2014 IEEE 34th International Conference on Distributed Computing Systems

Cooperative and Efficient Real-time Scheduling for Automotive Communications

Yu Hua* Lei Rao† Xue Liu‡ Dan Feng*
*Wuhan National Lab for Optoelectronics, School of Computer
Huazhong University of Science and Technology, Wuhan, China
{csyhua, dfeng}@hust.edu.cn
†General Motors Research Lab
Palo Alto, CA, USA
lei.rao@gm.com
‡School of Computer Science
McGill University, Montreal, Quebec, Canada
xueliu@cs.mcgill.ca

***Abstract*—FlexRay is an automotive network communication protocol. It provides support to transmit time-sensitive messages in automobiles. FlexRay transmits periodic messages in a static segment and aperiodic messages in a dynamic segment. To improve transmission reliability, FlexRay offers hybrid data management schemes for both static and dynamic segments. However, existing approaches only schedule static segment and dynamic segment separately, leading to poor bandwidth utilization and transmission delay. Moreover, due to the bandwidth limitation, existing best-effort retransmission for all segments fails to achieve high reliability. To address these two concerns, we propose a novel and efficient scheduling scheme, called CoEfficient. The idea behind CoEfficient is to cooperatively schedule the static and dynamic segments, while judiciously stealing the selective slacks for reliable transmission based on practical fault models. CoEfficient schedules both static and dynamic segments in the dual-channel manner based on practical fault models. Extensive experiments based on real-world case studies demonstrate that CoEfficient meets the needs of both real-time transmission and reliability requirements, and delivers significant performance improvements.**

I. Introduction

The paradigm of vehicular systems is shifting. Automobile manufacturers have already begin investigating new power train, chassis, and by-wire control systems. They require a very fast, deterministic, and fault-tolerant protocol that meets the needs of the speed, reliability, and safety of such applications as brake-by-wire and steer-by-wire. Conventional in-vehicle networking solutions, such as Controller Area Network (CAN) [1], do not meet these requirements. FlexRay [2] has been developed to provide higher data rates and better fault tolerance to support automotive applications.

FlexRay provides communication infrastructure for future generation real-time control applications in vehicles. These applications are mostly real-time and safety-critical [1], [3]–[5]. Existing automotive real-time control systems include Anti-lock Braking System (ABS), electronic steering system, and Electronic Stability Control (ESC) system. These systems work in the backgrounds, involving large amounts of sensors, actuators and Electronic Control Units (ECU) working together. This highly sophisticated interaction heavily relies on a communication system that connects different parts in an efficient manner [3], [6], [7]. Although FlexRay has been used as an in-vehicle communication network, its applicability is severely hindered in high-speed safety-critical control systems [8]–[11]. The reason is that real-time safety-critical control applications require data integrity even with the occurrence of transient faults.

The faults in a FlexRay network often occur due to radiation, interference and temperature variation. We can classify these faults into two types, i.e., permanent and transient faults [12]. Specifically, physical damages generally cause the permanent faults that incur long-term malfunctioning. Moreover, the transient faults lead to the miscalculations in the logic and data corruption. The corruption often lasts for a short duration. A FlexRay generally contains a large fraction of transient faults.

The number of rich electronic devices in cars significantly increases. For example, 70 ECUs in luxury cars need to exchange around 2500 signals [1], [13]). Hence, how to deal with transient faults becomes important and requires efficient fault-tolerant techniques to improve communication reliability. Unfortunately, in spite of such reliability concerns, existing frame packing techniques assume a fault-free transmission of frames over the transmission bus. To address this problem, FlexRay is used, and however it is not efficient in practice due to two main challenges.

Challenge 1: Separate scheduling upon heterogeneous segments. FlexRay messages transmit in the heterogeneous form of static and dynamic segments. Existing work schedules either static segments [3], [4], [14], [15] or dynamic segments [16]–[18]. Since they overlook the fact that both static and dynamic segments transmit together in one frame, the separate scheduling exacerbates the performance in terms of bandwidth utilization and transmission latency. Specifically, the scheduling on the dynamic segments relies on the priority-based policy, which potentially causes heavy delays and even data loss for low-priority frames. On the other hand, static segments generally consume a large fraction of available frames. These frames often contain idle slacks that unfortunately can not used by dynamic segments. Hence, cooperative scheduling becomes very important to the performance improvements of

1063-6927/14 $31.00 © 2014 IEEE
DOI 10.1109/ICDCS.2014.22

the entire FlexRay system.

Challenge 2: Best-effort retransmission for alleviating potential faults. In order to offer communication reliability, FlexRay leverages static and pre-defined schedules that contain redundant transmission tasks. This scheme, in practice, fails to efficiently improve the reliability, since it does not support acknowledgement or retransmission schemes. In order to alleviate the inefficiency of scheduling real-time and burst segments in the presence of faults, performing the retransmission of FlexRay's static and dynamic segments is a simple and easy-to-use approach. Existing work [4], [9] performs the retransmissions in a best-effort way. However, due to the fact of the limited bandwidth, the best-effort retransmission for *all* segments is hardly implemented, while leading to severe inefficiency in the data transmission and communication reliability. The reason comes from overlooking the fact that not all segments will fail at the same time and the potential failure of the segments occurs with some probability. Hence, it is unnecessary to retransmit all segments. Instead, we can select some segments, which have higher failure probability, for retransmission.

In order to address the above challenges, we propose a novel and efficient scheduling scheme, called CoEfficient. The design goals of CoEfficient are twofold: (1): *Guarantee the desired reliability goals against transient faults*; (2): *Satisfy the real-time transmission requirements*. Existing real-time scheduling schemes can offer fault-tolerant services well for either scheduling homogenous data segments or managing a single channel, which failing to meet the needs of FlexRay's real-time scheduling requirements. By proposing a selective slack stealing technique, CoEfficient becomes the first work, to the best of our knowledge, that schedules both static and dynamic segments in the dual-channel manner, while providing fault tolerance and delivering real-time performance. Specifically, we make the following contributions.

Cooperative Scheduling with Selective Dual-channel Slacks. By taking into account the properties of data segments, we respectively model the transmission of static, retransmitted and dynamic segments, respectively as *hard-deadline periodic, hard-deadline aperiodic* and *soft-deadline aperiodic* scheduling tasks. These models make the FlexRay transmission practical due to their ease-of-use and simplicity. CoEfficient further employs a novel slack stealing scheme to pilfer selective, rather than any available, slacks to schedule both static and dynamic segments in a unified manner. The selective slack stealing scheme first determines which segments should be retransmitted in order to meet the reliability goal and then steals the idle slacks whose timing lengths are larger than the segments to be retransmitted. We can significantly reduce the computation overhead on the limited, not all, idle slacks. Fast and accurate slack computation allows CoEfficient to capture available slacks that can be pilfered by the tasks of transmitting hard-deadline aperiodic and soft-deadline aperiodic segments. Idle slacks are minimized. As a result, CoEfficient achieves high transmission efficiency and bandwidth utilization.

Differentiated Retransmission for Guaranteed Reliability. Guaranteed reliability refers to the ability to meet the needs of predefined reliability goals. CoEfficient offers the reliability guarantee by leveraging the differentiated retransmission of segments. The differentiated retransmission is adaptive and efficient to select and retransmit the segments based on the analysis of failure probabilities. Different reliability goals may produce different sets of retransmitted segments. Compared with conventional best-effort retransmission for all segments, the differentiated retransmission not only achieves the reliability goal, but also obtains significant performance improvements in terms of bandwidth utilization and transmission latency. In the meantime, CoEfficient is compliant with and complementary to existing schemes for scalable fault tolerance, and focuses on cooperatively scheduling the fault-tolerant segments and efficiently optimizing bandwidth utilization.

Real System Implementation. In order to comprehensively examine the performance of our proposed CoEfficient scheme in the FlexRay network, we implement CoEfficient in a prototype testbed. The real prototype contains all the mentioned components and functionalities of CoEfficient. We use real-world case studies from the automotive industry to evaluate the system performance in terms of overall running time, bandwidth utilization, transmission latency and deadline miss ratios for the transmitted segments. By the comparisons with state-of-the-art FlexRay based scheduling schemes, experimental results demonstrate the efficiency and efficacy of CoEfficient.

The rest of this paper is organized as follows. Section II introduces the backgrounds of FlexRay scheduling. Section III describes the cooperative scheduling and slack stealing schemes. We present the performance evaluation and related work respectively in Sections IV and V. Section VI concludes our paper.

II. FlexRay Network

In this section, we present real-time signals and hybrid segments of FlexRay to support data transmission. We further describe the FlexRay architecture.

A. Real-time Signals and Hybrid Segments

In a FlexRay network, the communications among N Electronic Control Units (ECU) $\{E_1, E_2, \ldots, E_N\}$ are multiplexed over one or more shared buses. An ECU generates multiple signals. For example, the ith ECU generates N_i signals, i.e., $S_i = \{s_1^i, s_2^i, \ldots, s_{N_i}^i\}$. The ECU signals consist of four parts, i.e., period, offset, deadline and length. Specifically, the *period* (P_j^i) is the rate at which the node E_i produces signal s_j^i. The *offset* (O_j^i) is the time after which the node E_i produces the first instance of signal s_j^i. The *deadline* (D_j^i) is the time by which the transmission of the signal s_j^i must be completed. The *length* (W_j^i) is the size of the signal s_j^i in bits.

In order to transmit and schedule real-time signals, FlexRay leverages time-triggered and event-triggered messages. By using these messages, FlexRay supports the transmission of periodic messages in Static Segments (SS) and offers the priority-based scheduling upon event-triggered messages in Dynamic Segments (DS). Moreover, the periodic messages are transmitted in the unique static slots of SS according to Time Division Multiple Access (TDMA), which is similar to the operations in Time-Triggered Protocol (TTP) [19]. On the other hand, aperiodic messages are sent in the dynamic slots

of DS, which is similar to ByteFlight [20] and uses Flexible TDMA (FTDMA).

FlexRay transmits data by using both static and dynamic segments. These segments leverage different scheduling policies to transmit periodic and aperiodic messages in the communication slots. Specifically, a static communication slot is an interval of time that can be exclusively used by a specific node for transmitting a frame. The static segment in a communication cycle can support the transmission of time-critical messages according to a periodic cycle. Within this cycle, a time slot is always reserved to the same network node. This node can use fixed length and locate in a given position. In a static slot, the frame ID corresponds to the slot. Each static communication slot contains a constant number of *macroticks* to support data transmission. Moreover, in the static segment, all communication slots are identical and can be statically configured.

Unlike the configuration and scheduling on static segments, the dynamic segments offer more flexible scheduling scheme for aperiodic messages. The communication slot of transmitting dynamic segments contains one or more *minislots*. The minislots can be interpreted as the smallest time unit, which can be represented and measured by the value of $gdMinislot$. Different from a static communication slot, the duration of a dynamic communication slot can vary by considering the length of the frame. A variable $vSlotCounter$ contains the ID of the current dynamic slot that starts from a pre-configured initial value. A dynamic slot can transmit a frame with the corresponding ID. Furthermore, FlexRay determines the duration of the dynamic slot via the computation of the length of the transmitted frame. In some cases, the duration of a dynamic communication slot becomes one minislot when no frames are sent in the FlexRay network.

B. FlexRay Architecture and Data Segments

In order to describe a FlexRay architecture, we need to study the cluster topology and node structures. A FlexRay cluster consists of multiple nodes that are connected. The connection model depends on the topology. For example, a bus topology uses direct communication channel, while a star topology uses star couplers. In order to support various applications, FlexRay allows a cluster to flexibly configure network topology. The topology includes bus, star or hybrid connection. Moreover, each node in a FlexRay cluster contains a host and a Communication Controller (CC). These two components are connected by a Controller-Host Interface (CHI). CHI becomes a buffer between the host and CC. In general, the host is a part of an ECU and can carry out the application software to deal with incoming messages and generate outgoing messages. The functions of offering FlexRay protocol services can be implemented and executed in the communication controller.

A bus driver contains a transmitter and a receiver. In order to improve the real-time efficiency of transmitting messages, FlexRay offers inter-node connection, in which a bus driver can connect with the communication controller to one communication channel. To further guarantee the synchronization performance, the bus driver needs to contain clock synchronization with other nodes, while constructing and checking cyclic redundancy code verification. The periodic and aperiodic real-time messages can be transmitted in FlexRay communication cycles via multiple network nodes.

One salient feature of a FlexRay network is the high transmission reliability via the design of dual-channel communication. The node architecture supports the scheduling on static and dynamic segments in a dual-channel manner. Specifically, each node contains a *schedule table*. The schedule table maintains the scheduling sequences of transmitting the messages within the static segments. The *priority queues* serve for scheduling the dynamic segments.

In practice, due to the distinct functions and design principles, the static and dynamic segments have different scheduling schemes. Specifically, in order to schedule static segments, we need to maintain a timing based sequence, i.e., the number of cycles and slots, as well as the associated message in the schedule table. Moreover, in order to schedule dynamic segments, we need to define the slot number to each node. In the meantime, we schedule all messages in each priority queue in the fixed priority manner. In order to improve scheduling efficiency, we require a buffer in the CHI to maintain the messages that can be written by the host and read by the communication controller.

The important problem in scheduling FlexRay messages is to determine which messages are transmitted during the allocated slots in an efficient manner. We use different schemes respectively for scheduling static and dynamic messages. For scheduling static messages, a schedule table is used to maintain the transmission time in each network node. We further place a given message into its associated static buffer in the CHI. On the other hand, for scheduling dynamic messages, a node in the FlexRay is able to send different messages using the same dynamic Frame ID. We schedule two or more messages with the same frame ID to be sent in the same bus cycle based on the comparisons in terms of their priorities. In order to guarantee the data transmission quality, FlexRay sends the message from the head of the priority queue in the current bus cycle. FlexRay also needs to insert the messages with the same Frame ID into a local output queue.

III. Cooperative Scheduling upon Fault-tolerant FlexRay Segments

This section presents the cooperative scheduling scheme upon fault-tolerant segments in a dual-channel manner. The scheduling tasks are classified into periodics and aperiodics to transmit various data segments. We also present the fault model and probability analysis. Selective slacks are further computed to support efficient FlexRay communications.

A. Periodic and Aperiodic Tasks

In order to improve communication reliability and optimize the bandwidth utilization, we use a fault-tolerant cooperative scheduling upon the static and dynamic segments in the FlexRay network. Specifically, we model the transmission of static, retransmitted and dynamic segments respectively as hard deadline periodic, hard deadline aperiodic and soft deadline aperiodic tasks. The design goal is to *schedule both periodic and aperiodic tasks to meet all periodic deadlines and achieve the reliability goal* ρ.

1) Periodic Tasks: Consider a FlexRay network with H periodic tasks, $\tau_1, \tau_2, \cdots, \tau_H$. Each task, $\tau_i, 1 \leq i \leq H$, has a worst-case computation requirement C_i, a period T_i, an offset relative to time origin ϕ_i where $0 \leq \phi_i \leq T_i$, and a hard deadline d_i ($d_i \leq T_i$). We assume that the parameters C_i, T_i, ϕ_i and d_i, are known. The tasks with smaller value of d_i are allocated higher priority. For a periodic task τ_i, it leads to an infinite sequence of jobs. The kth job, represented as τ_{ik}, operates at time $\phi_i + (k_i - 1)T_i$ and needs to be completed by $\phi_i + (k_i - 1)T_i + d_i$.

2) Aperiodic Tasks: For the aperiodic tasks, we need to place them into a queue. A slot value is associated with the enqueued aperiodic task to demonstrate how many additional slots can be allocated to facilitate its processing, while its deadline is still met. In general, for an aperiodic task J_k, it has an associated arrival time α_k, a processing requirement p_k and a hard deadline D_k. Moreover, if the deadline is not hard, we set $D_k = \infty$ and need to minimize the response time. The aperiodic tasks can be indexed such that $0 \leq \alpha_k \leq \alpha_{k+1}, k \geq 1$. A cumulative aperiodic workload process, $W(t) = \sum_{k|\alpha_k \leq t} p_k$, accumulates all the aperiodic work that arrives in the interval $[0,t]$.

Since periodic tasks generally require static and pre-defined scheduling, we mainly present the schemes for soft and hard aperiodics respectively for scheduling dynamic and retransmitted segments.

B. Soft Aperiodics Tasks for Scheduling Dynamic Segments

We schedule dynamic segments as soft aperiodic tasks. Specifically, at time t, we need to determine the largest amount of aperiodic processing at priority level $k, 1 \leq k \leq n$, when there exist n priorities. We add the aperiodic processing capacity to the system workload, while not causing any deadlines of any periodic tasks to be missed. For example, if a lower priority level is chosen, the task may be delayed by higher-priority periodic processing. In general, aperiodic processing at priority level k generally does not delay periodic tasks with priority levels $k-1$ or higher. We need to examine the scheduling latency upon the periodic tasks with the priority k through n.

In the FlexRay network, we describe the scheduling scheme by using the following representation. Specifically, the priority level i is defined as $k \leq i \leq n$. $C_i(t)$ denotes the cumulative aperiodic processing consumed during $[0,t]$ at level i or higher. $I_i(t)$ is the processing of level i inactivity during $[0,t]$ for $1 \leq i \leq n$ and $t \geq 0$. $r_i(t)$ is the number of jobs of τ_i completed by time t. We use A_{ij} to denote the total aperiodic processing available in an interval. C_i denotes aperiodic processing time already consumed. At time t, the task $\tau_{ir_i(t)}$ has been completed and however the task $\tau_{i(r_i(t)+1)}$ has not. During $[0,t]$, we use $A_{i(r_i(t)+1)}$ to represent aperiodic processing units that can be done at level k without causing any miss of deadlines at level i.

At time t, we add an aperiodic task with priority i of size $S_{i,t}$ without missing the deadlines of τ_i. $S_{i,t}$ represents the amount of slacks available for aperiodic processing at priority level i or higher, and is defined as $S_{i,t} = A_{i(r_i(t)+1)} - C_i(t) - I_i(t)$.

In order to meet the needs of all deadlines of periodic tasks with priority level k or lower, we need to minimize S_i over $k \leq i \leq n$. The largest aperiodic task with priority k that can be added to the system at time t is given by $S^\star_k$, where $S^\star_{k,t} = \min_{k \leq i \leq n} S_{i,t}$.

C. Hard Aperiodics Tasks for Retransmitted Segments

In order to efficiently schedule the retransmitted segments that are used for improving the reliability, we need to determine whether there exists sufficient time available during the interval between the arrival time and the completion deadline. In the meantime, all the guaranteed tasks, including periodics and previously guaranteed but not yet completed aperiodics, needs to meet their deadlines. Formally, for an aperiodic task J_k, its arrival time, processing requirement, and deadline are respectively α_k, p_k, and D_k. We need to determine an appropriate priority level to process the task.

In order to compute the soft deadline aperiodics, we need to take into account the previously guaranteed aperiodics and the processing in a series of intervals, rather than a single interval. Hence, we compute the total aperiodic processing available at priority level i during $[\alpha_k, \alpha_k + D_k]$. A new aperiodic task arrives at a time in which there are no pending aperiodic tasks that have been guaranteed but have not yet completed.

In practice, in the interval $[t_a, t_b]$ ($t_a = \alpha_k$ and $t_b = \alpha_k + D_k$), we compute the maximum amount of slacks that are available for aperiodic processing at the highest priority level. In order to maintain the history information and facilitate the computation, we use an aperiodic processing time accumulator, θ, that has an initial value of 0. At time t_a, we compute the slack that is immediately available, say $\theta^\star$. We then add $min\{\theta^\star, t_b - t_a\}$ to θ. We leverage the minimization to guarantee that aperiodic processing time $\theta^\star$ does not exceed the time available to execute that processing. The slack is immediately used since aperiodic tasks are serviced at the highest priority.

Aperiodic processing will not occur until the job at the lowest priority level without slacks. For example, τ_L, at priority level L, completes. At its worst-case finishing time, $F_L = t_\beta$, this task will complete. Based on the analysis, we argue that there is no slack available for aperiodic processing in the interval $[t_a + \theta^\star, t_\beta]$. Hence, it is not necessary to recompute the slack $\theta^\star$ at any other time t in the interval $[t_a, t_\beta]$. Furthermore, due to the processing activity in the interval $[t_a, t_\beta]$, performing the computation of the slack available at time t_β becomes difficult. In order to obtain correct results, we need to update the variables that have been changed in terms of task slack values and level i inactivity. We hence consider every task with priority L or higher, which is either active at time t_a or arrives during $[t_a, t_\beta]$. Each of these tasks needs to be completed along with τ_L by the time no later than t_β.

The FlexRay schedules hard aperiodics by executing tasks τ_k. This task contains the level i inactivity to be accumulated at higher priority levels. In order to compute level i inactivity, we need to determine the total execution time in each priority level. In fact, there exist multiple jobs of one task that executes during the interval. The slack, that is obtained when each task completes, can be added to the slack for its corresponding priority level. The execution time moves to t_β when the appropriate computation of the slack and level i inactivity at each level i complete. The same computation can be used for the interval $[t_\beta, t_b]$, in which the computation process

continues until $t_\beta \geq t_b$. We finally obtain the available aperiodic processing time in $[t_a, t_b]$.

D. The Dual-channel Design for FlexRay Segments

In order to improve transmission reliability, FlexRay specification [2] leverages a dual-channel design. This design supports flexible choices to transmit the static and dynamic segments. The scheduling on static segments occurs in each network node. We need to respectively maintain a slot counter variable $SlotCounter(A)$ for channel A and a slot counter variable $SlotCounter(B)$ for channel B to schedule segments in channels A and B. These slot counters have the initial value of 1 at the beginning of each communication cycle. FlexRay further increases the slot values until the end of each communication slot in scheduling the segments.

The lengths of dynamic segments are important to deliver high performance in transmitting message of practical FlexRay networks. FlexRay uses the number of "minislots" to represent the length of the dynamic segment. The number of "minislots", in fact, is equal to $gNumberOfMinislots$. When transmitting dynamic segments, their lengths in the corresponding slot are generally small. Moreover, if there are messages to be sent during a slot, the transmission length in the dynamic slot is equal to the number of minislots, thus transmitting the whole message in an efficient manner.

The efficiency of transmitting dynamic messages relies on the network configuration and states. In a communication cycle, in order to initialize the network configurations, FlexRay needs to reset the counters of slots and minislots in the communication controller of a node. After examining exist messages to be transmitted in the controller, FlexRay allows to configure them into the frames. Furthermore, when there are sufficient idle slots before completing the dynamic segment, FlexRay can transmit the selected messages in the dynamic segment of the bus cycle.

FlexRay makes use of parameter management in the real implementations to schedule the segments. If the dynamic slot counter is equal to the value of the Frame ID of the transmitted message, a parameter $pLatestTx$ is used. FlexRay compares the current value of the minislot counter with $pLatestTx$. Each network node uses a fixed value $pLatestTx$ that is tightly associated with the size of the largest dynamic frame.

E. Fault Model and Probability Analysis

FlexRay networks leverage signals as elementary communication units from one ECU to another in automotive applications. These automotive signals can be packed together into frames. FlexRay further transmits these frames via the communication bus. In practice, the frames on the bus are possible to lose due to the potential transient faults. Electronic devices hence become increasingly vulnerable.

Automotive industry proposes an international standard (IEC61508 [21], [22]) for functional safety of electronic safety-related systems, which has been well-recognized and widely used as the fault model in the research community [4], [7], [9], [23]. The standard consists of multiple levels of system reliability. For each level, the standard specifies the probability of system level failure in a time unit, u. Furthermore, we leverage γ to determine the maximum probability of a system failure. Given γ, we define $\rho = 1 - \gamma$ as the reliability goal.

The design goal, in essence, represents quantitative measure with respect to transient faults. Each message $M_z (1 \leq z \leq N)$ has the failure probability, p_z. In order to compute this failure probability, we consider the message's size, W_z, and leverage existing fault-injection tools, such as Vector [24] and Elektrobit [25] that can be used to compute Bit Error Rates (BER). For a BER, p_z can be computed as $p_z = 1-(1-BER)^{W_z}$. Given the reliability goal ρ over a time unit u, we aim to derive the probability that all messages can be successfully transmitted.

Theorem 1: **The Probability of Successful Transmission.** Given a time unit u, the probability that all messages' deadlines are met is $\prod_{z=1}^{N}(1-p_z^{k_z+1})^{\frac{u}{T_z}}$. Each message has the retransmission number, k_z, and the failure probability, p_z.

Proof: In a FlexRay network, the transmission failure means that one instance of a message M_z fails to transmit in the first transmission and the following k_z retransmissions. This probability is $p_z^{k_z+1}$. Moreover, the probability that the message M_z has at least one transmission without faults is $1-p_z^{k_z+1}$. When extending the analysis from one instance to the entire time unit u, the message M_z occurs in a period T_z for $\frac{u}{T_z}$ times. Hence, for all instances of the message M_z, the probability that at least one transmission without faults over time interval u is $(1-p_z^{k_z+1})^{\frac{u}{T_z}}$. This is actually the probability of successful transmission of message M_z. Finally, when considering all messages and their instances, the successful probability is $\prod_{z=1}^{N}(1-p_z^{k_z+1})^{\frac{u}{T_z}}$. ■

In a practical FlexRay network, the system reliability goal ρ needs to be satisfied, i.e., $\prod_{z=1}^{N}(1-p_z^{k_z+1})^{\frac{u}{T_z}} \geq \rho$. In order to achieve this goal, CoEfficient needs to select the retransmitted frames by computing the successful transmission probability. In the meantime, our cooperative scheduling, that uses the slack stealing, needs to judiciously choose the slacks whose lengths match the sizes of the retransmitted frames.

F. Reliability-aware Slack Computation

The dual-channel structure in a FlexRay network offers the opportunity to significantly improve network transmission reliability. However, the dual channels, in the meantime, need to consume more network resources. A suitable tradeoff between reliability and system optimization becomes more important in a practical FlexRay network. In this paper, we make use of reliability-aware slack stealing scheme to improve network resource utilization without any loss of system reliability.

Conventional slack stealing algorithms [26], [27] have the salient property of optimizing bandwidth utilization and reducing transmission latency. The idea is to allocate the same priority to periodic tasks. The slack stealing can hence meet the needs of satisfying all periodic task deadlines and dealing with the aperiodic tasks in the FIFO order.

In order to accurately and efficiently support cooperative scheduling on the static and dynamic segments, we need to determine the maximum amount of stolen slacks without violating hard timing constraints. CoEfficient hence constructs a slot stealer that uses fixed priority preemptive dispatcher to

meet the deadlines of hard periodics and minimize the response time of soft aperiodic tasks.

In order to facilitate the computation of selective slacks, CoEfficient handles the hard periodic tasks by examining the selective slacks between the deadlines on calling a task and the next. We further use a table to store and maintain the identified values. A set of counters can be helpful to keep track of the selective slacks, which have larger probability to be used at different priority levels. By using this table, CoEfficient decreases the values of these counters that can be executed or updated. We identify the maximum amount of processing time from calling a hard deadline task in order to guarantee the reliability.

In the context of CoEfficient design, $l_{i,t}$ is the time when task i was last released, and $x_{i,t}$ is the earliest possible next release of task i, $x_{i,t} = l_{i,t} + T_i$. $d_{i,t}$ is the next deadline on calling task i, and $c_{i,t}$ is the remaining execution time for currently calling task i. In practice, we define that $l_{i,t}, x_{i,t}$ and $d_{i,t}$ are all measured relative to time t. When task i completes, we have $d_{i,t} = x_{i,t} + D_i$. In fact, $d_{i,t}$ is the deadline following the next release. In addition, we obtain the value of $c_{i,t}$ by subtracting the execution time used from the worst case execution time, C_i. We obtain $c_{i,t} = 0$ if task i completes at time t.

Reliability-aware slack computation needs to identify $S_{i,t}^{max}$ that is the maximum amount of selective slack time in the interval $[t, t + d_{i,t})$. This slack has the larger probability to be stolen at priority level i. In the meantime, task i should meet its deadline. In order to efficiently compute the maximum slack time, $S_{i,t}^{max}$, we examine the interval $[t, t + d_{i,t})$ by studying a number of level i busy and idle periods. For a level i busy period, it is a continuous time interval and we can place one or more tasks of priority level i or higher in the execution queue. On the other hand, a level i idle period is a time interval. The corresponding execution queue is free of level i or higher priority tasks.

CoEfficient can obtain the level i idle time between the completion of task i and its deadline for task i computation without causing the deadline to be missed. Furthermore, since $S_{i,t}^{max}$ is equal to the overall level i idle time in the interval, we can compute the maximum slack. We describe the main parameters in Table I.

TABLE I. PARAMETERS IN THE SCHEDULING SCHEME.

Parameters	Description
p_z	Failure probability of message M_z
W_z	The size of message M_z
k_z	Retransmission number
u	Entire time unit
$d_{i,t}$	The next deadline on calling task i
$S_{i,t}$	The level i slot processing released at t
$w_{i,t}$	The length of a level i busy period from t

In order to achieve the reliability goal, we need to compute the retransmission number k_z by using the known parameters. This scheme works for the retransmitted segments, when $k_z \geq 1$. To drive the recurrence to continue, a parameter ε is used to represent the granularity of time. Specifically, we use $S_{i,t}$ to denote the selective slack that is possible to be stolen. Its initial value is zero. Performing the computation obtains the end of a busy period in the interval $[t, t + d_{i,t})$. Moreover, we increase the selective slack processing, $S_{i,t}$, by using the amount of idle time found in the last step. We can obtain the maximum selective slack, represented as $S_{i,t}$, once task i has reached its deadline. Otherwise, the above operations need to be repeated.

IV. PERFORMANCE EVALUATION

In this section, we show the experimental results of our proposed CoEfficient running on multiple datasets.

A. Experiments Configuration

We perform the experiments in 10 FlexRay nodes that are connected to a bus analysis tool. This tool can help record the information of message transmission. We implement and configure the FlexRay nodes by using multiple networked boards. These boards consist of a 16-bit Flash-based controller unit to support the FlexRay protocol operations. We also make use of FlexRay-enabled transceivers to support the physical layer of the FlexRay bus. To order to examine the real-time transmission performance, an independent module is used to receive and maintain all messages that are transmitted on the FlexRay bus.

In order to comprehensively evaluate the performance, we configure the communication cycles that contain both static and dynamic segments. FlexRay partitions available bus bandwidth into multiple communication cycles in a time-triggered manner. The communication cycle, which is periodically repeated, is an instance of FlexRay communication structure.

For static segments, the datasets consist of synthetic test cases and two real-world scenarios. The synthetic test cases are generated by randomly changing message parameters, such as periods and deadlines, to represent various possible scenarios. Specifically, we define the periods varying from 5ms to 50ms, and the deadlines varying from 1ms to 20ms. By considering the experiences from the real-world industry [15], we set the FlexRay communication cycle period to be 5ms and the static cycle length to be 3ms. Moreover, we use two real-world applications, i.e., Brake-By-Wire (BBW) and Adaptive Cruise Controller (ACC). Tables II and III respectively show the details of the used parameters.

For dynamic segments, we define the values of the used parameters and add the dynamic segments into the communication cycles. Our experiments make use of a suitable timing property in terms of aperiodic messages by studying a message set from Society for Automotive Engineers [28]. We hence set aperiodic messages to be a period and a deadline to be 50ms. Moreover, we use 30 aperiodic messages with the IDs 81 to 110 or 121 to 150, respectively corresponding to the number of 80 and 120 slots.

The configuration parameters for dynamic segments can be summarized as follows. The values of gdMacrotick $[\mu s]$, gdMinislot(macrotick), gdSymbolWindow(macrotick), gdDynamicSlotIdlePhase(minislot) and gdMinislotActionPointOffset(macrotick) are respectively 1, 8, 0, 1 and 2. The gNumberOfStaticSlots(macrotick) are 80 and 120. The values of gdCycle $[\mu s]$, gdStaticSlot(macrotick) and gdMacroPerCycle are 5000, 40 and 5000.

TABLE II. BRAKE-BY-WIRE MESSAGE PARAMETERS.

Message	Offset (ms)	Period (ms)	Deadline (ms)	Size (bits)
1	0.28	8	8	1292
2	0.76	8	8	285
3	0.58	1	1	1574
4	0.72	1	1	552
5	0.87	1	1	348
6	0.92	1	1	469
7	0.34	1	1	1184
8	0.28	8	8	875
9	0.75	8	8	759
10	0.52	8	8	932
11	0.95	8	8	1261
12	0.62	8	8	633
13	0.72	8	8	452
14	0.85	8	8	342
15	0.91	8	8	856
16	0.47	8	8	1578
17	0.56	1	1	1742
18	0.58	1	1	553
19	0.92	1	1	1172
20	0.68	1	1	878

TABLE III. ADAPTIVE CRUISE CONTROLLER MESSAGE PARAMETERS.

Message	Offset (ms)	Period (ms)	Deadline (ms)	Size (bits)
1	0.42	16	16	1024
2	0.62	16	16	1024
3	0.58	16	16	1024
4	0.25	16	16	1024
5	0.39	16	16	1024
6	0.48	24	24	1024
7	0.22	24	24	1024
8	0.51	24	24	1024
9	0.32	24	24	1024
10	0.47	24	24	1024
11	0.65	24	24	1024
12	0.42	24	24	1024
13	0.31	32	32	1280
14	0.56	32	32	1280
15	0.48	32	32	1280
16	0.32	32	32	256
17	0.66	32	32	256
18	0.42	32	32	256
19	0.26	32	32	1280
20	0.35	32	32	256

The experiments uniformly distribute the aperiodic messages into 10 FlexRay nodes. In each network node, the aperiodic messages are generated by using an interrupt-based routine running as the host process. One a 16-bit reload timer is used to count down the time. Moreover, we use $gNumberOfMinislots$ to determine the minimum length of dynamic segment. We evaluate the performance of transmitting dynamic segments by using 50 and 100 minislots. The value of parameter $gNumberOfMiniSlots$ can be adjusted to show the changes of the length of dynamic segment. Moreover, we examine the performance for $BER = 10^{-7}$ and $BER = 10^{-9}$, which correspond to different reliability goals. The communication cycle is 1ms and the static segment timing length is 0.75ms, based on industrial and academic experiences [2], [4], [6], [9], [23], [29].

B. Evaluation Results

We compare CoEfficient with the standard implementation of FlexRay specification (FSPEC) [2], in terms of overall running time, bandwidth utilization, average transmission latency for static and dynamic segments, and deadline miss ratios.

1) Running Time: We examine the running time of Brake-By-Wire (BBW), Adaptive Cruise Controller (ACC) scenarios and synthetic test cases under $BER = 10^{-7}$ and $BER = 10^{-9}$, which are respectively shown in Figures 1 and 2. These results demonstrate the average running time with respect to the increasing number of messages. Specifically, BBW and ACC scenarios describe relatively light overhead as shown in Figure 1(a) and 2(a).

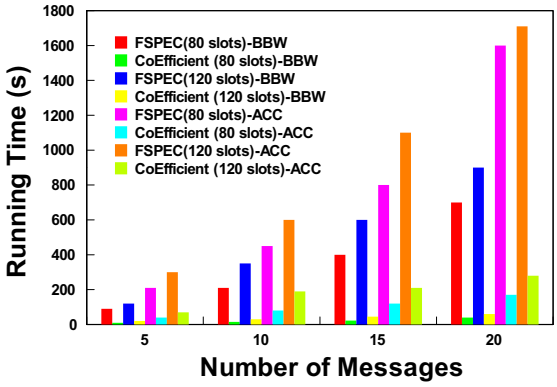
When $BER = 10^{-7}$, the proposed CoEfficient scheme completes the message transmission within 76.2 seconds (for 80 slots) or 92.3 seconds(for 120 slots), which are much smaller than 1670 or 1910 seconds of the standard FSPEC. The reason is that CoEfficient can efficiently schedule both static and dynamic segments in a dual-channel manner.

When $BER = 10^{-9}$, we obtain the similar observations as shown in Figure 2. In order to offer higher reliability, the number of retransmitted segments increases and hence the overall transmission delays are larger, compared with $BER = 10^{-7}$. Moreover, since the 120-slot data incur more idle slots and decrease the bandwidth utilization, the running time for 120 slots is larger than that for 80 slots.

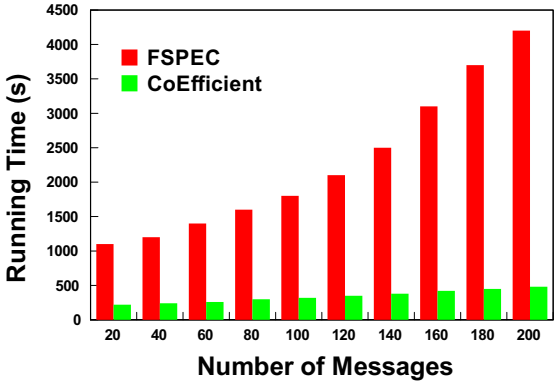
The scalability of the proposed CoEfficient scheme is important to offer efficient and reliable transmission service in the FlexRay network. The cooperative scheduling upon synthetic test is examined due to its larger-scale message set, as shown in Figure 1(b). Compared with FSPEC, CoEfficient significant decreases the latency. We obtain the similar observations in Figure 2(b) when $BER = 10^{-9}$. These experimental results demonstrate that the running time of CoEfficient is much smaller than that of the standard FSPEC.

2) Bandwidth Utilization: The metric of bandwidth utilization describes the ratio of the bandwidth that is actually used to the whole bandwidth. CoEfficient provides reliable and efficient scheduling services and obtains significant performance improvements upon bandwidth utilization with the aid of the selective slack stealing technique. Figure 3 shows the bandwidth utilization of CoEfficient and FSPEC from 25 to 100 minislots. We observe that CoEfficient improves 56.2%, 55.3%, 53.8% and 52.2% bandwidth utilization over the standard FSPEC, respectively in 25, 50, 75 and 100 minislots. CoEfficient can therefore significantly improve the bandwidth utilization.

3) Transmission Latency: In the context of FlexRay scheduling, we compute the transmission latency from the generation time to the ending time. Figure 4 shows the average transmission latency of both static and dynamic segments (50 and 100 minislots) under $BER = 10^{-7}$ and $BER = 10^{-9}$. Specifically, we examine the transmission latency respectively in the synthetic cases, BBW and ACC scenarios. First, Figure 4(a) shows the latency of transmitting static segments (from 1 to 80 ID) in synthetic test cases. Although CoEfficient and FSPEC attempt to provide hard transmission guarantee to static segments, FSPEC incurs larger transmission latency than CoEfficient. The delays of FSPEC are on average 8.2ms and 5.8ms respectively in 50 and 100 minislots when $BER = 10^{-7}$, and those of CoEfficient are 4.7ms and 3.8ms. When $BER = 10^{-9}$, the delays of FSPEC are on average 12.9ms and 10.7ms respectively in 50 and 100 minislots, and those of CoEfficient are 9.6ms and 7.8ms. The main reason is that FSPEC makes use of best-effort retransmission for all segments, which often fail in the limited bandwidth. Unlike FSPEC, CoEfficient

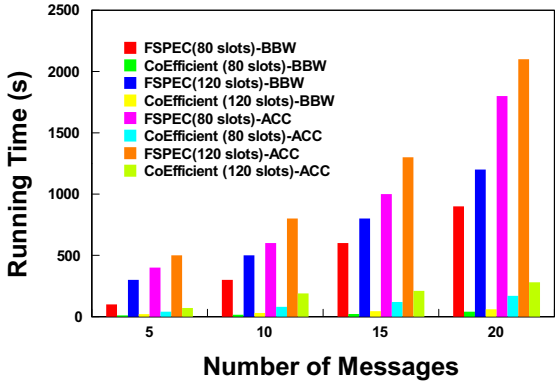


(a) BBW and ACC messages.

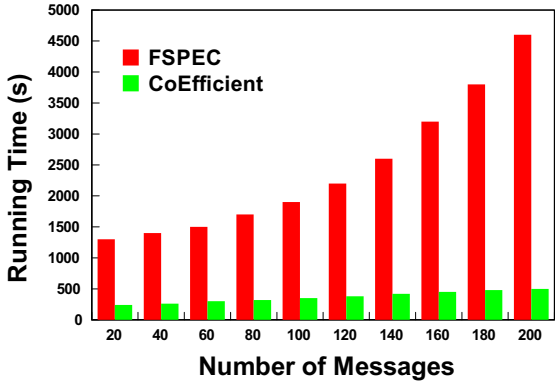


(b) Synthetic test cases.

Fig. 1. Running time for real-world application and synthetic test cases when $BER = 10^{-7}$.



(a) BBW and ACC messages.



(b) Synthetic test cases.

Fig. 2. Running time for real-world application and synthetic test cases when $BER = 10^{-9}$.

achieves the reliability goal by leveraging statistical analysis to execute selective transmission, while efficiently improving the bandwidth utilization. In the meantime, we obtain the similar observations from the delays of BBW and ACC as shown in Figure 4(b).

For dynamic segments, Figure 4(c) shows the transmission latency of the synthetic test cases. When $BER = 10^{-7}$, we observe that CoEfficient produces on average 67.5% and 59.3% smaller latencies than the standard FSPEC, respectively in 50 and 100 minislots. In the BBW and ACC, those are 51.6% and 42.5%. When $BER = 10^{-9}$, CoEfficient produces on average 43.2% and 38.7% smaller latencies than the standard FSPEC, respectively in 50 and 100 minislots. In BBW and ACC, those are 33.6% and 30.1%. By leveraging the selective slack stealing, CoEfficient not only obtains significant performance improvements in terms of transmission latency, but also offers real-time transmission for the dynamic segments.

4) Deadline Miss Ratios: The metric of deadline miss ratio describes the number of missing-deadline messages divided by the total number of the transmitted messages. Figure 5 demonstrates the deadline miss ratio when taking into account the minislots from 25 to 100. Since CoEfficient significantly reduces the transmission delay and improves the bandwidth utilization, the average ratios of missed messages of CoEfficient are 4.8% when $BER = 10^{-7}$ and 3.2% when $BER = 10^{-9}$, while those are respectively 21.3% and 19.5% in the FSPEC scheme.

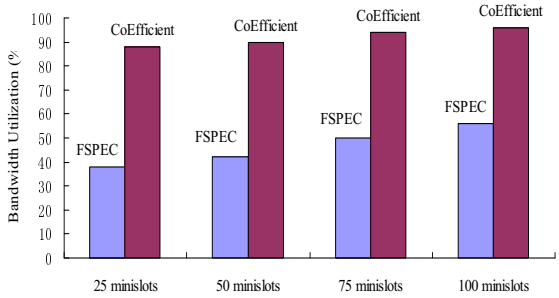


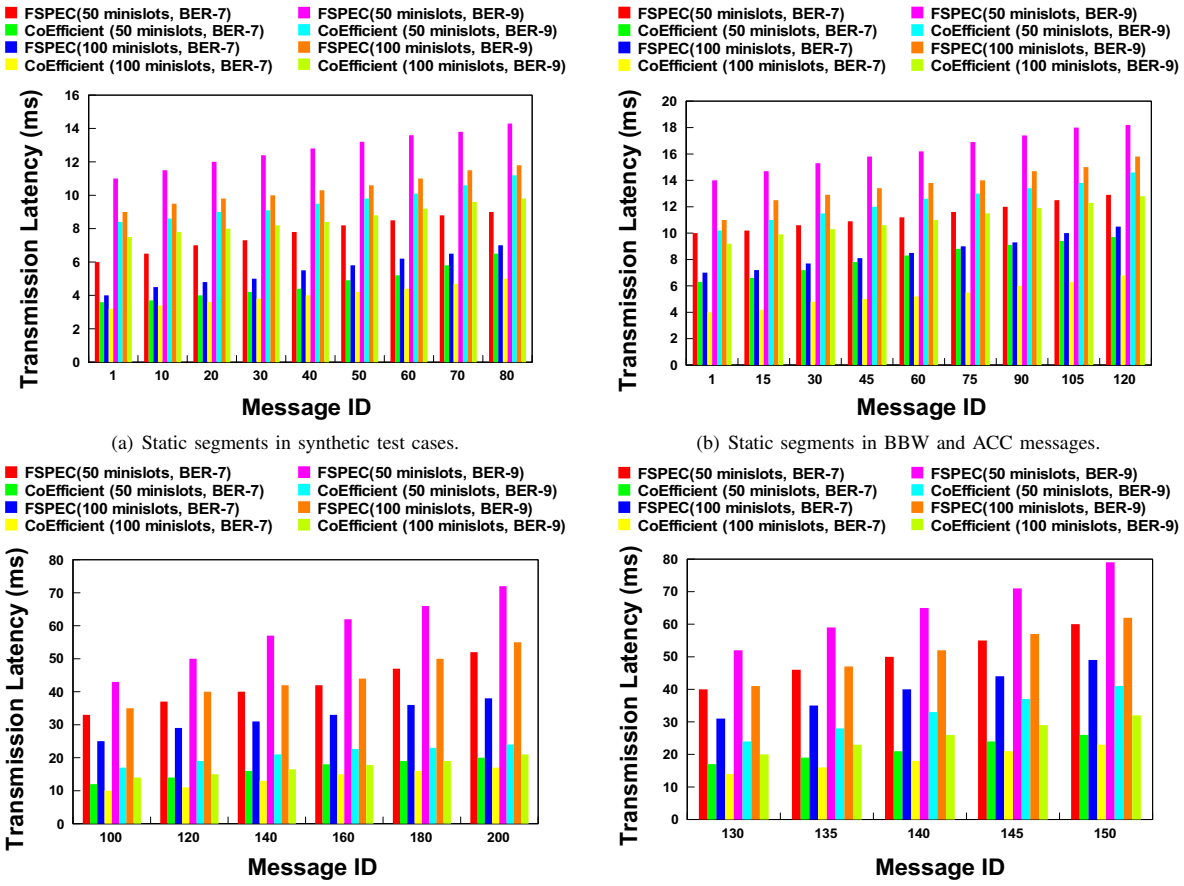
Fig. 3. Bandwidth utilization.

V. Related Work

We categorize related work of FlexRay into protocol improvements, segments transmission efficiency and fault-tolerance management.

A. Protocol Improvements

For protocol improvements, Controller Area Network (CAN) [1] provides the bounded delay communication at data rates between 125kb/s and 1Mb/s. However, it is not suitable for FlexRay that is hard real time in essence and requires high-speed, robust, and predictable communication. Recent attempts to meet these demands include Time-Triggered CAN (TTCAN [30]), Time-Triggered Protocol (TTP [19]), and Byte-Flight [20]. TTCAN and TTP are time-triggered technology



(a) Static segments in synthetic test cases.

(b) Static segments in BBW and ACC messages.

(c) Dynamic segments in synthetic test cases.

(d) Dynamic segments in BBW and ACC messages.

Fig. 4. Average transmission latency of both static and dynamic segments, when dynamic segments respectively have 50 and 100 minislots. For simplicity, we use $BER-7$ and $BER-9$ to respectively denote $BER=10^{-7}$ and $BER=10^{-9}$.

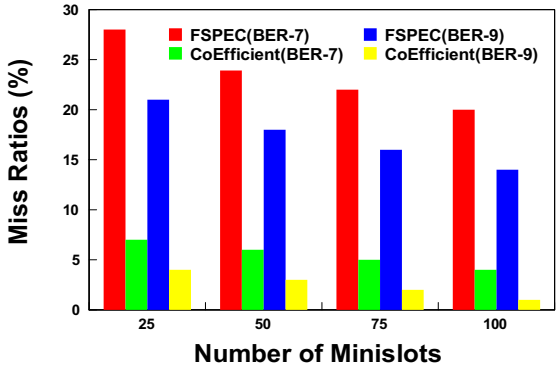


Fig. 5. Deadline miss ratio. For simplicity, we use $BER-7$ and $BER-9$ to respectively denote $BER=10^{-7}$ and $BER=10^{-9}$.

with predictable medium access, while ByteFlight is based on FTDMA.

B. Segments Transmission Efficiency

For segments transmission, existing work mainly considers the schemes for scheduling either static segments or dynamic segments. A frame packing algorithm [31] can efficiently minimize bandwidth consumption in the dynamic segments by packing different signals into a message frame. Scheduling static segments [14] and dynamic segments [16] can be separately formulated into nonlinear integer programming problem to maximize their own bandwidth utilization. Moreover, slot multiplexing [10] offers schedulability analysis for the dynamic segments of FlexRay. Based on mixed-integer linear programming, an optimization framework [3] is proposed to schedule the transactions that consist of tasks and signals in a FlexRay-based system. A schedulability analysis [32] determines the timing properties of transmitted messages. Furthermore, although HOSA [7] uses the dual channel for data transmission, its best-effort retransmission consumes substantial bandwidth to support fault tolerance.

C. Fault-tolerance Management

For fault-tolerance management, optimizing bandwidth utilization [4] is formulated into constraint logic programming to support fault-tolerant schedule in the presence of transient and intermittent faults. A systematic probabilistic analysis is used to provide formal guarantee on desired reliability levels. However, this work only considers the static segments of FlexRay. Moreover, the problem of scheduling FlexRay segments is formulated as a mixed integer linear programming algorithm [23]. Its design goal however is to retransmit as

many faulty messages as possible, which can not offer reliability guarantee due to choosing the retransmitted messages in an ad-hoc manner. The retransmission may improve the reliability with extra loads in the transmission bandwidth and additional communication latency. In its optimizing application-level acknowledgement and retransmission scheme, transmission time is allocated on top of an existing schedule. Unlike them, CoEfficient provides cooperative scheduling upon both static and dynamic segments, while using real-world fault models and providing transmission reliability.

VI. Conclusion

The reliability of transmitting messages is important to the efficiency and performance in the FlexRay networks since the rapid growth of hybrid segments need to be transmitted in an efficient and reliable manner. While most existing work fails to efficiently offer scalable fault tolerance, our work identifies this important problem to meet the needs of reliability and deliver high performance. In order to meet the needs of transmission efficiency and reliability, this paper proposes a cooperative and efficient scheme, called CoEfficient. CoEfficient offers comprehensive solution and delivers high performance. It supports fault-tolerant scheduling on both static and dynamic segments by considering real-world fault models. CoEfficient significantly decreases transmission latency and improves bandwidth utilization and reliability, using the improved slack stealing technique. By the comparisons with state-of-the-art FlexRay based scheduling schemes, extensive experimental results based on real-world test cases demonstrate the efficiency and efficacy of the proposed CoEfficient scheme.

Acknowledgements

This work was supported in part by National Natural Science Foundation of China (NSFC) under Grant 61173043, National Basic Research 973 Program of China under Grant 2011CB302301, NSFC under Grant 61025008, and NSERC Discovery Grant 341823, US National Science Foundation Award 1116606.

References

[1] N. Navet, Y. Song, F. Simonot-Lion, and C. Wilwert, "Trends in automotive communication systems," *Proceedings of the IEEE*, vol. 93, no. 6, pp. 1204–1223, 2005.

[2] "The flexray communication system specification, version 2.1," *http://www.flexray.com*.

[3] H. Zeng, M. Di Natale, A. Ghosal, and A. Sangiovanni-Vincentelli, "Schedule optimization of time-triggered systems communicating over the flexray static segment," *IEEE Transactions on Industrial Informatics*, vol. 7, no. 1, pp. 1–17, 2011.

[4] B. Tanasa, U. D.Bordoloi, P. Eles, and Z. Peng, "Scheduling for fault-tolerant communication on the static segment of flexray," *Proc. IEEE RTSS*, 2010.

[5] Y. Hua and X. Liu, "Scheduling heterogeneous flows with delay-aware deduplication for avionics applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 9, pp. 1790–1802, 2012.

[6] I. Park and M. Sunwoo, "Flexray network parameter optimization method for automotive applications," *IEEE Transactions on Industrial Electronics*, vol. 58, no. 4, pp. 1449 – 1459, 2011.

[7] Y. Hua, X. Liu, and W. He, "HOSA: Holistic Scheduling and Analysis for Scalable Fault-tolerant FlexRay Design," *Proc. INFOCOM*, pp. 1233–1241, 2012.

[8] Y. Sedaghat and S. Miremadi, "Categorizing and analysis of activated faults in the flexray communication controller registers," *Proc. IEEE European Test Symposium,*, pp. 121–126, 2009.

[9] B. Tanasa, U. Dutta Bordoloi, P. Eles, and Z. Peng, "Reliability-aware frame packing for the static segment of flexray," *Proc. ACM International Conference on Embedded Software (EMSOFT)*, 2011.

[10] B. Tanasa, U. Bordoloi, S. Kosuch, P. Eles, and Z. Peng, "Schedulability Analysis for the Dynamic Segment of FlexRay: A Generalization to Slot Multiplexing," *Proc. IEEE RTAS*, pp. 185–194, 2012.

[11] Y. Hua, X. Liu, W. He, and D. Feng, "Design and Implementation of Holistic Scheduling and Efficient Storage for FlexRay," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2014.

[12] H. Kopetz, R. Obermaisser, P. Peti, and N. Suri, "From a federated to an integrated architecture for dependable real-time embedded systems," *Technical Report, TECHNISCHE UNIV VIENNA (AUSTRIA)*, 2004.

[13] A. Albert, "Comparison of event-triggered and time-triggered concepts with regard to distributed control systems," *Embedded World*, vol. 2004, pp. 235–252, 2004.

[14] K. Schmidt and E. Schmidt, "Message scheduling for the flexray protocol: The static segment," *IEEE Transactions on Vehicular Technology*, vol. 58, no. 5, pp. 2170–2179, 2009.

[15] M. Lukasiewycz, M. Glaß, J. Teich, and P. Milbredt, "Flexray schedule optimization of the static segment," *Proc. CODES+ ISSS*, 2009.

[16] E. Schmidt and K. Schmidt, "Message scheduling for the flexray protocol: The dynamic segment," *IEEE Transactions on Vehicular Technology*, vol. 58, no. 5, pp. 2160–2169, 2009.

[17] K. Schmidt, E. G. Schmidt, A. Demirci, E. Yuruklu, and U. Karakaya, "An Experimental Study of the FlexRay Dynamic Segment," *Proc. Advances in Automotive Control*, 2010.

[18] K. Jung, M. Song, D. Lee, and S. Jin, "Priority-based scheduling of dynamic segment in FlexRay network," *Proc. International Conference on Control, Automation and Systems*, 2008.

[19] H. Kopetz and G. Bauer, "The time-triggered architecture," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 112–126, 2003.

[20] J. Berwanger, M. Peller, and R. Griessbach, "A new high performance data bus system for safety-related applications," *BMW AG, EE-221, Munich, Germany, Available: http://www.byteflight.com/specification*, 1999.

[21] I. IEC, "61508 functional safety of electrical/electronic/programmable electronic safety-related systems," *International Electrotechnical Commission*, 1998.

[22] S. Brown, "Overview of iec 61508. design of electrical/electronic/programmable electronic safety-related systems," *Computing & Control Engineering Journal*, vol. 11, no. 1, pp. 6–12, 2000.

[23] W. Li, M. Di Natale, W. Zheng, P. Giusto, A. Sangiovanni-Vincentelli, and S. Seshia, "Optimizations of an application-level protocol for enhanced dependability in flexray," *Proc. DATE*, 2009.

[24] Vector, "[Online]. Available: http://www.vector.com,"

[25] E. A. GmbH, "[Online]. Available: http://www.elektrobit.com,"

[26] R. Davis, K. Tindell, and A. Burns, "Scheduling slack time in fixed priority pre-emptive systems," *Proc. RTSS*, 1993.

[27] S. Thuel and J. Lehoczky, "Algorithms for scheduling hard aperiodic tasks in fixed-priority systems using slack stealing," *Proc. Real-Time Systems Symposium*, pp. 22–33, 1994.

[28] SAE, "Class C Application Requirements, SAE J2056/1," *SAE Handbook, Soc. Automotive Engineers, Warrendale, PA*, vol. 2, pp. 23.366–23.371, June, 1993.

[29] B. brake system relies on FlexRay, "http://www.automotivedesignline.com/news/218501196," July, 2009.

[30] TTCAN, "[Online]. Available: http://www.cancia.org/can/ttcan/," May 2005.

[31] M. Kang, K. Park, J. Choi, and M.-s. Kong, "Minimizing the bandwidth consumption of the flexray dynamic segment," *Proc. IEEE International Conference on Consumer Electronics (ICCE)*, pp. 407–408, 2013.

[32] T. Pop, P. Pop, P. Eles, Z. Peng, and A. Andrei, "Timing analysis of the flexray communication protocol," *Real Time Systems*, vol. 39, no. 1, pp. 205–235, 2008.