames does not make sense. Based on this assumption, an egress port i can be defined as $ep_i = \{Q_i, V_i, r_{TX,i}, \zeta_i\}$ where

- Q_i: the set of queues in the egress port i. $Q_i = \{q_{i,0}, q_{i,1}, ..., q_{i,n}\}, n \leq 7$. $q_{i,0}$ is the

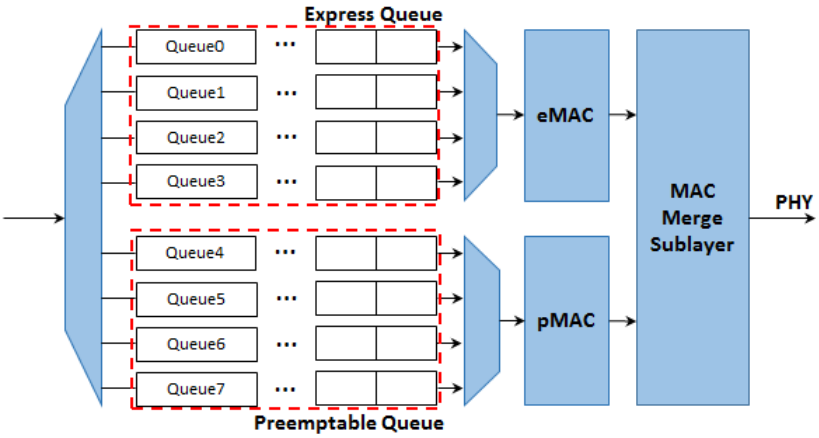


Figure 5.3: Frame preemption supportive port

highest-priority queue and $q_{i,n}$ is the lowest- priority queue in order;

- V_i: the corresponding end-station or switch with egress port i;
- $r_{TX,i}$: physical link speed; and
- ζ_i: boundary between express and preemptable queues.

For example, if ζ_i is 3, then $q_{i,0}, \ldots, q_{i,3}$ are express queues and $q_{i,4}, \ldots, q_{i,n}$ are preemptable queues.

5.2.2.2 Traffic Flow

An in-vehicle traffic flow is initiated periodically and has fixed route because of deterministic operation requirement. We group in-vehicle traffic flows into four classes; control signal, sensor data (vehicle dynamics), raw data from camera or radar (lidar) and the others as shown in Table 5.1. The control signal, sensor data, raw data from camera or radar (lidar) are usually used by time-critical applications, and thus these data flows have deadlines. On the other hand, non-time-critical traffic flows do not have deadlines. A traffic flow is defined as $f_i = \{T_i, D_i, L_i, P_i, J_i, \chi_i, \rho_i\}$ where

- T_i: period of f_i;
- D_i: relative deadline of f_i;
- L_i: maximum payload size of f_i;
- P_i: the value of priority code point (PCP) of f_i;

	Critical Frame			Non-Critical Frame
	Class 1	Class 2	Class 3	Class 4
Type	Control	Sensor (Vehicle Dynamics)	CAM, Radar	Others
Period	10ms-40ms	40ms-100ms	20ms-100ms	50ms-1s
Deadline	same as period			No deadline
Payload	10-100byte	100-200byte	400-1500byte	1500byte

Table 5.1: Ethernet traffic flow classes

- J_i: release jitter of f_i;
- χ_i: class of f_i;
- ρ_i: route of f_i.

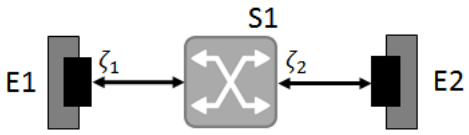
As shown in Fig. 2.4, the PCP value is specified in the header of frames of a traffic flow. The frames are forwarded to one of eight queues based on their PCP value. For example, suppose $PCP = k$ is mapped to $q_{i,k}$. If the PCP value of a frame is 0, then the frame is queued into $q_{i,0}$. Because queues in egress ports have priority, the priority of frames are intrinsically determines by their PCP values. We assume that, for any port i, $PCP = k$ is mapped to $q_{i,k}$.

5.3 Synthesis Problem

To utilize the standardized frame preemption, we must synthesize not only the assignment of frames of traffic flows to queues but also the assignment of queues to MAC interface. That is, we need to determine not only a set of P_i for given traffic flows but also a set of ζ_i for given egress ports because we assume that express queues have higher priority than preemptable queues. However, to the best of our knowledge, there is no solution which deals with the two assignments at once — we are the first to address these problems at once for the standardized frame preemption.

The example shown in Fig. 5.4 stresses the importance of ζ_i decision. As explained before, the placement of type boundary at an egress port determines the type of frames of traffic flows. For example, the frames of f_1 are express and the frames of f_2 and f_3 are preemptable at the egress port ep_1 when $\zeta_1 = 2$. Hence, placement of the type boundary affects the maximum queuing delay of the frames at the egress port.

Suppose the maximum transmission time of frames of f_1, f_2 and f_3 is 5. That is, the maximum waiting time for the entire transmission of a frame is 5 at an egress port. Also, suppose that the maximum waiting time is 1 when an express frame waits for the



Traffic Flow	P_i	D_i	L_i	ρ_i
f_1	2	20	100B	E1->S1->E2
f_2	4	25	100B	E1->S1->E2
f_3	6	30	100B	E1->S1->E2

Figure 5.4: Example of showing importance of ζ_i decision.

ζ_1	ζ_2	R_1^+	R_2^+	R_3^+
2	2	12	30	30
2	4	16	26	30
2	6	16	30	30
4	4	20	22	30
4	6	20	26	30
6	6	20	30	30

Table 5.2: Worst-case E2E latency with different type boundary configurations

transmission of a fragment of a preemptable frame. Then, with the configuration of $\zeta_1 = 2$ and $\zeta_2 = 4$, the maximum queuing delay of frames of f_1 at ep_1 is 1 because the frames can preempt any frames of f_2 and f_3. Also, that at ep_2 is 5 because a frame of f_1 needs to wait for the entire transmission of a frame of f_2 in the worst case. Thus, the worst-case end-to-end (E2E) latency of frames of f_1 is $1+5+5+5 = 16$. The worst-case E2E latency of given traffic flows (R_i^+) with different boundary configurations are shown in Table 5.2. The given traffic flow is schedulable ($R_i^+ \leq D_i$) only with configuration of $\zeta_1 = 4$ and $\zeta_2 = 4$.

As shown in the above example, synthesis of PCP and placement of type boundaries directly affect the scheduling sequence of competing frames at egress ports. In other words, the synthesis affects the worst-case end-to-end (E2E) latency of the frames, and the competing frames have a seesaw relationship with respect to the worst-case E2E latency.[2] That is, we can configure the worst-case E2E latency of the frames by controlling the assignments to achieve a given optimization goal (G).

However, finding the optimal assignments for a given goal is non-trivial because there are millions of ways of making assignments, so it is computationally infeasible to explore

[2]The worst-case E2E latency is analyzed using an existing worst-case E2E latency analysis[100]. See the Appendix for detail.

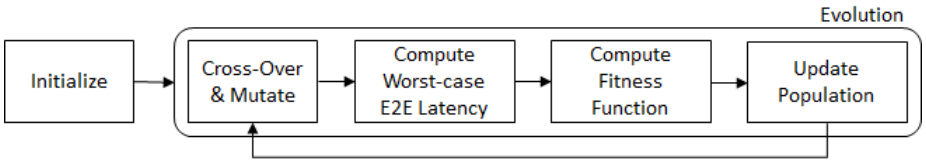


Figure 5.5: The overview of GA-based framework

all possible combinations even for a small number of traffic flows. Suppose, for example, there are n traffic flows and m egress ports. Then, there are at least 8^{n+m} different combinations because there are 8 possible priority levels for each traffic flow and 8 queues in an egress port.

5.4 Generic Solution Approach

As mentioned in the previous section, it is computationally infeasible to find the optimal assignments for a given goal G. Thus, we propose a heuristic framework based on the well-known genetic algorithm (GA).

Definition 2 (Individual)**.** *Each individual has two genes PS and TBS, where PS is a set of P_i for given traffic flows and TBS is a set of ζ_i for given egress ports. An individual is thus defined as $s_i = (PS, TBS)$.*

Definition 3 (Population)**.** *The k^{th} population ϕ_k is a set of individuals. The individuals in a population are always sorted by the fitness function. $\phi_{k,i}$ denotes the individual that has the i^{th} best result of fitness function in the in the k^{th} population ϕ_k*

5.4.1 Overview of GA-Based Framework

Fig. 5.5 provides an overview of our GA-based framework. The initialization step sets up an initial population (ϕ_0), with which the algorithm starts and continues evolution until the number of evolutions exceeds the pre-defined maximum. Since we want to optimize the set of P_i's and the set of ζ_i's, PS and TBS are the evolution parameters. During the evolution step, the algorithm generates a new individual repeatedly by inheriting the evolution parameters from two randomly chosen individuals in the current population (cross-over) and mutating the inherited parameters.

After the mutation, the framework computes the worst-case E2E latency for the given traffic flows using the state-of-the-art worst-case E2E analysis technique. With the computed worst-case E2E latency, the framework obtains the result of fitness function for the

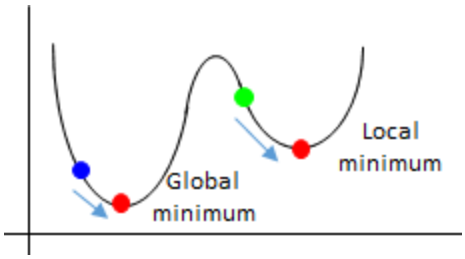


Figure 5.6: Initial parents significantly affects the outcome of the genetic algorithm

new individual. The given optimization goal G is often used as the fitness function. The result for the new individual is compared to that for individuals in the current population. If the new individual has a better result than any individuals in the current population (ϕ_c), the current population is updated.

5.4.2 Initialization

The individuals in the initial population (ϕ_0) are used as initial parents. From the initial parents, the GA-based framework gradually approaches a convergence point through the defined evolution procedure. Because the convergence point can be either the global optimum or a local optimum point, the initial parents significantly affect the final result from the GA-based framework. As shown in Fig. 5.6, the final result with a well-chosen initial parent (blue dot) can outperform the final result with a poor-chosen initial parent (green dot). Due to the importance of initial parents, we need an initialization algorithm to generate an initial population for a given optimization goal.

5.4.3 Evolution Procedure

The evolution procedure consists of selection, cross-over, mutation and population update as illustrated in Fig. 5.7. The selection step randomly selects two individuals from the current population (ϕ_c) as parents. In the cross-over step, a new individual is generated, and the individual inherits the evolution parameters from the selected individuals. The new individual inherits PS from the first-selected parent and TBS from the second-selected parent.

In the mutate step, a traffic flow or an egress port is randomly selected to be mutated. When a traffic flow is chosen, the traffic flow's PCP value is replaced with a random value. When an egress port is chosen, the placement of type boundary of the port is replaced with a random value. Through the cross-over and mutate steps, the evolution parameters are shaken, and thus the new individual can have a better or worse result than its parents. To

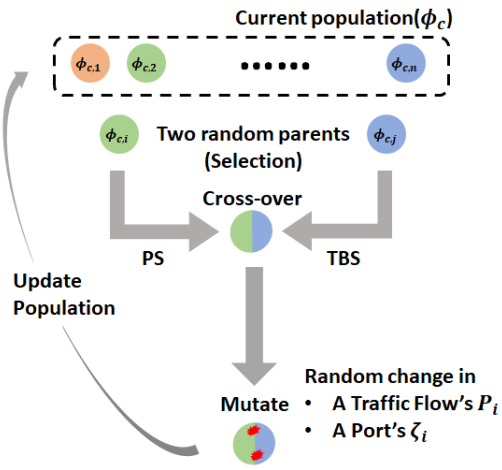


Figure 5.7: Evolution procedure

evaluate the new individual, the worst-case E2E latency and the fitness function are recomputed with the changed PS and TBS. If the new individual has a better result than the lowest-ranked individual ($\phi_{c,n}$) in the current population (ϕ_c), the current population is updated by placing the new individual in the population and removing the lowest-ranked individual from the population.

5.5 Case Study 1: Reliability

Reliable communication is one of the most important requirements for vehicle applications. Unexpected functional failures could lead to disastrous results, such as damage/loss of properties or lives. However, the standardized error-handling mechanism, *Seamless Redundancy* (IEEE 802.1CB), is expensive to implement because it requires at least two separate physical routes. Moreover, it requires complex status management in switches, so only specified frames in switches can be corrected. Thus, a general and cheap error-handling mechanism is required. Recent studies [68, 13] claim that the well-known automatic repeat request (ARQ) protocol [62] is a promising way to improve reliability for real-time switched Ethernet.

5.5.1 ARQ Protocol

We assume that every end-station supports the following ARQ protocol and the ARQ protocol is implemented in Layer2.

When a frame of traffic flow i is initiated by a sender, the frame is first forwarded into an ARQ buffer. Then, ARQ sets an expiration time for the frame. The worst-case

roundtrip time of the traffic flow is used as the expiration time. ARQ then copies the frame and forwards the frame to the egress port without waiting for the completion of transmission of any prior frames. When the receiver receives the frame correctly, it sends an acknowledgement (ACK) frame to the sender. The ACK frame has same priority as its corresponding data frame. If the ACK frame arrives at the sender within the expiration time, the frame is removed from the ARQ buffer. If the ACK frame does not arrive at the sender within the expiration time or the negative acknowledgement (NAK) frame arrives at the sender, ARQ copies and re-transmits the frame again.

ARQ improves reliability via this re-transmission mechanism. We will explain below how the proposed GA-based framework can be applied to maximize the number of "allowable" re-transmissions.

5.5.2 Applying the GA-Based Framework

5.5.2.1 Optimization Goal (Fitness Function)

Critical Scaling Factor (CSF) [83, 81] is a well-known metric that represents the number of times each task can be (re-)executed to recover a system from unexpected failures without missing its deadline. Thus, a higher CSF value means higher reliability. In the context of networking, the metric represents the maximum number of re-transmissions allowed without missing deadlines. Thus, CSF for a traffic flow is defined as:

$$CSF_i = \frac{D_i - R_i^+ - R_{ACK,i}^+}{R_i^+ + R_{ACK,i}^+}, \tag{5.1}$$

where R_i^+ is the worst-case E2E latency of frames of traffic flow i, and $R_{ACK,i}^+$ is the worst-case E2E latency of ACK frames of traffic flow i.

Maximizing the number of allowable re-transmissions for given traffic flows translates to maximizing the minimum CSF among all traffic flows. Thus, it can be formulated as a max-min optimization problem as:

$$\mathbf{G}: Maximize \left\{ \underset{i}{Min} \left\{ CSF_i \right\} \right\} \tag{5.2}$$

5.5.2.2 Generating initial population

As mentioned above, competing frames have a seesaw relationship with respect to the worst-case E2E frame latency. Since CSF depends on the worst-case round trip time ($RTT_i^+ = R_i^+ + R_{ACK,i}^+$), the competing frames also have a seesaw relationship with

respect to CSF. Thus, the above optimization problem can be cast into the problem of balancing CSF for given traffic flows.

We want to generate an initial individual which has a well-balanced CSF among given traffic flows. We make three hypothesis about a set of P_i's which leads well-balanced CSF as:

H1: The lower the class, the higher the PCP;
H2: Traffic flows are well distributed to eight queues;
H3: The smaller the payload size, the higher the PCP.

(**H**1) In general, frames of lower-class traffic flows have relatively short deadlines than those of higher class traffic flows. So, the frames of lower-class traffic flows have relatively less room for re-transmission than higher-class traffic. Thus, assigning higher priorities to lower-class traffic might result in a well-balanced CSF.

(**H**2) If queue length is not balanced, it will increase queuing delay included in the worst-case E2E latency of traffic flows. Suppose, for example, there are two frames; one is a frame of traffic flow A and the other is a frame of traffic flow B. If the frames are forwarded into the same queue, then they interfere with each other in terms of the worst-case E2E latency. On the other hand, if the frames are forwarded to different queues, only lower-priority frames would be interfered with by higher-priority frames. So, the latter case may result in a well-balanced CSF.

(**H**3) Transmission time of a frame with a smaller payload is smaller than that with a larger payload size. Since the interference by a frame with large transmission time is less likely to keep the safety margin large than that by a frame with small transmission time, a frame with smaller payload should have a higher PCP than that with larger payload to enable well-balanced CSF.

Based on the above hypotheses, we generate an initial population. For **H**1, the algorithm assigns separate queue resource to each class. The first ω_1 queues are assigned to class 1 and the next ω_2, ω_3 and ω_4 queues are assigned to class 2, 3 and 4 as illustrated in Fig. 5.8. $\Omega = \{\omega_1, \omega_2, \omega_3, \omega_4\}$ denotes the queue resource isolation. For example, the frames of class 1 traffic flows should be buffered in the first ω_1 queues, and thus the PCP of the frame should be $\{0, \ldots, \omega_1 - 1\}$. There are a total of 60 different Ω's. Because of the manageable number of ways to consider, the algorithm explores the entire search space to find the best Ω.

For **H**2, the maximum payload of each queue is limited in the initialization process. The payload limit is computed for each class, and thus the queues assigned to the same

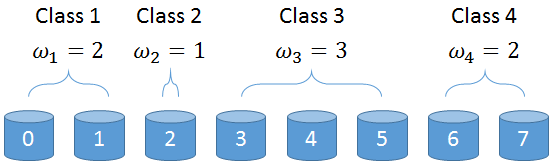


Figure 5.8: Assign the isolated queue resource to each class

class have the same payload limit. The payload limit for each class is computed as:

$$PL_k = \frac{\sum_{f_i \in F_k} L_i}{\omega_k},$$

where F_k is the set of traffic flows whose class is k and ω_k is the number of queues assigned to class k. For example, suppose that queue 0 and 1 are assigned to class 1 and $PL_1 = 100$. Also, suppose that the sum of payloads of traffic flows assigned to queue 0 already reached 100. Then, the remainder of class 1 traffic flows should be assigned to queue 1.

For **H**3, the algorithm sorts the given frames in ascending order by the payload size, and fills up from the highest priority queue to the lowest priority queue with the sorted frames. Thus, a frame with the smaller payload is buffered into the higher priority queue.

5.6 Case Study 2: Extensibility

Vehicle owners will soon be able to update their vehicles, just like smartphones, by downloading newly developed functions. Thus, the extensibility of in-vehicle networks will be a very important non-functional requirement in the near future.

5.6.1 Applying GA-Based Framework

5.6.1.1 Optimization Goal (Fitness Function)

Suppose system designers want to improve the extensibility of a configured in-vehicle network to accept newly introduced traffic flows from firmware updates, application updates, and communication with other entities (e.g., V2X messages). Then, to accept the new traffic flows, existing traffic flows must have enough *time-cushion* (TC):

$$TC_i^- = D_i - R_i^+$$

, where TC_i^- is the smallest time-cushion of frames of traffic flow i.

The system designers, therefore, need to maximize the minimum time-cushion among frames of given traffic flows:

$$G : Maximize \left\{ \underset{i}{Min} \{TC_i^-\} \right\}$$

5.6.1.2 Generating the initial population

To find a good initial population for the given optimization goal, we design an initialization algorithm based on a parametric search. The algorithm tries to find a set of P_i's which makes the minimum TC of given traffic flows larger than a pre-specified value α.

The maximum possible TC (α_{max}) is smaller than the largest deadline of given traffic flows and the minimum possible TC (α_{min}) is larger than zero if all the flows are schedulable. Thus, we initialize three variables as: $\alpha_{max} = max_i\ D_i$, $\alpha_{min} = 0$ and $\alpha = (\alpha_{min} + \alpha_{max})/2$.

Frames of traffic flows which have relatively short deadlines will have small TCs. So, to make the minimum TC for given traffic flows as large as possible, assigning higher priorities to shorter-deadline traffic flows is a good strategy. The algorithm works as follows.

Step 1: Sort the given traffic flows in ascending order by deadline to assign higher priorities to flows with shorter deadlines. Suppose that the set of sorted traffic flows are $F_s = \{f_{s,0}, f_{s,1}, \ldots, f_{s,n}\}$, where $f_{s,0}$ is the traffic flow with the shortest deadline, $f_{s,1}$ is the traffic flow with the second shortest deadline, and so on.

Step 2: Assign the current priority (P_{curr}) to $f_{s,i}$ (priorities of the traffic flows from $f_{s,0}$ to $f_{s,i-1}$ have already been assigned). Note that the current priority starts from the highest priority (0).

Step 3: Analyze the worst-case E2E latency for each priority-assigned traffic flow ($\{f_{s,0}, \ldots, f_{s,i}\}$) and compute TC for the flows.

Step 4: If any of the priority-assigned traffic flows have lower TC than the pre-specified value α, then increase the current priority ($P_{curr} \leftarrow P_{curr} + 1$), re-assign the increased current priority to $f_{s,i}$, and go to Step 3. Otherwise, increment i by 1 and go to Step 1.

At this time, if the current priority becomes larger than maximum possible value ($P_{curr} > 7$), then our algorithm cannot find the set of P_i's which makes the minimum TC of given traffic flows larger than α. Thus, $\alpha_{max} \leftarrow \alpha - 1$ and α are re-computed with the updated α_{max}. Or, if all the given traffic flows are assigned their priorities, our algorithm succeeds in finding the set of P_i's which makes the minimum TC of given traffic flows larger than α. Thus, $\alpha_{min} \leftarrow \alpha + 1$ and α are re-computed with the updated α_{min}.

5.7 Evaluation

5.7.1 Methodology

We have evaluated the performance of our proposed GA-based framework with the two above optimization goals by developing an evaluation tool, *Priority Assignment Evaluation for TSN (PAET)*. The roles of PAET are to (1) generate a parameterized random network and traffic, (2) apply baselines or our proposed GA-based framework to the generated network and traffic, (3) compute fitness function and (4) perform a series of tests. Describe below is how to generate the network and traffic.

5.7.1.1 Baselines

The baselines we used in this evaluation are:

- FPCP: Fixed PCP for each class. (Class1: 0, Class2: 2, Class3: 4, Class4: 7).
- AOPA: Audsley's Optimal Priority Assignment [8].
- RPA: Robust Priority Assignment [25]

Because these baselines only determine a set of P_i for given traffic flows, we have to decide the ζ_i for given egress ports. Thus, we assign fixed value to all egress ports. For example, $\forall i\ \zeta_i = 0$. Because the performance of the baselines depends on the value of ζ_i, we test all possible values (0, 1, ..., 7) and use the best performance with the fixed borderline assignment as the results of baselines. For example, if the performance of FPCP with $\zeta_i = 2$ is better than the performance of FPCP with $\zeta_i = 4$, we use results with $\zeta_i = 2$ as the results of FPCP.

5.7.1.2 Network Generation

PAET receives the number of frames, the number of end stations and the number of switches as input parameters. The ranges of these parameters are shown in Table. 5.3. With the given parameters, the network generator in PAET generates a random tree-topology network according to the following sequences.

- Step 1. Assign a random corresponding switch to each end station.
- Step 2. Select two random switches to make connection between them.
- Step 3. If there already exists a valid route between the randomly chosen switch, go to step 2 to avoid making any cycle.

Number of End Stations	5 - 20
Number of Switches	3 - 7
Number of Frames	100 - 500
Topology	Tree
Portion of Class 1 Frames	5%
Portion of Class 2 Frames	5%
Portion of Class 3 Frames	50%
Portion of Class 4 Frames	40%
Number of Evolution	100000

Table 5.3: Simulation Configuration

- Step 4. Otherwise, Make a connection between them.
- Step 5. Check whether all the switches have at least two links or not. If all the switches have at least two links, the process of network generation terminates. Otherwise, go to step 2.

5.7.1.3 Traffic Generation

PAET receives the total number of traffic flows as an input parameter. With the given total number of traffics, the traffic generator in PAET generates random traffic flows according to the following sequences.

- Step 1: Generate a traffic flow and decide its class with fixed portion of each class shown in Table 5.3.
- Step 2: Decide period, deadline, maximum payload size and release jitter for the traffic flow. The range of these values are shown in 5.1.
- Step 3: Choices two random different end stations. One becomes source of the traffic flow and the other becomes sink of the traffic flow. Because the network is tree-topology, route of the traffic between the source and the sink automatically determined.
- Repeat until the number of generated traffic flows is same as the given total number of traffic flows.

5.7.1.4 Metrics

We measure the schedulability, CSF and TC for the different assignment algorithms as the evaluation metrics. CSF is used to quantify the effectiveness of the proposed GA-based

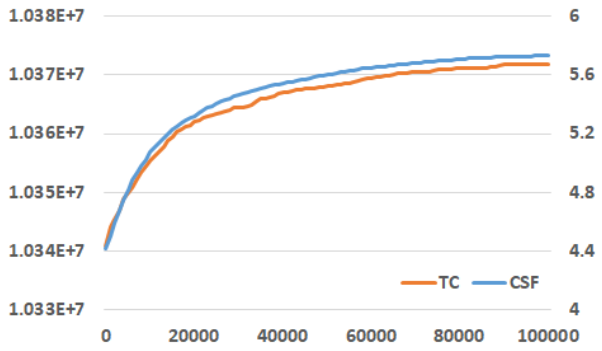


Figure 5.9: Average CSF and TC according to the number of evolutions for the generated 400 test cases. Y-axis on the left is average TC and Y-axis on the right is average CSF. X-axis is the number of evolutions.

framework and initialization algorithm in terms of reliability. TC is used for extensibility.

5.7.2 Evaluation Results & Analysis

We compare our algorithm with the baseline explained above by generating 400 different test cases. The 'Init' is the results after applying the designed initialization algorithm, and the 'Genetic' is the results after 100000 evolutions. Fig. 5.9 shows the average CSF and TC according to the number of evolutions for the 400 test cases. As shown in the figures, the improvement rapidly slowdown near the 100000 evolutions, and thus we can infer that it almost reaches to the convergence point. So, even though we can obtain the better result with more evolutions, the benefit from the additional evolutions would not be significant.

5.7.2.1 Schedulability

We first evaluate the baselines and our algorithms in terms of schedulability. With ARQ, if $RTT_i^+ \leq D_i \forall i$, the given set of traffic flow is schedulable. The results with ARQ are shown in Fig. 5.10 (Left). Without ARQ, if $R_i^+ \leq D_i \forall i$, the given set of traffic flow is schedulable because there is no acknowledgement. The results without ARQ are shown in Fig. 5.10 (Right). Because $RTT_i^+ > R_i^+$, the schedulability without ARQ is higher than that with ARQ.

According to the results, while the every generated test cases are schedulable with our algorithm, some of the test cases are unschedulable with the baselines. These schedulability difference mainly comes from the queue usage. We know that there are eight queues in an egress port. However, FPCP assigns priorities to traffic flows based on their class, and thus only four queues in an egress port are used. In other words, the other four queues are not

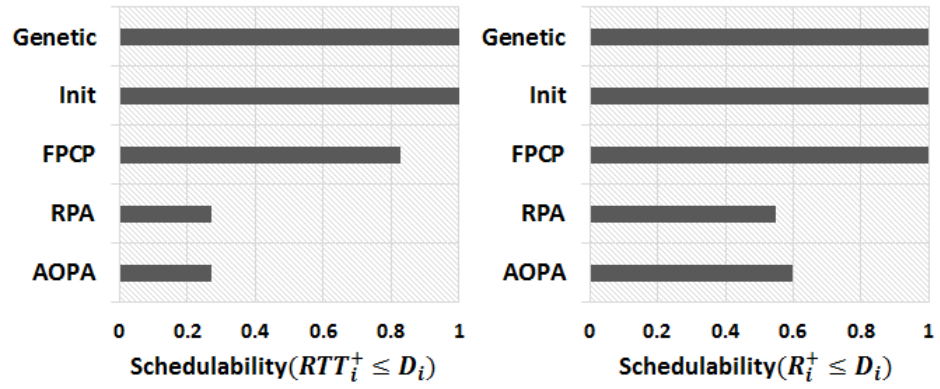


Figure 5.10: (Left) Schedulability with ARQ (Right) Schedulability without ARQ

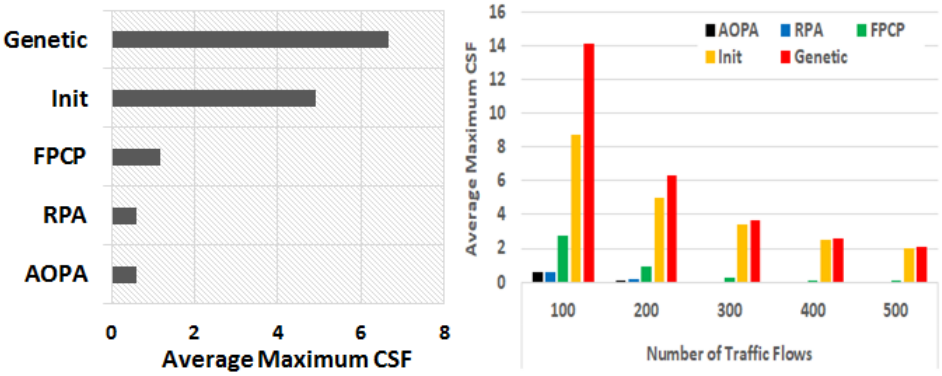


Figure 5.11: (Left) The average of maximum CSF for the generated test cases with different assignment algorithms (Right) The average of maximum CSF for different number of traffic flows

used. Also, AOPA and RPA assign the lowest possible priority to each traffic flow, and thus lower priority queues are more likely used than higher priority queues. As a result, the given resources are not used efficiently, and these algorithms cannot benefit from load balancing effect in the worst-case E2E latency. For example, if there are two traffics and the two traffics are queued into same queues, then they interfere each other in terms of the worst-case E2E latency. On the other hand, if the two traffics are queued into different queues, the traffic queued in the higher priority queue does not interfered by the traffic queued in the lower priority queue.

5.7.2.2 CSF: Reliability

The average maximum CSF for the generated 400 test cases are shown in Fig. 5.11 (Left). Because CSF represents the number of possible re-transmission, we assume that CSF is 0 if a given test case is unschedulable. As shown in Fig. 5.11 (Right), in terms of CSF, FPCP,

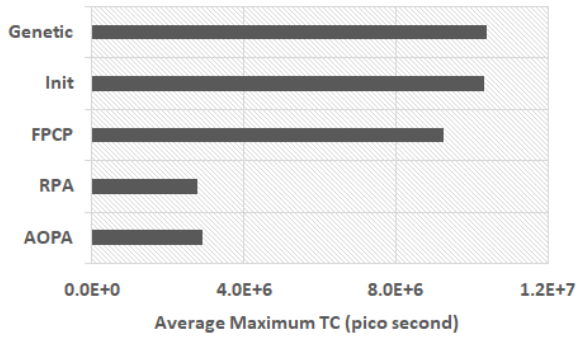


Figure 5.12: The average of maximum TC for the generated test cases with different assignment algorithms.

AOPA and RPA have the low performance because these algorithms do not efficiently utilize the given queue resources in egress ports as we explained in the previous subsection. Therefore, results with Init and Genetic are better than that with FPCP, AOPA, and RPA.

We think that the initialization algorithm for CSF optimization is the advanced version of FPCP. It assigns priorities to given traffic flows based on their class like FPCP. However, it well distributes the given traffic flows to the eight queues. Thanks to the efficient use of given resources, Init shows better results than the baselines. The results of Genetic shows not only the effect of efficient use of given queue resources but also the effect of optimization of P_i and ζ_i. Even though the results of Genetic is not the optimal value, the average maximum CSF with Genetic is about 8.5 times higher than that with AOPA and RPA, about 3.9 times higher than that with FPCP, and 1.32 times higher than that with Init.

As shown in Fig. 5.11 (Right), Genetic shows much better performance with the small number of traffic flows. This is because, with the small number of traffic flows, frame with small deadline can have small worst-case E2E latency and it makes large number of re-transmission available.

5.7.2.3 TC: Extensibility

The average maximum TC for the generated 400 test cases are shown in Fig. 5.12. When we compute the average maximum, we assume that TC is 0 if a given test case is unschedulable. Similar to the average maximum CSF, Init and Genetic outperforms the baselines. In detail, the average maximum TC with Genetic is about 2.74 times longer than that with AOPA and RPA, about 1.11 times longer than that with FPCP. However, Genetic very slightly improves TC from Init. This is because the initialization algorithm (Init) can find a very good set of P_i using parametric search. We believe that the set of P_i found by Init is almost optimum. Thus, Genetic might benefit from the optimization of ζ_i, and it seems

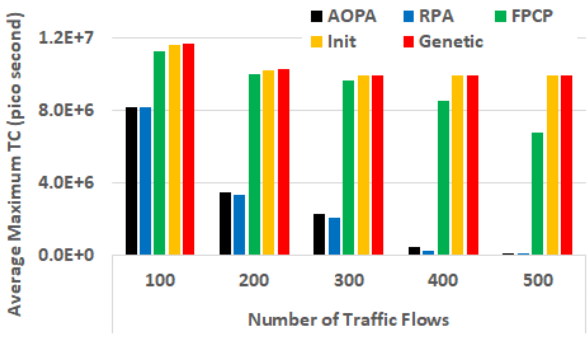


Figure 5.13: The average of maximum TC for different number of traffic flows

that ζ_i optimization does not affect the TC improvement much.

As shown in Fig. 5.13, FPCP and Genetic show almost same performance for the test cases with low utilization. This is because the deadline of frames of traffic flow is large enough for the cases. However, the benefit from using Genetic increases according to the increase of utilization.

5.7.2.4 Scalability

We have measured the average execution time of 1 evolution for different number of traffic flows to evaluate the scalability of our approach. The measured execution time on our machine (Intel Xeon E5-2683 @ 2.10GHz, 128GB Memory) are shown in Table. 5.4. The results show that the execution time of 1 evolution is exponentially increased along the number of traffic flows. For example, the execution time of 1 evolution for 400 traffic flows is about 16 times larger than the execution time of 1 evolution for 100 traffic flows. Because of the exponential increase, this approach is not suitable for large number of traffic flows cases. However, we believe that the performance is acceptable for practical automotive use cases, in particular due to that our method is applied at design time.

Traffic Flows	100	200	300	400	500
Execution Time	0.3s	1.15s	2.86s	5.02s	8.02s

Table 5.4: Execution time of 1 evolution

5.8 Conclusions

In this chapter, We address the synthesis problem for the standardized frame preemption (802.1Qu, 802.3br). Because frame preemption can be utilized various optimization pur-

pose, we propose a GA-based framework to determine a set of priority and a set of type borderline for given traffic flows. We show that the proposed framework can be applied to optimize reliability and extensibility by designing an initialization algorithm. Our experimental results demonstrate that the GA-based framework with the proposed initialization algorithms outperforms the existing assignment algorithms (AOPA, RPA) used in general real-time systems and an intuitive approach (FPCP).

CHAPTER 6

OPMB: Optimal Priority Assignment for Multi CAN/CAN-FD Buses with a Central Gateway

6.1 Introduction

Automakers keep adding new functions to their products to attract more customers. Since such newly-introduced functions usually require communication with other electronic control units (ECUs) to acquire & deliver sensor (e.g., speedometer, radar, etc.) data, the amount of in-vehicle network traffic keeps rising. Due to this increase of in-vehicle traffic, the controller area network (CAN) — *de facto* standard of the in-vehicle network which supports up to 1Mbps — reaches its bandwidth limit. Thus, the automakers are adding more CAN buses and connecting them via a *central gateway* to handle the increasing vehicle data traffic. In other words, a system of multiple buses connected via a central gateway has become a norm in new vehicles. Recently, automakers have also begun replacing CAN with a higher bandwidth protocol, Controller Area Network with Flexible Data rate (CAN-FD), which can support up to 12Mbps. However, since CAN has enough bandwidth and speed to support certain domains like powertrain and is also cheaper than CAN-FD, both CAN and CAN-FD are expected to coexist in the foreseeable future.

Automakers are very conscious of production cost, and hence want to design a cost-minimized in-vehicle network. It is, therefore, important to know if a designed system satisfies the requirements of given in-vehicle messages at design time. If the system cannot meet the requirements, the system designer must modify the designed system by either adding more resources (e.g., more CAN/CAN-FD buses) or optimizing the system further. Thus, it is necessary to have an "optimal" priority assignment algorithm[1] available at design time. The optimal algorithm can then be used to determine the (non)existence of a schedulable priority assignment for the given set of messages on the designed system.

[1]If the optimal priority-assignment algorithm cannot find a schedulable priority assignment, no other assignment algorithm can find a schedulable priority assignment either.

The priority-assignment problem has been studied by many researchers for decades. Of them, Audsley's optimal priority assignment (AOPA) [82] is proven optimal for a *single* CAN/CAN-FD bus. Even though the central-gateway-based architecture is commonly used in modern vehicles, to the best of our knowledge, Joshi *et al.* [50] are the first and the only one with focus on priority assignment for multi-buses with a central gateway. Their algorithm is claimed to be optimal for the multi-bus system under the assumption that central gateway cannot change priority of in-coming network traffic. However, the central gateway for automotive (e.g., AUTOSAR gateway) can easily change the priority of in-coming network messages with very little cost, and hence there is no reason to restrict priority changes at central gateway in practice. Also, to the best of our knowledge, there is no optimal algorithm proposed thus far for the multi-bus system where the central gateway can change the priority of in-coming network messages. Thus, we need an optimal priority assignment algorithm for the system.

To meet this need, we propose an **O**ptimal **P**riority-assignment algorithm for **M**ultiple CAN/CAN-FD **B**uses with a central gateway (**OPMB**). It builds on backtracking (brute-force search with theoretically-proven pruning) to assign priorities to given messages. In particular, OPMB can tell the system designers the (non)existence of priority assignments which satisfy the timing requirements of the given set of messages in a networked system. However, the brute-force search incurs exponential time complexity, making it essential to prune unnecessary searches. We have identified several ways of pruning unnecessary searches and proved that the identified pruning does not affect the discovery of a schedulable priority assignment. We have also conducted extensive simulation by generating realistic in-vehicle messages. Our simulation results show OPMB is able to determine whether schedulable priority assignment exists or not for 96.9% of realistic automotive message-sets *within 1 second*. Also, the results show that OPMB outperforms the state-of-art priority assignment algorithms in terms of *schedulability coverage*.[2]

This chapter makes the following main contributions:

- A counter-example showing that global priority assignment cannot be optimal for a multi-bus system where the central gateway can alter the priority of in-coming network messages.
- Development of an optimal priority-assignment algorithm, OPMB, for a multiple CAN/CAN-FD bus system with a central gateway;
- Demonstration of utility of OPMB for industry-size problems and its superiority to

[2]Defined as the ratio of the number of schedulable cases (by using a priority assignment algorithm) to the number of tested cases.

existing priority-assignment algorithms for the system consisting of multiple CAN/CAN-FD buses with a central gateway.

The rest of chapter is organized as follows. We discuss the related work in Section 2. We provide the primers of CAN and CAN-FD in Section 3, and describe the system model in Section 4. We prove that global priority assignment algorithm cannot be optimal in Section 5, and state the priority-assignment problem in Section 6. In Section 7, we present the new optimal priority-assignment algorithm, OPMB, for multiple CAN/CAN-FD buses with a central gateway. Section 8 evaluates the utility of OPMB by measuring its execution time and performance in comparison with the existing priority-assignment algorithms. We discuss the limitation of OPMB in Section 9 and conclude the chapter in Section 10.

6.2 Related Work

6.2.1 Priority assignment for CAN/CAN-FD

Since priority assignment to CAN/CAN-FD messages greatly affects the schedulability of a given set of messages, there have been various priority-assignment algorithms proposed for CAN/CAN-FD buses. The most representative of them is Audsley's optimal priority assignment (AOPA) [7], which is proven optimal by Davis *et al.* [82] for a *single* CAN/CAN-FD bus without priority inversion, but priority inversions can occur in practice [26, 51, 52].

The variants of AOPA have also been proposed to address practical problems. For example, Davis *et al.* [27] proposed a robust priority assignment by maximizing the number of successive tolerable transmission errors without any timing violation, in order to account for transmission errors that may happen in practice. Schmidt *et al.* [89] considered backward compatibility in a priority assignment to reflect a condition in which some messages have fixed IDs. Additionally, the limitation of their approach due to the insufficient gap between fixed IDs is addressed in [28]. Joshi *et al.* [50] proposed an algorithm for multiple CAN-FD buses with a central gateway. Even though their algorithm is claimed to be optimal for multi-bus systems, we find that claim does not hold when buses have different link speeds.

There are also different types of ID/priority assignment algorithms for CAN/CAN-FD messages. Pölzlbauer *et al.* [79] considered the extensibility problem of ID assignment for CAN. Park *et al.* [78] proposed a message priority-assignment algorithm for a bus shared by both CAN nodes and CAN-FD nodes where changing the operation mode of CAN

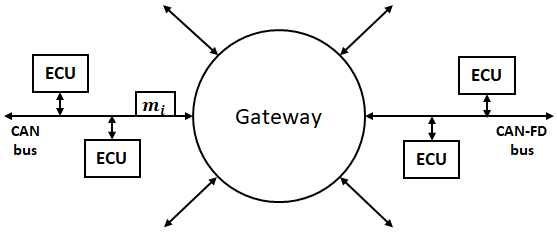


Figure 6.1: System model of multiple CAN/CAN-FD buses with a central gateway

controller is required. The algorithm is designed based on NP-EDF [48] to minimize the number of operation-mode changes.

6.2.2 Priority assignment for distributed real-time system

Multiple buses with a central gateway can be regarded as a distributed real-time system since we need to schedule each message on two buses to meet its end-to-end deadline. The task priority assignment in distributed real-time systems has been studied extensively. This problem is NP-hard [103], and hence various heuristic algorithms have been proposed. Garcia *et al.* [36] proposed a heuristically optimized priority assignment (HOPA) by leveraging those design parameters affecting the worst-case response time. Azketa *et al.* [15] proposed use of the genetic algorithm to find a schedulable priority assignment. Yoon *et al.* [107] proposed zero slack priority assignment (ZSPA), which decomposes an end-to-end deadline into local per-task deadlines and assigns priorities to the sub-tasks using AOPA. If the worst-case response time of a sub-task after assigning a priority is smaller than its local deadline, the remaining slack is used for other sub-tasks. Thus, initially-determined local deadlines are automatically adjusted in the priority assignment.

6.3 System Model

Fig. 6.1 illustrates the system under consideration which is composed of multiple CAN/CAN-FD buses, ECUs, and a central gateway. This central gateway-based networked system is commonly used in modern vehicles (e.g., Volkswagen Atlas 2018). Several ECUs and the gateway are connected to a bus, and they use the shared medium to transmit messages to other ECUs or the gateway on the bus.

6.3.1 Bus and message models

If every ECU connected to a shared bus has the ability to receive a CAN-FD data frame using its CAN-FD controller/transceiver, we call the shared bus *CAN-FD bus*. Otherwise, we call the shared bus *CAN bus*. We assume that only messages compatible with the CAN data frame format can be transmitted on a CAN bus. Likewise, we assume that only messages compatible with the CAN-FD data frame format can be transmitted on a CAN-FD bus. Thus, a bus (b_i) can be modeled as $b_i = \{\vec{ecu}_i, type_i, ls_i^{arb}, ls_i^{data}\}$ where

- $\vec{ecu}_i$: a set of ECUs connected to b_i;
- $type_i \in \{CAN, FD\}$: type of b_i;
- ls_i^{arb}: link speed of b_i during the arbitration phase;
- ls_i^{data}: link speed of b_i during the data phase.

ECUs generate periodic messages (m_i) which usually contain sensor data and/or control commands. We call the ECU that generates (receives) a message m_i *source* (*destination*) ECU of m_i and the bus connected to the ECU source (destination) bus. Since a sensor data can be used by multiple applications on different ECUs, a message can have more than one destination ECU. When the source ECU of a message and its destination ECU(s) are on different buses, the message is forwarded and routed to the destination ECU(s) via the central gateway. In addition, due to the characteristics of vehicle functions, the sensor data or control command carried in a message has a *valid time*, thus imposing a timing constraint on the message. For example, a braking command from an ADAS[3] application (e.g, cruise control) should be delivered to the brake module within 10ms to avoid a crash. Consequently, a message is modeled as $m_i = \{src_i, \vec{dest}_i, p_i, d_i, l_i\}$, where

- src_i: source ECU of m_i;
- $\vec{dest}_i$: a set of destination ECUs of m_i;
- p_i: period of m_i;
- d_i: relative deadline of m_i;
- l_i: payload size of m_i.

[3]Advanced Driver-Assistance System

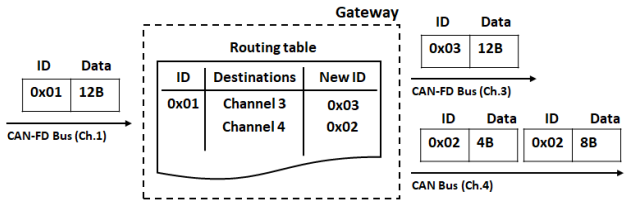


Figure 6.2: Message routing in the gateway based on the embedded routing table

Since a message can be transmitted on multiple (source and destinations) buses, we need to treat a message on different buses as different messages so as to assign different priorities for each bus. Thus, we let $m_{i,j}$ denote m_i on bus j. In our priority-assignment algorithm, $m_{i,j}$ has a "local" deadline $d_{i,j}$, so $m_{i,j}$ is described as $\{p_{i,j}, d_{i,j}, l_{i,j}, tr_{i,j}, \chi_{i,j}\}$, where

- $p_{i,j}$: period of $m_{i,j}$, ($p_{i,j} = p_i$);
- $d_{i,j}$: relative deadline of $m_{i,j}$;
- $l_{i,j}$: payload size of m_i, ($l_{i,j} = l_i$);
- $tr_{i,j}$: transmission time of $m_{i,j}$;
- $\chi_{i,j}$: class of $m_{i,j}$ ($\chi_{i,j} \in \{NOF, SRC, DEST\}$).

We classify a message $m_{i,j}$ to be one of three types:

- $\chi_{i,j} = NOF$ if the message is not forwarded through the central gateway, i.e., m_i is only transmitted on bus j;
- $\chi_{i,j} = SRC$ if m_i is forwarded via the gateway and j is the source bus;
- $\chi_{i,j} = DEST$ if m_i is forwarded via the gateway and j is one of the destination buses.

6.3.2 Gateway model

The central gateway processes in-coming messages to be routed to their destination ECUs based on the embedded routing table in the gateway. Fig. 6.2 illustrates the message routing in the gateway. If the source ECU and a destination ECU are connected to the same type of buses (e.g., source and destination ECUs are connected to a CAN bus), only the

identifier (ID) field of the in-coming message is changed and the message is forwarded to the destination ECU like in AUTOSAR [4] PDU-based routing [10].

On the other hand, if the source and destination ECUs are connected to different types of bus (e.g., the source ECU is connected to CAN bus, but the destination ECU is connected to CAN-FD bus), a frame format conversion is required. A conversion from CAN frame to CAN-FD frame is simple because only the header and tail format changes are required.

However, a conversion from CAN-FD frame to CAN frame is tricky because the maximum payload size of CAN-FD frame is much larger than that of a CAN frame. For example, a 16-byte CAN-FD frame cannot be transformed into a single CAN frame because the maximum payload size of CAN frame is 8 bytes. If a CAN-FD frame with the payload of > 8 bytes needs to be forwarded to a CAN bus, the gateway must split the CAN-FD frame into multiple CAN frames. For example, if a 12-byte CAN-FD frame has to be forwarded to a CAN bus, then the CAN-FD frame will be fragmented into one 8-byte CAN frame and one 4-byte CAN frame by the gateway. The gateway then applies the same ID to the fragmented frames based on the routing table as described in Fig. 6.2.

Our model accounts for the conversion of a CAN-FD frame to multiple CAN frames by summing up the transmission times of the fragmented frames. For example, if 12-byte CAN-FD frame is fragmented into one 8-byte CAN frame and another 4-byte CAN frame to transmit on CAN bus j, then $tr_{i,j}$ is the sum of the transmission times of 8-byte and 4-byte CAN frames.

6.4 Global priority assignment vs. per-bus priority assignment for a CAN/CAN-FD multi-bus system

There are two possible ways of assigning priorities to messages for a multi-bus system: (1) global priority assignment and (2) per-bus priority assignment. Global priority assignment algorithms assign a unique priority to each message for the entire system. For example, under a global priority assignment policy, if the priority of m_i on b_1 is 1, then that of m_i on any other bus is also 1. Also, the priority of m_i should be different from the priority of m_j if $m_i \neq m_j$. The Modified Audsley's Algorithm (MAA) [50] is an example of global priority assignment. On the other hand, per-bus priority assignment algorithms assign a unique priority to each message on a bus. For example, under a per-bus priority assignment policy, any value can be the priority of m_i on b_j if the value is a unique priority for b_j. ZSPA [107] is an example of per-bus priority assignment. We compare these two priority

[4] AUTomotive Open System ARchitecture

assignments with respect to implementation and schedulability.

6.4.1 Implementation

Since the value in the identifier field represents the priority of a CAN/CAN-FD message, to implement the per-bus priority, the central gateway must be able to change the ID value of in-coming messages according to the embedded routing table. However, to implement global priority, the central gateway only needs to forward incoming messages to destination buses according to the embedded routing table. Thus, additional memory space is needed for the per-bus assignment as a penalty to store new IDs for incoming messages in the central gateway. The required additional memory space increases with the size of routing table. For example, if we assume there are 500 entries in the routing table, additional $500 \times 2Bytes$ (assuming 2Bytes used for an ID) for the new ID are required. Also, the execution time for changing ID is also the penalty of per-bus assignment. However, because changing the value of ID requires a single memory copy instruction, the execution time would be very small.

6.4.2 Schedulability

Even though global priority assignment has advantage in memory usage over per-bus priority assignment, it has a disadvantage in finding schedulable priority assignment. The following example shows that global priority assignment cannot be optimal for a CAN/CAN-FD multi-bus system where the gateway can change the IDs of messages.

- $b_1 = \{\{ecu_1\}, 0, 1Mbps, 1Mbps\}$;
- $b_2 = \{\{ecu_2\}, 1, 500Kbps, 5Mbps\}$;
- $m_1 = \{ecu_1, ecu_2, 400us, 400us, 8byte\}$;
- $m_2 = \{ecu_1, ecu_2, 400us, 400us, 8byte\}$.

With a global priority assignment, there are two ways of priority assignment for the above example. The first way is that m_1 has priority 1 (higher) and m_2 has priority 2 (lower). In this case, the WCRT of m_1 on b_1 is $135\mu s$, and that of m_1 on b_2 is $86.6\mu s$ according to Eq. (2.6). Thus, the worst-case end-to-end (e2e) delay of m_1 ($135\mu s + 86.6\mu s = 221.6\mu s$) is smaller than its deadline ($400\mu s$). However, the WCRT of m_2 on b_1 is $270\mu s$, and that of m_2 on b_2 is $173.2\mu s$. Thus, the worst-case e2e delay of m_2 ($270\mu s + 173.2\mu s = 443.2\mu s$) exceeds its deadline ($400\mu s$).

The second way is that m_1 has priority 2 (lower), and m_2 has priority 1 (higher). However, the worst-case e2e delay of m_1 is larger than its deadline. Thus, any global priority assignment algorithms cannot find a schedulable priority assignment for this example.

However, there is a schedulable priority assignment if the same messages on different buses can have different priorities (per-bus priority assignment) as follows: (priority 1 to $m_{1,1}$), (priority 2 to $m_{2,1}$), (priority 1 to $m_{2,2}$), and (priority 2 to $m_{1,2}$). With these priority assignments, the WCRT of m_1 on b_1 is $135\mu s$, and that of m_1 on b_2 is $173.2\mu s$ according to Eq. (2.6). Thus, the worst-case e2e delay of m_1 ($135\mu s + 173.2\mu s = 308.2\mu s$) is less than its deadline ($400\mu s$). Also, the worst-case e2e delay of m_2 ($270\mu s + 86.6\mu s = 356.6\mu s$) is less than its deadline ($400\mu s$). Hence, global priority assignment cannot be optimal.

6.5 Problem Statement

Determining whether a designed in-vehicle network can meet the requirements of a given set of messages is of great importance to minimization of the in-vehicle network cost, thus calling for an optimal priority-assignment algorithm.

To meet this need, we first want to determine whether there exists a schedulable priority assignment for a given set of messages $M = \{m_1, \ldots, m_n\}$ on a designed network of buses $B = \{b_1, \ldots, b_m\}$ such that

$$\forall i, delay^{+}_{i,src} + delay^{+}_{cgw} + delay^{+}_{i,dest} \leq d_i,$$

where $delay^{+}_{i,src}$ is the worst-case delay on the source network (the time between m_i's release at src_i and its arrival at a Rx buffer of the central gateway), $delay^{+}_{cgw}$ is the worst-case processing delay in the gateway (the time between m_i's arrival at the Rx buffer of the source network and its arrival at the Tx buffer of the destination network within the gateway), and $delay^{+}_{i,dest}$ is the worst-case delay on the destination network (time between m_i's arrival at the Tx buffer of the destination network in the gateway and its arrival at the destination ECU).

Our additional goal is to find and provide a schedulable priority assignment, if exists, for the given set M of messages on the set B of buses.

6.6 OPMB

We now present an Optimal Priority assignment for Multi CAN/CAN-FD Buses (OPMB). OPMB is a backtracking-based priority-assignment algorithm. Since backtracking is ba-

Algorithm 5: OPMB

Input : S: the current state of buses
Output: S_{fail}: the failure state

```
1  if isSolutionFound(S) == true then
2  |   return NULL;
3  end
4  S_fail ← NULL;
5  schd ← getSchedulableAssignments(S);
6  fix ← getFixableAssignment(S);
7  if j = failureCheck(schd) then
8  |   S_fail.bus ← j;
9  |   S_fail.state ← S.S_j;
10 |   return S_fail;
11 end
12 if fix ≠ ∅ then
13 |   S' ← applyFixableAssignment(S, fix);
14 |   S_fail ← OPMB(S')
15 |   return S_fail;
16 end
17 for i ← 1 to |schd| do
18 |   correctable ← false;
19 |   if S_fail ≠ NULL then
20 |   |   if schd[selIdx].bus == S_fail.bus or isCrpdInUM_f(schd[selIdx], S_fail) then
21 |   |   |   correctable = true;
22 |   |   end
23 |   |   if correctable == false then
24 |   |   |   continue;
25 |   |   end
26 |   end
27 |   selIdx ← i;
28 |   S' ← applySchedulableAssignment(S, schd[i]);
29 |   S_fail ← OPMB(S');
30 |   if S_fail == NULL then
31 |   |   return NULL;
32 |   end
33 end
```

sically a brute-force search with theoretically-proven pruning, it can always determine whether a solution exists or not. That is, our algorithm is intrinsically optimal priority assignment algorithm. However, without efficient pruning, it may consume a huge amount of time before it terminates. Thus, we need to overcome the key challenge of identifying theoretically-proven pruning.

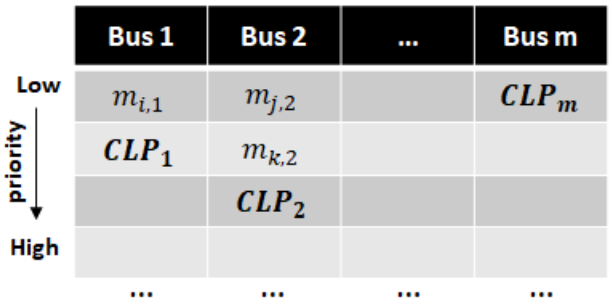


Figure 6.3: Priority-assignment table

6.6.1 Input parameters and return values

Before delving into OPMB, we first need to define the state of buses and the failure state, S and S_{fail}, respectively, which are used as the input parameter and the return value in OPMB.

The state S of buses consists of the states of individual buses, i.e., $S = \{S_1, \ldots, S_n\}$. The state S_j of a bus j consists of a set of assigned messages (AM_j), a set of unassigned messages (UM_j) on the bus, and the current lowest priority (CLP_j) of the bus. Thus, $S_j = \{AM_i, UM_i, CLP_i\}$. For example, if there is no assigned message for bus j, then $CLP_j = 1$ (the lowest priority) as shown in Fig. 6.3 (Bus m). If there is one assigned message for bus j, then $CLP_j = 2$ (the second lowest priority) as shown in Fig. 6.3 (Bus 1), and so on. Also, If the source or the destination of a message (m_i) is on bus j, it ($m_{i,j}$) must belong to either AM_j or UM_j.

The failure state FS consists of the index of failed bus j and the state of bus j at the time of failure $S_j^f \leftarrow S_j$, and thus $FS = \{j, S_j^f\}$.

6.6.2 Initial state

In the initial state of a bus, every message belongs to the unassigned message set, i.e., $\forall i, j\ m_{i,j} \in UM_j,\ AM_j = \emptyset$. Also, CLP_j is initialized to 1 (the lowest priority for each bus) in the initial state.

Since $m_{i,j}$ needs a local deadline, we have to assign it a local deadline $d_{i,j}$. Initially, we assume that $m_{i,j}$ can fully consume the given time margin d_i for m_i. Thus, we assign the value of d_i to $d_{i,j}$ if $\chi_{i,j} = SRC$ or $\chi_{i,j} = NOF$, and we assign the value of $d_i - tr_{i,src_i}$ to $d_{i,j}$ if $\chi_{i,j} = DEST$. For example, if $d_i = 5$ and m_i is transmitted on buses 1 and 2, then $d_{i,1} = 5$ and $d_{i,2} = 5 - tr_{i,1}$. The local deadline is changed during the execution of OPMB.

After assigning the local deadline, we subtract $delay_{cgw}^{+}$ from $d_{i,j}$ if $\chi_{i,j} = SRC$ or $\chi_{i,j} = DEST$ to ignore the gateway processing time in the process of OPMB. For exam-

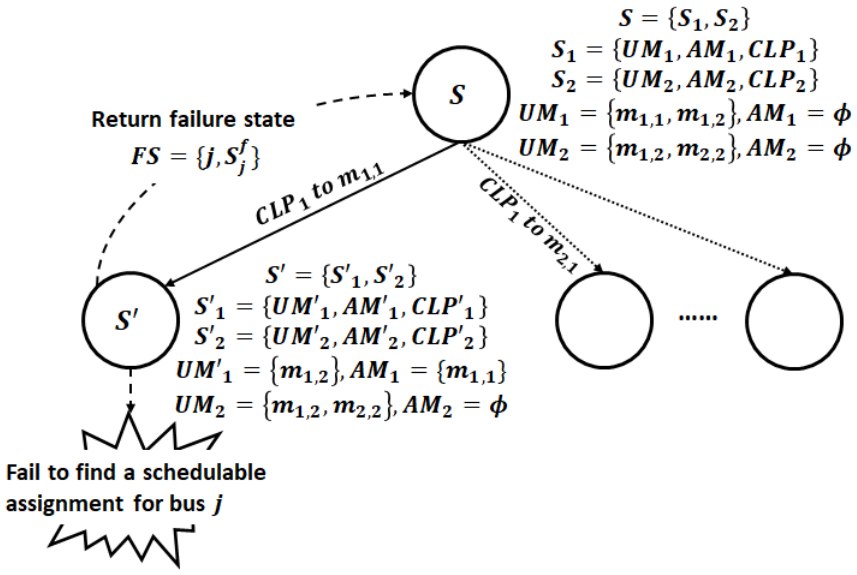


Figure 6.4: Overall procedure of OPMB

ple, if $delay^{+}_{cgw}$ is 1, then $d_{i,1} = 4$ and $d_{i,2} = 4 - tr_{i,1}$.

6.6.3 OPMB overall procedure

OPMB is designed to fill the priority assignment table shown in Fig. 6.3. Algorithm 5 and Fig. 6.4 describe the overall algorithm flow of OPMB, which is implemented recursively. On each function call, OPMB assigns CLP_j to unassigned messages $m_{i,j} \in UM_j$, and calls OPMB recursively for the reduced problem (S') as illustrated in Fig. 6.4. If assigning priority to every message is completed successfully, OPMB returns $NULL$. To determine the assignment to apply, OPMB first finds the set of schedulable assignments and a fixable assignment for the given state S. Note that we will define and detail the schedulable and fixable assignments. Before applying either a schedulable or a fixable assignment, OPMB checks whether there is a failure condition in the given state S as stated in Lines 7-11 of Algorithm 5. If there is no schedulable assignment for bus j and $UM_j \neq \emptyset$, the *failureCheck* procedure returns the failed bus j as well as its state.

If there is a fixable assignment, then OPMB applies that assignment for the given state S. Otherwise, OPMB applies a schedulable assignment. If OPMB gets a failure return for the reduced problem (with a schedulable assignment), OPMB tries another schedulable assignment. For instance, suppose $m_{1,1}$, $m_{1,2}$, $m_{2,1}$ and $m_{2,2}$ are unassigned messages for the given state S as illustrated in Fig. 6.4. Also, suppose OPMB assigns priority CLP_1 to $m_{1,1}$ and calls OPMB with the reduced problem (S'), recursively. If OPMB gets a failure return for the reduced problem, OPMB tries another way of assigning priorities to

Algorithm 6: getSchedulableAssignment

Input : S: the current state of buses
Output: $schd$: a set of schedulable assignment

```
1  schd ← φ
2  UM = UM_1 ∪ ... ∪ UM_m // m is the number of buses
3  for m_{i,j} ∈ UM do
4  |  compute delay^+_{i,j} by applying CLP_j to m_{i,j}
5  |  if delay^+_{i,j} ≤ d_{i,j} then
6  |  |  add (CLP_j to m_{i,j}) to schd
7  |  end
8  end
9  delList ← φ
10 for (CLP_j to m_{i,j}) ∈ schd do
11 |  for (CLP_j to m_{p,j}) ∈ schd do
12 |  |  removeFlag ← true
13 |  |  for k ← 1 to |B| do
14 |  |  |  if m_{i,k} exists and m_{p,k} exists then
15 |  |  |  |  if min(d_{i,k}, d_i − delay^+_{i,j}) ≤ min(d_{p,k}, d_p − delay^+_{p,j}) then
16 |  |  |  |  |  removeFlag ← false
17 |  |  |  |  end
18 |  |  |  end
19 |  |  end
20 |  |  if removeFlag == true then
21 |  |  |  add (CLP_j to m_{p,j}) to delList
22 |  |  end
23 |  end
24 end
25 return schd − delList;
```

unassigned messages (e.g., CLP_1 to $m_{2,1}$).

6.6.4 Pruning unnecessary searches

6.6.4.1 Schedulable assignments

For a given state S, if a certain assignment immediately violates the requirement, we have to discard the branch with that assignment and select a different assignment to find a solution for the given state. Thus, for the given state, OPMB discards any assignment that violates timing constraints ($delay^+_{i,j} > d_{i,j}$) as shown in Lines 5-7 of Algorithm 6. Note that we compute $delay^+_{i,j}$ based on Eq. (2.6). OPMB only selects schedulable assignments which are defined as:

- CLP_j to $m_{i,j}$ when $delay^+_{i,j} \leq d_{i,j}$.

Suppose OPMB applies a schedulable assignment (CLP_j to $m_{i,j}$), then $delay^+_{i,j}$ is determined. So, the local deadline of $m_{i,j}$'s corresponding messages ($\forall k$ $m_{i,k}$ such that $\chi_{i,k} \neq \chi_{i,j}$) has to be re-evaluated because the amount of time can be used by $m_{i,k}$ is reduced, i.e., $d_{i,k}$ is re-evaluated after applying a schedulable assignment as:

$$d_{i,k} = \min(d_{i,k}, d_i - delay^+_{i,j}).$$

We can further prune the unnecessary searches by excluding some schedulable assignments from the set of schedulable assignments. Suppose a schedulable assignment A assigns CLP_j to the message $m_{i,j}$ and a schedulable assignment A' assigns CLP_j to the message $m_{p,j}$. Also, suppose that if $m_{i,k}$ exists, then $m_{p,k}$ also exists for every bus k. We can exclude the assignment A' from the set of schedulable assignments if $\min(d_{p,k}, d_i - delay^+_{p,j}) < \min(d_{i,k}, d_i - delay^+_{i,j})$ for every bus k. Because the corresponding messages on the other bus can have a larger local deadline by selecting A than selecting A', A is a better choice than A'. Note that large time margin is always better than small time margin to be schedulable. Thus, we exclude A' from the set of schedulable assignments as stated in Line 25 of Algorithm 6.

6.6.4.2 Fixable assignments

Suppose state S becomes S' when we select an assignment as illustrated in Fig. 6.5. If the non-existence of a solution for S' can guarantee the non-existence of a solution for S, we do not need to search a solution with other assignments for the given state S. Thus, the number of assignments we have to explore for the given state S becomes 1. We call such assignments *fixable-assignments* for the given state S. For example, in Fig. 6.4, if failure to find a solution with the assignment of CLP_1 to $m_{1,1}$ can guarantee the non-existence of a solution for the given state, we do not need to try any other assignments such as CLP_1 to $m_{2,1}$. We have identified three fixable-assignments as:

- FA1: CLP_j to $m_{i,j}$ when $\chi_{i,j} = NOF$, $delay^+_{i,j} \leq d_{i,j}$.
- FA2: CLP_j to $m_{i,j}$ when $\chi_{i,j} = DEST$, $m_{i,src_i} \in AM_{src_i}$, $delay^+_{i,j} \leq d_{i,j}$.
- FA3: CLP_j to $m_{i,j}$ and CLP_k to $m_{i,k}$ when $\chi_{i,j} = SRC$, $\chi_{i,k} = DEST$, $\forall k$ $delay^+_{i,j} + delay^+_{i,k} \leq d_i - delay^+_{cgw}$.

How to obtain these fixable assignments for a given state S is described in Algorithm 7.

Lemma 5. *Suppose the given state S is changed to S' after applying a FA1. Then, the non-existence of a solution for S' guarantees the non-existence of a solution for S.*

Algorithm 7: getFixableAssignment

Input : S: the current state of buses
Output: fix: a set of fixable assignments

```
1  fix ← φ;
2  UM = UM_1 ∪ ... ∪ UM_m; // m is the number of buses
3  for m_{i,j} ∈ UM_j do
4      compute delay^+_{i,j} by applying CLP_j to m_{i,j};
5      if (χ_{i,j} == NOF and delay^+_{i,j} ≤ d_{i,j}) or (χ_{i,j} == DEST and m_{i,src_i} ∈ AM_{src_i}
         and delay^+_{i,j} ≤ d_{i,j}) then
6          add (CLP_j to m_{i,j}) to fix;
7          continue;
8      end
9      addFlag ← true;
10     for k ← 1 to m do
11         if m_{i,k} exists and χ_{i,j} == SRC and χ_{i,k} == DEST and
             delay^+_{i,j} + delay^+_{i,k} > d_i − delay^+_{cgw} then
12             addFlag ← false;
13             break;
14         end
15     end
16     if addFlag == true then
17         for k ← 1 to |B| do
18             if m_{i,k} exists then
19                 add (CLP_k to m_{i,k}) to fix;
20             end
21         end
22     end
23 end
24 return fix;
```

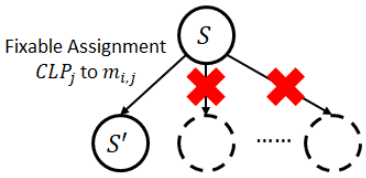


Figure 6.5: If CLP_j to $m_{i,j}$ is a fixable assignment for a given state S, we do not need to move forward with other assignments from S.

Proof: Since $\chi_{i,j} = NOF$, m_i is not forwarded via a central gateway. Thus, the assignment (CLP_j to $m_{i,j}$) only affects the state of bus j (S_j) from the given state S. Hence, the assignment only affects the message schedulability on bus j.

Suppose there is no solution for S', but a solution exists for S. That is, every message on bus j is schedulable in state S, but the assignment makes $\exists k\ m_{k,j} \in UM_j$ unschedulable.

When CLP_j is assigned, $delay^+_{k,j}$ of messages ($\forall k\ m_{k,j} \in UM_j$) is not affected by the assignment. Also, the assignment does not change $d_{k,j}$. Hence, the assignment does not affect the schedulability of $\forall k\ m_{k,j} \in UM_j$. That is, the assignment cannot make $\forall k\ m_{k,j} \in UM_j$ unschedulable. It contradicts the supposition. Thus, if there is no solution for S', then there is no solution for S.

Lemma 6. *Suppose the given state S is changed to S' after applying a FA2. Then, the non-existence of a solution for S' guarantees the non-existence of a solution for S.*

Proof: Since $\chi_{i,j} = DEST$ and $m_{i,src_i} \in AM_{src_i}$, the assignment CLP_j to $m_{i,j}$ only changes the state of bus j (S_j) from the given state S. Thus, the assignment can only affect the message schedulability on bus j. Similarly to the proof of Lemma 5, we can show that the assignment cannot make $\forall k\ m_{k,j} \in UM_j$ unschedulable. Thus, if there is no solution for S', then there is no solution for S.

Lemma 7. *Suppose the given state S is changed to S' after applying a FA3. Then, the non-existence of a solution for S' guarantees the non-existence of a solution for S.*

Proof: Unlike FA1 and FA2, FA3 makes multiple assignments at once, and changes the state of multiple buses (j and k) from the given state S. Thus, FA3 can affect the message schedulability on the multiple buses.

Suppose there is no solution for S', but a solution exists for S. It means that every message on the multiple buses is schedulable in the state S, but the assignments make at least one unassigned message on the buses unschedulable.

Since CLP_j is assigned to $m_{i,j}$ and CLP_k is assigned to $\forall k\ m_{i,k}$ at once, the assignments do not change any local deadline of messages on the buses. Also, the worst-case delay of unassigned messages on the buses is not affected by the assignments. Thus, like the proofs of Lemma 5 and 6, the assignments cannot make a message unschedulable on the buses. Thus, if there is no solution for S', then there is no solution for S.

6.6.4.3 Restriction from Failure State

Suppose OPMB performs a schedulable assignment A in the given state S, and the assignment returns a failure state. To find a solution for the state S, OPMB tries a different schedulable assignment (A') if exists. However, if A' does not resolve the cause of failure OPMB experienced with A, OPMB will encounter the same failure again as illustrated in Fig. 6.6. Thus, to prune the unnecessary search (A'), OPMB checks whether A' can potentially resolve the received failure state or not.

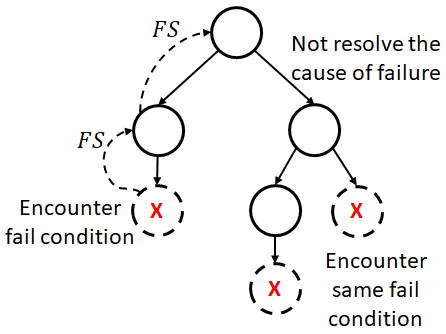


Figure 6.6: Encountering failure without resolving its cause

When OPMB cannot find any schedulable assignment for bus j, OPMB returns the failure state ($FS = \{j, S_j^f\}$). To resolve the failure state (there is no schedulable message in UM_j^f), at least one applied assignment related to UM_j^f has to be revoked. Thus, OPMB allows A' to be tried only when A is related to UM_j^f. In other words, OPMB allows an assignment (A') to be tried only when (1) A assigns CLP_j to a message on the failed bus j or (2) A assigns a priority to the corresponding message in UM_j^f as stated in Lines 19–26 in Algorithm 5.

Lemma 8. *Suppose* OPMB *fails to find a schedulable priority assignment for a given state S with an assignment A and get a failure state $FS = \{j, S_j^f\}$. Also, suppose A does not assign priority to a message on the failed bus j and A does not assign priority to the corresponding message in UM_j^f. Then,* OPMB *cannot find a schedulable priority assignment with any other schedulable assignment (A') for the given state S.*

Proof: Suppose OPMB can find a schedulable priority assignment with an arbitrary assignment A', i.e., every message on bus j is schedulable with A'. In other words, assignment A makes messages in UM_j^f unschedulable. However, the local deadlines of messages in UM_j^f are not affected by the assignment A because A does not assign priority to the corresponding message in UM_j^f. Also, the worst-case delays of messages in UM_j^f are not affected by the assignment A because A does not assign priority to a message on the failed bus j. Hence, A cannot make a message in UM_j^f unschedulable. It contradicts the supposition. Thus, OPMB cannot find a schedulable priority assignment with an arbitrary assignment A'.

6.7 Evaluation

aWe have conducted extensive simulation to evaluate OPMB in comparison with (i) deadline-monotonic (DM) — simple heuristic, (ii) ZSPA — the state-of-art fixed-priority assignment

Table 6.1: System model configuration

Number of buses	3 - 8
Bus type	CAN or CAN-FD
CAN bus link speed	250Kbps, 500Kbps
CAN-FD bus arbitration phase link speed	500Kbps
CAN-FD bus data phase link speed	2Mbps, 5Mbps, 8Mbps
Number of ECUs	250Kbps: 3 - 4ECUs 500Kbps: 4 - 7 ECUs 2Mbps: 7 - 10 ECUs 5Mbps: 8 - 12 ECUs 8Mbps: 10 - 15 ECUs

algorithm for general distributed real-time systems [107] — and (iii) MAA (Algorithm 2 in [50]) — the state-of-art fixed-priority assignment algorithm for a multi-domain system with a central gateway [50].

The main goals of this evaluation are to (1) compare the schedulability coverage of different algorithms for industry-size problems, (2) identify the strength of OPMB over MAA, (3) understand the reason for the identified OPMB's strength, and (4) check the feasibility of OPMB for the industry-size problems. To meet these goals, we measured schedulability coverage and execution time of each algorithm. To assess the schedulability coverage, we have designed and implemented a simulator[5] which randomly generates multi-domain system models and message sets. Generation of the system models and the message sets is detailed next.

6.7.1 Simulator Setup

6.7.1.1 System model generation

The simulator generates a multi-domain system model based on the configuration in Table 6.1. To scale up to industry-size problems, we use the maximum number of domains in [50] as the maximum number of buses. Our simulation adopted the CAN bus link speeds commonly used in the automotive industry. Also, the listed CAN-FD data phase speeds are supported by current commercial CAN-FD transceivers/controllers. The maximum number of ECUs on each bus is restricted to its data phase link.

[5]Available at https://github.com/TaejuPark/OPMB_RTSS_2020

Table 6.2: Signal characteristics

Period (ms)	share	Size (Bytes)	share
1	4%	1	35%
2	3%	2	49%
5	3%	4	13%
10	31%	5 - 8	0.8%
20	31%	9 - 16	1.3%
50	3%	17 - 32	0.5%
100	20%	33 - 64	0.4%
200	1%		
1000	4%		

Table 6.3: Configuration for signal generation

Number of signals	Number of buses $\times$ (10 - 200)
Number of destinations	1 - 4
Probability of gatewayed signal	$10 - 100\%$

6.7.1.2 Message set generation

Our simulator generates signals (instead of messages) and packs them using Algorithm 1 in [50]. We also use the same signal characteristics used in [50] because they generate realistic in-vehicle signals based on real-world automotive benchmarks [94]. The signal characteristics are provided in Table 6.2.

When the simulator generates a signal, the source ECU of the generated signal is randomly chosen with an equal probability. Also, the simulator determines whether the signal is forwarded to other buses or not. If a signal is determined to be forwarded to other buses, the destination ECUs are randomly chosen among all ECUs with an equal probability. Otherwise, only those ECUs that share the same bus with the source ECU of the signal are chosen as the destination ECUs for the signal.

6.7.1.3 Gateway processing delay

We use the state-of-art analysis in [55] for CAN message processing to compute the worst-case gateway processing delay ($delay^{+}_{cgw}$):

$$T_{wait}(ISR_{rx}) + T_e(ISR_{rx}) + \sum_{i=1}^{k} \theta + T_c + T_e(Task_{tx}), \tag{6.1}$$

where $T_{wait}(ISR_{rx})$ is the waiting time for Rx interrupt service routine (ISR), $T_e(ISR_{rx})$ is the execution time for Rx ISR, θ is the time to compare the received ID with the ID value in the routing table, T_c is the time for converting the source bus message to the destination bus message, $T_e(Task_{tx})$ is the execution time for Tx task, and k is the number of elements in the routing table.

We set $T_e(ISR_{rx}) = 5\mu s$, $\theta = 1\mu s$, $T_c = 5\mu s$, $T_e(Task_{tx}) = 20\mu s$ according to the measurement results in [55], and set $T_{wait}(ISR_{rx}) = 0$ since the gateway is assumed to have a dedicated core for each bus in this simulation.

6.7.1.4 OPMB timeout

Even though we try to reduce the execution time of OPMB as much as possible, its execution time could be unacceptably large. Thus, OPMB is forced to terminate upon expiration of a timer. That is, OPMB is terminated whenever the execution time exceeds a predefined expiration time (10,000s in this simulation) on Intel Xeon E5-2683 @ 2.10GHz, 128GB Memory. If OPMB is terminated due to a timeout, we regard it as OPMB's failure to find a schedulable priority assignment.

6.7.2 Test cases

To compare OPMB with the existing algorithms for the various system configurations, we generated test cases by randomly selecting parameters in Tables 6.1 and 6.3 as follows:

- **Overall:** generate 36,000 cases randomly.
- **Fixed number of buses:** generate 6,000 test cases for each fixed number of buses (e.g., 6,000 cases of a 3-bus system, 6,000 cases of a 4-bus system,. . .).
- **Fixed bus-type:** generate 6,000 test cases for each fixed bus-type (CAN only, CAN-FD only, and mixed CAN/CAN-FD). Note that the system is set to have 3 bus types for these test cases.
- **Fixed bus link speed:** generate 6,000 test cases for each fixed bus link speed (250Kbps, 500Kbps, 2Mbps and 5Mbps). Note that the system is set to have 3 bus-types for these test cases.
- **Fixed the maximum number of destination ECUs:** generate 36,000 test cases for each configuration (the maximum number of destinations of a signal is 1,2,3 and 4).

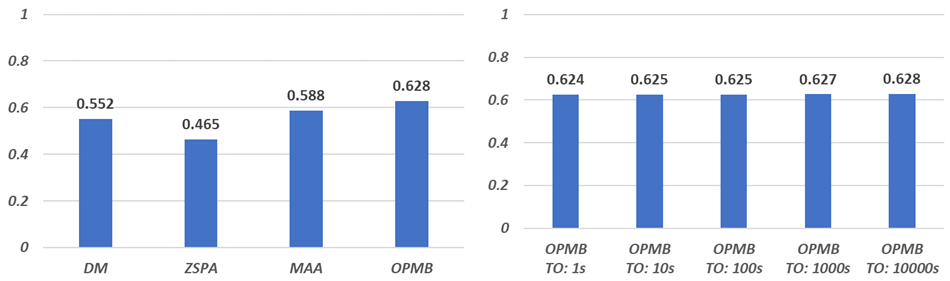


Figure 6.7: (a) Schedulability coverage of the applied algorithm for 'overall'; (b) Schedulability coverage of OPMB for 'overall' with different timeouts

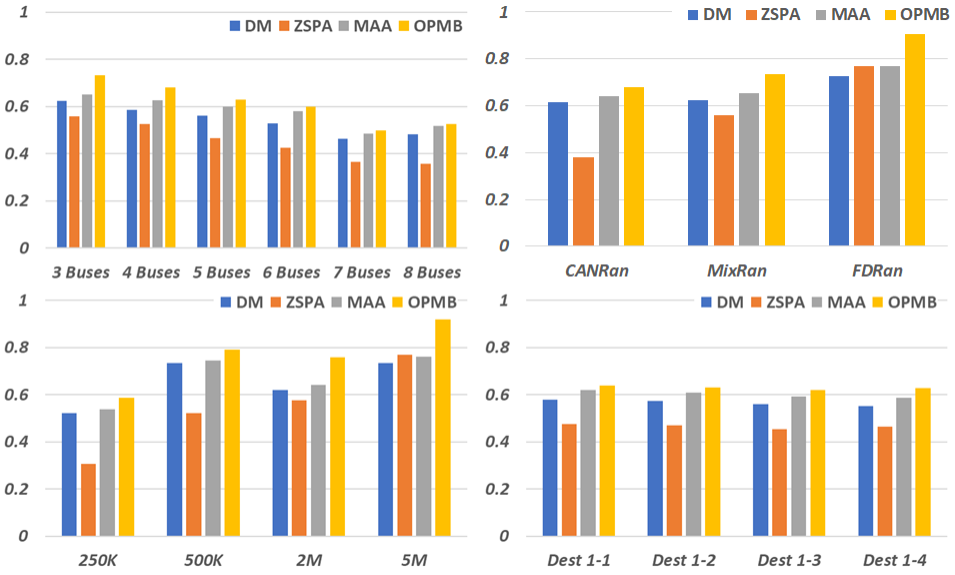


Figure 6.8: Schedulability coverage of the applied algorithm for (a) fixed number of buses, (b) fixed bus-type, (c) fixed bus link speed and (4) different maximum number of destinations of a signal

Note that we only use *valid cases*, where every bus has less than 1.0 utilization (load), from the generated test cases to find a schedulable priority assignment using DM, ZSPA, MAA, and OPMB. Also, when we generate the test cases, the number of buses, the average number of signals per bus and the probability of gatewayed-signals are given manually as command line arguments.[6]

[6] ./run -c 3 -s 50 -p 70 means 3 buses, average 50 signals per bus, 70% of gateway signals.

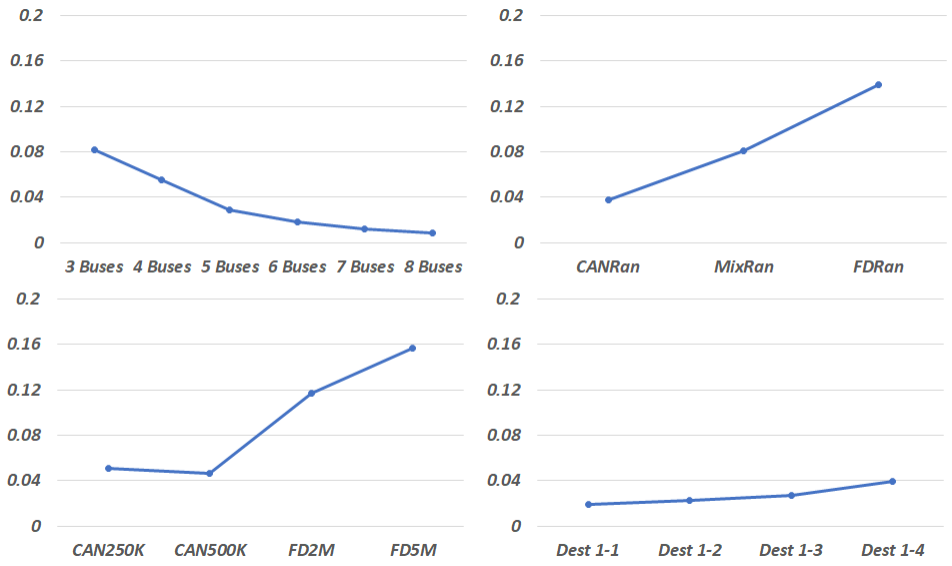


Figure 6.9: Schedulability coverage gap between OPMB and MAA for (a) fixed number of buses, (b) fixed bus-type, (c) fixed bus link speed, and (4) different maximum number of destinations of a signal

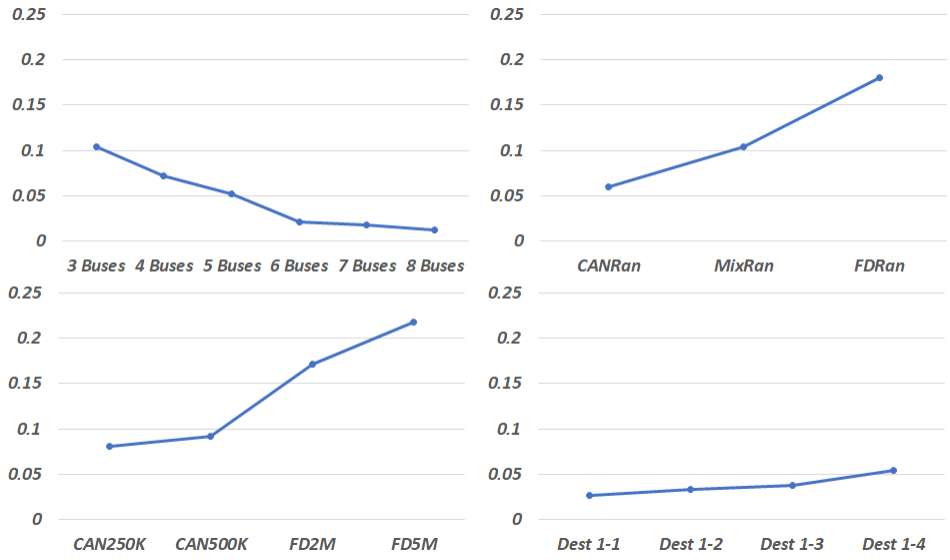


Figure 6.10: Maximum room (schedulability coverage) to improve by using per-bus priority assignment over global priority assignment for (a) fixed number of buses, (b) fixed bus-type, (c) fixed bus link speed, and (4) different maximum number of destinations of a signal

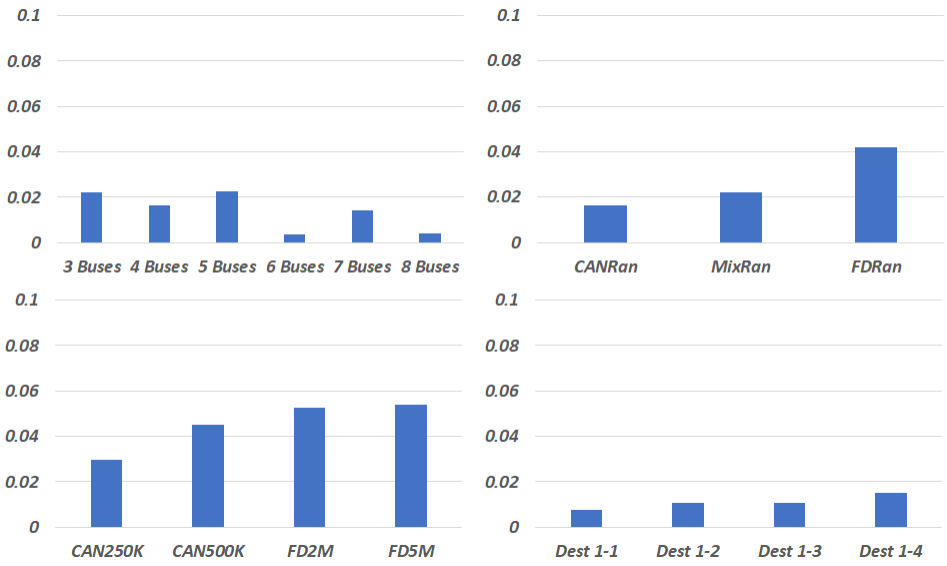


Figure 6.11: OPMB timeout ratio for (a) fixed number of buses, (b) fixed bus-type, (c) fixed bus link speed, and (4) different maximum number of destinations of a signal

6.7.3 Evaluation results and analyses

6.7.3.1 Comparison of schedulability coverage

First, we compare the schedulability coverage of the algorithms. Fig. 6.7(a) shows that the schedulability coverage of the evaluated algorithms for industry-size problems. As we expected, for 'overall' test cases, OPMB outperforms DM by 7%, ZSPA by 16%, and MAA by 4% in schedulability coverage. Also, Fig. 6.7(b) shows that even though increasing the expiration time of OPMB helps to cover more cases, the increase of schedulability coverage is not significant. Fig. 6.8 shows OPMB outperforming the existing algorithms regardless of system configuration.

6.7.3.2 Where does OPMB have strength and why?

The schedulability coverage gap between OPMB and MAA shown in Fig. 6.9. It shows that OPMB has strength for systems with a smaller number of buses, higher bus link speed, and a larger number of destinations of a signal. In fact, OPMB also has strength for systems composed of only CAN-FD buses, and the strength comes from the higher bus link speed of CAN-FD buses.

This can be reasoned about as follows: higher bus link speed and larger number of destinations of a signal make the problem more complex as it increases the available number of combinations for message priority assignment. For example, a higher link speed means

Table 6.4: OPMB execution times (in seconds)

	$t <= 1$	$t <= 10$	$t <= 10^2$	$t <= 10^3$	$t <= 10^4$	Timeout
Schedulable	2327	3	2	5	3	0
Unschedulable	1286	14	17	7	9	56
Total	3613	17	19	12	12	56

a larger number of signals can be scheduled on a bus without exceeding the bus utilization limit ($= 1.0$). Also, a larger number of destinations of a signal means that a signal can have more priorities, e.g., priority 1 for bus 1, priority 2 for bus 2, ... Thus, OPMB shows strength over MAA in these cases.

However, it is difficult to infer the reasons for strength in the case of smaller number of buses. So, we have investigated how much of room (in perspective of schedulability) to improve by using per-bus priority assignment over global priority assignment for various system configurations. If there is more room to improve for the system with a smaller number of buses, OPMB can have more chances to outperform MAA. Thus, it makes sense that OPMB has strength for systems with a smaller number of buses. To investigate the maximum room to improve, we count the following test cases since OPMB is the optimal per-bus priority assignment and MAA is the optimal global priority assignment.

- OPMB finds a schedulable priority assignment while MAA cannot;
- OPMB cannot find a schedulable priority assignment due to the timeout (not decidable).

The percentage of room to improve (counted cases / valid cases) is shown in Fig. 6.10. We can see the percentage of the maximum room to improve increases with the decrease of the number of buses and the increase of bus link speed and the number of signal destinations. This trend is exactly same as the pattern in schedulability coverage gap between OPMB over MAA. That is, the amount of benefit from OPMB is proportional to the possible room to improve by using per-bus priority assignment.

6.7.3.3 Feasibility of OPMB

Since DM, ZSPA and MAA are polynomial-time algorithms, their execution times are expected to be small enough for industry-size (automotive) problems. In contrast, OPMB is basically an exponential-time algorithm, and hence its completion could take very long. To see the differences in execution time, we first measure the average execution time and standard deviation of each algorithm for 'overall' test cases as shown in Table 6.5.

Table 6.5: Execution time (in second)

	DM	ZSPA	MAA	OPMB
Min	0.0004	0.0007	0.0001	0.0001
Max	0.1166	3.885	0.8056	10000
Average	0.0099	0.0912	0.0136	164.6877
Standard deviation	0.0108	0.2012	0.0343	1245.41
5% Percentile	0.0012	0.0028	0.0003	0.0005
25% Percentile	0.0032	0.011	0.0012	0.0018
50% Percentile	0.0065	0.0308	0.0035	0.0049
75% Percentile	0.0127	0.089	0.0111	0.0134
95% Percentile	0.0298	0.3641	0.0595	0.1306

The average execution times of DM, ZSPA and MAA are less than 1 second and the standard deviations are also small. Thus, about 1 second would be expected for industry-size problems. However, the average execution time of OPMB is about 164x larger than those of ZSPA and MAA, and its execution time varies widely (i.e., a large standard deviation) for the test cases. Thus, we investigate the distribution of OPMB's execution time as shown in Table 6.4. OPMB is shown to take less than 1 second for most of the test cases (96.9% for 'overall' test cases).

We also measured the timeout ratio for the other test cases as shown in Fig. 6.11 with the expiration time of 10,000s. The results show that OPMB has the worst timeout ratio for systems with 5Mbps bus link speed. Because the worst timeout ratio is about 5.3%, we expect OPMB to be feasible for about 95% of real-world scenarios.

6.8 Extensions

Switched Ethernet is prevalent in modern vehicles for ADAS and infotainment to handle large amounts of network traffic from camera/lidar/radar sensors. The raw data is processed and transformed into smaller-sized data and then forwarded to other (e.g., powertrain or body) domains. However, the current OPMB only covers the system composed of multiple CAN/CAN-FD buses with a central gateway, and thus cannot handle the switched Ethernet. However, OPMB can be extended to the system that includes the switched Ethernet. From OPMB's perspective, the differences between CAN/CAN-FD and switched Ethernet are (1) computing the worst-case delay in a network and (2) the limited number of priorities. We can use the timing analysis for the switched Ethernet [100] instead of Eq. (2.6). For the limited number (up to 8) of priorities, OPMB needs to assign the same priority to multiple messages on a network because the number of messages might be greater than 8. Thus,

CLP_k should not be incremented by 1 after assigning priority to a message, but CLP_k should be incremented when there is no schedulable message with CLP_k.

6.9 Conclusion

Determining whether or not a designed in-vehicle network can meet the timing requirements of a given set of messages is important to minimize the in-vehicle network cost, thus calling for optimal priority assignment. To meet this need, we have proposed an optimal priority-assignment algorithm, OPMB, for multiple CAN/CAN-FD buses with a central gateway. It is designed based on backtracking (brute-force search with theory-proven pruning). Our in-depth simulation has demonstrated that OPMB outperforms the state-of-art priority-assignment algorithms for multi-bus systems, and has strength especially in high-speed systems which represent future automotive systems. OPMB is also shown to be feasible for most realistic automotive message sets.

CHAPTER 7

PRMB: Priority Assignment and Routing Table Synthesis for Multi CAN/CAN-FD Buses with a Central Gateway

7.1 Introduction

Modern vehicles like Volkswagen Atlas 2018 are equipped with multiple network buses — such as the *Controller Area Network* (CAN) bus, the *de facto* standard in-vehicle network — interconnected via a central gateway for information collection and distribution. The AUTomotive Open System ARchitecture (AUTOSAR) standardizes a software framework for the gateway as part of in-vehicle communication services, and the AUTOSAR Association has been continuously updating the standard documents for the gateway to reflect advances in the automotive industry, e.g., adding features for *CAN with Flexible Data-rate* (CAN-FD).

The AUTOSAR gateway supports both *PDU-direct* and *signal-based* routing. Under PDU-direct routing, an incoming message is directly forwarded from a source network to destination network(s). On the other hand, under signal-based routing, signals in an incoming message are extracted by the gateway which then generates new messages by assembling the signals extracted from multiple incoming messages and forwards the newly generated message to destination networks. This way, signal-based routing can reduce network loads by avoiding unnecessary signal forwarding and combining signals of multiple messages into a single message. Since minimizing production cost is a primary goal for every automaker to attract potential customers at a low price, efficient resource usage while satisfying the given performance (especially timing) requirement is essential. Signal-based routing is a better option than PDU-direct routing for efficient resource usage. However, the procedure associated with signal-based routing is more complex than that with PDU-direct routing, and thus the processing time required for signal-based routing is much larger than

that for PDU-direct routing. Thus, applying signal-based routing to in-vehicle messages while guaranteeing the timely delivery of given signals is a non-trivial problem.

The end-to-end (e2e) delay of a signal depends on the priority assignment of messages on source and destination in-vehicle networks. Also, if the gateway uses signal-based routing, the set of messages on the network depends greatly on how the gateway generates and forwards messages to networks. Hence, if the gateway uses signal-based routing, priority assignment to messages on the network and synthesis of the routing table (describing how messages are generated and forwarded) in the gateway should be considered together to guarantee the timely delivery of given signals. However, to the best of our knowledge, there are no existing solutions that consider message priority assignment and routing table synthesis together. There are a few studies [50, 77] that consider priority a ssignment in multi-bus systems with a central gateway, assuming the central gateway forwards incoming messages based on PDU-direct (*not* signal-based) routing.

To use signal-based routing in in-vehicle networks, we propose **P**riority assignment and **R**outing table synthesis for **M**ultiple CAN/CAN-FD **B**uses with a central gateway (**PRMB**). PRMB consists of two main components. The first component executes the extended version of OPMB— an optimal priority assignment algorithm for a multi-bus system with a central gateway — to find a schedulable priority assignment for a given set of messages. Since OPMB is designed without considering signal-based routing, we extend OPMB to account for differences between PDU-direct routing and signal-based routing. In particular, to consider the difference of gateway processing time between the two routing methods, we have built an experimental platform where the AUTOSAR gateway is ported and the processing times of the two routing methods are measured. Based on the measurement results, we create processing time models for the two routing methods. The second component selects two messages to merge at the gateway and updates the routing table to reflect the chosen merge. Especially, PRMB tries to find a merge that maximally reduces the network load. To evaluate PRMB, we have conducted extensive simulation by generating realistic in-vehicle signals. Our simulation results show that PRMB outperforms state-of-the-art priority assignment algorithms in schedulability coverage. This paper makes the following main contributions:

- Modeling the worst-case gateway processing times for PDU-direct routing and signal-based routing;
- Extending OPMB to support signal-based routing;
- Proposing PRMB to solve priority assignment and routing table synthesis problems together for a prevalent in-vehicle system consisting of multiple CAN/CAN-FD

buses interconnected via a central gateway; and

- Demonstration of PRMB's superiority to existing priority-assignment algorithms for representative automotive message sets.

7.2 Related Work

Since priority assignment to CAN/CAN-FD messages greatly influences the schedulability of a given set of messages, there have been numerous priority-assignment algorithms proposed for CAN/CAN-FD buses.

For a *single* CAN(-FD) bus, Audsley's optimal priority assignment (AOPA) [7] is proven optimal by Davis *et al.* [82] if there is no priority inversion. However, priority inversions can happen in practice [26, 51, 52]. Variants of AOPA have also been proposed to address practical problems. For example, Davis *et al.* [27] considered transmission errors i n priority assignment. The authors of [89, 28, 79] addressed the extensibility of CAN for future changes of message set.

Park *et al.* [78] first considered a bus shared by both CAN and CAN-FD nodes where changing the operation mode of CAN controller is required. They proposed a message priority-assignment algorithm based on NP-EDF [48] to minimize the number of operation-mode changes.

There have also been a few studies that consider the multiple CAN/CAN-FD buses with a central gateway. Joshi *et al.* [50] proposed an algorithm based on AOPA for multi-bus systems. However, they assumed that the central gateway does not change the priority of an incoming message for the destination bus. Park *et al.* [77] addressed the limitation of [50] by allowing the central gateway to change message priorities. They proposed an optimal priority assignment for multi-bus systems using backtracking. However, all of these studies do not consider signal-based routing at the central gateway, so they cannot achieve the benefit of reducing bus load with signal-based routing.

7.3 System Model

The system consists of multiple CAN/CAN-FD buses interconnected via a central gateway as illustrated in Fig. **??**. Each ECU is connected to a single CAN/CAN-FD bus and transmit/receive in-vehicle messages via the bus and possibly the central gateway.

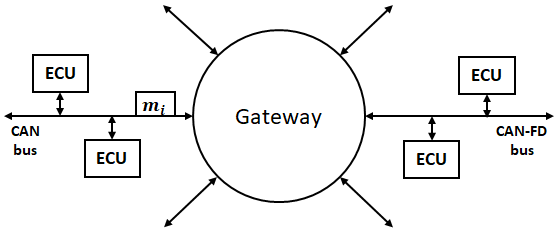


Figure 7.1: System model of multiple CAN/CAN-FD buses with a central gateway

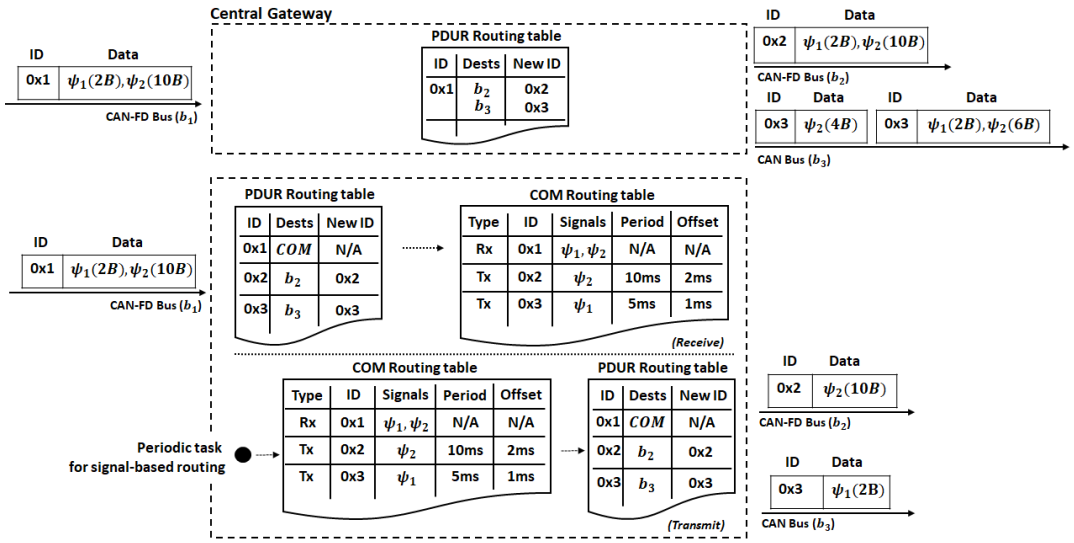


Figure 7.2: Message routing based on (Up) PDU-direct routing and (Down) signal-based routing

7.3.1 Bus model

If each ECU connected to a shared bus can receive a CAN-FD data frame using its CAN-FD controller/transceiver, then the shared bus is called a *CAN-FD bus* else a *CAN bus*. We assume that only messages compatible with the CAN data frame format can be transmitted on a CAN bus. Likewise, we assume that only messages compatible with the CAN-FD data frame format can be transmitted on a CAN-FD bus. Thus, a bus b_i can be modeled as $b_i = \{\vec{ecu}_i, type_i, ls_i^{arb}, ls_i^{data}\}$, where $\vec{ecu}_i$ = the set of ECUs connected to b_i; $type_i \in \{$CAN, CAN-FD$\}$ = the type of b_i; ls_i^{arb} = the link speed of b_i during the arbitration phase; and ls_i^{data} = the link speed of b_i during the data phase.

7.3.2 Signal and message model

Apps running on ECUs periodically generate signals like sensor data, control commands, etc. The generated signals are used as the input to other apps running on other ECUs in

the system. Due to the characteristics of automotive apps, each signal has a *valid time*. A signal can, therefore, be modeled as $\psi_i = \{src(\psi_i), \vec{dest}(\psi_i), p(\psi_i), d(\psi_i)\}$, where $src(\psi_i)$ = source ECU of ψ_i; $\vec{dest}(\psi_i)$ = destination ECUs of ψ_i; $p(\psi_i)$ = period of ψ_i; $d(\psi_i)$ = deadline of ψ_i; and $l(\psi_i)$ = size of ψ_i (in bytes).

The generated signals are packed into CAN/CAN-FD frames and then transmitted to the destination ECUs in the form of CAN/CAN-FD frames via CAN/CAN-FD buses and the central gateway. Since a frame has to be delivered to all destination ECUs of the embedding signals within its specified valid time, the characteristics of a frame depends on the characteristics of the embedding signals. Thus, a frame can be modeled as $m_i = \{b(m_i), \Psi(m_i), p(m_i), d(m_i), l(m_i), tt(m_i), \vec{\omega}(m_i), \chi(m_i)\}$, where $b(m_i)$ = bus where m_i is transmitted; $\Psi(m_i)$ = set of embedding signals in m_i; $p(m_i)$ = period of m_i; $d(m_i)$ = deadline of m_i; $l(m_i)$ = payload size of m_i (in bytes); $tt(m_i)$ = transmission time of m_i; and $\vec{\omega}(m_i)$ = set of corresponding messages of m_i; if $\psi_k \in \Psi(m_i)$ and $\psi_k \in \Psi(m_j)$, then m_j is the corresponding message of m_i. NOF means that not every signal in $\Psi(m_i)$ is forwarded via the central gateway. SRC means that a signal ($\psi_k \in \Psi(m_i)$) is forwarded via the central gateway and $b(m_i)$ is the bus to which $src(\psi_k)$ is attached. $DEST$ means that a signal ($\psi_k \in Psi(m_i)$) is forwarded via the central gateway and $b(m_i)$ is the bus to which one of $dest(\psi_k)$ is attached.

The period of m_i is set to the greatest common divisor of the period of the signals in $\Psi(m_i)$, the deadline of m_i is set to the minimum deadline of signals in $\Psi(m_i)$ or $p(m_i)$.

7.3.3 Gateway model

We consider an AUTOSAR-compliant gateway that forwards incoming messages based on either PDU-direct routing [10] or signal-based routing [9]. The architecture of AUTOSAR g ateway for CAN(-FD) communication is illustrated in Fig. 7.3.

7.3.3.1 Procedures of PDU direct routing

When a message arrives at the Rx buffer of a CAN(-FD) controller, the interrupt routine defined in the CAN(-FD) driver is called to process the incoming message. The incoming message is forwarded to the PDUR (PDU Router) module. PDUR tries to find a matched entry from a pre-defined PDUR routing table for the message based on its ID as shown in Fig. 7.2 (Up). If the value of 'route' in the matched entry is 'CANIF', the message is forwarded to the CAN(-FD) driver from the PDUR module with the destination bus information through CANIF (CAN Interface) module. The CAN(-FD) driver stores the message in the Tx buffers of a CAN(-FD) controller for the destination buses. Thus, if a

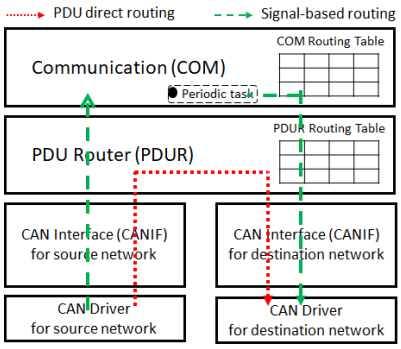


Figure 7.3: AUTOSAR gateway architecture for CAN(-FD) communications

message is forwarded based on PDU-direct routing, a transmission request is synchronized with the message reception.

7.3.3.2 Procedures of signal-based routing

Under signal-based routing, the incoming message should be forwarded to the COM (Communication) module from the PDUR module within the interrupt routine. Thus, the value of 'route' in the matched entry for the incoming message should be 'COM' as shown in Fig. 7.2 (Down). Once the COM module receives a message, it extracts all signals from the message. It updates Tx-type messages (in the COM routing table) which contain some of the extracted signals. After the updates, the reception interrupt routine terminates.

Unlike PDU-direct routing, there is no transmission request within the interrupt routine. The Tx procedures for signal-based routing are executed in a dedicated periodic task as shown in Fig. 7.3. Note that the periodic task is called by the timer interrupt supported by either operating system or hardware. During the execution of the periodic task, it scans the entire Tx-type messages in the COM routing table. If the timer for a Tx-type message expires, the task triggers a transmission request for the message. Then, the message is forwarded from the COM module to the CAN(-FD) driver, and then finally copied into the Tx buffers of the CAN(-FD) controller for the destination buses. After triggering the transmission, the expired timer for the message is reset to the period of the Tx-type message.

7.4 PDU-direct vs. Signal-based Routing

Applying PDU-direct and signal-based routing to an incoming message has pros and cons. We compare them in terms of network load and processing delay at the central gateway.

7.4.1 Network load

When PDU-direct routing is applied to an incoming message, the gateway does not modify the data in the message. So, unnecessary signals would be forwarded to the network. For example, suppose the destination bus of ψ_1 is b_1 and that of ψ_2 is b_2. Also, suppose the signals in a message ($m_1, \Psi(m_1) = \{\psi_1, \psi_2\}$) is forwarded to b_1 by the central gateway using PDU-direct routing. Then, ψ_2 is sent on b_1 even though no ECU connected to b_1 needs ψ_2.

On the other hand, with signal-based routing, the central gateway can generate a new message by picking up signals from incoming messages, and transmit the newly-generated message to the buses. For example, suppose the destination bus of ψ_1 and ψ_3 is b_1 and that of ψ_2 is b_2. Also, suppose the signals in messages ($m_1, \Psi(m_1) = \{\psi_1, \psi_2\}$, $m_2, \Psi(m_2) = \{\psi_3\}$) are forwarded to b_1 by the central gateway using signal-based routing. Then, the central gateway can pick up ψ_1 from m_1 and ψ_3 from m_2, and generate and transmit a new message ($m_3, \Psi(m_3) = \{\psi_1, \psi_3\}$) to b_1. We can thus avoid transmission of unnecessary signals (ψ_2 here), and also reduce overheads by sending fewer messages (sending m_3 once instead of sending m_1 and m_2). So, we can reduce the load of the given network buses.

7.4.2 Processing delay

The worst-case processing delay at the gateway for a signal ($delay^{+}_{cgw}(\psi_i)$[1]) can be divided into three parts: (1) copying an incoming message from the Rx buffer of CAN(-FD) controller to memory, (2) core procedure of PDU-direct routing (or signal based routing) at the gateway, (3) copying an outgoing message from memory to the Tx buffer of CAN(-FD) controller. The first part is the same for both PDU-direct routing and signal-based routing. However, the second and the third parts have different effects on $delay^{+}_{cgw}(\psi_i)$.

The core procedure of signal-based routing is more complex than that of PDU-direct routing as we explained in Section 7.3.3. Thus, the delay for the second part becomes larger when a signal is forwarded based on signal-based routing.

For the third part, there is no clear winner between the two routing methods because the delay for the third part depends on the payload size of the outgoing message, and the payload size of a destination message of a signal could become larger (or smaller) than that of the source message of the signal. For example, suppose $\Psi(m_1) = \{\psi_1, \psi_2\}$, $\Psi(m_2) = \{\psi_3\}$ and b_3 is the destination bus of ψ_1 and ψ_3. If signals in m_1 and m_2 are forwarded to b_3 based on signal-based routing by the central gateway, there are two ways

[1]The time between ψ_i's arrival at the Rx buffer of the source bus and its arrival at the Tx buffer of the destination bus within the central gateway

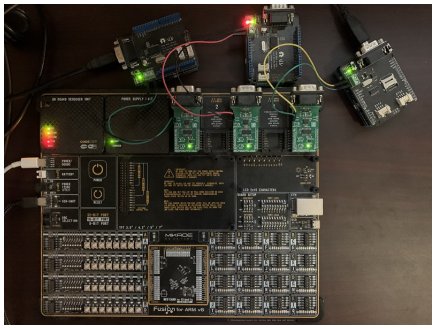


Figure 7.4: AUTOSAR gateway evaluation platform

of forwarding. First, ψ_1 and ψ_3 are merged into m_3. In this case, the payload size of m_3 can be larger than that of m_1 (say $l(\psi_3) > l(\psi_2)$). Second, $\Psi(m_3) = \{\psi_1\}$, $\Psi(m_4) = \{\psi_3\}$. In this case, the payload size of m_1 is smaller than m_3.

7.4.3 Measuring and analyzing processing delay

To learn the difference of processing delay between the two routing methods, we have built an evaluation platform as shown in Fig. 7.4 and measured the execution time of the two routing procedures on the platform. Specifically, we have implemented an AUTOSAR gateway by porting Arctic Core AUTOSAR 4.0 [4] and FreeRTOS [34] on mikroC fusion for ARM v8 board [65] w ith MCP2517FD click board [66]. Also, we implemented ECUs using Arduino Uno [5] and MCP2515/MCP2517FD shields [64]. The click board on the gateway board and the CAN(-FD) shields on Arduino are connected to each other via a copper wire, and thus the ECUs are connected via the implemented AUTOSAR gateway.

7.4.3.1 Delay for copying message

We measured the worst-case execution time of copying messages by varying the payload size of the copied message. The results in Fig. 7.5 show that the worst-case execution time for the copy increases as the payload size increases. Especially, it increases almost by an identical amount for every 4 bytes. We conjecture the reason to be: the API for CAN(-FD) chip control provided by mikroC makes 4-byte data transactions to store (load) the given data to the Tx buffer (memory). Thus, we can model the delay as:

$$delay^{+}_{cgw,rx}(m_i) = \eta_{rx,1} \times \left\lceil \frac{l(m_i)}{4} \right\rceil + \eta_{rx,2} \tag{7.1}$$

$$delay^{+}_{cgw,tx}(m_j) = \eta_{tx,1} \times \left\lceil \frac{l(m_j)}{4} \right\rceil + \eta_{tx,2}, \tag{7.2}$$

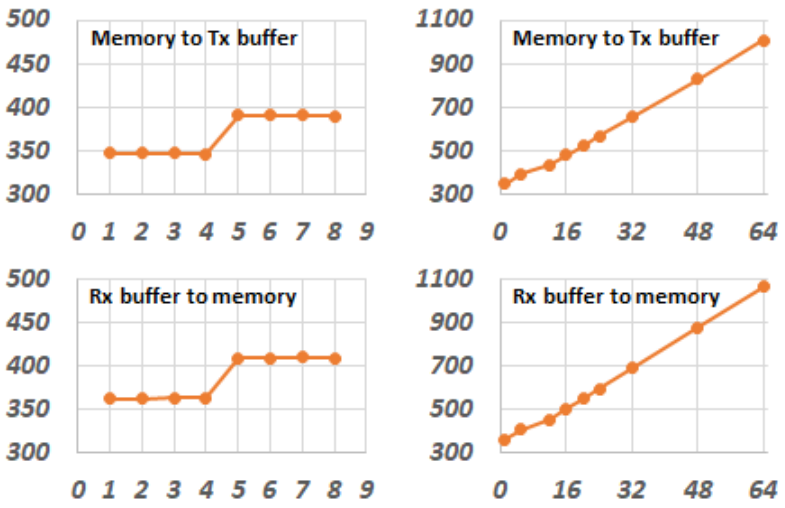


Figure 7.5: Execution time for copying message from/to CAN(-FD) controller (X-axis: payload size (in byte), Y-axis: time (in μs))

where m_i is the message loaded from Rx buffer, m_j is the message stored to Tx buffer, $\eta_{rx,1}$, $\eta_{rx,2}$, $\eta_{tx,1}$ and $\eta_{tx,2}$ are coefficients which depend on the gateway hardware.

7.4.3.2 Delay for core routing procedures

Fig. 7.6 (Left) shows the measured worst-case execution time of the core procedure of PDU-direct routing. We measured the execution time while varying N_{prt} (the number of entries in the PDUR routing table) because N_{prt} may affect the time to find a matched entry for an in-coming message. As we expected, the measured worst-case execution time linearly increases with the increase of N_{prt}. But the measured worst-case execution time remained the same for different payload sizes. That is, the payload size does not have significant impact on the execution time of PDU-direct routing procedure. Thus, we can model the processing delay for PDU-direct routing as:

$$delay^{+}_{pdr} = \eta_{pdr,1} \times N_{prt} + \eta_{pdr,2}, \tag{7.3}$$

where $\eta_{pdr,1}$ and $\eta_{pdr,2}$ are coefficients which depend on the running platform hardware.

Unlike PDU-direct routing, the Rx process is separated from the Tx process for signal-based routing. So, we separately measured the execution time for the Rx and the Tx processes. Fig. 7.6 (Right) shows the measured worst-execution time of the Rx procedure of signal-based routing while varying the number of signals in a processed message. The result shows that the worst-case execution time of the Rx procedure of signal-based routing linearly increases with the increase of the number of signals in a message. This is because

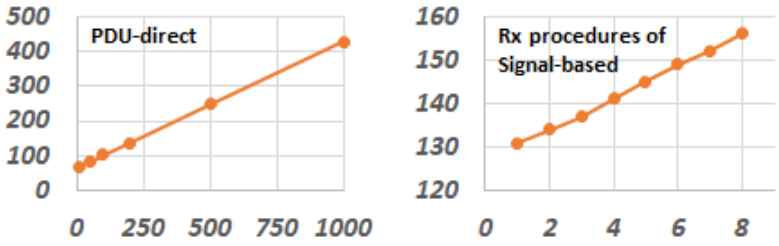


Figure 7.6: (Left) Execution time of core procedure of PDU-direct routing (X-axis: the number of entries in the PDUR routing table, Y-axis: time (in μs)). (Right) Execution time of Rx procedure of signal-based routing (X-axis: the number of signal in a processed message, Y-axis: time (in μs)).

the core procedure extracts signals from the received message and updates Tx-type messages with the received signal value. The receiving procedure also needs to find a matched entry in the PDR-routing table for the received message like PDU-direct routing, and hence we can calculate the processing delay for the Rx procedure of signal-based routing as:

$$delay^{+}_{srx} = \eta_{srx,1} \times N_{prt} + \eta_{srx,2} \times |\Psi(m_i)| + \eta_{srx,3}, \tag{7.4}$$

where $|\Psi(m_i)|$ is the number of signals in m_i, and $\eta_{srx,1}$, $\eta_{srx,2}$ & $\eta_{srx,3}$ are coefficients that depend on the gateway hardware.

Fig. 7.7 shows the measured worst-execution time of the Tx procedure of signal-based routing while varying the number of COM routing table entries and the number of triggered messages to be transmitted in a periodic task. The worst-execution is shown to increase linearly with the increase of the number of entries. This is because the periodic task scans all the messages in the COM routing table to find messages to be transmitted, and thus a destination message of ψ_i triggered last in the worst case. Also, the worst-case execution time increases linearly with the increase of the number of triggered messages to be transmitted. This is because when the transmission of a message is triggered, the message is copied from memory to Tx buffer of CAN(-FD) controller. That is, if there are multiple triggered messages to be transmitted, then the delay for copying messages from memory to the Tx buffer is applied multiple times. Thus, we can compute the processing delay of the Tx procedure of signal-based routing as:

$$\begin{aligned} delay^{+}_{stx}(TM) =& \eta_{stx,1} \times N_{crt} + \eta_{stx,2} + \\ & \sum_{m_j \in TM} \sum_{m_k \in FG(m_j)} (delay^{+}_{cgw,tx}(m_k)) \end{aligned} \tag{7.5}$$

where N_{crt} is the number of entries in the COM routing table, TM is a set of triggered messages, $FG(m_j)$ is the set of fragments of m_j, $\eta_{stx,1}$ and $\eta_{stx,2}$ are coefficients which

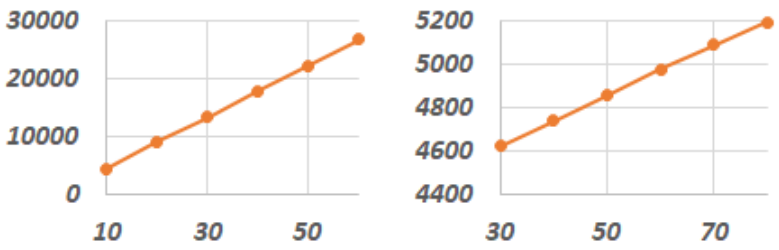


Figure 7.7: (Left) Execution time of Tx procedure of signal-based routing while varying the number of triggered messages to be transmitted (Right) and varying the number of entries in the COM routing table (Y-axis: time (in μs))

depend on the gateway hardware. Note that if the payload size of m_j is larger than 8 bytes and m_j is forwarded to a CAN bus, the message is split into multiple fragments before transmission.

7.4.3.3 Gateway processing delay for a signal

Putting all things together, we can compute the gateway processing delay for a signal under PDU-direct routing as:

$$delay^{+}_{cgw}(\psi_i) = delay^{+}_{cgw,rx}(m_i) + delay^{+}_{pdr} + \sum_{m_k \in FG(m_j)} (delay^{+}_{cgw,tx}(m_k)) \tag{7.6}$$

where m_i is the source message containing ψi, m_j is the destination message containing ψ_i.

Also, we can compute the gateway processing delay for a signal under signal-based routing as:

$$delay^{+}_{cgw}(\psi_i) = delay^{+}_{cgw,rx}(m_i) + delay^{+}_{srx} + P_{stx} + delay^{+}_{stx}(TM(m_j)) + \sum_{m_k \in FG(m_j)} (delay^{+}_{cgw,tx}(m_k)) \tag{7.7}$$

where m_i is the source message containing ψ_i, m_j is a destination message containing ψ_i, P_{stx} is the period of the task for signal-based routing and $TM(m_j)$ is the set of messages triggered to be transmitted with m_j in an instance of the task.

7.5 Problem Statement

Signal-based routing has great potential for the efficient use of resources since it can reduce network loads by avoiding unnecessary signal forwards and combining signals from

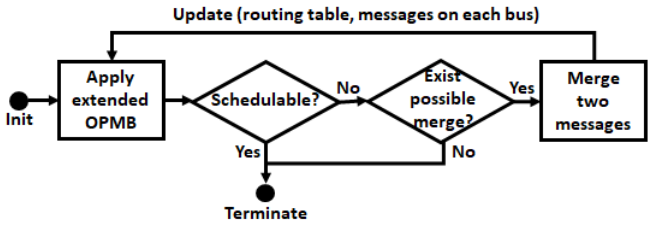


Figure 7.8: Overall procedures of PRMB

multiple incoming messages into one message. However, it is not easy to use signal-based routing because timely delivery of the given signals should be guaranteed as:

$$\forall_i,\ delay^{+}_{src}(\psi_i) + delay^{+}_{cgw}(\psi_i) + delay^{+}_{dest}(\psi_i) \leq d(\psi_i),$$

where $delay^{+}_{src}(\psi_i)$ is the worst-case delay on the source network (the time between ψ_i's release at $src(\psi_i)$ and its arrival at a Rx buffer of the central gateway), and $delay^{+}_{dest}(\psi_i)$ is the worst-case delay on the destination network (time between ψ_i's arrival at the Tx buffer of the destination network in the gateway and its arrival at the destination ECU).

Since (1) $delay^{+}_{src}(\psi_i)$ and $delay^{+}_{dest}(\psi_i)$ are affected by the priority assignment of messages which contain the given signals, (2) $delay^{+}_{cgw}(\psi_i)$ is affected by the routing method and merge of signals at the central gateway, and (3) the generated messages on destination buses are affected by the signal merge at the central gateway, priority assignment and routing table synthesis must be addressed together to guarantee the timing requirements.

Thus, our goal is to solve the priority assignment and routing table synthesis problems together to maximize schedulability coverage for given signals ($\Psi = \{\psi_1, \ldots, \psi_n\}$) on a designed network of buses ($B = \{b_1, \ldots, b_m\}$).

7.6 PRMB

7.6.1 Overall Procedure

We now present a Priority assignment and Routing table synthesis for Multi CAN/CAN-FD Buses (PRMB).

Fig. 7.8 illustrates its overall procedure. First, PRMB pre-processes the given signals (Ψ) to apply the extended OPMB (or OPMB+) (see [77] the original OPMB). PRMB then performs OPMB+ for a given message set. If OPMB+ finds a schedulable priority assignment for the message set, PRMB returns the current routing table configuration and a schedulable priority assignment. Otherwise, PRMB tries to reduce the network load by

merging two messages under signal-based routing. So, it selects two messages to merge, and updates routing tables to reflect the merge. After performing the merge, PRMB re-applies OPMB+ to find a schedulable priority assignment. Until PRMB finds such an assignment, the process of performing OPMB+ and merging messages is repeated. If there is no more possible merge, then PRMB declares that it cannot find any schedulable priority assignment for the given set of signals on a designed network (B).

7.6.2 Pre-processing (Init)

PRMB converts the given signals to the input parameters of OPMB+. The input parameters of OPMB+ (same as that of OPMB) are the set of bus states ($S = \{S_1, \ldots, S_m\}$) where a bus state ($S_j = \{AM_j, UM_j, CLP_j\}$) contains messages on the bus whose priorities have not yet been assigned (UM_j), priority assigned messages on the bus (AM_j), and the current lowest priority of the bus (CLP_j)).

For the conversion, the given set of signals are packed into CAN/CAN-FD messages using the state-of-art signal packing algorithm [50] for a multi-bus system. The generated messages are assigned to the corresponding buses by assuming that every message is forwarded by the central gateway using PDU-direct routing. For example, suppose a message contains ψ_1 and ψ_2 and the source ECU of ψ_1 is on b_1 and the destination ECU of ψ_1 is on b_2. Also, suppose the destination ECU of ψ_2 is on b_3. Then, the message is assigned to b_1, b_2 and b_3 as m_1, m_2 and m_3 each. So, $m_1 \in UM_1, m_2 \in UM_2$ and $m_3 \in UM_3$ after the initialization step.

7.6.3 Extending OPMB

OPMB is proven to be optimal for the system consists of multiple CAN/CAN-FD buses and a central gateway where the gateway can change the ID (or priority) of a message. That is, OPMB can find a schedulable priority assignment for a given set of messages on the system, if any. However, since OPMB is not designed for signal-based routing, we cannot use the original OPMB directly. So, we extend the original OPMB to account for the following distinct characteristics of signal-based routing.

- A destination message ($\chi(m_i) = DEST$) can have multiple source messages as a result of merging signals at the central gateway;
- Due to the difference in the arrival time of merged signal, the delay at the central gateway should account for the difference of arrival times.

Algorithm 8: Extended OPMB (or OPMB+)

Input : S: the current state of buses
Output: S_{fail}: the failure state

```
1  if isSolutionFound(S) == true then
2  |   return NULL;
3  end
4  S_fail ← NULL;
5  schd ← getSchedulableAssignments(S);
6  fix ← getFixAssignmentExt(S);
7  if j = failureCheck(schd) then
8  |   S_fail.bus ← j;
9  |   S_fail.state ← S.S_j;
10 |   return S_fail;
11 end
12 if fix ≠ ∅ then
13 |   S' ← applyFixedAssignment(S, fix);
14 |   S_fail ← OPMB(S')
15 |   return S_fail;
16 end
17 for i ← 1 to |schd| do
18 |   correctable ← false;
19 |   if S_fail ≠ NULL then
20 |   |   if schd[selIdx].bus == S_fail.bus or isCrpdInUM_f(schd[selIdx], S_fail) then
21 |   |   |   correctable = true;
22 |   |   end
23 |   |   if correctable == false then
24 |   |   |   continue;
25 |   |   end
26 |   end
27 |   selIdx ← i;
28 |   S' ← applySchdAssignmentExt(S, schd[i]);
29 |   S_fail ← OPMB(S');
30 |   if S_fail == NULL then
31 |   |   return NULL;
32 |   end
33 end
```

7.6.3.1 Algorithm of OPMB+

Algorithm 8 describes OPMB+ and has almost the same algorithmic flow as the original OPMB. However, the conditions for fixed assignment and the way of deadline adjustment after applying a schedulable assignment need to be changed. So, we replace the functions 'getFixableAssignment' and 'applySchedulableAssignment' with 'getFixAssignmentExt' and 'applySchdAssignmentExt' (see lines 6 and 28 in Algorithm 8). Next, we describe how those functions are revised.

7.6.3.2 Schedulable assignment with signal-based routing

In the original OPMB, when a schedulable assignment ($CLP_{b(m_i)}$ to m_i) is applied, OPMB updates the deadlines of corresponding messages ($m_k \in \vec{\omega}(m_i)$) because the available time for m_k decreases. For example, suppose that message m_i consumes $3ms$, then m_k can spend up to $d(m_i) - 3ms$.

Under signal-based routing, the message (m_k) corresponding to m_i may have a signal not in m_i if m_k is the result of signal merge at the central gateway. So, the difference of arrival times of merged signals should be considered when OPMB+ updates the deadline of the corresponding messages. For example, suppose $\Psi(m_1) = \{\psi_1\}, \Psi(m_2) = \{\psi_2\}, \Psi(m_3) = \{\psi_1, \psi_2\}$. Then, the the deadline adjustment for m_3 should consider the arrival time difference of ψ_1 and ψ_2.

Then, how much of deadline reduction is needed for a merged message? We analyze the amount of reduction required from the following two cases.

Case 1: $p(m_i) - delay^+(m_i) > p(m_k)$. For example, Suppose $\Psi(m_i) = \{\psi_1\}, \Psi(m_j) = \{\psi_2\}, \Psi(m_k) = \{\psi_1, \psi_2\}$, $p(m_i) = 20ms$, $delay^+(m_i) = 4ms$, $p(m_k) = 10ms$. Then, this example belongs to Case 1. In this example, ψ_1 can arrive at the central gateway before $t = 4ms$, and can be contained in Tx-type messages which were transmitted after $t = 4ms$ because $delay^+(m_i) = 4ms$. If the transmission offset of m_3 is set to $delay^+(m_i) \mod p(m_k)$, then m_k is transmitted at $t = 4ms,\ 14ms, \ldots$. Because the deadline of m_k is $10ms$, the transmitted instance of m_k at $t = 4ms$ can arrive before $t = 20ms$ if m_k is guaranteed to satisfy its timing requirement. Thus, the deadline adjustment for m_k is not required when OPMB+ applies the schedulable assignment ($CLP_{b(m_i)}$ to m_i).

Case 2: $p(m_i) - delay^+(m_i) \leq p(m_k)$. For example, Suppose $\Psi(m_i) = \{\psi_1\}, \Psi(m_j) = \{\psi_2\}, \Psi(m_k) = \{\psi_1, \psi_2\}$, $p(m_i) = 20ms$, $delay^+(m_i) = 14ms$, $p(m_k) = 10ms$. Then, this example belongs to Case 2.

In this example, ψ_1 can arrive at the central gateway before $t = 14ms$, and can be contained in Tx-type messages which were transmitted after $t = 14ms$ because $delay^+(m_i) = 14ms$. If the transmission offset of m_k is set to $delay^+(m_i) \mod p(m_k)$, then m_k is transmitted at $t = 4ms,\ 14ms, \ldots$. Because the deadline of m_k is $10ms$, the transmitted instance of m_k at $t = 14ms$ cannot arrive before $t = 20ms$ although m_k is guaranteed to be delivered before its deadline. Thus, reduction of the deadline of m_k is required when OPMB+ applies the schedulable assignment ($CLP_{b(m_i)}$ to m_i). Thus, for Case 2, $d(m_k)$ has to be updated to:

$$d(m_k) = min\{p(m_k) - (delay^+(m_i)\ mod\ p(m_k)), d(m_k)\} \tag{7.8}$$

7.6.3.3 Fixed assignment with signal-based routing

In the original OPMB, three fixed assignments are identified: $FA1, FA2$ and $FA3$ (see [77]for details). Since $FA1$ only considers not-yet-forwarded messages ($\chi(m_i) = NOF$), it is not affected by signal-based routing. So, we only revise FA2 and FA3 to account for signal merges at the central gateway.

$FA2$ assigns a priority to a destination message ($\chi(m_i) = DEST$) if the timely delivery of signals in $\Psi(m_i)$ can be guaranteed. Hence, $\forall_k\ delay^+(m_k)$ should be known to apply $FA2$ where $m_k \in \vec{\omega}(m_i)$. In other words, m_k should be a priority-assigned message. Because a destination message can have multiple corresponding messages under signal-based routing, $FA2$ has to be revised to:

- FA2-Ext: $CLP_{b(m_i)}$ to m_i when $\chi(m_i) = DEST$, $\forall_k\ m_k \in AM_{b(m_k)}$ where $m_k \in \omega(m_k)$, $delay^+(m_i) \leq d(m_i)$.

$FA3$ assigns priorities to a source message ($\chi(m_i) = SRC$) and $\forall_k\ m_k \in \vec{\omega}(m_i)$ at once if timely delivery of signals in $\Psi(m_i)$ and $\Psi(m_k)$ can be guaranteed. Under PDU-direct routing only, a source message can have multiple corresponding messages, but a destination message can have one corresponding message ($1 : N$ relation). So, when we select a source message (m_i) to apply $FA3$, we only need to guarantee the timely delivery of signals in $\Psi(m_i)$. Thus, the set of messages to which priorities need to be assigned are automatically determined: $m_i \cup \vec{\omega}(m_i)$.

On the other hand, under signal-based routing, a destination message can also have multiple corresponding messages ($N : N$ relation). Due to the $N : N$ relation, when we select a source message (m_i) to apply $FA3$, guaranteeing the timely delivery of signals, which do not belong to m_i, is also required. For example, suppose $\Psi(m_i) = \{\psi_1\}, \Psi(m_j) = \{\psi_2\}, \Psi(m_k) = \{\psi_1, \psi_2\}$. If we select m_i to apply $FA3$, guaranteeing the timely delivery of ψ_2 is also required.

Thus, OPMB+ needs to discover a set of messages to assign priorities when it selects a source message (m_i) to apply $FA3$. The discovery process has a signal set and a message set as the variables. The process starts with copying m_i to the message set. It then extracts every signal in the message set and copies the extracted signals into the signal set. It then tries to find all messages which contain any signal in the signal set. The process of extracting signals and finding messages is repeated until no more message is found. Let $\Omega(m_i)$ is the discovered message set and $\Psi(\Omega(m_i))$ is the set of signals in the discovered messages. Then, $FA3$ has to be revised to:

- FA3-Ext: $\forall_k\ CLP_{b(m_k)}$ to $m_k \in \Omega(m_i)$ when $\chi(m_i) = SRC$, $\forall_j\ delay^+_{src}(\psi_j) +$

$delay^{+}_{dest}(\psi_j) \leq d(\psi_j) - delay^{+}_{cgw}(\psi_j)$ where $\psi_j \in \Psi(\Omega(m_i))$, $delay^{+}(m_k) \leq d(m_k)$ and $\chi(m_k) = DEST$.

7.6.4 Merging two messages

7.6.4.1 Selecting two messages to merge

When OPMB+ cannot find a schedulable priority assignment, PRMB tries to reduce the network load by merging two messages. To select two messages to merge, PRMB first finds every possible merge (merging m_i and m_j into m_k) which holds the following conditions:

- $\chi(m_i) = DEST$ and $\chi(m_j) = DEST$;
- $b(m_i)$ is equal to $b(m_j)$;
- $\frac{tt(m_i)}{p(m_i)} + \frac{tt(m_j)}{p(m_j)} > \frac{tt(m_k)}{p(m_k)}$.

The first condition checks if the messages to merge are forwarded by the central gateway. The second condition checks whether the messages to merge are transmitted on the same bus or not. The third condition checks if the merge can reduce the network load. After finding the every possible merge, PRMB selects the merge with the largest load reduction.

7.6.4.2 Update routing tables and message set

When PRMB determines two messages to merge, PRMB updates the routing table as described in Algorithm 9.

Suppose m_i and m_j are merged into m_k. To update the routing table, PRMB finds matched entries for m_i and m_j in PDUR and COM routing tables. If the matched entry for m_i (m_j) is in the PDUR routing table and its 'route' is not the 'COM' module, its 'route' and 'dests' columns must be updated: (1) removing $b(m_k)$ from the 'dests' because m_i (m_j) should not be forwarded to $b(m_k)$ based on PDU-direct routing, and (2) adding 'COM' to the 'route'. Also, because m_i (m_j) is forwarded to the COM module, m_i (m_j) has to be added to the COM routing table as an RX-type entry.

Existence of m_i (m_j) in the COM routing table means that m_i (m_j) was already generated by merging some messages, and PRMB tries another merge on already merged messages. m_i (m_j) no longer exists after merging with m_j (m_i) into m_k, and hence the entry for m_i (m_j) has to be removed from the COM and the PDUR routing tables. After its removal, we need to add an entry for the newly-merged message (m_k) to the COM and the PDUR routing tables.

Algorithm 9: updateRoutingTable

Input : rt : routing table, $merge$: merge

```
1  idxPdurt1 ← getMatchedInPdurt(rt, merge.m1);
2  idxComrt1 ← getMatchedInComrt(rt, merge.m1);
3  idxPdurt2 ← getMatchedInPdurt(rt, merge.m2);
4  idxComrt2 ← getMatchedInComrt(rt, merge.m2);
5  if idxPdurt1 ≠ -1 then
6      dests ← getPdurtEntry(rt, idxPdurt1, 'Dests');
7      if dests ≠ {'COM'} then
8          dests ← dests ∪ {'COM'} - merge.bus;
9          updatePdurtEntry(rt, idxPdurt1, 'Dests', dests);
10         addComrtEntry(rt, merge.m1.id, RX, merge.m1.signals);
11     end
12 end
13 if idxComrt1 ≠ -1 then
14     deleteComrtEntry(rt, idxComrt1);
15     deletePdurtEntry(rt, idxPdurt1);
16 end
17 if idxPdurt2 ≠ -1 then
18     dests ← getPdurtEntry(rt, idxPdurt2, 'Dests');
19     if dests ≠ {'COM'} then
20         dests ← dests ∪ {'COM'} - merge.bus;
21         updatePdurtEntry(rt, idxPdurt2, 'Dests', dests);
22         addComrtEntry(rt, merge.m2.id, RX, merge.m2.signals);
23     end
24 end
25 if idxComrt1 ≠ -1 then
26     deleteComrtEntry(rt, idxComrt2);
27     deletePdurtEntry(rt, idxPdurt2);
28 end
29 addComrtEntry(rt, rt.maxID, TX, merge.mm.signals);
30 addPdurtEntry(rt, rt.maxID, merge.bus);
```

With the updated routing table, PRMB also has to update the current message set to reflect the applied merge. First, PRMB has to remove m_i and m_j from the message set and add m_k to the message set. Second, PRMB updates the message deadlines to reflect the change of central gateway processing time caused by the change of PDUR and COM routing tables.

7.7 Evaluation

We have conducted extensive simulations to evaluate PRMB in comparison with (i) deadline-monotonic (DM), simple heuristic for priority assignment, and (ii) OPMB, the state-of-art

Table 7.1: System model configuration

Number of buses	3 - 8
Bus type	CAN or CAN-FD
CAN bus link speed	250Kbps, 500Kbps
CAN-FD bus arbitration phase link speed	500Kbps
CAN-FD bus data phase link speed	2Mbps, 5Mbps, 8Mbps
Number of ECUs	250Kbps: 3 - 4ECUs 500Kbps: 4 - 7 ECUs 2Mbps: 7 - 10 ECUs 5Mbps: 8 - 12 ECUs 8Mbps: 10 - 15 ECUs

fixed-priority assignment algorithm for a multi-domain system with a central gateway [77]. From this evaluation, we want to know how effectively the evaluated algorithms use given resource, thus focusing on the comparative schedulability coverage of these algorithms.

7.7.1 Simulator Setup

To assess the schedulability coverage of the evaluated algorithms, we have designed and implemented a simulator which (1) generates a multi-bus system and a set of signals, (2) applies the state-of-art signal packing algorithm [50] to the generated signals, (3) applies DM, OPMB and PRMB to a generated system and a set of messages resulting from signal-packing. We run the simulator on a machine equipped with Intel Xeon E5-2683 @ 2.10GHz and 128GB Memory.

7.7.1.1 System and signal generation

The simulator generates a multi-bus system and a set of signals based on the configuration in Tables 7.1, 7.2, and 7.3 used in [77] because they try to emulate the real-world vehicle scenarios by referring to [50] and [94]. When the simulator generates a system, the value of a parameter is randomly chosen within the parameter's range. For example, for a CAN-FD bus, the data phase link speed for the bus is randomly chosen from {2Mbps, 5Mbps, 8Mbps}. When the simulator generates a signal, its period and size are determined with the probabilities in Table 7.2. For example, a signal would have $10ms$ with a 31% probability. Note that the number of buses, the average number of signals per bus and the percentage of gatewayed signals are given as command line arguments to the simulator, so the values of the three parameters are not randomly chosen.

Table 7.2: Signal characteristics

Period (ms)	share	Size (Bytes)	share
1	4%	1	35%
2	3%	2	49%
5	3%	4	13%
10	31%	5 - 8	0.8%
20	31%	9 - 16	1.3%
50	3%	17 - 32	0.5%
100	20%	33 - 64	0.4%
200	1%		
1000	4%		

Table 7.3: Configuration for signal generation

Number of signals	Number of buses $\times$ (10 - 200)
Number of destinations	1 - 4
Probability of gatewayed signal	$10 - 100\%$

Table 7.4: Values of coefficients used in the simulation (in μs)

$\eta_{rx,1}$	$\eta_{rx,2}$	$\eta_{tx,1}$	$\eta_{tx,2}$	$\eta_{pdr,1}$	$\eta_{pdr,2}$
0	30	1.11	27	0.37	65
$\eta_{srx,1}$	$\eta_{srx,2}$	$\eta_{srx,r}$	$\eta_{stx,1}$	$\eta_{stx,1}$	P_{stx}
0.37	3.3	22.5	0.56	80	1000

7.7.1.2 Time to terminate OPMB and the extended OPMB

The execution time of OPMB and the extended OPMB could be unacceptably large because they have exponential time complexity. Thus, we must force them to terminate by setting an expiration time. According to [77], OPMB can make a result for 97% of realistic automotive message sets within 1 second. So, we set the expiration time to 1 second in this simulation.

7.7.1.3 Gateway processing delay

To consider the effect of gateway processing delay in this simulation, we need to set the values of coefficients in Eqs. (7.1)–(7.7) and the period of task for signal-based routing in the simulator. We obtain the values of coefficients via a regression analysis using the measured execution time on the evaluation platform. The task period is set to $1ms$ in this simulation. The values of the coefficients used in this simulation are provided in Table 7.4.

In fact, to compute the value for $\eta_{rx,1}, \eta_{rx,2}, \eta_{tx,1}$ and $\eta_{tx,2}$, we use data collected from Arduino boards because the time for loading/storing data from/to registers in CAN(-FD)

controller takes abnormally large time on the mikroC board. For example, on the Arduino boards, loading/storing data takes about 20 30μs which is consistent with the reported number in [55]. However, it takes about 400μs (20x than normal) on the mikroC board. We conjecture the reason to be: every CAN(-FD) click board shares a single serial peripheral interface (SPI) on the mikroC board, and thus continuous change of SPI slave is required to load/store data from/to registers in a specific CAN(-FD) controller.

7.7.2 Test Cases

We have generated 12000 different test cases for our evaluation. From the generated test cases, we only use valid test cases by excluding the following cases.

- Lower bound of utilization (load) of a bus exceeds 1.0
- Best end-to-end latency of a signal is larger than the deadline of the signal.

To compute the lower bound of utilization of a bus (b_i), we sort signals for each ECU attached to b_i by their size. And we packed the sorted signals into CAN(-FD) messages. In this process, every message fully utilize the payload size limit and a signal is separated into multiple messages if its size is larger then remaining space of a message. Because we compute the lower bound of utilization, period of a message is the maximum period of signals in the message. After that, we sum the utilization of each message on the bus. Also, to compute the best end-to-end latency of a signal, we assume a message contains the signal has the highest priority on its bus. Also, the signal experiences the minimum gateway processing time by assuming the signal forwarded based on PDU-direct routing and the matched entry for the signal in the PDUR routing table is located at the first place.

7.7.3 Evaluation Results: Schedulability Coverage

For each case, we try to find a schedulable priority assignment using different priority-assignment algorithms. The results (schedulability coverage) are shown in Fig. 7.9. As we expected, priority assignment algorithms (OPMB and PRMB) for a multi-domain CAN/CAN-FD network are shown to have much better performance than the simple heuristic algorithm. Also, as expected, PRMB gains 5.7% out of 100% over OPMB from the bus load reduction using signal-based routing.

We also measured the schedulability coverage difference between PRMB and OPMB for different network configurations. The results are shown in Fig. 7.10. In the results, PRMB shows strength over OPMBwhen the multi-bus system has slow link-speed buses.

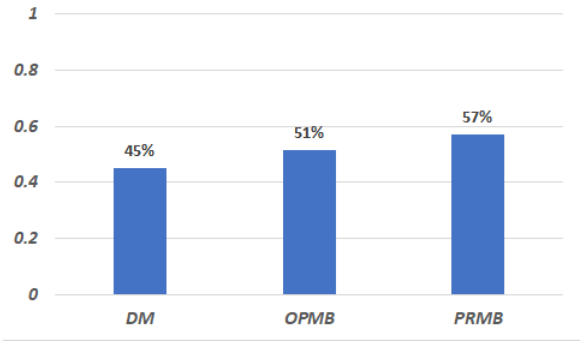


Figure 7.9: Schedulability coverage of priority assignment algorithms for the test cases

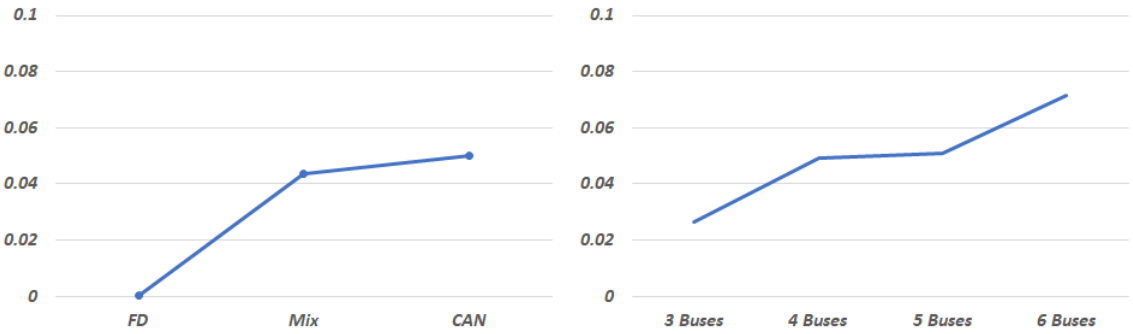


Figure 7.10: Schedulability coverage difference between PRMB and OPMB in different network configurations (schedulability coverage of PRMB - schedulabiity coverage of OPMB)

When PRMB performs a merge, the amount of data reduction on a bus is fixed from the merge, and thus a single merge can have large impact in load reduction on the slow linkspeed bus. Also, the gateway processing time for signal-based routing increases with the number of messages forwarded based on signal-based routing, signal-based routing lose its benefit if the number of entries in the COM routing table. Thus, PRMB shows strengths over OPMBwith the slow link-speed buses. Also, PRMB shows more strength over OPMBwith the larger number of of buses in a system. This is because, with the large number of buses, the probability of existence of a slow link-speed bus in the system increases.

7.7.4 Evaluation Results: Execution Time

We also measured the execution time of PRMB to know how much time needed to obtain results from PRMB. The result is shown in Table. 7.5. The maximum execution time of PRMB took in the extensive simulation was about 6192 seconds. Since the priority assignment and routing table synthesis is done at design time (not at runtime), the maximum

Table 7.5: Execution time (in second)

Average	Standard deviation	Max
106.10	150.77	6196.33

number shows the usability of PRMB.

7.8 Conclusion

Reducing production cost is a primary goal of every automaker, and thus efficient use of given resource is essential. With signal-based routing, we can reduce network loads by merging multiple in-coming messages into one outgoing message at the central gateway. To exploit signal-based routing for efficient use of in-vehicle network resources, we have proposed PRMB which performs priority assignment and routing table synthesis together to guarantee the timely delivery of given signals. Our extensive simulation has demonstrated the superior performance of PRMB over the state-of-art algorithms in terms of schedulability coverage.

CHAPTER 8

Conclusion and Future Directions

The amount of in-vehicle network traffic keeps rising, and automakers have dealt with the increasing trend by re-designing the in-vehicle network architecture and adopting high-bandwidth protocols, such as CAN-FD and Ethernet TSN. Such changes have led to increasing the cost of in-vehicle networks, and thus designing minimum-cost in-vehicle networks becomes important to automotive system designers. This thesis has made five contributions by finding optimal/near-optimal network configurations (PAMT, DOFP, OPMB, and PRMB) and by removing pessimism in the design verification (EACAN). We summarize the contributions and discuss future research directions.

8.1 Contributions

8.1.1 Optimizing network configurations

In Chapter 2, we proposed an optimal priority assignment algorithm, PAMT, for a mixed CAN/CAN-FD bus where CAN and CAN-FD nodes share a single bus. On this mixed bus, CAN nodes are required to change their operation mode to avoid generating error messages. PAMT minimizes the negative impact of mode transition by performing type-based clustering.

In Chapter 5, we identified two key configurable parameters that affect the worst-case end-to-end latency of a message for the standardized frame preemption (802.1Qbu) feature of TSN. We also proposed a genetic algorithm-based optimization framework, called DOFP, to optimize the identified parameters.

In Chapter 6, we showed finding a schedulable priority assignment for multiple CAN and CAN-FD buses with a central gateway is an NP-hard problem where the central gateway can change ID (priority) of messages. Thus, we proposed an exponential time-complexity optimal priority assignment algorithm, called OPMB, for the system, and demonstrated that OPMB is feasible for automotive-size problems.

In Chapter 7, we extended OPMB to support signal-based routing. Also, we proposed PRMB that performs priority assignment and routing table synthesis together to benefit from signal-based routing in resource usage.

8.1.2 Removing pessimism in design verification

In chapter 4, we proposed a runtime adaptation scheme, called EACAN. Instead of using the worst-case transmission error rate in the timing verification, timing verification with EACAN assumes zero transmission error rate at the design time. Then, EACAN monitors the runtime behavior of transmission error, reconfigures the system based on the observation. Verification with the best-case and runtime system reconfiguration increases the acceptance rate at the verification phase while guaranteeing the satisfaction of the given timing requirement.

8.2 Future Work

8.2.1 Analyze and minimize gateway processing time for the limited number of processing cores

In Chapters 6 and 7, we used the gateway processing delay for PDU-based routing and signal-based routing by assuming there is a dedicated processing core for each bus. Thus, we did not consider the waiting time between the arrival of a message at the Rx buffer of CAN(-FD) controller and the execution of the interrupt routine for message reception. However, in practice, the number of processing cores could be lower than the number of buses. So, the beginning of the interrupt routine needs to wait for the completion of another interrupt routine. Thus, we need to analyze how the worst-case gateway processing time for the two routing methods is changed to guarantee the satisfaction of timing requirements.

Moreover, because the waiting time to execute the interrupt routine depends on the bus-to-core assignment, we should find an optimal bus-to-core assignment that minimizes the average gateway processing time, and reflect the optimal assignment within the analysis to remove pessimism in the verification. We can also reduce the gateway processing time by finding the optimal period of each periodic task for signal-based routing or by finding a better merge.

8.2.2 Considering both security and routing at the central gateway

In Chapters 6 and 7, we only considered routing tasks in the central gateway. However, with the growing concerns about secure in-vehicle networking, the central gateway is expected to be the hub of secure in-vehicle networking because the central gateway [31] manages the security keys and performs the access control for incoming messages. However, running the security tasks on the central gateway could interfere with the execution of routing tasks due to the limited processing resource, and will likely cause schedulability degradation. Because the security requirement is not the same for the entire in-vehicle messages, satisfying just the minimum requirement for each message is the best to minimize the schedulability degradation.

In Chapter 7, we showed that signal-based routing enhances the schedulability of an in-vehicle network. However, when signals with different security requirements are packed into one message (e.g., packing signals with the low-security requirement and those with the high-security requirement together), the signal packing could raise the security requirement. So, it could offset the benefit of signal-based routing. Thus, to maximize the benefits from signal-based routing, we must consider the security requirement of each signal when merging signals.

8.2.3 Zone-based architecture

Because of the rapid advancement of System-on-Chip (SoC) and high-bandwidth communication technologies for vehicle systems, a zone-based architecture is now emerging as the next-generation in-vehicle network architecture. In the zone-based architecture, instead of distributing functions to multiple ECUs, a domain controller (or a high-performance ECU) covers functions for a specific domain. A small number of ECUs within a vehicle communicate with each other using high-bandwidth communication protocols (e.g., Ethernet). Thus, the architecture would become much simpler than the current gateway-based architecture. However, the implementation of dozens of time-critical functions on a single high-performance ECU is not trivial because interference between the functions makes it difficult to guarantee the satisfaction of timing requirements. Numerous studies have already proposed (1) scheduling time-critical tasks on a multiprocessor by considering interference between tasks [16, 41] and (2) timing isolation [61, 106, 85] to solve the problem. However, we still need to leverage the trade-off between isolation and efficient use of the resource.

BIBLIOGRAPHY

[1] https://web.archive.org/web/20201116220119/https://www.worldometers.info/cars/.

[2] IEC 61508. Functional safety of electric / electronic / programmable electronic safety-related systems, 2011.

[3] T. Adamson. Hybridization of CAN and CAN FD networks. In *15th International CAN Conference*, 2015.

[4] ARCCORE. https://web.archive.org/web/20200210125428/http://dev.arccore.com/public/user-doc/UD441x/Arctic-Studio-4.0_9503258.html.

[5] Arduino. https://web.archive.org/web/20201201232114/https://www.arduino.cc/.

[6] A. A. Atallah, G. B. Hamad, and O. A. Mohamed. Fault-resilient topology planning and traffic configuration for ieee 802.1qbv tsn networks. In *2018 IEEE 24th International Symposium on On-Line Testing And Robust System Design (IOLTS)*, pages 151–156, 2018.

[7] N. C. Audsley. On priority assignment in fixed priority scheduling. *Information Processing Letters*, pages 79(1):39–44, 2001.

[8] N.C. Audsley. On priority assignment in fixed priority scheduling. *Information Processing Letters*, 79(1):39 – 44, 2001.

[9] AUTOSAR. Specification of Communication, AUTOSAR CP Release 4.4.0, 2018.

[10] AUTOSAR. Specification of PDU Router, AUTOSAR CP Release 4.4.0, 2018.

[11] G. Avni, S. Guha, and G. Rodriguez-Navas. Synthesizing time-triggered schedules for switched networks with faulty links. In *2016 International Conference on Embedded Software (EMSOFT)*, pages 1–10, Oct 2016.

[12] P. Axer, M. Sebastian, and R. Ernst. Probabilistic response time bound for CAN messages with arbitrary deadlines. In *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1114–1117, March 2012.

[13] P. Axer, D. Thiele, and R. Ernst. Formal timing analysis of automatic repeat request for switched real-time networks. In *Proceedings of the 9th IEEE International Symposium on Industrial Embedded Systems (SIES 2014)*, pages 78–87, June 2014.

[14] H. Aysan, A. Thekkilakattil, R. Dobrin, and S. Punnekkat. Efficient fault tolerant scheduling on controller area network (can). In *Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on*, pages 1–8, Sept 2010.

[15] E. Azketa, J. P. Uribe, M. Marcos, L. Almeida, and J. J. Gutierrez. Permutational genetic algorithm for the optimized assignment of priorities to tasks and messages in distributed real-time systems. In *2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 958–965, Nov 2011.

[16] Sanjoy Baruah, Marko Bertogna, and Giorgio Buttazzo. *Multiprocessor Scheduling for Real-Time Systems*. Springer Publishing Company, Incorporated, 2015.

[17] Sofiene Beji, Sardaouna Hamadou, Abdelouahed Gherbi, and John Mullins. Smt-based cost optimization approach for the integration of avionic functions in ima and ttethernet architectures. In *Proceedings of the 2014 IEEE/ACM 18th International Symposium on Distributed Simulation and Real Time Applications*, DS-RT '14, pages 165–174, Washington, DC, USA, 2014. IEEE Computer Society.

[18] U. D. Bordoloi and S. Samii. The frame packing problem for can-fd. In *2014 IEEE Real-Time Systems Symposium*, pages 284–293, Dec 2014.

[19] Christelle Braun, Lionel Havet, and Nicolas Navet. NETCARBENCH: A benchmark for techniques and tools used in the design of automotive communication systems. In *7th IFAC International Conference on Fieldbuses & Networks in Industrial & Embedded Systems - FeT'2007*, pages 321–328, Toulouse, France, November 2007.

[20] I. Broster, A. Burns, and G. Rodriguez-Navas. Probabilistic analysis of can with faults. In *Real-Time Systems Symposium, 2002.*, pages 269–278, 2002.

[21] I. Broster, A. Burns, and G. Rodriguez-Navas. Comparing real-time communication under electromagnetic interference. In *Real-Time Systems, 2004. ECRTS 2004. Proceedings. 16th Euromicro Conference on*, pages 45–52, 2004.

[22] A. Burns and R. I. Davis. Mixed criticality on controller area network. In *2013 25th Euromicro Conference on Real-Time Systems*, pages 125–134, July 2013.

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

[24] Silviu S. Craciunas, Ramon Serna Oliver, Martin Chmelík, and Wilfried Steiner. Scheduling real-time communication in ieee 802.1qbv time sensitive networks. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, RTNS '16, pages 183–192, New York, NY, USA, 2016. ACM.

[25] R. I. Davis and A. Burns. Robust priority assignment for fixed priority real-time systems. In *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, pages 3–14, Dec 2007.

[26] R. I. Davis, S. Kollmann, V. Pollex, and F. Slomka. Controller Area Network (CAN) Schedulability Analysis with FIFO Queues. In *2011 23rd Euromicro Conference on Real-Time Systems*, pages 45–56, July 2011.

[27] Robert I. Davis and Alan Burns. Robust priority assignment for messages on Controller Area Network (CAN). *Real-Time Systems*, 41(2):152–180, 2009.

[28] Robert I. Davis, Alan Burns, Victor Pollex, and Frank Slomka. On Priority Assignment for Controller Area Network when Some Message Identifiers Are Fixed. In *Proceedings of the 23rd International Conference on Real Time and Networks Systems*, RTNS '15, pages 279–288, New York, NY, USA, 2015. ACM.

[29] Robert I. Davis, Steffen Kollmann, Victor Pollex, and Frank Slomka. Schedulability Analysis for Controller Area Network (CAN) with FIFO Queues Priority Queues and Gateways. *Real-Time Syst.*, 49(1):73–116, January 2013.

[30] Eude Cezar de Oliveira. Electrical Architectures and In-Vehicles Networks. In *SAE Technical Paper*. SAE International, 04 2007.

[31] TATA ELEXI. Gateway Architecture for Secured Connectivity and in Vehicle Communication.

[32] Reinhard Felgenhauser. Electromagnetic interference (emi) in e-vehicles. https://web.archive.org/web/20201208050934/https://www.eenewsautomotive.com/content/electromagnetic-interference-emi-e-vehicles [Online; posted 12-Octobuer-2011], October 2011.

[33] J. Ferreira, A. Oliveira, P. Fonseca, and J. Fonseca. An experiment to assess bit error rate in can. In *In Proceedings of 3rd International Workshop of Real-Time Networks (RTN2004*, pages 15–18, 2004.

[34] FreeRTOS. https://web.archive.org/web/20201127150258/https://www.freertos.org//.

[35] Freescale. Future advances in body electronics. https://web.archive.org/web/20190827084634/https://www.nxp.com/docs/en/white-paper/BODYDELECTRWP.pdf.

[36] J. J. G. Garcia and M. G. Harbour. Optimized priority assignment for tasks and messages in distributed hard real-time systems. In *Proceedings of Third Workshop on Parallel and Distributed Real-Time Systems*, pages 124–132, April 1995.

[37] Voica Gavriluţ and Paul Pop. Traffic class assignment for mixed-criticality frames in ttethernet. *SIGBED Rev.*, 13(4):31–36, November 2016.

[38] Voica Gavriluţ and Paul Pop. Traffic-type assignment for tsn-based mixed-criticality cyber-physical systems. *ACM Trans. Cyber-Phys. Syst.*, 4(2), January 2020.

[39] Voica Gavrilut, Bahram Zarrin, Paul Pop, and Soheil Samii. Fault-tolerant topology and routing synthesis for ieee time-sensitive networking. In *Proceedings of the 25th International Conference on Real-Time Networks and Systems*, RTNS '17, page 267–276, New York, NY, USA, 2017. Association for Computing Machinery.

[40] V. Gavriluţ, L. Zhao, M. L. Raagaard, and P. Pop. Avb-aware routing and scheduling of time-triggered traffic for tsn. *IEEE Access*, 6:75229–75243, 2018.

[41] Zhishan Guo, Kecheng Yang, Fan Yao, and Amro Awad. Inter-task cache interference aware partitioned real-time scheduling. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, SAC '20, page 218–226, New York, NY, USA, 2020. Association for Computing Machinery.

[42] A. R. Guraliuc, M. Zhadobov, R. Sauleau, L. Marnat, and L. Dussopt. Millimeter-wave electromagnetic field exposure from mobile terminals. In *Networks and Communications (EuCNC), 2015 European Conference on*, pages 82–85, June 2015.

[43] Z. Hanzalek, P. Burget, and P. Sucha. Profinet io irt message scheduling with temporal constraints. *IEEE Transactions on Industrial Informatics*, 6(3):369–380, Aug 2010.

[44] F. Hartwich. Bit Time Requirements for CAN FD. In *14th International CAN Conference*, 2013.

[45] SAE International. Class c application requirement considerations. *SAE Technical Report J2056/1*, Feb 2000.

[46] ISO/TC22. Iso26262: Road vehicles - functional safety. Technical report, International Organization for Standardization, 2011.

[47] ISO/TC22/SC31. Iso17458-1: Road vehicles - flexray communications system - part1: General information and use case definition. Technical report, International Organization for Standardization, 2013.

[48] K. Jeffay, D. F. Stanat, and C. U. Martel. On non-preemptive scheduling of period and sporadic tasks. In *Proceedings of Twelfth Real-Time Systems Symposium*, pages 129–139, Dec 1991.

[49] P. Joshi, S. S. Ravi, Q. Liu, U. D. Bordoloi, S. Samii, S. K. Shukla, and H. Zeng. Approaches for assigning offsets to signals for improving frame packing in can-fd. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(5):1109–1122, 2020.

[50] Prachi Joshi, Haibo Zeng, Unmesh D. Bordoloi, Soheil Samii, S. S. Ravi, and Sandeep K. Shukla. The Multi-Domain Frame Packing Problem for CAN-FD. In *29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*, 2017.

[51] U. Keskin. Evaluating Message Transmission Times in Controller Area Network (CAN) without Buffer Preemption Revisited. In *Vehicular Technology Conference (VTC Fall), 2013 IEEE 78th*, pages 1–5, Sept 2013.

[52] D. A. Khan, R. J. Bril, and N. Navet. Integrating hardware limitations in CAN schedulability analysis. In *Factory Communication Systems (WFCS), 2010 8th IEEE International Workshop on*, pages 207–210, May 2010.

[53] D. A. Khan, R. I. Davis, and N. Navet. Schedulability analysis of CAN with non-abortable transmission requests. In *ETFA2011*, pages 1–8, Sept 2011.

[54] J. Kim, K. Lakshmanan, and R. Rajkumar. Rhythmic tasks: A new task model with continually varying periods for cyber-physical systems. In *2012 IEEE/ACM Third International Conference on Cyber-Physical Systems*, pages 55–64, April 2012.

[55] J. H. Kim, S. Seo, N. Hai, B. M. Cheon, Y. S. Lee, and J. W. Jeon. Gateway framework for in-vehicle networks based on can, flexray, and ethernet. *IEEE Transactions on Vehicular Technology*, 64(10):4472–4486, 2015.

[56] Daniel Kästner, Marek Jersak, Christian Ferdinand, Peter Gliwa, and Reinhold Heckmann. An integrated timing analysis methodology for real-time systems. In *SAE Technical Paper*. SAE International, 04 2011.

[57] R. Lange, A. C. Bonatto, F. Vasques, and R. S. de Oliveira. Timing analysis of hybrid flexray, can-fd and can vehicular networks. In *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, pages 4725–4730, Oct 2016.

[58] Sune Mølgaard Laursen, Paul Pop, and Wilfried Steiner. Routing optimization of avb streams in tsn networks. *SIGBED Rev.*, 13(4):43–48, November 2016.

[59] J. P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *[1990] Proceedings 11th Real-Time Systems Symposium*, pages 201–209, 1990.

[60] K. Lennartsson. CAN FD filter for Classical CAN controllers. In *15th International CAN Conference*, 2015.

[61] Y. Lim and H. Kim. Cache-aware real-time virtualization for clustered multi-core platforms. *IEEE Access*, 7:128628–128640, 2019.

[62] Shu Lin, D. J. Costello, and M. J. Miller. Automatic-repeat-request error-control schemes. *IEEE Communications Magazine*, 22(12):5–17, December 1984.

[63] Rouhollah Mahfuzi, Amir Aminifar, Soheil Samii, Ahmed Rezine, Petru Eles, and Zebo Peng. Stability-aware integrated routing and scheduling for control applications in ethernet networks. 2018.

[64] Microchip. *Stand-Alone CAN Controller with SPI Interface*, August 2012. Rev. G.

[65] MikroElktronika. https://web.archive.org/web/20201112042535/https://www.mikroe.com/fusion-for-arm//.

[66] MikroElktronika. https://web.archive.org/web/20200809195701/https://www.mikroe.com/mcp2517fd-click.

[67] S. Monroe, D. Stout, and J. Griffith. Solutions of CAN and CAN FD in a mixed network topology. In *14th International CAN Conference*, 2013.

[68] Mischa Möstl, Daniel Thiele, and Rolf Ernst. Invited - towards fail-operational ethernet based in-vehicle networks. In *Proceedings of the 53rd Annual Design Automation Conference*, DAC '16, pages 53:1–53:6, New York, NY, USA, 2016. ACM.

[69] M. Di Natale. Scheduling the CAN bus with earliest deadline techniques. In *Proceedings 21st IEEE Real-Time Systems Symposium*, pages 259–268, 2000.

[70] N. Navet and H. Perrault. Can in automotive applications: a look forward. In *13th International CAN Conference*, 2012.

[71] N. Navet and F. Simonot-Lion. In-vehicle communication networks - a historical perspective and reveiw. *Technical report, University of Luxembourg*, 2013.

[72] N. Navet, Y.-Q. Song, and F. Simonot. Worst-case Deadline Failure Probability in Real-time Applications Distributed over Controller Area Network. *J. Syst. Archit.*, 46(7):607–617, April 2000.

[73] Nicolas Navet, Schehnaz Louvart, Jose Villanueva, Sergio Campoy-Martinez, and Jorn Migge. Timing verification of automotive communication architectures using quantile estimation. In *2014 Embedded Real Time Software and Systems*, Feburary 2014.

[74] Naresh Ganesh Nayak, Frank Dürr, and Kurt Rothermel. Routing algorithms for ieee802.1qbv networks. *SIGBED Rev.*, 15(3):13–18, August 2018.

[75] Shuichi Oikawa and Ragunathan Rajkumar. Linux/rk: A portable resource kernel in linux. In *In 19th IEEE Real-Time Systems Sumposium*, 1998.

[76] R. Serna Oliver, S. S. Craciunas, and W. Steiner. Ieee 802.1qbv gate control list synthesis using array theory encoding. In *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 13–24, April 2018.

[77] T. Park, J. Lyu, and K. G. Shin. Optimal priority assignment for multiple can/can-fd buses with a central gateway. In *2020 IEEE Real-Time Systems Symposium (RTSS)*, Dec 2020.

[78] T. Park and K. G. Shin. Optimal priority assignment for scheduling mixed can and can-fd frames. In *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2019.

[79] Florian Pölzlbauer, Robert I. Davis, and Iain Bate. A Practical Message ID Assignment Policy for Controller Area Network That Maximizes Extensibility. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, RTNS '16, pages 45–54, New York, NY, USA, 2016. ACM.

[80] Florian Pölzlbauer, Robert I. Davis, and Iain Bate. Analysis and Optimization of Message Acceptance Filter Configurations for Controller Area Network (CAN). In *Proceedings of the 25th International Conference on Real-Time Networks and Systems*, RTNS '17, pages 247–256, New York, NY, USA, 2017. ACM.

[81] Sasikumar Punnekkat, Rob Davis, and Alan Burns. Sensitivity analysis of real-time task sets. In R. K. Shyamasundar and K. Ueda, editors, *Advances in Computing Science — ASIAN'97*, pages 72–82, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[82] R. I. Davis and A. Burns and R. J. Bril and J. J. Lukkien. Controller Area Network (CAN) Schedulability Analysis: Refuted, Revisited and Revised. *Real-Time Syst.*, 35(3):239–272, April 2007.

[83] J. Regehr. Scheduling tasks with mixed preemption relations for robustness to timing faults. In *23rd IEEE Real-Time Systems Symposium, 2002. RTSS 2002.*, pages 315–326, 2002.

[84] F. Ren, Y. R. Zheng, M. Zawodniok, and J. Sarangapani. Effects of Electromagnetic Interference on Control Area Network Performance. In *Region 5 Technical Conference, 2007 IEEE*, pages 199–204, April 2007.

[85] James Robb and Björn B. Brandenburg. Nested, but Separate: Isolating Unrelated Critical Sections in Real-Time Nested Locking. In Marcus Völp, editor, *32nd Euromicro Conference on Real-Time Systems (ECRTS 2020)*, volume 165 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:23, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[86] Robert Bosch GmbH. *CAN Specification*, 1991. Ver 2.0.

[87] Robert Bosch GmbH. *CAN with Flexible Data-Rate Specificaton*, April 2012. Ver 1.0.

[88] L. Rodrigues, M. Guimaraes, and J. Rufino. Fault-tolerant clock synchronization in can. In *Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No.98CB36279)*, pages 420–429, Dec 1998.

[89] K. W. Schmidt. Robust Priority Assignments for Extending Existing Controller Area Network Applications. *IEEE Transactions on Industrial Informatics*, 10(1):578–585, Feb 2014.

[90] M. Schreiner, H. Mahmoud, M. Huber, S. Koc, and J. Waldmann. Safe-guarding CAN FD for applications in trucks. *CAN Newsletter*, 01/2013.

[91] M. Schreiner, H. Mahmoud, M. Huber, S. Koc, and J. Waldmann. CAN FD from an OEM point of view. In *14th International CAN Conference*, 2013.

[92] M. Schreiner, H. Mahmoud, M. Huber, S. Koc, and J. Waldmann. Can fd from an oem point of view. In *14th International CAN Conference*, 2013.

[93] J. W. Shin, J. H. Oh, S. M. Lee, and S. E. Lee. CAN FD controller for in-vehicle system. In *2016 International SoC Design Conference (ISOCC)*, pages 227–228, Oct 2016.

[94] Arne Hamann Simon Kramer, Dirk Ziegenbein. Real world automotive benchmark for free. In *6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2015)*, 2015.

[95] W. Steiner. An evaluation of smt-based schedule synthesis for time-triggered multi-hop networks. In *2010 31st IEEE Real-Time Systems Symposium*, pages 375–384, Nov 2010.

[96] W. Steiner. Synthesis of static communication schedules for mixed-criticality systems. In *2011 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*, pages 11–18, March 2011.

[97] Symtavision. https://web.archive.org/web/20201208050402/https://www.luxoft.com/automotive/.

[98] Domitian Tamas-Selicean, Paul Pop, and Wilfried Steiner. Synthesis of communication schedules for ttethernet-based mixed-criticality systems. In *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, CODES+ISSS '12, pages 473–482, New York, NY, USA, 2012. ACM.

[99] L. Tan, C. Du, and Y. Dong. Control-performance-driven period and deadline selection for cyber-physical systems. In *2015 10th Asian Control Conference (ASCC)*, pages 1–6, May 2015.

[100] D. Thiele and R. Ernst. Formal worst-case performance analysis of time-sensitive ethernet with frame preemption. In *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–9, Sept 2016.

[101] D. Thiele, R. Ernst, and J. Diemer. Formal worst-case timing analysis of ethernet tsn's time-aware and peristaltic shapers. In *2015 IEEE Vehicular Networking Conference (VNC)*, pages 251–258, 2015.

[102] K. W. Tindell and A. Burns. Guaranteed message latencies for distributed safety-critical hard real-time control networks. Technical Report YCS 229, University of York, Department of Computer Science, 1994.

[103] K. W. Tindell, A. Burns, and A. J. Wellings. Allocating hard real-time tasks: An np-hard problem made easy. *Real-Time Systems*, 4(2):145–165, Jun 1992.

[104] K. W. Tindell, H. Hansson, and A. J. Wellings. Analysing real-time communications: controller area network (can). In *Real-Time Systems Symposium, 1994.*, pages 259–263, Dec 1994.

[105] Volcano. https://web.archive.org/web/20201026163851/https://www.mentor.com/embedded-software/autosar/.

[106] Meng Xu, Robert Gifford, and Linh Thi Xuan Phan. Holistic multi-resource allocation for multicore real-time virtualization. In *Proceedings of the 56th Annual Design Automation Conference 2019*, DAC '19, New York, NY, USA, 2019. Association for Computing Machinery.

[107] H. Yoon and M. Ryu. Guaranteeing end-to-end deadlines for autosar-based automotive software. *International Journal of Automotive Technology*, 16(4):635–644, Aug 2015.

[108] L. Zhao, P. Pop, and S. S. Craciunas. Worst-case latency analysis for ieee 802.1qbv time sensitive networks using network calculus. *IEEE Access*, 6:41803–41815, 2018.

[109] L. Zhao, P. Pop, Z. Zheng, and Q. Li. Timing analysis of avb traffic in tsn networks using network calculus. In *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 25–36, 2018.

[110] K. M. Zuberi and K. G. Shin. Non-preemptive scheduling of messages on controller area network for real-time control applications. In *Proceedings Real-Time Technology and Applications Symposium*, pages 240–249, May 1995.

[111] A. Zuhily and A. Burns. Optimality of (D-J)-monotonic priority assignment. *Information Processing Letters*, page 103(6), 2007.