HOSA: Holistic Scheduling and Analysis for Scalable Fault-tolerant FlexRay Design

Yu Hua
School of Computer, WNLO
Huazhong University of Science and Technology
Wuhan, China
csyhua@hust.edu.cn

Xue Liu
School of Computer Science
McGill University
Montreal, Quebec, Canada
xueliu@cs.mcgill.ca

Wenbo He
School of Computer Science
McGill University
Montreal, Quebec, Canada
wenbohe@cs.mcgill.ca

***Abstract*—FlexRay is a new industry standard for next-generation communication in automotives. Though there are a few recent researches on performance analysis of FlexRay, two important aspects of the FlexRay design have been overlooked. The first is a holistic integrated scheduling scheme that can handle both static and dynamic segments in a FlexRay network. The second is cost-effective and scalable fault-tolerance. In order to address these aspects, we propose a novel holistic scheduling scheme, called HOSA, which can provide scalable fault tolerance by using flexible and ease-of-use dual channel communication in FlexRay. HOSA is built upon a novel slot pilfering technique to schedule and optimize the available slots in both static and dynamic segments. Moreover, in order to achieve efficient implementation, we propose approximate computation, which can efficiently support cost-effective and holistic scheduling by judiciously obtaining the tradeoff between computation complexity and available pilfered slots. HOSA hence offers two salient features, i.e., providing fault-tolerance and improving bandwidth utilization. Extensive experiments based on synthetic test cases and real-world case studies demonstrate the efficiency and efficacy of HOSA.**

I. Introduction

Modern automobiles are often driven by wire (X-by-wire), including Anti-lock Braking System (ABS), electronic steering, and Electronic Stability Control (ESC) systems. These systems involve 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. FlexRay [1] is an automotive network communications infrastructure developed by the FlexRay Consortium. It has become the *de facto* standard in the automotive industry. FlexRay provides a communication infrastructure for future generation high-speed X-by-wire applications in vehicles. These applications are mostly real-time and safety-critical [2]. FlexRay hence aims to provide hard real-time capabilities through cycle-based and time-triggered communications. The FlexRay standard is being deployed in the major line of new vehicles. For example, the new BMW-7 series are equipped with FlexRay-based brake system [3]. FlexRay provides two channels with a high bandwidth of 10 Mb/s each and offers multiple benefits compared with previous protocols, say Controller Area Network (CAN) [4], across a wide range of automotive applications. These benefits include high speed, fault tolerance, and a deterministic cycle-based message transport, along with a synchronized, common time base to all nodes in the system.

FlexRay is an open standard. It aims to provide scalable, deterministic and high performance communication for automotive applications. However, in order to be practical in automotive products and obtain significant performance improvements, two challenges need to be carefully and efficiently dealt with.

Isolated Scheduling: FlexRay is a real-time system to schedule time-triggered and event-triggered messages. FlexRay supports the transmission of periodic messages in static segments (SS) and priority-based scheduling of event-triggered messages in dynamic segments (DS). Periodic messages are transmitted in the unique static slots of SS according to time division multiple access (TDMA). The operation of the FlexRay SS is similar to the time-triggered protocol (TTP) [5]. Moreover, aperiodic messages are sent in the dynamic slots of DS that is similar to ByteFlight [6] and employs a flexible TDMA (FTDMA) approach. In both cases, the timely message delivery depends on the message schedule that is statically configured before the network starts to operate. The scheduling computation involves assigning the static slots for the periodic messages as well as the priority based dynamic slots assignment for the aperiodic messages.

Most existing work, however, only considers the scheduling for either static segments [2], [7]–[9] or dynamic segments [10]–[12]. This isolated scheduling severely limits the performance in terms of bandwidth utilization and transmission latency.

Limited Fault Tolerance: In FlexRay networks, faults may be frequent and ubiquitous due to radiation, interference and temperature variation. Such faults can be classified into permanent and transient faults [13]. Permanent faults are usually caused by physical damages and lead to long-term malfunctioning. Transient faults usually result in the miscalculations in the logic and data corruption and last for a short duration. *X-by-wire* automotive applications are safety-critical. They require data integrity even with the occurrence of transient faults. Moreover, with the increasing numbers of rich electronic devices in cars (e.g., around 2500 signals are exchanged among 70 ECUs of luxury cars [4], [14]), handling transient faults demands efficient fault-tolerant techniques to improve the system reliability.

Unfortunately, existing work fails to efficiently address the

978-1-4673-0775-8/12/$31.00 ©2012 IEEE

above challenges. Specifically, although FlexRay has been widely used as an in-vehicle communication network, its applicability is severely hindered in high-speed safety-critical X-by-wire systems [15]. FlexRay does not provide acknowledgement or re-transmission schemes and hence there is limited guarantee on message delivery for reliability. Moreover, the authors in [16] formulated the scheduling problem as a mixed integer linear programming algorithm, but its design goal was to re-transmit as many faulty messages as possible, which may fail to offer reliability guarantee due to that the re-transmitted messages are chosen in an ad-hoc manner. The re-transmission of faulty messages can improve the reliability with extra loads in the bandwidth and additional transmission latency. Recently, authors in [8] uses systematic probabilistic analysis to provide formal guarantee on desired reliability levels. However, this work only considers the static segments of FlexRay.

In order to address the above challenges, we propose a novel holistic scheduling scheme, called HOSA. HOSA considers both SS and DS in the holistic scheduling design and can support scalable fault tolerance and improve bandwidth utilization. Specifically, we make the following contributions.

Holistic Scheduling. FlexRay is a high-bandwidth communication protocol with a cyclic operation. Each FlexRay cycle consists of a static segment and a dynamic segment. The former is designed for the periodic transmission of real-time data, while the latter supports the transmission of low-priority data and event-triggered (aperiodic) real-time data. HOSA employs a novel holistic scheme to schedule both static segments and dynamic segments in a unified manner. Fast and accurate slot computation allows HOSA to identify available static slots that can be pilfered by the task that transmits dynamic messages. Idle slots are hence minimized and HOSA achieves high bandwidth utilization.

Scalable Fault Tolerance. Scalable fault tolerance refers to the ability of the FlexRay protocol to operate in the configurations that provide various degrees of fault tolerance. HOSA is compliant with existing schemes for scalable fault tolerance, and focuses on flexibly scheduling dual channel communication and efficiently optimizing bandwidth utilization. HOSA implements this through the design of a novel slot pilfering technique. It further leverages approximation computation to significantly reduce the complexity with slight impact on the available pilfered slots.

System Implementation. In order to examine the performance of our proposed HOSA scheme in FlexRay networks, we implement HOSA in a prototype testbed. The prototype contains all the mentioned components and functionalities. We use synthetic test cases and real-world case studies from the automotive industry to evaluate the system performance in terms of overall running time, bandwidth utilization, deadline miss ratio and average transmission latency for both static and dynamic segments. Experimental results demonstrate the efficiency and efficacy of HOSA. For instance, compared with the existing industrial FlexRay implementation [1], HOSA obtains about 50% improvements on bandwidth utilization and 61.5% reduction in transmission latency.

The rest of this paper is organized as follows. Section II presents the FlexRay scheduling model. Section III describes the holistic scheduling on both static and dynamic segments. We present the performance evaluation and related work respectively in Section IV and V. Finally, we conclude our paper in Section VI.

II. Scheduling Model

In this Section, we illustrate the architecture of a FlexRay cluster and its communication cycle that contains static and dynamic segments. We further describe the dual-channel based design in HOSA, which supports holistic scheduling on the FlexRay segments.

A. FlexRay Cluster

A FlexRay cluster consists of the network nodes connected by FlexRay communication channels as shown in Figure 1. FlexRay allows a cluster to be flexible configuration of network topology, such as bus, star or hybrid connection. A cluster is a communication system that contains multiple nodes connected via at least one communication channel directly in a bus topology or by star couplers in a star topology. Moreover, each node in a FlexRay cluster consists of a host and a communication controller (CC), which are connected by a controller-host interface (CHI). The host is a part of an Electronic Control Units (ECU) where the application software is executed to handle incoming messages and generates outgoing messages. The communication controller implements the FlexRay protocol services. CHI serves as a buffer between the host and the CC.

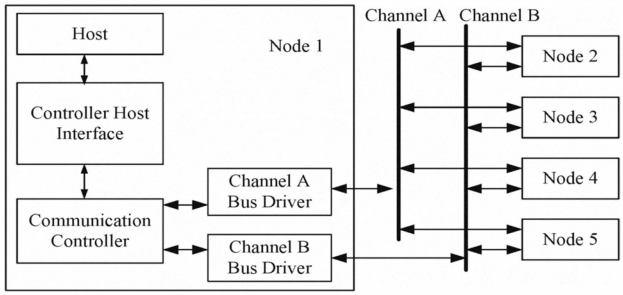


Fig. 1. Illustration of a FlexRay Cluster.

In order to support real-time message communication, a bus driver that has a transmitter and a receiver connects with the communication controller to one communication channel that supports the inter-node connection. The bus driver also maintains clock synchronization with other nodes, constructs and checks cyclic redundancy code verification. The network nodes thus exchange periodic and aperiodic real-time messages that are transmitted in FlexRay communication cycles.

B. Communication Cycle

FlexRay divides available bus bandwidth into multiple communication cycles in a time-triggered manner. The communication cycle is an instance of the communication structure that is periodically repeated. As shown in Figure 2, the communication cycle consists of a static segment (SS), a dynamic

segment (DS), a symbol window (SW), and a network idle time (NIT). The symbol window describes a communication period and in this period, FlexRay allows a symbol to be transmitted on the network. The network idle time describes a communication-free period and contains the remaining number of macroticks, which have not been allocated to the previous three parts (i.e., static segment, dynamic segment, and symbol window).

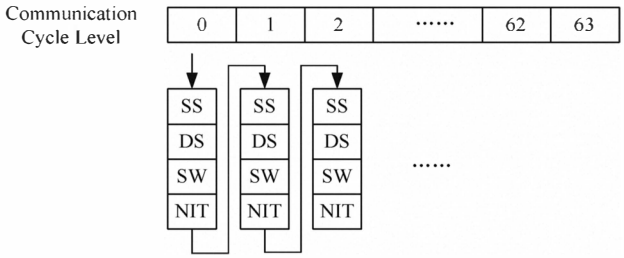


Fig. 2. Communication Cycles.

FlexRay uses static and dynamic segments for message transmission. Specifically, the static segment of the communication cycle supports 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. The used slot has the fixed length and is assigned to a given position in the entire segment. On the other hand, the dynamic segment offers flexible communications, in which message transmission is arbitrated by identifier priority (For example, the lowest identifier messages are transmitted first). Each node only needs to know the time slots for its incoming and outgoing communications. The specification of these time slots is maintained in local scheduling tables. The dynamic property of the slot comes from the fact that its duration may vary according to the length of the transmitted frame. Overall, the time-triggered model of FlexRay can hence provide time determinism for message delivery.

The scheduling on message delivery depends on the management of frame ID. A frame ID indicates the slot in which the frame should be transmitted. A frame ID is used no more than once in each channel in a communication cycle. Each frame to be transmitted in a cluster has a frame ID assigned to it. FlexRay distinguishes the frame IDs between static and dynamic segments.

C. Static and Dynamic Segments

Static and dynamic segments are the structures of message delivery in a FlexRay network. Static segment is a portion of the communication cycle where the media access is controlled via a TDMA scheme. FlexRay can determine the access to the media in a static segment only by the progression of time. Furthermore, the dynamic segment portion of the communication cycle makes use of Flexible Time Division Multiple Access (FTDMA) to schedule the media access via a mini-slotting scheme. The minislot is a time interval of the dynamic segment to support flexible timing configuration. FlexRay then allows the dynamic segment access to the media based on a priority manner for the nodes to transmit data.

Static and dynamic segments demonstrate different formats and functionalities in the communication slots. Specifically, static communication slot is an interval of time. The access to a communication channel is allowed exclusively to a specific node for transmitting a frame with a frame ID that corresponds to the slot. Each static communication slot contains a constant number of *macroticks* regardless of whether or not a frame is sent in the slot. In the static segment, all communication slots are of identical and static configuration.

Furthermore, dynamic communication slot contains one or more *minislots*. The smallest time unit in a DS is the minislot with a duration representation of *gdMinislot*. A DS contains a maximum number of *gNumberOfMinislots* (between 0 and 7986) minislots. Unlike a static communication slot, FlexRay allows the duration of a dynamic communication slot to vary depending on the length of the frame. A variable *vSlotCounter* contains the ID of the current dynamic slot starting from a pre-configured value. In each dynamic slot, a frame with the corresponding ID is transmitted, and hence the duration of the dynamic slot is determined by the length of the transmitted frame. If no frame is sent, the duration of a dynamic communication slot is equal to that of one minislot. In fact, frames are transmitted within dynamic slots that are superimposed on the minislots.

D. Dual Channel Design

Dual channel design in the FlexRay specification [1] offers flexible transmission patterns for the static and dynamic segments. Specifically, for scheduling static segments, each network node maintains a slot counter variable $SlotCounter(A)$ for channel A and a slot counter variable $SlotCounter(B)$ for channel B. Both slot counters are initialized with 1 at the beginning of each communication cycle and further incremented at the end of each communication slot.

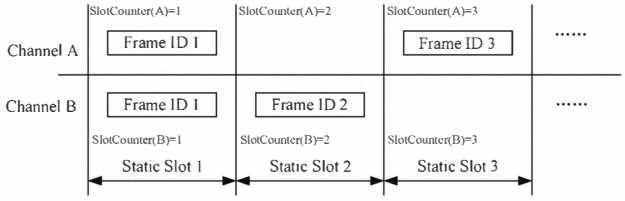


Fig. 3. Scheduling static segments.

Figure 3 illustrates the transmission patterns in a single node that makes use of the static segments. The scheduling on static segments depends upon the operations defined in a schedule table. For example, in slot 1 the node transmits a frame on channel A and a frame on channel B. In slot 2 the node transmits a frame only on channel B. For scheduling dynamic segments, each network node maintains two slot counters, respectively for channels A and B, in scheduling the dynamic segments. Figure 4 illustrates the scheme of scheduling the dynamic segments. Note that although the slot counters for channel A and for channel B are incremented simultaneously within the static segment, their values can be incremented independently according to the dynamic arbitration scheme.

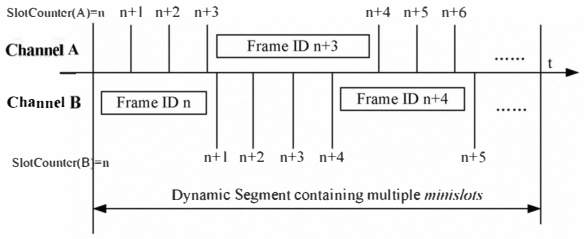


Fig. 4. Scheduling dynamic segments.

III. Holistic Scheduling

This Section presents the holistic scheduling for both static and dynamic segments with the aid of slot pilfering technique.

A. FlexRay Node Architecture

A FlexRay network supports dual-channel communication to offer the guarantee of transmission reliability. Figure 5 shows the node architecture for scheduling static and dynamic segments in the dual channel. Each node uses a *schedule table* to maintain and schedule the messages to be transmitted in the static segments, while using *priority queues* for the dynamic segments. Here, we use capital letter to represent the original messages and lower case for corresponding redundant ones.

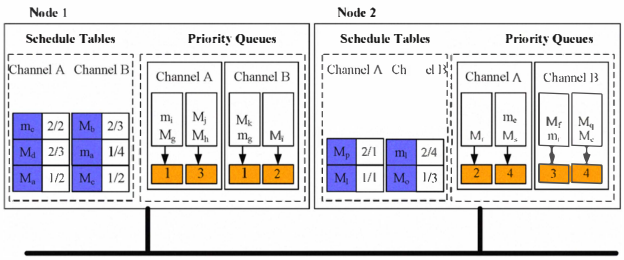


Fig. 5. Node architecture for static and dynamic segments in a dual channel.

Static and dynamic segments have different scheduling schemes to allow a message to be transmitted. First, for scheduling static segments, a message in the schedule table has a timing based sequence, i.e., the number of cycles and slots. For example, the message M_a is transmitted on the second slot of the first cycle, represented as "1/2". On the other hand, for dynamic segments, we allocate the slot number to each node and all messages in each priority queue will be scheduled in the fixed priority way. For example, node 1 sends the messages to slots 1 and 3 of channel *A*. For each of these slots, CHI provides a buffer that can be written by the host and read by the communication controller. At the beginning of each slot, the communication controller needs to read the messages in the buffers so as to facilitate the transmission of frames.

In order to significantly reduce the potential transmission collision and obtain the performance improvement, during any communication slot, FlexRay only allows one node to send messages on the bus. This node needs to transmit the message with the frame ID that is equal to the current value of the slot counter. We set two slot counters that respectively correspond to the static and dynamic segments. In the design phase, we decide and allocate the frame identifiers to nodes. Each node to send messages has one or more static and/or dynamic slots.

For static and dynamic messages, we further leverage different schemes to decide which messages are transmitted during the allocated slots. For static messages, there exists a schedule table with the transmission time in each network node. When transmitting a static message starts, a given message is placed into its associated static buffer in the CHI. For example, static message M_a sent from node 1 has an entry "1/2" in the schedule table specifying that it should be sent in the second slot of the first static cycle.

On the other hand, for scheduling dynamic messages, there is an assumption that Frame ID is specified in advance. For example, as shown in Figure 5, dynamic message M_h has the frame identifier "3". Moreover, FlexRay allows a node to send different messages using the same dynamic Frame ID. For example, messages M_j and M_h on node 1 have both Frame ID 3. If two or more messages with the same frame ID prepare to be sent in the same bus cycle, a priority scheme is used to decide which message will be sent first. By considering the dual-channel transmission, each dynamic message M_i or m_i has their associated priority, say $priority_{M_i}$ or $priority_{m_i}$. Messages with the same Frame ID will be inserted into a local output queue, in which we order them based on their priorities. The message from the head of the priority queue will be sent in the current bus cycle. For example, message M_h will be sent before M_j because it has a higher priority.

In addition, original and redundant messages may be not identical in the receiver node, although this case occurs with very small probability. In this case, the receiver node will require a retransmission.

B. Dual Channel Scheduling

Figure 6 shows the periodic communication that has two cycles of length in channel *A* and *B*. Each cycle contains two time intervals with different access policies (a static and a dynamic segment). They have different lengths that are fixed over the cycles. Moreover, both the static and dynamic segments have multiple slots. In the static segment, FlexRay allows the slots number to be fixed. The length of these slots are constant and equal, regardless of whether static messages are sent or not in that cycle. FlexRay uses the global configuration parameter *gdStaticSlot* to specify the length of a static slot [1]. As shown in Figure 6, there are four static slots for the static segment. Note that a FlexRay cycle generally contains a symbol window and a network idle time. Since they are actually not related with our scheduling analysis, for simplicity, we ignore them in the examples.

The performance in practical FlexRay networks relies on the definition of the dynamic segments' lengths. FlexRay specifies the length of the dynamic segment in the number of "minislots", which is equal to *gNumberOfMinislots*. During the transmission of dynamic segments, if there is no message to be sent during a slot, the length of this slot becomes very small. Otherwise, the dynamic slot offers a transmission length, i.e., the number of minislots, to allow for transmitting the whole message.

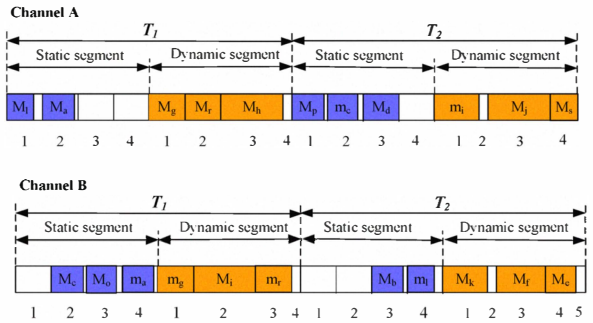


Fig. 6. Dual channel scheduling on static and dynamic segments.

At the beginning of each communication cycle, the communication controller of a node resets the counters of slots and minislots for initialization configuration. Moreover, the controller also needs to check if there exist messages to be transmitted, which will be further organized into the frames. As shown in Figure 6, there exists an assumption that all messages to be transmitted are ready before the first bus cycle. In practice, due to different schemes in scheduling static and dynamic segments, the transmission scenarios would be different. Specifically, static segments use a schedule table to select the messages into static frames to transmit in the bus cycle. For example, messages M_l and M_p are placed into the associated static buffers in the CHI in order to be transmitted in the first bus cycle.

Moreover, transmitting a dynamic message is constrained and conditional. Only if there exist enough idle slots until the end of the dynamic segment, the selected messages can be transmitted during the dynamic segment of the bus cycle. In the real implementations, when the dynamic slot counter reaches the value of the Frame ID of the transmitted message, FlexRay needs to check if the current value of the minislot counter is smaller than a given value $pLatestTx$. For each network node, the value $pLatestTx$ is fixed and depends upon the size of the largest dynamic frame. For example, message M_e prepares for transmission before the first bus cycle starts. However, after message m_r is transmitted, there are not enough slots left in the dynamic segment. This will delay the transmission of M_e for the next bus cycle.

C. Holistic Scheduling Segments

In order to optimize the bandwidth utilization and offer substantial performance improvements, we use holistic scheduling upon the static and dynamic segments in the FlexRay network. Specifically, we consider the transmission of static and dynamic segments respectively as hard deadline periodic and soft deadline aperiodic tasks. The design goal is to *schedule a mixture of periodic and aperiodic tasks in a dual channel to guarantee that all periodic deadlines are met and the response time for the aperiodic tasks can be as small as possible in the FlexRay network*. Holistic scheduling scheme hence offers available time for completing the aperiodic tasks by "*pilfering*" all the processing time from the periodic tasks without causing their deadlines to be missed.

The main idea behind slot pilfering comes from the practical observations and long-term experiences. When an aperiodic request arrives, the slot judiciously pilfers all the available slots from periodic tasks, which are used to satisfy the aperiodic requests. On the other hand, when there are no pending aperiodic requests, we schedule the periodic tasks as usual. We further formulate the slot pilfering technique in a FlexRay network that contains n periodic tasks, $\tau_1, \tau_2, \cdots, \tau_n$.

Definition 1. *Each task, $\tau_i (1 \leq i \leq n)$, is denoted by a 4-tuple $\tau_i = \{C_i, T_i, \phi_i, d_i\}$, where C_i is the worst-case computation requirement, T_i is a period, $\phi_i (0 \leq \phi_i \leq T_i)$ is an offset relative to time origin, and d_i ($d_i \leq T_i$) is a hard deadline. We assume that the parameters C_i, T_i, ϕ_i and d_i, are the known deterministic quantities.*

A fixed priority algorithm, say deadline monotonic algorithm [17], can schedule these tasks. In the meanwhile, the tasks with smaller value of d_i are allocated higher priority.

We leverage differentiated representation for scheduling the tasks of static and dynamic segments. For a periodic task τ_i for the static segment, it leads to an infinite sequence of jobs. We further consider the scheduling on aperiodic tasks for dynamic segments as the problem of parameter optimization. Specifically, we place an aperiodic task for a dynamic segment into the queues based on deadline orders. The slot value, associated with an enqueued aperiodic task for the dynamic segment, demonstrates how many available slots can be allocated to facilitate its processing, while its deadline is still met. In order to achieve this goal, we need to use the value of available slots to offer transmission guarantee. New aperiodic tasks for dynamic segments will not incur its deadline to be violated. At the same time, all periodic deadlines for scheduling static segments are also guaranteed. Therefore, the remaining processing time can be competed between hard and soft aperiodics. We further describe the aperiodic task for scheduling a dynamic segment.

Definition 2. *The aperiodic task J_k for scheduling a dynamic segment is represented as a 3-tuple, $J_k = \{\alpha_k, p_k, D_k\}$, where α_k is the associated arrival time, p_k is the processing requirement and D_k is the hard deadline. In order to support the retrieval of aperiodic tasks, HOSA defines that $0 \leq \alpha_k \leq \alpha_{k+1}, k \geq 1$.*

Based on the above definitions, HOSA aims to minimize the response time of J_k, represented as R_k. Specifically, we consider $W(t) = \sum_{k|\alpha_k \leq t} p_k$ as a cumulative aperiodic workload process. This process collects all the aperiodic tasks. These tasks share the same property of the arrival time within the interval $[0,t]$. Moreover, a cumulative aperiodic execution process, ε_t, is a continuous function with the property of $\varepsilon_t \leq W(t), t \geq 0$. HOSA thus describes the completion time of J_k, as $T_k = min\{t | \varepsilon_t = \sum_{i=1}^{k} p_i\}$. Therefore, HOSA achieves $R_k = T_k - \alpha_k$.

In order to efficiently support the holistic scheduling on the static and dynamic segments in the dual channel, we need to determine the maximum processing time that can be pilfered from hard deadline periodic tasks. FlexRay communication system can use a slot pilferer to schedule the static and

dynamic segments. The slot pilferer can efficiently address the problem of minimizing the response times of soft aperiodic tasks, while offering the guarantee that the deadlines of hard periodics are also met.

D. Slot Pilfering

We leverage a slot pilfering algorithm [18]–[20] to optimize bandwidth utilization and offer flexible dual-channel scheduling. The slot pilfering algorithm can minimize the response times of soft aperiodic tasks. In order to support aperiodic requests, a slot pilferer needs to find spare processing time by effectively pilfering (i.e., stealing) the slots from the hard deadline periodic tasks. HOSA hence needs to determine the maximum amount of slots, that can be pilfered, without violating the hard timing constraints.

The slot pilfering algorithm presents the method to find the available slots for transmitting both static and dynamic segments in a dual-channel way. Specifically, we first implement this by modeling the processing schedule for the hard periodic tasks. We then check the slots among the deadlines of executing sequential tasks. HOSA stores and maintains the values found in a table. In practice, we make use of a set of counters to record the slots that are possible to be pilfered at different priority levels. We further decrease the values of these counters by considering the tasks that are carried out or updated with the reference to the table. We finally determine the maximum amount of processing time that is possible to be pilfered from executing a hard deadline task without causing its deadline to be missed.

In the context of FlexRay design, we have the following assumption: (1) Each instance uses its worst case execution time; (2) The deadline of each task for scheduling either static or dynamic segment is less than or equal to its minimum inter-arrival time; (3) There is no synchronization or jitter in the task set for scheduling either static or dynamic segment. We also present the used notation for describing task i. Specifically, given an interval time $[0,t]$, $l_{i,t}$ is the time when task i was last released, and $x_{i,t}$ is the earliest time when task i was next released. Moreover, $d_{i,t}$ is the next deadline on executing task i and $c_{i,t}$ is the remaining time for executing task i. HOSA thus has $x_{i,t} = l_{i,t} + T_i$. When task i is complete, we have $d_{i,t} = x_{i,t} + D_i$, which is actually the deadline for the next release. In addition, we can obtain the value of $c_{i,t}$ by subtracting the execution time used in the worst case execution time, C_i, and $c_{i,t} = 0$ if task i is complete at time t.

In order to efficiently carry out holistic scheduling on both static and dynamic segments, during the execution interval $[t, t+d_{i,t})$, we aim to find the maximum amount of slot time, represented as $S^{max}_{i,t}$. The time may be pilfered in the priority level i. In the meantime, we need to guarantee that task i meets its deadline. In order to efficiently compute the maximum slot time, i.e., $S^{max}_{i,t}$, we need to examine the slot computation in the interval $[t, t+d_{i,t})$, in which there exist a number of level i busy and idle periods for holistic scheduling. Specifically, a level i busy period refers to a continuous time interval, during which the execution queue contains one or more tasks with priority level i or higher. Instead, a level i idle period is a time interval during which the execution queue is free of level i or higher priority tasks. Therefore, any level i idle time between the completion of task i and its deadline could be pilfered for task i computation without causing the deadline to be missed. We argue that the maximum slot that may be pilfered is equal to the overall level i idle time in the interval. We further leverage this result to calculate $S^{max}_{i,t}$.

HOSA computes the level i idle time with the aid of two important equations, i.e., $w^{m+1}_{i,t}$ and $v_{i,t}(w_{i,t})$. The former demonstrates how to determine $w_{i,t}$ that represents the length of a level i busy period beginning at time t. By considering a given start time, the latter can determine the length of a level i idle period. We hence execute the iteration computation over the interval $[t, t+d_{i,t})$ and summarize all the idle times to find $S^{max}_{i,t}$. We first compute $w^{m+1}_{i,t} = S_{i,t} + \sum_{\forall j \in HighPriority(i) \cup i}(c_{j,t} + \lceil \frac{\max(w^m_{i,t} - x_{j,t}, 0)}{T_j} \rceil C_j)$, in which $S_{i,t}$ represents the level i slot processing that is released at time t for FlexRay's segments. This equation in fact takes into account two components to decide the extent of the busy period for scheduling FlexRay's segments. The first part shows the level i or higher priority processing at time t. The second part shows the level i or higher priority processing released during the busy period, which actually exhibits a recursive definition during the scheduling for FlexRay's segments.

We further discuss how to obtain the value of $w_{i,t}$ in a FlexRay network. First, the increments of the processing show the property of being monotonic with the length of the busy period. We thus can leverage a recurrence to compute the $w_{i,t}$. The recurrence procedure for scheduling FlexRay's segments starts to operate when $w^0_{i,t} = 0$ and ends when $w^{m+1}_{i,t} = w^n_i$ or $w^{m+1}_{i,t} > d_{i,t}$. On the other hand, we can consider $t + w_{i,t}$ to define the start of a level i idle period during the scheduling for FlexRay's segments. By considering a given start time of a level i idle period, the end of this idle time occurs either at the next release of a task of priority i or higher or at the end of the interval $[t, t+d_{i,t})$. Therefore, through computing the $v_{i,t}(w_{i,t})$, we obtain the length of the level i idle window, $v_{i,t}(w_{i,t}) = \min\{\max(d_{i,t} - w_{i,t}, 0), \min_{\forall j \in hp(i) \cup i}(\max(\lceil \frac{w_{i,t} - x_{j,t}}{T_j} \rceil, 0) T_j + x_{j,t} - w_{i,t})\}$.

Combining $w^{m+1}_{i,t}$ and $v_{i,t}(w_{i,t})$, slot pilfering scheme for scheduling FlexRay's segments determines the maximum slot, $S^{max}_{i,t}$. Specifically, first, $S_{i,t}$ denotes the slot that is possible to be pilfered. Its initial value is set to zero. Second, in order to obtain the end of a busy period in the interval $[t, t+d_{i,t})$, HOSA performs the computation in the $w^{m+1}_{i,t}$. Third, by considering the end of the busy period as the start of an idle period, the $v_{i,t}(w_{i,t})$ can return the lengths of idle times. Fourth, by computing the amount of the idle time in the last step, HOSA increases the slot processing, $S_{i,t}$. Fifth, if task i has reached its deadline, HOSA can pilfer the maximum slot, which is represented as $S_{i,t}$. Otherwise, HOSA repeats previous steps to carrying out the slot pilfer scheme for scheduling FlexRay's segments.

E. Approximation

Although the standard slot pilfer scheme works well for improving bandwidth utilization, it suffers from the highly complex computation that in fact severely limits its use in practical applications. An approximation technique is hence necessary and important, which offers a suitable tradeoff between computation complexity and available pilfered slots.

To this aim, we leverage an efficient approximation approach for scheduling FlexRay's segments. The basic idea behind this approximation is to use simpler computation to significantly reduce the computation complexity but at the expense of obtaining slightly less available slots to be pilfered. Specifically, through using the time between the completion of a task and its next deadline, the approximation can find a lower bound on the available slots. Since hard real-time task sets contain periodic tasks for scheduling static segments, we only need to recalculate the slots available at priority level i when task i completes. We hence argue that a lower bound on the level i slot, i.e., S_i^A, can be available immediately after task i completes. In the FlexRay network, the lower bound on level i idle time depends on the length of interval ω. We use $S_i^A(\omega)$ as the notation for a function of ω that can return this lower bound. In fact, the approximation can find the level i idle time in the interval between the completion of task i and the deadline on its next instance.

It is worth noting that a variation of the algorithm of slot computation can be used to calculate $S_i^A(\omega)$. Specifically, $D_i + T_i \geq \omega \geq T_i$ represents the completion of task i. Since the task i completes, we do not need to carry out any task with higher priority than i. When all tasks of higher priorities than i are released immediately, task i completes and in the meantime the least level i idle time is available. This result can help compute the values of $S_i^A(\omega)$ for each possible value of ω. Therefore, performing the computation of $S_i^A(\omega)$ is to capture the level i idle time in the interval $[0, \omega)$ to support the approximation computation for scheduling FlexRay's segments.

IV. Performance Evaluation

In this Section, we show the experimental results of implementing our proposed HOSA running on mixed datasets (including static and dynamic segments).

A. Experimental Configurations

Our experiments are performed using 10 FlexRay nodes that are connected to a bus analysis tool that helps record the information of message transmission in the FlexRay network. The FlexRay nodes are implemented and configured by multiple networked boards that consist of a 16-bit Flash-based controller unit to support the FlexRay protocol operations, 2 IP-modules for the dual-channel design, and FlexRay-enabled transceivers to support the physical layer of the FlexRay bus. In order to facilitate the real-time transmission analysis, we use an independent module to receive and maintain all messages that are transmitted on the FlexRay bus.

The experiments make use of the mixed datasets that contain both static and dynamic segments. Specifically, for the datasets of static segments, the datasets consist of synthetic test cases and one real-world scenario. The synthetic test cases were generated by varying message parameters, such as periods and deadlines, to cover a wide range of possible scenarios. The periods are varied between 2 ms and 50 ms. The deadlines are varied between 1 ms to 20 ms. The FlexRay communication cycle period is 5 ms and the static cycle length is 3 ms, based on the experiences from the industry [9]. The test cases contain a large number of messages. Moreover, we consider a real-world x-by-wire application, i.e., brake-by-wire, which has been widely used in performance evaluation of the FlexRay-based design. Table I shows the details of the associated parameters.

TABLE I
BRAKE-BY-WIRE MESSAGE PARAMETERS.

Message	Offset (ms)	Period (ms)	Deadline (ms)	Size (bits)
1	0.26	8	8	1280
2	0.72	8	8	272
3	0.52	1	1	1560
4	0.88	1	1	563
5	0.92	1	1	345
6	0.96	1	1	425
7	0.22	1	1	1172
8	0.27	8	8	852
9	0.76	8	8	763
10	0.39	8	8	915
11	0.91	8	8	1245
12	0.52	8	8	628
13	0.69	8	8	427
14	0.81	8	8	338
15	0.93	8	8	847
16	0.42	8	8	1560
17	0.61	1	1	1730
18	0.53	1	1	532
19	0.95	1	1	1154
20	0.77	1	1	861

For the datasets of dynamic segments, we configure the parameters introduced in Section II-B in each communication cycle. We set the values of the parameters as shown in Table II. The suitable timing properties of aperiodic messages used in our experiments are taken from a message set that is published by the Society for Automotive Engineers [21]. Hence, we consider aperiodic messages with a period (minimum inter-arrival time) and a deadline of 50 ms. We use 30 aperiodic messages with the IDs, from 81 to 110 or from 121 to 150, respectively corresponding to the sequential numbers in 80 and 120 slots. The maximum number of their transmission slots *cSlotIDMax* are 110 and 150, respectively.

TABLE II
CONFIGURATION PARAMETERS FOR DYNAMIC SEGMENTS.

Configuration Parameter	**Value**
gdMacrotick $[\mu s]$	1
gNumberOfStaticSlots[macrotick]	80, 120
gdCycle $[\mu s]$	5000
gdStaticSlot[macrotick]	40
gdMacroPerCycle	5000
gdMinislot[macrotick]	8
gdSymbolWindow[macrotick]	0
gdDynamicSlotIdlePhase[minislot]	1
gdMinislotActionPointOffset[macrotick]	2

We uniformly distribute the aperiodic messages in 10 FlexRay nodes. In each network node, an interrupt-based routine running as the host process generates the aperiodic messages. We use a 16-bit reload timer to count down the

time until the next generation of each message. Furthermore, for generating the event-based messages, a *rand*() function in C standard library computes the next generation time.

The minimum length of the dynamic segment is determined by *gNumberOfMinislots*. We select the dynamic segments with 50 and 100 minislots in our evaluation. In order to adjust the length of the dynamic segment, We vary the value of parameter *gNumberOfMiniSlots*. In order to compensate the modification of the dynamic segment length, we change the parameter *gdNIT* (duration of network idle time) so as to keep the frame cycle duration of 5000 μs as a constant.

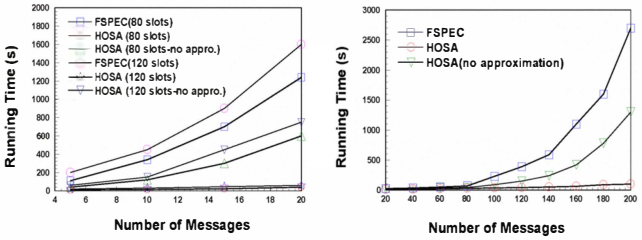
We compare the HOSA scheme with the standard implementation of FlexRay specification (FSPEC) [1] and HOSA without the approximation for reducing computation complexity, in terms of overall running time, bandwidth utilization, average transmission latency for static and dynamic segments, and deadline miss ratio.

B. Results

Figure 7 demonstrates the average running time with respect to the increments of messages in both brake-by-wire scenario and synthetic test cases. Specifically, the brake-by-wire scenario describes relatively light overhead as shown in Figure 7(a). The proposed HOSA scheme completes the message transmission within 40 seconds (for 80 slots) or 62 seconds (for 120 slots), which are much smaller than 1240 or 1600 seconds from using the standard FSPEC. The reason is that HOSA leverages flexible slot scheduling scheme for both static and dynamic segments, which significantly improves the message transmission and decreases deadline miss ratio (as shown in Figure 10).

Moreover, we also observe that the running time for 120 slots is larger than that for 80 slots since the former potentially leads to more idle slots and the utilization of the overall bandwidth decreases, which is also verified by the results in Figure 8. In addition, if there is no approximation, the running time will increase to 652 and 761 seconds respectively for the cases of 80 and 120 slots. This result also demonstrates the efficiency of the approximation for reducing the complexity of the slot computation.

In order to examine the scalability of the proposed HOSA scheme, we execute the holistic scheduling upon synthetic test cases that contain much larger message set. Figure 7(b) shows the experimental results and HOSA requires much smaller running time than the standard FSPEC, in particular in the large scale.



(a) Brake-by-wire case. (b) Synthetic test case.

Fig. 7. Running time for brake-by-wire application and synthetic test cases.

Bandwidth utilization refers to the ratio of the bandwidth that is actually used to the whole bandwidth. HOSA offers flexible scheduling and obtains significant improvements upon bandwidth utilization with the aid of slot pilfering technique. Figure 8 shows the bandwidth utilization of HOSA and FSPEC under 50 and 100 minislots. We observe that HOSA improves 48.6% and 51.2% bandwidth utilization over the standard FSPEC, respectively in 50 and 100 minislots. HOSA can therefore optimize the bandwidth utilization. Moreover, without the approximation, the slot computation incurs longer waiting time and leads to more potential re-transmission that reduces the practical bandwidth utilization.

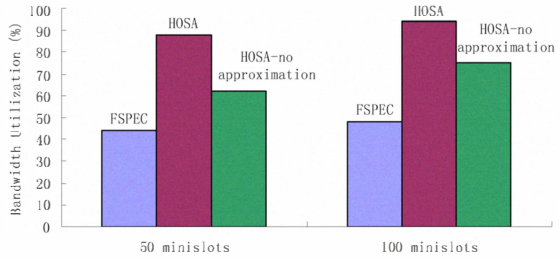
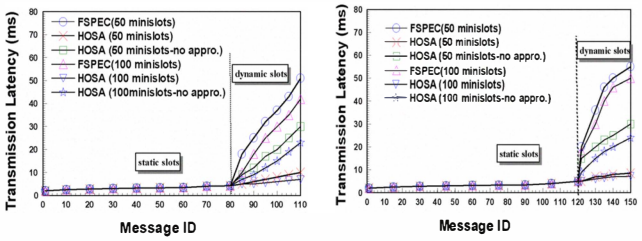


Fig. 8. Bandwidth utilization.



(a) Synthetic test cases. (b) Brake-by-wire messages.

Fig. 9. Average transmission latency of both static and dynamic segments.

The transmission latency is counted from the generation time to the time when the message is completely transmitted. Figure 9 shows the average transmission latency of both static and dynamic segments. Specifically, we examine the transmission latency respectively in the synthetic cases and brake-by-wire scenario, both of which exhibit similar observations. For instance, in the Figure 9(a), we divide the messages into two types, i.e., static (from 1 to 80 IDs) and dynamic (from 81 to 110 IDs) segments. Both HOSA and FSPEC can provide hard transmission guarantee to static segments. They hence obtain the same latency (from 1 to 80 message IDs).

Moreover, for dynamic segments (from 81 to 110 message IDs), HOSA requires on average 61.5% and 68.2% smaller latency than the standard FSPEC. The decrements in the Figure 9(b) are 71.5% and 78.2% respectively. HOSA can hence efficiently support the transmission for both static and dynamic segments. In addition, the results also demonstrate the approximation (represented as *appro.* in Figure 9) technique can significantly reduce transmission latency due to its simple computation.

The deadline miss ratio is defined as the number of messages with missed deadlines divided by the number of

messages transmitted. Figure 10 demonstrates the deadline miss ratio when taking into account 50 and 100 minislots. Since HOSA significantly reduces the transmission delay and improves the bandwidth utilization, the ratio of missed messages is on average 3.5%, which is much smaller than 7.2% in no approximation and 27.8% in the FSPEC scheme.

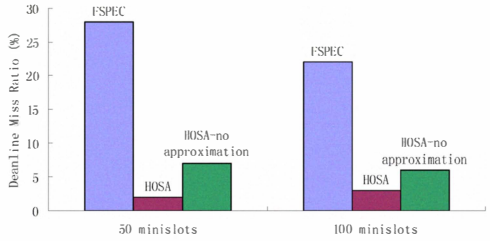


Fig. 10. Deadline miss ratio.

V. Related Work

One of the first in-vehicle communication networks is the controller area network (CAN) [4] that provides bounded delay communication at data rates between 125 kb/s and 1 Mb/s. However, it is not suitable for new applications, e.g., x-by-wire applications, which are hard-real-time in essence, and require high-speed, robust, and predictable communication. The attempts to meet these demands are time-triggered CAN (TTCAN [22]), time-triggered protocol (TTP [5]), and ByteFlight [6]. TTCAN and TTP are time-triggered technology with predictable medium access, while ByteFlight is based on FTDMA.

Existing work on FlexRay mainly considers a single channel scenario, in which either static segments or dynamic segments are scheduled. Optimizing bandwidth utilization [8] is formulated into constraint logic programming to provide fault-tolerant schedule in the presence of transient and intermittent faults. Scheduling static segments [7] and dynamic segment [10] is formulated into nonlinear integer programming problem to maximize bandwidth utilization. Based on mixed-integer linear programming, an optimization framework [2] is proposed to schedule transactions consisting of tasks and signals on a FlexRay-based system. Although authors in [23] propose the schedulability analysis for determining the timing properties of message transmitted in both static and dynamic segments, the analysis only considers the scenario of a single channel.

VI. Conclusion

Providing scalable fault-tolerance is important to the FlexRay networks. This paper proposes a cost-effective scheme, called HOSA, that supports holistic scheduling on both static and dynamic segments in the dual-channel communication system. HOSA leverages a slot pilfering technique to significantly minimize idle slots, improve bandwidth utilization and decrease transmission latency. HOSA efficiently handles the complexity of slot computation with the aid of a proper approximation approach. Extensive experimental results based on synthetic and real-world test cases demonstrate the efficiency and efficacy of the proposed HOSA scheme.

Acknowledgment

This work was supported in part by National Natural Science Foundation of China (NSFC) under Grant 61173043 and 60703046, and NSERC Discovery Grant 341823-07, and FQRNT grant 2010-NC-131844.

References

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

[2] 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.

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

[4] 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.

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

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

[7] 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.

[8] B. Tanasa, U. D.Bordoloi, P. Eles, and Z. Peng, "Scheduling for Fault-Tolerant Communication on the Static Segment of FlexRay," *Proc. IEEE Real-Time Systems Symposium*, 2010.

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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

[15] 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.

[16] 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. Design, Automation and Test in Europe*, 2009.

[17] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: Exact characterization and average case behavior," *Proc. Real-Time Systems Symposium*, pp. 166–171, 1987.

[18] J. Lehoczky and S. Ramos-Thuel, "An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems," *Proc. Real-Time Systems Symposium*, pp. 110–123, 1992.

[19] R. Davis, K. Tindell, and A. Burns, "Scheduling slack time in fixed priority pre-emptive systems," *Proc. Real-Time Systems Symposium*, pp. 222–231, 1993.

[20] 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.

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

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

[23] 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.