Quantifying Notions of Extensibility in FlexRay Schedule Synthesis

REINHARD SCHNEIDER, TU Munich
DIP GOSWAMI, Eindhoven University of Technology
SAMARJIT CHAKRABORTY, TU Munich
UNMESH BORDOLOI, PETRU ELES, and ZEBO PENG, Linkoping University

FlexRay has now become a well-established in-vehicle communication bus at most original equipment manufacturers (OEMs) such as BMW, Audi, and GM. Given the increasing cost of verification and the high degree of crosslinking between components in automotive architectures, an incremental design process is commonly followed. In order to incorporate FlexRay-based designs in such a process, the resulting schedules must be *extensible*, that is: (i) when messages are added in later iterations, they must preserve deadline guarantees of already scheduled messages, and (ii) they must accommodate as many new messages as possible without changes to existing schedules. Apart from extensible scheduling having not received much attention so far, traditional metrics used for quantifying them cannot be trivially adapted to FlexRay schedules. This is because they do not exploit specific properties of the FlexRay protocol. In this article we, for the first time, introduce new notions of extensibility for FlexRay that capture all the protocol-specific properties. In particular, we focus on the dynamic segment of FlexRay and we present a number of metrics to quantify extensible schedules. Based on the introduced metrics, we propose strategies to synthesize extensible schedules and compare the results of different scheduling algorithms. We demonstrate the applicability of the results with industrial-size case studies and also show that the proposed metrics may also be visually represented, thereby allowing for easy interpretation.

Categories and Subject Descriptors: C.3 [**Special-Purpose and Application-Based Systems**]: Real-Time Systems

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: FlexRay, extensibility, schedule synthesis, automotive

ACM Reference Format:
Reinhard Schneider, Dip Goswami, Samarjit Chakraborty, Unmesh Bordoloi, Petru Eles, and Zebo Peng. 2014. Quantifying notions of extensibility in FleXray schedule synthesis. ACM Trans. Des. Autom. Electron. Syst. 19, 4, Article 32 (August 2014), 37 pages.
DOI: http://dx.doi.org/10.1145/2647954

1. INTRODUCTION

FlexRay has taken a veritable lead as the next-generation automotive in-vehicle communication network. The FlexRay protocol has been developed by a consortium of more than 100 leading companies in the automotive industry between the years 2000 and 2010, which recently has completed its work with the finalization of the protocol

---

This article extends an earlier version that appeared at the 48th Design Automation Conference (DAC2011).
Authors' addresses: R. Schneider (corresponding author), Electrical Engineering Department, TU Munich, Germany; email: reinhard.schneider@res.ei.tum.de; D. Goswami, Electrical Engineering Department, Eindhoven University of Technology, The Netherlands; S. Chakraborty, Electrical Engineering Department, TU Munich, Germany; U. Bordoloi, P. Eles, and Z. Peng, Computer Science Department, Linkoping University, Sweden.

DOI: http://dx.doi.org/10.1145/2647954

specifications [FlexRay 2013]. Due to its high bandwidth, deterministic temporal behavior, and fault-tolerant mechanisms, the FlexRay bus has become an inherent part of in-vehicle networks in today's premium-class automobiles such as the Audi A8 and the BMW 7 series [Fuchs 2010]. The design process for FlexRay follows the iterative design paradigm widely followed in the automotive industry, where new components and functionalities are incrementally added and tested at each iteration. Thus, new messages are added and scheduled on the FlexRay bus at each design cycle. At the initial design cycle, the system designer decides on the physical-layer configuration, for example, the configuration of the bus topology and other global parameters such as bus speed and bus period. Once these parameters are validated and fixed, they should not be changed in later iterations in order to avoid a complete redesign and reevaluation of the system from scratch. Moreover, time-triggered systems such as FlexRay exhibit tight dependencies of local task schedules and global bus schedules, and hence the overall timing behavior of the system is very sensitive to changes in these schedules. In particular, changes to bus schedules may result in the following.

—There may be a reconfiguration of the FlexRay controllers of all ECUs affected by the change, that is, all ECUs that transmit or receive the message whose schedule has been changed.
—Also there can be changes in task schedules that need to be synchronized with the new bus schedules to meet the timing constraints.
—Furthermore, these changes can cause reevaluation of the overall timing behavior of the system that is affected by the change *and* all other applications that might be affected by the change, for example, a new schedule for message m_1 might result in interference with an existing message m_2 and increase its response time.
—Finally, there may be test and validation procedures that are extremely costly in terms of time and money.

The test and validation process is of particular importance and aims at several important goals such as verification of functional properties (e.g., stability of control applications, conformance tests to verify compliance with different standards such as AUTOSAR [2013], robustness and stress tests, as well as performance tests [Armengaud et al. 2008]). To this end, test concepts strive for maximum test coverage to test all important protocol and application features in all possible modes of operation. As a result, test procedures are very time consuming and expensive, involving: (i) several hardware setups such as prototyping hardware and measurement equipment, electronic control units (ECUs), Hardware-in-the-Loop (HiL) systems, and fully equipped vehicles; (ii) software and tools for test management, monitoring, debugging, test, and validation; and (iii) domain experts from different departments and suppliers.

Hence, it is extremely important that, in an iterative design process, the existing design and schedules are unaffected by the addition of new applications or features. To address these challenges, it is important that new messages can be added in future iterations without disturbing and changing the schedules of the existing messages while guaranteeing their real-time constraints. In other words, schedules generated at each iteration should be *sustainable*, that is, *forward compatible*. More precisely, new messages being added in future iterations (*future versions*) should not cause any deadline violations of existing messages (*current version* of the design). Second, new messages should be able to be accommodated without rescheduling existing messages while satisfying all the timing constraints. As a result, *extensibility* [Sangiovanni-Vincentelli et al. 2009; Sangiovanni-Vincentelli and Di Natale 2007] is a highly desirable attribute in the design of automotive embedded systems to realize cost-efficient development and to reduce time-to-market.

Related Work and Our Contributions. Along the lines of the research we present in this article, two major directions of related work have become apparent. These can be classified into work on: (i) extensible scheduling for real-time systems, and (ii) scheduling and timing analysis for FlexRay. While there has been some recent effort to capture extensibility in real-time systems, there is still ambiguity regarding the interpretation of these notions. In this context, sustainability has been discussed in the context of uniprocessor and multiprocessor scheduling [Baruah and Burns 2006; Burns and Baruah 2008; Baker and Baruah 2009]. Accordingly, a *scheduling policy* is defined as sustainable with respect to a system parameter if a system that was determined to be schedulable remains so even after the parameter is changed for the better, that is, either increased or decreased. A similar definition for sustainability has also been presented in Anand and Lee [2008] and Poon and Mok [2010]. There are three major reasons why the existing notions of sustainability are not suitable for FlexRay schedules.

—First, they do not consider the automotive product-line design approach where schedules are generated incrementally at different design iterations [Zheng et al. 2005]. In an incremental design approach, schedules must be generated at each iteration before the full range of functionalities and messages to support is known. For this reason, a novel notion of sustainability is required that also captures the spirit of incremental scheduling.
—Second, the analysis techniques presented along the preceding lines of research are restricted to processor scheduling and do not consider FlexRay-specific protocol and scheduling properties. As a result, the presented notions cannot be applied to FlexRay schedules and, hence, FlexRay-specific design methodologies are required.
—Finally, in the incremental design process that arises in the automotive scenario, it is hardly likely that parameters like periods of already scheduled messages will be changed once they have been determined. Rather, it is natural that new messages are *added* to the system. For this reason, existing notions of sustainability that are defined with respect to changes in a system parameter are not applicable in this scenario. Rather, we are concerned with the question as to whether an *existing design* or *version* is sustainable with respect to its temporal behavior, that is, whether previously scheduled messages are feasible (do not violate their deadlines) even when new messages are added in future iterations. To accurately reflect this property and to distinguish between the originally introduced notion of sustainability in real-time systems by Baruah et al., in this article, a notion of *forward compatibility* for FlexRay schedules is introduced. For ease of exposition, we simply refer to *compatibility* in what follows.

Various papers focused on extensibility of distributed real-time systems in which the notions of extensibility differ in their interpretations. The work in Zheng et al. [2005] proposes an extensibility metric for messages on a Time Division Multiple-Access (TMDA) bus. Here, extensibility is defined as the maximum message worst-case transmission time extension a schedule can accommodate by rescheduling the finish time of the message between the source ECU and the destination ECU. Extensibility has been discussed in the context of incremental scheduling for distributed real-time embedded systems [Pop et al. 2004]. The work addresses the problem of adding new functionalities such that the timing requirements are fulfilled while the already running applications are disturbed as little as possible and new functionality can easily be added to the existing system. Further, extensibility has been defined as the cost one has to pay when a new functionality, such as a task or a message, is added on to an existing system [Scheler and Schroeder-Preikschat 2006]. Furthermore, uncertainty in frame

payloads has been studied to quantify extensibility for FlexRay messages [Ghosal et al. 2010].

—Among the previous lines of work there exist many different interpretations of extensibility. In this work, we are interested in a notion of extensibility for FlexRay schedules in an incremental design scenario. In contrast to the existing notions, we quantify the capacity to accommodate additional (*future*) messages on FlexRay as a measure of extensibility.

Apart from existing work on extensibility, of late, there has also been tremendous research interest towards building tools and algorithms for scheduling and timing analysis of messages on automotive networks [Schliecker et al. 2009]. In particular, the synthesis of schedules for FlexRay has drawn the attention of several researchers [Schmidt and Schmidt 2009, 2010; Tanasa et al. 2010; Zeng et al. 2010, 2009; Schneider et al. 2010]. Of relevance is an efficient approach to schedule messages on the static (time-triggered) segment of FlexRay [Lukasiewycz et al. 2009]. Timing analysis methods for the dynamic (event-triggered) segment of FlexRay have been presented [Pop et al. 2006; Hagiescu et al. 2007] and recent research efforts have improved earlier results and incorporated details of the FlexRay protocol [Tanasa et al. 2012; Neukirchner et al. 2012]. The work in Ghosal et al. [2010] addresses incremental FlexRay scheduling and incorporates uncertainty of design parameters. However, metrics to quantify extensibility for FlexRay schedules were not presented. Moreover, the simplifying assumptions in the proposed scheduling techniques do not exploit all the special characteristics of FlexRay, such as slot multiplexing, or were restricted to the static segment.

In contrast, our schedule synthesis approach is designed to systematically optimize the schedules towards extensibility. In this article we, for the first time, propose the following.

—*Notions* for extensibility from the perspective of FlexRay in an incremental design scenario are given. In this context, we also show that traditional notions of extensibility cannot be trivially adapted to FlexRay schedules.
—*Metrics* to quantify the quality of FlexRay schedules with respect to extensibility are provided. Towards this aim, metrics proposed in this work are applicable to both static and dynamic segments. Apart from having a mathematical basis, our metrics also allow easy visualization and thus may be easily interpreted by automotive engineers and help to identify resource bottlenecks in early design phases.
—*Scheduling algorithms* are given that incorporate the proposed metrics and analysis techniques to *synthesize* extensible FlexRay schedule parameters.

As already described earlier, there exist several works on timing analysis for the FlexRay dynamic segment that: (i) capture different levels of protocol details, (ii) provide different accuracy in the message delay estimates, and (iii) use different models of computation resulting in different computational complexity [Bordoloi et al. 2012]. It is also foreseeable that other delay models will come up in the near future. As a consequence, new concepts presented in this article that are, to some extent, based on a specific delay model require an understanding of the underlying analysis techniques and may require specific adaptations to the delay model under consideration. As the focus of this work lies in the definition, quantification, and synthesis of extensible FlexRay schedules, for simplicity of exposition we use a simpler delay model that computes more pessimistic message delays but is intuitive and suitable to demonstrate our proposed metrics and techniques. However, we emphasize that our results are general and hence independent from the specific delay model under consideration. In fact, we give insights on how the proposed concepts may be easily integrated into other delay models.

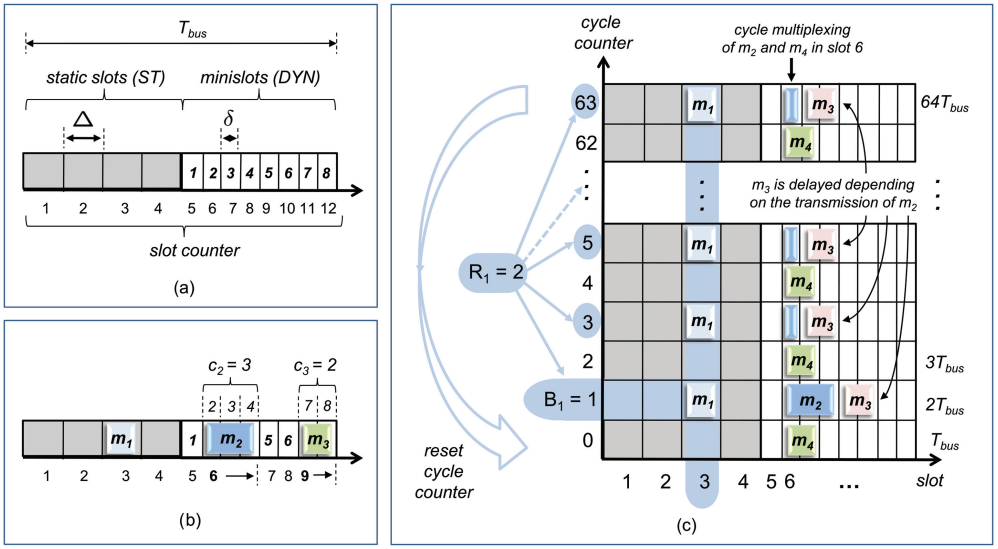

Fig. 1. Figure (a) illustrates a single FlexRay cycle of length T_{bus} with $N = 4$ static slots in the ST segment and $M = 8$ minislots in the DYN segment; (b) transmission of messages m_1, m_2, m_3 in their assigned slots $S_1 = 3, S_2 = 6$, and $S_3 = 9$; (c) FlexRay matrix and the transmission of the messages m_1, m_2, m_3 and m_4 according to the schedules $\Theta_1 = \{3, 1, 2\}$, $\Theta_2 = \{6, 1, 2\}$, $\Theta_3 = \{7, 1, 2\}$ and $\Theta_4 = \{6, 0, 2\}$.

2. THE FLEXRAY PROTOCOL

In the following we introduce the basics of the FlexRay protocol [FlexRay 2013]. The FlexRay communication protocol is organized as a periodic sequence of cycles, where each cycle is of fixed length T_{bus} as depicted in Figure 1(a). Each cycle is further subdivided into two major segments, namely a static (ST) and a dynamic (DYN) segment. In the following we discuss the ST and the DYN segments, followed by specific properties of FlexRay schedules.

Static segment. The ST segment of FlexRay follows a TDMA-based communication paradigm. It is partitioned into a number of equal-length time windows, called static *slots*, of duration Δ. The slots are labeled with a *slot counter* in ascending order starting with 1 and ending with N and the set of static slots is denoted by $\mathcal{S}_{ST} = \{1, \ldots, N\}$, for example, $N = 4$ in Figure 1(a). Each message m_i to be transmitted in the ST segment is assigned a slot number $S_i \in \mathcal{S}_{ST}$. If m_i is not ready at the beginning of the slot, the slot remains empty. For example, in Figure 1(b) the static segment is defined as $\mathcal{S}_{ST} = \{1, \ldots, 4\}$. Message m_1 is transmitted in its assigned static slot $S_1 = 3$. Slot number 4 is not assigned to any message and hence a full static slot of length Δ elapses without any message transmission.

Dynamic segment. The DYN segment is comprised of M equal-length *minislots* of much smaller size δ that is $\delta \ll \Delta$. The set of dynamic slots is given by $\mathcal{S}_{DYN} = \{N + 1, \ldots, N + M\}$. For example, in Figure 1(a) the DYN segment consists of $M = 8$ minislots and the set of available dynamic slots is denoted by $\mathcal{S}_{DYN} = \{5, \ldots, 12\}$. A *dynamic slot* is a logical entity that instead specifies the priority of a message in the DYN segment. Thus, each message m_i is assigned a slot number $S_i \in \mathcal{S}_{DYN}$, specifying that m_i may be transmitted at the beginning of slot S_i. Messages m_j having a higher priority than m_i are assigned lower slot numbers $S_j < S_i$ so that they have access to the bus first. In case a message m_i is transmitted in a slot S_i, then this slot consumes a certain number of minislots c_i depending on the message size and hence dynamic slots

are of variable length. However, if no message is transmitted in a certain slot, then only one minislot of length δ is consumed. For example, in Figure 1(b), message m_2 starts its transmission at the beginning of minislot 2 in slot 6 and occupies three successive minislots as $c_2 = 3$. Consequently, slot 6 is of length 3δ. Thus, after transmission of m_2 the minislot counter value is equal to 5 while the slot counter changes from 6 to 7, as illustrated in the figure. In slots 7 and 8 no message is transmitted and hence only one minislot is consumed at each of these slots. Finally, message m_3 is assigned slot 9, that is, $S_3 = 9$, and consumes two minislots. Note that, in contrast to Figure 1(a), the set of available dynamic slots is $\mathcal{S}_{DYN} = \{5, \ldots, 9\}$, that is, the slot numbers 10, 11, and 12 are not available due to the transmission of the messages m_2 and m_3. Further, a message is transmitted only if the current minislot counter does not exceed the value of *pLatestTx*, which denotes the highest minislot number a message transmission is allowed to begin for a certain ECU. The value of *pLatestTx* is statically configured during design time and depends on the maximum dynamic payload size that is allowed to be transmitted by a certain ECU (more details can be found in the FlexRay specification [FlexRay 2013]).

FlexRay schedules. In the preceding we described the ST and DYN segments of a FlexRay communication cycle. Further, a set of 64 cycles is repeated in a periodic sequence that we refer to as the *FlexRay matrix*. As illustrated in Figure 1(c), each cycle is indexed by a cycle counter incremented from 0 to 63 and reset to 0 again. Apart from the slot number S_i, two further parameters specify the actual transmission cycles of a message m_i within the 64 cycles: (i) the base cycle B_i indicating the offset within 64 communication cycles, and (ii) the cycle repetition rate R_i denoting the number of cycles that must elapse between two consecutive allowable transmissions. Thus, any FlexRay message m_i is assigned S_i, B_i, and R_i to uniquely specify admissible transmission points within 64 cycles. We refer to this as the *schedule* $\Theta_i = \{S_i, B_i, R_i\}$ of m_i. For instance, in Figure 1(c), the schedule for m_1 in the ST segment is specified as $\Theta_1 = \{3, 1, 2\}$, that is, every odd cycle in slot 3 is available for transmission of message m_1. This is because the first cycle is indicated by base cycle $B_1 = 1$, and repetition rate $R_1 = 2$ specifies that two cycles must elapse between allowable transmission points. Similarly, the schedule for m_2 in the DYN segment is specified as $\Theta_2 = \{6, 1, 2\}$. However, an instance of m_2 is only transmitted in cycle 1 of slot 6 consuming three minislots. As a result, the next slot number 7 assigned to m_3 according to $\Theta_3 = \{7, 1, 2\}$ is delayed by 2 minislots in cycle 1, whereas in the other cycles no instance of m_2 is transmitted and hence m_3 is transmitted earlier. Consequently, the delay of a message in the DYN segment may vary in every cycle depending on the transmission of messages with a higher priority and a lower slot number, respectively. Note that every even cycle in slot 6, namely cycle $0, 2, 4, \ldots, 62$, is assigned to message m_4 with $\Theta_4 = \{6, 0, 2\}$. Such scheduling leads to *slot multiplexing*, that is, the same slot being used by multiple messages in different cycles. Since any message will be scheduled within the 64 cycles, the base cycle can be assigned a value within 0 and 63, namely $B_i \in \{0, \ldots, 63\}$. According to the specification of the AUTOSAR FlexRay interface [AUTOSAR 2013] and the FlexRay protocol [FlexRay 2013], the following relations hold:

—repetition rate $R_i = \{2^r \mid r \in \mathbb{N}_0, r \leq 6\}$;
—base cycle $B_i < R_i$;
—the set of feasible cycles related to Θ_i is defined as
$\Gamma_i := \{\gamma \mid \gamma = (B_i + n \cdot R_i),\ n \in \mathbb{N}_0, n \leq (\frac{64}{R_i} - 1)\}$; and
—$(S_i = S_j) \rightarrow (\Gamma_i \cap \Gamma_j = \emptyset)$, where $S_i, S_j \in \mathcal{S}$, and $\mathcal{S} = \mathcal{S}_{ST} \cup \mathcal{S}_{DYN}$.

The last relation defines the slot multiplexing condition where several messages assigned to the same slot may not interfere in any cycle according to their schedules.

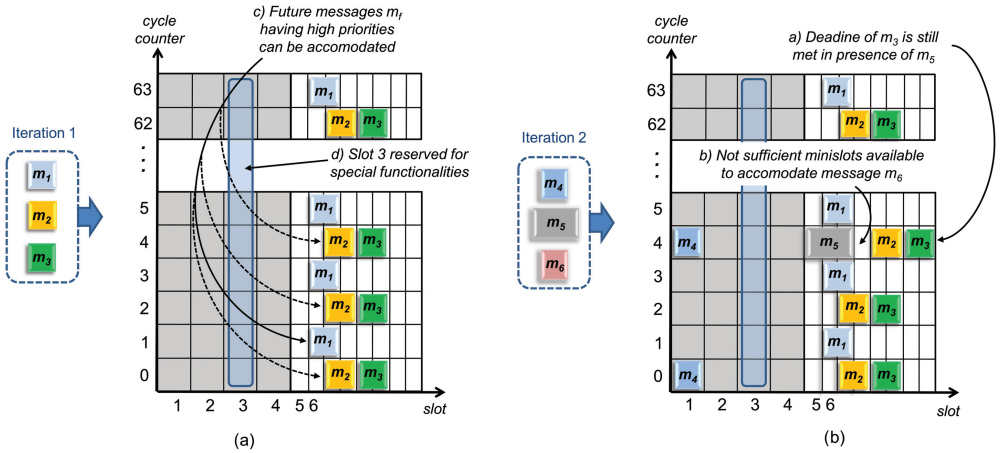

Fig. 2. (a) Design iteration 1 where three new messages m_1, m_2, m_3 are schedules on the FlexRay bus; (b) subsequent design iteration 2 where two new messages m_4 and m_5 have been scheduled on top of the already scheduled messages m_1, m_2, m_3.

3. MOTIVATION AND CHALLENGES

In this section, we illustrate the challenges involved in quantifying extensibility from the perspective of the FlexRay protocol discussed in the previous section. Towards this end, we show that conventional metrics do not apply to FlexRay and motivate the need for new techniques that quantify extensible schedules considering platform-specific properties. Let us consider the examples in Figure 2 that illustrate several schedules in two consecutive design iterations, namely $I = 1$ in Figure 2(a) and $I = 2$ in Figure 2(b). Figure 2(a) shows three messages m_1, m_2, and m_3 that have been scheduled at design iteration 1 on the DYN segment. Messages m_4, m_5, and m_6 are to be scheduled on the ST and DYN segment, respectively, at iteration 2. Note that messages in the ST segment do not experience interference from messages in future iterations as static slots are of fixed and equal length and hence guarantee temporal isolation. Consequently, a schedule Θ_i in the ST segment is *compatible* if the message m_i is assigned to a slot that appears frequently enough to meet the deadline d_i. Thus, it is a straightforward check, thus compatibility in this case follows from the schedule synthesis itself.

Example (a). Assume message m_5 has been assigned slot 5 (that has the highest priority in the DYN segment) at iteration 2 as illustrated in Figure 2(b). Note that, at iteration 1 in Figure 2(a), m_3 was already assigned slot 8 in cycle 4. Hence, the schedule of m_3 must be synthesized such that its deadline d_3 is met even in the presence of the new message m_5, that is, the schedule of m_3 must be *compatible*. Towards this aim, two pieces of information are crucial during schedule synthesis at design iteration 1: (i) the number of future messages having higher priorities than m_3 and (ii) the worst-case workload generated by the future messages. Clearly, designers cannot precisely predict such information. However, based on the product-line, current design stage and class of applications expected in the future, it is reasonable to assume some knowledge about the range of typical message sizes. Given such a range, we will show in Section 4 how protocol properties allow to predict the worst-case workload in the future with reasonable accuracy. Moreover, given a schedule Θ_i, we may bound the number of higher-priority messages that may be assigned according to the slot number $S_i \in \Theta_i$. Based on such observations, we will specify a notion of compatibility and present a compatibility test in Section 4.

Example (b). Note that m_3 is only allowed to be transmitted in the DYN segment if there are sufficient minislots available, that is, m_3 can only be transmitted if the minislot count does not exceed *pLatestTx* (see Section 2). Let message m_6 from an iteration 2 consume two minislots and $pLatestTx = 7$. Then, as depicted in Figure 2(b), m_6 will not be allowed to allocate cycle 4 of slot 6 as there will not be sufficiently many minislots available for a transmission of m_3 before *pLatestTx*. Hence, schedules such as $\Theta_6 = \{6, 0, 2\}$, $\Theta_6 = \{6, 0, 4\}$, $\Theta_6 = \{6, 4, 8\}$, etc., should not be assigned to m_6. Our metric for defining compatibility must be able to capture such details as well.

Example (c). In this example we show that conventional metrics like counting the total number of empty slots do not accurately quantify extensibility for the FlexRay protocol. For example, slot 6 is assigned to message m_1 in every odd cycle at iteration 1 in Figure 2(a). Conventional metrics that count the total number of empty or occupied slots would declare slot 6 as occupied. However, this does not reflect the true nature of the schedule properties as discussed in Section 2. This is because slot 6 can still be assigned in the even cycles to a message m_f in a future iteration, for example, m_6 in iteration 2, using slot multiplexing. Hence, the schedules $\Theta_f = \{S_f, B_f, R_f\}$ where $S_f = 6$, $R_f \in \{2, 4, 8, 16, 32, 64\}$, and $B_f = 2n < R_f, n \in \mathbb{N}_0$, are available for m_f. In order to quantify such available schedules accurately, we require novel metrics of extensibility that we will present in Section 5.

Example (d). System designers often reserve certain slots that are provided for specific protocols, such as XCP [AUTOSARXCP 2013] which is a measurement and calibration protocol, or protocols for the diagnosis and transport layer. Such slots may not be assigned to application messages even if they are empty. For instance, in Figure 2(a), slot 3 is reserved for such special functionalities. Hence, future application messages that must be scheduled in the ST segment may not be assigned to any cycles of slot 3, for instance, m_4 has been scheduled in the first (and not the third) slot at iteration 2 (see Figure 2(b)). Consequently, we do not quantify the extensibility of such reserved slots in order to avoid distortion of the extensibility metric (see Section 5).

Running example. For the sake of demonstration we now introduce a large-scale running example to provide meaningful results and to show visual interpretation of our proposed metrics. We consider a FlexRay bus configuration with $T_{bus} = 5ms$, $\mathcal{S}_{ST} = \{1, \ldots, 17\}$, $\mathcal{S}_{DYN} = \{18, \ldots, 258\}$, and $\delta = 0.015ms$. The value of *pLatestTx* is set to 238 for each ECU. The FlexRay network parameters have been generated compliant with the specification using the SIMTOOLS [2013] configuration software. The message parameters have been selected as commonly found in automotive applications with payload sizes in the range $n_i = [2bytes, 40bytes]$ [Lim et al. 2011] and periods p_i between $10ms$ and $500ms$ with deadlines $d_i \leq p_i$ [Grenier et al. 2008]. All schedules $\Theta_i = \{S_i, B_i, R_i\}$ have been generated in compliance with the FlexRay specification as described in Section 2. For ease of exposition, we consider an iterative design scenario with only two consecutive design iterations, $I \in \{1, 2\}$, where 100 messages[1] are scheduled in the DYN segment at each iteration. For the generated schedules, we evaluate extensibility according to the introduced metrics for each design iteration. In particular, we study how extensibility properties evolve during several design iterations.

4. COMPATIBILITY ANALYSIS

In this section we define a notion of (forward) *compatibility* for FlexRay schedules. The basic idea of compatibility for FlexRay schedules in an incremental design scenario

[1]The details of all message and schedule parameters are depicted in the Appendix in Tables VII and VIII.

is: Given a *version* of a FlexRay design with already scheduled, so-called existing messages, forward compatibility describes whether the existing messages will also meet their deadlines in future versions of the design where new messages might be added in any design iteration. In particular, a schedule Θ_i is referred to as *compatible* if the following conditions are met.

—*COMP1.* The deadline d_i is guaranteed even in the presence of messages from future iterations that may interfere with m_i and increase its response time.
—*COMP2.* The workload due to messages from future iterations that interfere with m_i, is bounded.
—*COMP3.* Existing schedules are not changed at any time.

We present a compatibility analysis and introduce a compatibility index that serves as a performance measure to quantify the compatibility capabilities of scheduling algorithms. With such an aim, we first show that compatibility in the ST segment is rather trivial and hence we focus our analysis on the DYN segment. The ST segment of FlexRay provides temporal isolation between the slots $S_i \in \mathcal{S}_{ST}$ as all slots are of fixed and equal length. Hence, messages transmitted in the static slots do not interfere with each other and have no impact on the message delay. Consequently, COMP1 is fulfilled if the deadline d_i is guaranteed for a certain static segment schedule Θ_i. Similarly, COMP2 does not impose any constraints on the compatibility analysis as there is no interference between messages in the ST segment. The worst-case delay in the static segment depends on: (i) the static slot length Δ and (ii) the bus blocking time $R_i \cdot T_{bus}$. The bus blocking time considers the case that m_i just missed its static slot S_i and has to wait for R_i cycles until the next available slot. This gives us the following expression for the worst-case delay D_i of a message m_i in the ST segment

$$D_i = R_i \cdot T_{bus} + \Delta, \tag{1}$$

and the condition for compatibility in the ST segment yields

$$D_i \leq d_i. \tag{2}$$

Note that (1) considers the case where the application tasks are not synchronized with the FlexRay bus schedules. Further, (1) does not depend on the slot number S_i and the base cycle B_i as these parameters are already captured by the worst-case scenario that considers the slot has just been missed. If the worst-case delay D_i satisfies the deadline d_i then the schedule Θ_i is referred to as *compatible*, independent of the number and properties of future messages that will be scheduled in the ST segment.

4.1. Compatibility in the Dynamic Segment

As mentioned in Section 2, the quantification of compatibility for DYN segment schedules with $S_i \in S_{DYN}$ is more complex due to the dynamic nature of the priority-based communication paradigm. In the following, we first present: (i) a *delay model* and a *schedulability test* to verify message deadlines, and (ii) a *workload estimation model* to account for interference due to future messages. In general, the number and workload of future messages are unpredictable. However, we exploit certain FlexRay-specific properties to bound the number of higher-priority messages and make reasonable assumptions on the expected workload. Finally, we present a *compatibility test* to check whether any schedule Θ_i satisfies real-time constraints in the presence of messages from future iterations interfering with m_i.

Delay model and schedulability test. The transmission time of a message m_i consuming c_i minislots is given by $e_i = c_i \cdot \delta$, where δ is the duration of a minislot. Further, we denote the *effective transmission time* by $\overline{e}_i = (c_i - 1) \cdot \delta$. This captures the additional

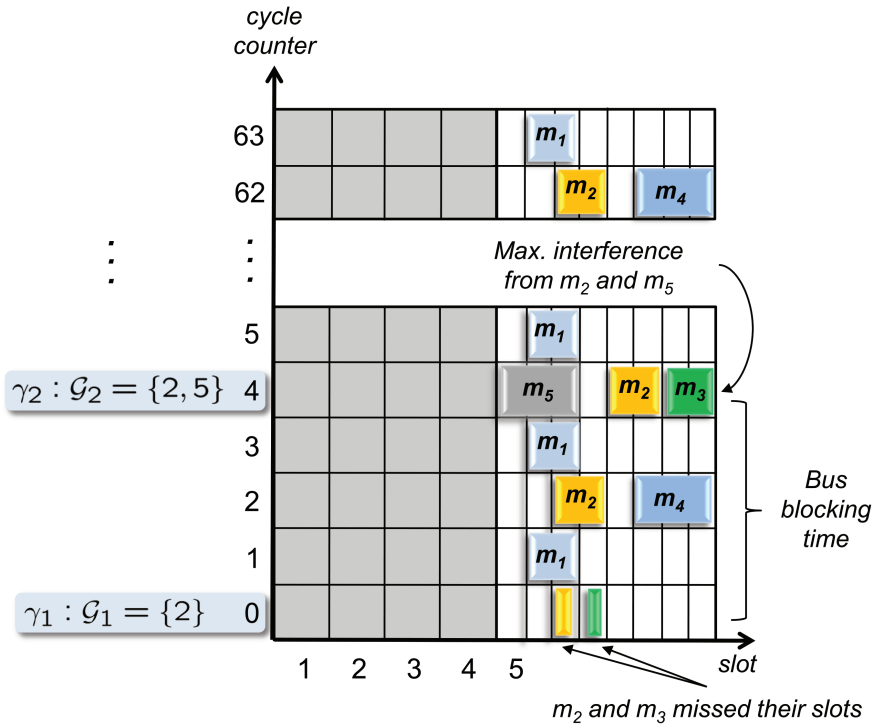

Fig. 3. Worst-case delay scenario for message m_3.

transmission time in case a message m_i of size c_i is actually transmitted in its assigned slot $S_i \in \mathcal{S}_{DYN}$. Since one minislot is consumed even if no message is transmitted on the bus, the effective workload generated by m_i is $\overline{c}_i = c_i - 1$. This is also illustrated in Figure 4 where message m_1, transmitted in slot 6, consumes $c_1 = 4$ minislots, resulting in an effective workload of $\overline{c}_1 = 3$ minislots. Let $\mathcal{G}_k$ be the sets of message indices $j \in \mathcal{G}_k$ such that $\Gamma_j \cap \gamma \neq \emptyset$ and $S_j < S_i$, $\forall j, \forall \gamma \in \Gamma_i$, and $k \in \{1, \ldots \frac{64}{R_i}\}$, k being an integer. Hence, m_j are messages having a priority higher than m_i, that is, they share at least one cycle with m_i and have a lower slot number such that their transmissions might affect the delay of m_i.

Provided that a sufficiently large number of minislots is available to transmit m_i in any cycle, the worst-case delay due to messages with higher priorities than m_i's may be computed as

$$D_i = R_i \cdot T_{bus} + \max_{k \in \{1, \ldots, \frac{64}{R_i}\}} \sum_{j \in \mathcal{G}_k} \overline{e}_j + e_i. \tag{3}$$

The first term in the preceding equation accounts for the bus blocking time $R_i \cdot T_{bus}$ in case m_i just missed its slot S_i and has to wait for R_i cycles until the next available slot. The second component $\max_{k \in \{1, \ldots, \frac{64}{R_i}\}} \sum_{j \in \mathcal{G}_k} \overline{e}_j$ captures the worst-case interference due to messages m_j having a higher priority than m_i and e_i denotes the transmission time of m_i. Let us look at the example illustrated in Figure 3 where five messages are scheduled in the DYN segment with schedules $\Theta_1 = \{6, 1, 2\}$, $\Theta_2 = \{7, 0, 2\}$, $\Theta_3 = \{8, 0, 4\}$, $\Theta_4 = \{9, 2, 4\}$, and $\Theta_5 = \{5, 4, 8\}$. Let us compute the worst-case delay for message m_3 with $\Theta_3 = \{8, 0, 4\}$. The corresponding set of available cycles is defined as (see Section 2) $\Gamma_3 = \{0, 4, 8, 12, 16, \ldots, 60\}$, namely $\gamma_1 = 0, \gamma_2 = 4, \gamma_3 = 8, \ldots, \gamma_{16} = 60$. Further, we

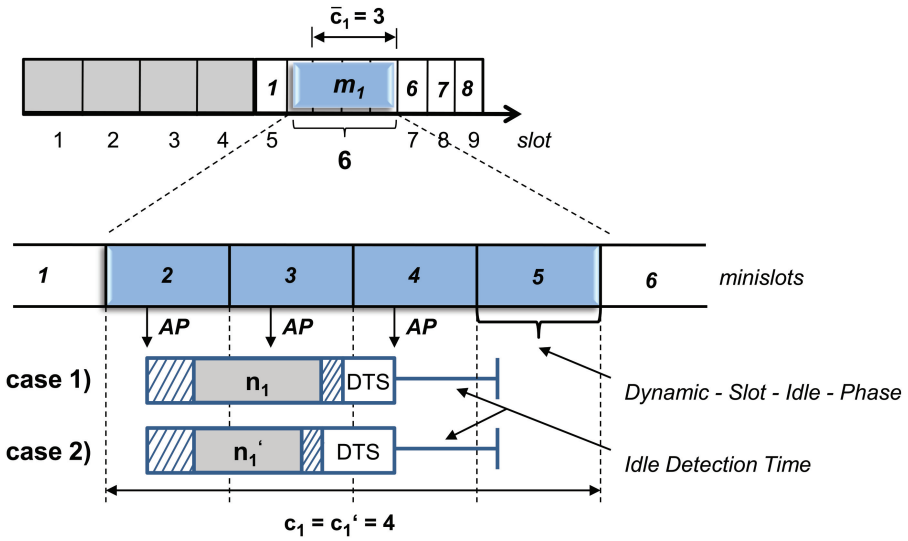

Fig. 4. Transmission scheme in the DYN segment. In *case 1*, message m_1 has a payload of n_1 whereas in *case 2*, the payload size for m_1 is $n_1' < n_1$. In both cases, the actual minislot consumption in the DYN segment is $c_1 = c_1' = 4$.

can see from the figure that the two messages m_2 and m_5 have a smaller slot number (higher priority) and interfere with m_3 in certain cycles, for instance, in cycle 4 the corresponding sets of feasible cycles of m_2 and m_5 are defined as $\Gamma_2 = \{0, 2, 4, 6, \ldots, 62\}$ and $\Gamma_5 = \{4, 12, 20, 28, 36, 44, 52, 60\}$. Hence, the set of message indices $\mathcal{G}_k$ that indicate the messages interfering with m_3 are defined as $\mathcal{G}_1 = \{2\}$, namely m_2, $\mathcal{G}_2 = \{2, 5\}$, namely m_2, m_5, $\mathcal{G}_3 = \{2\}$, $\mathcal{G}_4 = \{2, 5\}, \ldots, \mathcal{G}_{16} = \{2, 5\}$. Let $T_{bus} = 5ms$ and $\delta = 0.015ms$, then the worst-case delay for message m_3 can be computed as $D_3 = 4 \cdot 5ms + (1 + 2) \cdot 0.015ms + 2 \cdot 0.015ms = 20.075ms$ with $c_2 = 2$, $c_3 = 2$, and $c_5 = 3$ minislots. Note that (3) assumes that the delay is safely bounded and no message exceeds the value of $pLatestTx$, that is, is *displaced* to the next admissible cycle. To account for this we also compute the actual minislot counter and check whether $\mu_i < pLatestTx$. The number of empty slots with higher priorities than S_i according to $\mathcal{G}_k$ is bounded by

$$x_k = (S_i - 1) - N - |\mathcal{G}_k|, \tag{4}$$

where $|\mathcal{X}|$ denotes the cardinality of set $\mathcal{X}$. In our example we get $x_1 = 2$ because $S_3 = 8$ ($\Theta_3 = \{8, 0, 4\}$) and $N = 4$, further $\mathcal{G}_1 = \{2\}$, and hence $|\mathcal{G}_1| = 1$. Similarly, for the remaining cycles we get $x_2 = 1$, $x_3 = 2$, $x_4 = 1, \ldots, x_{16} = 1$. Finally, we formulate the schedulability test as

$$(D_i \leq d_i) \wedge (\mu_i < pLatestTx), \tag{5}$$

where the minislot counter is defined as

$$\mu_i = \max_{k \in \{1, \ldots, \frac{64}{R_i}\}} \left(\sum_{j \in \mathcal{G}_k} c_j + x_k \right) \tag{6}$$

and the delay D_i must respect the deadline d_i.

Workload estimation model. As discussed earlier, x_k denotes the number of empty slots corresponding to a certain group $\mathcal{G}_k$. These slots can be assigned to messages m_f in future iterations. However, to evaluate the compatibility test, the transmission times

Table I. Example: Number of Minislots c per Payload Size n

Subset of payload sizes $\mathcal{N}_i$	payload n in bytes	$f(n)$ minislots
$\mathcal{N}_1$	$\{2, 4, 6\}$	2
$\mathcal{N}_2$	$\{8, 10, \ldots, 20\}$	3
$\mathcal{N}_3$	$\{22, 24, \ldots, 36\}$	4
$\vdots$	$\vdots$	$\vdots$
$\mathcal{N}_{18}$	$\{246, 248, \ldots, 254\}$	19

of the future messages need to be estimated, which will be discussed in what follows. Let $\mathcal{N}$ be the set of feasible payload sizes $n \in \{0, 2, 4, \ldots, 254\}$ in bytes[2] in accordance with the FlexRay protocol. Let $\mathcal{C}$ be the set of message sizes c in terms of minislots, including protocol header and physical-layer properties. Then, according to FlexRay [2013] and Rausch [2008], there exists a mapping function $f : \mathcal{N} \rightarrow \mathcal{C}$ where $f(n)$ is a function of the message payload size n. In other words, there exists a quantization in the number of minislots such that messages with different payload sizes may consume the same number of minislots in the DYN segment.

Every message transmission in the DYN segments starts and ends at a predefined point within a minislot, called *Action Point offset* (AP). The number of bits to be transmitted includes the actual payload data n, and the fixed FlexRay protocol header and trailer segments per message, indicated by the hatched boxes before and after the payload segment in Figure 4. The figure shows an illustrative example for transmission of a message m_1 in slot 6 in the DYN segment. The figure shows two different cases.

—*case 1.* m_1 has a payload of size n_1. This results in an actual minislot consumption on the bus of $c_1 = 4$ minislots on the bus.
—*case 2.* m_1 has a payload of size $n_1' < n_1$. Here, even if the message would have a slightly smaller payload segment, the number of minislots required to transmit m_1 on the bus is $c_1 = c_1' = 4$.

As the payload segments of different messages may differ (compare case 1 and 2) whereas the *Action Point offset* and the length of a minislot are predefined and fixed in the protocol configuration, a message might complete its transmission somewhere within a minislot, for example, in both cases of Figure 4 the transmission ends at different points in minislot 3. For this reason, each message transmission is extended until the next *Action Point offset* by adding a so-called *Dynamic Trailing Sequence* (DTS). Hence, the transmission time on the bus is always exactly an integer multiple of a minislot. In addition, there is a configurable *idle detection time* and a *dynamic slot idle phase* during which no other ECU is allowed to transmit any messages, and hence this time also counts to the message being currently transmitted. As we can see from the figure, messages might have different payload sizes but might still consume the same number of minislots on the bus, such as $c_1 = 4$ minislots for payload n_1 and n_1'. Formally, there exist subsets $\mathcal{N}_i \subseteq \mathcal{N}$ with $\mathcal{N}_i := \{n \mid \forall n \in \mathcal{N}_i : f(n) = c_i\}$. In other words, for several sets of payload sizes $\mathcal{N}_i$ the corresponding resource consumption c_i in terms of minislots is constant. Table I shows how the minislot consumption is related to the payload sizes for the FlexRay configuration of our running example. As a result, future messages that might be scheduled with priorities higher than m_i's can be considered by an estimated workload of c_f minislots. Hence, we account for their effective transmission time using $\overline{e}_f = (c_f - 1) \cdot \delta$. Note that the choice of c_f bounds the effective transmission time $\overline{e}_f$ for future messages that may have payload sizes up to

[2]We assume that every message m_i has a fixed payload size of n_i that is statically decided.

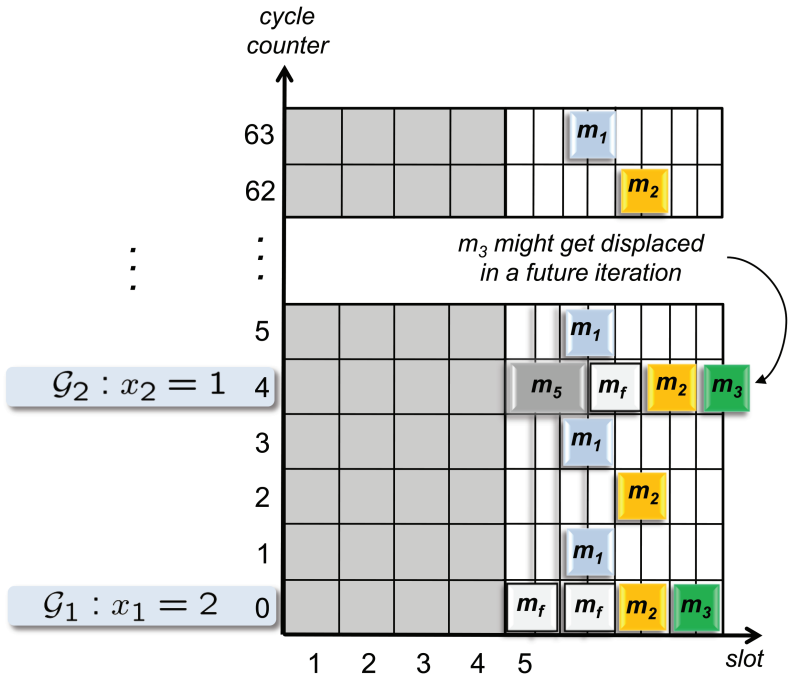

Fig. 5. Delay model and compatibility analysis.

$n \in \mathcal{N}_i$ for which $f(n) = c_f$. This allows for an expressive workload estimation as the system designer now does not need to know the exact payload sizes of future messages. In fact, it is sufficient to specify a range of expected payload sizes that can be bounded by a unique value of c_f and hence allows for an approximate payload estimation based on the designer's experience.

Compatibility test. In the following we present the schedulability condition for a compatible schedule. Using (3) and (4) the future worst-case delay $\bar{D}_i$ is computed as

$$\overline{D}_i = R_i \cdot T_{bus} + \max_{k \in \{1,\ldots,\frac{64}{R_i}\}} \left(\sum_{j \in \mathcal{G}_k} \overline{e}_j + x_k \cdot \overline{e}_f \right) + e_i \leq d_i. \tag{7}$$

Eq. (7) captures the interference due to already scheduled messages in the DYN segment as defined in (3). In addition, (7) considers the worst-case workload due to messages that might be scheduled in future iterations that consume up to c_f minislots and have an effective transmission time of $\bar{e}_f$, respectively. Note that the maximum interference from messages having a higher priority than m_i's is obtained by computing the maximum interference due to the existing messages and messages from future iterations among all cycles that might interfere with m_i's schedule. In other words, if a schedule Θ_i violates the deadline d_i while considering the possible messages in the future as well, then this schedule is not compatible. Let us look at the example illustrated in Figure 5 and evaluate (7) for message m_3 with $\Theta_3 = \{8, 0, 4\}$. Further, let us consider messages from future iterations m_f with $c_f = 2$ minislots as depicted in the figure. Recall that the value of c_f considers messages with several payload sizes. From Θ_2, Θ_3, and Θ_5 we again compute x_k for each $\mathcal{G}_k$ using (4). It can be seen that number of messages with a higher priority than m_i's can be different in every cycle, for instance,

in cycle 0, $x_1 = 2$, whereas in cycle 4, $x_2 = 1$. Even though in cycle 4 only one m_f can be assigned in the future, the resulting workload is more critical than in cycle 0, where two messages might be added. This is because the availability of minislots might expire in cycle 4 such that m_3 can get displaced in a future iteration. As a consequence, we also must consider the availability of minislots in future iterations. Hence, we require that the maximum minislot counter value $\overline{\mu}_i$ (considering the workload c_f of messages from future iterations) must not exceed the value of $pLatestTx$:

$$\overline{\mu}_i = \max_{k \in \{1,\ldots,\frac{64}{R_i}\}} \left(\sum_{j \in \mathcal{G}_k} c_j + x_k \cdot c_f \right) < pLatestTx. \tag{8}$$

If both (7) and (8) are fulfilled by any schedule Θ_i, such a schedule is referred to as *compatible*. Using (7) and (8) we formulate the compatibility test for the DYN segment.

$$(\overline{D}_i \leq d_i) \wedge (\overline{\mu}_i < pLatestTx) \tag{9}$$

Running example. Let us evaluate the compatibility test defined in (9) for design iteration $I = 1$ of our example. To achieve this, we consider future messages with a workload of $c_f = 4$ minislots. Consider, for example, message m_{10} that has been assigned the schedule $\Theta_{10} = \{101, 1, 2\}$ and a deadline $d_{10} = 22ms$. Currently, m_{10} easily meets its deadline constraints as the worst-case delay according to (3) is computed as $D_{10} = 11.305ms < d_{10}$ and the present minislot counter $\mu_{10} = 167 < pLatestTx$. Hence: (i) there is a sufficiently large slack of $10.695ms$, and (ii) enough minislots are available to transmit m_{10} in its assigned cycles. Even though m_{10} meets its deadline, considering messages from future iterations with $c_f = 4$ minislots resource consumption, the compatibility test in (9) fails because the worst-case minislot counter evaluates $\overline{\mu}_{10} = 317$, exceeding $pLatestTx = 238$. Consequently, Θ_{10} is not compatible although it meets its timing constraints at the current design iteration $I = 1$.

Similarly, message m_{94} with $\Theta_{94} = \{68, 3, 4\}$ also meets its deadline constraint at $I = 1$ because $D_{94} = 21.56ms$ and hence $D_{94} < d_{94} = 22ms$. Further, the maximum minislot counter $\mu_{94} = 152 < pLatestTx$. However, the compatibility test in (9) fails because the delay due to future messages is computed as $\overline{D}_{94} = 22.76ms$, which clearly may result in deadline violations at future design iterations $I > 1$. Thus, m_{10} and m_{94} will be declared not compatible by our compatibility test. Recall that this test is based on the presented workload estimation methods and the compatibility test and did not explicitly account for the actual messages in design iteration $I = 2$. Now, let us explicitly consider the messages from design iteration $I = 2$ and schedule 100 additional messages. We observe that Θ_{94} and Θ_{10}, marked as incompatible in the previous iteration, in fact violate their real-time requirements due to the presence of interfering messages from iteration $I = 2$. In particular, Θ_{94} now violates its deadline constraints, namely $D_{94} > d_{94}$. For Θ_{10}, the current maximum minislot counter μ_{10} is computed as $\mu_{10} = 241 > pLatestTx$, that is, m_{10} cannot be guaranteed to be transmitted in its assigned cycles.

4.2. Compatibility Index

Finally, we denote the number of schedules that pass the compatibility test as the *compatibility index* according to

$$\xi = \frac{|\mathcal{Q}|}{|\mathcal{M}|}, \tag{10}$$

where $\mathcal{M}$ denotes the set of all messages to be scheduled and $\mathcal{Q} \subseteq \mathcal{M}$ denotes the subset of messages that have been assigned compatible schedules, defined as $\mathcal{Q} := \{m_i \mid \forall m_i \in$

$\mathcal{Q} : (\overline{D}_i \leq d_i) \wedge (\overline{\mu}_i < pLatestTx)\}$. We will use ξ as a performance index in Section 7.1 to compare the compatibility capabilities of different scheduling algorithms.

4.3. Extensions to Other Delay Models

The defined notion of compatibility and the corresponding analysis introduced in this section are based on the delay model presented in Section 4.1. As already mentioned in Section 1, there exist several works on timing analysis for the FlexRay DYN segment that captures different levels of protocol details, provides different accuracy in the message delay estimates, and uses different models of computation. It is also foreseeable that new delay models will come up in the future that may further improve existing approaches. Hence, a natural question that may arise in this context is: how general is the presented compatibility analysis and how may it be integrated into other delay models? For this purpose, we want to consider recent works that compute upper bounds on the displacement of messages over multiple cycles in the DYN segment (e.g., Tanasa et al. [2012] and Neukirchner et al. [2012]) and thereby allow computation of less pessimistic worst-case delays compared to the delay model we use in this work.

Recall the definition of COMP1 that refers to a schedule as *compatible* iff the deadline d_i is guaranteed even in the presence of future messages. Essentially, we must check whether

$$\overline{D}_i \leq d_i, \tag{11}$$

where $\overline{D}_i$ denotes the future worst-case delay. For the sake of simplicity, in this work we computed $\overline{D}_i$ by extending the delay model of (3). However, this meant that we should be able to predict the workload of the future messages and thus we proposed the workload estimation model.

We want to emphasize that our workload estimation model can also be utilized by recently developed models (e.g., Tanasa et al. [2012] and Neukirchner et al. [2012]) to compute $\overline{D}_i$. Both, Tanasa et al. [2012] and Neukirchner et al. [2012] transformed the problem of computing the worst-case delays of messages on the DYN segment of FlexRay into a bin covering problem and then proposed different heuristics to solve the equivalent problem. As an input, they require the set of messages to be transmitted on the DYN segment as well as certain characteristics of these messages. These characteristics include the payload n_i of the messages and their schedule Θ_i. We have already discussed how to conservatively predict all these characteristics and the same techniques may also be used to find the input required by Tanasa et al. [2012] and Neukirchner et al. [2012]. First, the set of messages to be considered in the future consists of the existing messages as well as messages that may be accommodated in the empty higher-priority minislots in the existing iteration. For the models proposed in Tanasa et al. [2012] and Neukirchner et al. [2012], we propose to introduce one message for every empty minislot (this defines their priority) and for every cycle (this defines their base cycle B_i). These messages are assigned a repetition rate of $R_i = 64$. This repetition rate is chosen because as we have introduced one new message for every empty slot in the 64-cycle matrix, any repetition rate R_i less than 64 will add to duplicate messages and result in unnecessary pessimism. The payload size for these messages can be predicted exactly in the same fashion that we discussed in our proposed workload estimation model.

Also note that COMP2 and COMP3 retain their validity by definition. With this discussion, we want to convey that our metric on compatibility as well as our proposed workload estimation model can be used in a seamless plug-and-play fashion with other delay models that have been proposed in the literature. For simplicity of exposition, however, in this article we have used a relatively straightforward delay model.

5. EXTENSIBILITY ANALYSIS

The compatibility analysis discussed so far does not quantify the availability of schedules that may be assigned to messages added in future iterations. To account for this, we specify the notion of *extensibility* in what follows. The basic idea of extensibility for FlexRay schedules in an incremental design scenario is: given a FlexRay network with existing messages, extensibility is a property that quantifies the quality and number of schedules available to accommodate messages in future iterations. Provided existing schedules are not changed at any time, we introduce the following concepts.

—*EXT1.* The *grade of extensibility* indicates the number of available schedules that may be assigned to any messages in a particular slot S, that is, the availability of schedules Θ_f for future messages.
—*EXT2.* The *quality rating* of a slot S quantifies the real-time capabilities for future messages (e.g., slots with high priorities have higher-quality ratings).
—*EXT3.* The *extensibility index* combines the quality rating and grade of extensibility into a joint metric. This metric is used as an optimization objective in the scheduling framework we present in Section 6 to synthesize extensible schedules.

We will first explain the aforementioned concepts in detail and then introduce the *effective network extensibility* used as a performance measure to compare the extensibility capabilities of scheduling algorithms (see Section 7.1).

5.1. Grade of Extensibility

Let us consider the number of available schedules for a message m_i that is to be scheduled in slot S_i. Within S_i, m_i can be assigned any of the admissible repetition rates $R_i \in \{1, 2, 4, 8, 16, 32, 64\}$. For each of these repetition rates, there are R_i base-cycle values B_i available, where $B_i < R_i$. For example, if $R_i = 2$, m_i might be assigned a base cycle $B_i \in \{0, 1\}$, if $R_i = 4$, then $B_i \in \{0, 1, 2, 3\}$, and so on. Thus, the total number of available choices to schedule m_i in slot S_i is computed as $C(S_i) = \sum_{R_i} f(R_i),\ \forall R_i \in \{1, 2, 4, 8, 16, 32, 64\}$, where $f(R_i)$ denotes a function returning the number of available base cycles B_i for repetition rate R_i. As discussed before, $f(R_i) = R_i$ if there is no other message scheduled in slot S_i. On the other hand, let us consider that existing schedules from previous iterations already utilize some cycles of slot S_i and a message m_i is to be scheduled in the same slot. In this case, the number of available repetition rates R_i and base cycles B_i is less because of the presence of the existing schedules. For instance, let us consider the message m_4 with $\Theta_4 = \{9, 0, 2\}$ and study the influence of its schedule on the possible schedules for any m_i that is to be scheduled in the same slot. Here, $R_i = 1$, $B_i = 0$ is not available as the schedule $\Theta_i = \{9, 0, 1\}$ interferes with m_4's schedule in every even cycle as $\Gamma_4 \cap \Gamma_i = \{0, 2, 4, \ldots, 62\} \neq \emptyset$ and hence violates the slot multiplexing condition. Consequently, $f(1) = 0$. Concerning the other repetition rates, only half of the possible choices are available for m_i, namely $f(R_i) = \frac{R_i}{2}$ for $R_i \in \{2, 4, 8, 16, 32, 64\}$, as every even base cycle B_i is unavailable due to the presence of m_4's schedule and $C(S_i) = 0 + 1 + 2 + 4 + 8 + 16 + 32 = 63$. As the total number of available schedules in any empty slot $\hat{S}$ is computed as $C(\hat{S}) = \sum_{r=0}^{6} 2^r = 127$, we compute the grade of extensibility $P_1(S_i)$ of a slot S as

$$P_1(S) = \frac{C(S)}{C(\hat{S})}. \tag{12}$$

Let us look at another example. Consider the schedules $\Theta_1 = \{6, 1, 2\}$, $\Theta_2 = \{6, 0, 4\}$, $\Theta_3 = \{7, 0, 1\}$, and $\Theta_4 = \{9, 0, 2\}$. In slot 6, Θ_1 and Θ_2 are already assigned and cycles

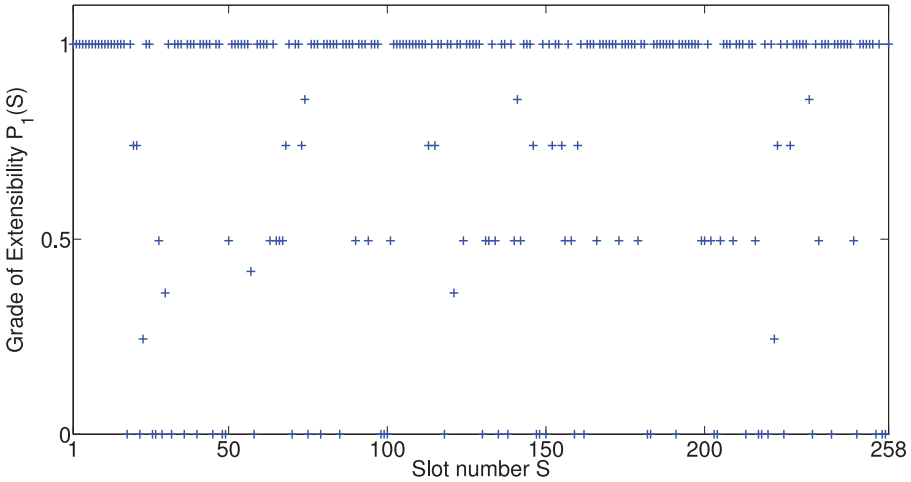

Fig. 6. Grade of extensibility for I = 1.

available to m_i in slot 6 are denoted by $(2 + n \cdot 4), n \in \{0, 1, \ldots, 15\}$. Using (12), we obtain $P_1(6) = \frac{0+0+1+2+4+8+16}{127} = 0.2441$, that is, 24.41% of all schedules are available to schedule any m_i in slot 6. Further, even though every fourth cycle is available, $P_1(6) \neq 0.25$ as schedules Θ_i where $R_i = \{1, 2\}$ are not available. In every cycle, slot 7 is already assigned m_3 (see Θ_3), hence $P_1(7) = 0$, and in every second cycle, slot 9 is assigned m_4 (see Θ_4), thus $P_1(9) = 0.4961$. This measure can be applied to the ST and DYN segment in the same manner. In fact, this metric allows us to quantify the grade of extensibility to accommodate messages in any slot $S \in \mathcal{S}$.

Running example. The grade of extensibility is illustrated in Figure 6. The figure shows $P_1(S)$ for 100 schedules generated for design iteration $I = 1$. Note that points in the figure where $P_1(S) = 0$ indicate slots where all cycles are allocated by schedules, such as $\Theta_1 = \{S_1, 0, 1\}$, or several slot-multiplexed schedules such as $\Theta_1 = \{S_1, 0, 2\}$ and $\Theta_2 = \{S_2, 1, 2\}$, with $S_1 = S_2$. This implies that no schedules are available for any further messages in that slot. In contrast, points where $P_1(S) = 1$ denote empty slots where all feasible schedules are available for future messages, that is, the grade of extensibility is maximum. $P_1(S) = 1$ is especially prominent in the ST segment, namely $S \in \{1, \ldots, 17\}$, as no messages have been scheduled in static slots at any iteration I. Points in the range $0 < P_1(S) < 1$ indicate slots where one or more slot-multiplexed schedules partially allocate cycles in S and therefore constrain the number of available schedules for messages in future iterations.

5.2. Quality Rating of Extensibility

In order to cope with a quantification of the quality rating of slots, we introduce a quality rating function $P_2(S)$ that specifies a weight for every communication slot $S \in \mathcal{S}$. Effectively, $P_2(S)$ maps a weight to every slot according to its ability to provide real-time guarantees for future messages, for example, according to slot priorities. We define a quality rating function $P_2(S)$ as follows:

$$P_2(S) = \begin{cases} 0, & \forall S \in \mathcal{R} \\ 1, & \forall S \in \mathcal{S}_{ST} \backslash \mathcal{R} \\ 1 - e^{-k\left(\frac{|S-(N+M)|}{S-(N+1)}\right)}, & \forall S \in \mathcal{S}_{DYN} \backslash \mathcal{R}. \end{cases} \tag{13}$$

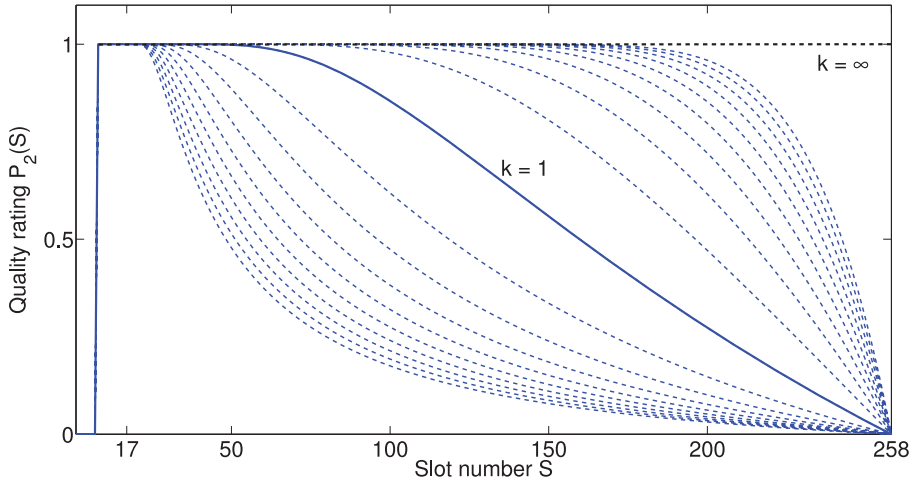

Fig. 7. Quality rating of slots.

Eq. (13) specifies different quality ratings for: (i) *reserved* slots, (ii) *static* slots, and (iii) *dynamic* slots and is discussed in detail in what follows.

(*i*) *Reserved slots.* The first term in (13) accounts for reserved communication slots $S \in \mathcal{R}$ that are supposed to be used for system functions or specific protocols (see Section 2). For these slots the schedule assignment is often predefined according to specific rules or standards and hence we do not quantify their extensibility, namely $P_2(S)$ evaluates zero. In our running example (see Figure 7), we used $\mathcal{R} = \{1, \ldots, 7\}$, that is, the first 7 slots in the ST segment are reserved and cannot be assigned to application messages. Note that slots can also be reserved in the DYN segment.

(*ii*) *Static slots.* The second term in (13) specifies the quality rating of static slots that are provided for scheduling application messages. All slots in the ST segment are of fixed, equal length and indicate equal priorities, that is, messages cannot experience interference from any other messages transmitted in the ST segment. Consequently, we weight the static slots with a constant maximum rating factor $P_2(S) = 1,\ \forall S \in \mathcal{S}_{ST} \backslash \mathcal{R}$, where $\mathcal{S}_{ST} = \{1, \ldots, N\}$. This is also reflected in Figure 7 where $P_2(S) = 1,\ \forall S \in \{8, \ldots, 17\}$.

(*iii*) *Dynamic slots.* The third term in (13) accounts for slots in the DYN segment that are not reserved, namely for $S \in \mathcal{S}_{DYN} \backslash \mathcal{R}$, where $\mathcal{S}_{DYN} = \{N + 1, \ldots, N + M\}$. The event-triggered communication paradigm in the DYN segment implies that the quality rating function must change weights according to the slot priorities. However, the rate of change is not constant because: (i) real-time guarantees can only be provided for messages assigned to *low* slot numbers (high priorities) where delays are bounded (see (5)); and (ii) real-time guarantees cannot be provided for messages assigned to *high* slot numbers (low priorities) as messages can only be transmitted in a certain cycle if sufficiently many minislots are available according to *pLatestTx*. Therefore, for the DYN segment, we propose a quality rating function according to (13). The exponential drop-off in (13) is based on the results presented in Zeng et al. [2010] and Neukirchner et al. [2012] showing that the message response times in the DYN segment exponentially increase with the slot number S. Thus, an exponential drop-off in $P_2(S), \forall S \in \mathcal{S}_{DYN}$ is a reasonable design choice to measure the quality rating of slots. Here, high-priority

slots are quantified with a high-quality rating, namely $P_2(S)$ evaluates towards 1 for $S \rightarrow (N+1)$. Similarly, $P_2(S)$ evaluates towards zero for $S \rightarrow (N+M)$. Intuitively, this indicates that messages assigned to high slot numbers are more likely to miss their deadlines due to the interference with messages having higher priorities, which may result in displacements and large delays, respectively. On the contrary, the messages assigned to low slot numbers (at the beginning of the DYN segment) hardly experience interference with other messages and hence their timing behavior is not much disturbed. Consequently, using a quality rating function, the system designer: (i) may determine for which slots extensibility is considered very important (slots have a high quality) and for which extensibility is considered to be of weak importance (slots have low quality); and (ii) accordingly direct the scheduling synthesis. Note that the characteristics of $P_2(S)$ may also be dynamically adapted in every iteration I based on the actual workload w of the already scheduled messages. This may be realized by formulating $k(\frac{1}{w})$ in (13) as a function of the workload. In the extreme case at $I = 0$ where none of the minislots is allocated to any messages, we have $w = 0$, which yields $k_I(\frac{1}{w}) = \infty$ and hence $P_2(S) = 1, \forall S \in \mathcal{S}_{DYN}$ as depicted in Figure 7. That is, if no message is assigned any slot, the quality rating of all slots is maximum. Further, with every design iteration, we may increase $k_I(\frac{1}{w})$ based on the actual workload w in the DYN segment resulting in a decrease of $P_2(S)$ as indicated in Figure 7 for different values of k for $\mathcal{S}_{DYN} = \{18, \ldots, 258\}$. For ease of exposition, in the rest of this article we assume $k = 1$.

5.3. Extensibility Index

In order to avoid the cumbersome process of interpreting multiple metrics, we introduce an *extensibility index* $E(S)$, a holistic quantification of extensibility that not only depends on the grade of extensibility but also on the quality rating

$$E(S) = P_1(S) \cdot P_2(S), \ \forall S \in \mathcal{S}. \tag{14}$$

Since $P_2(S)$ indicates a measure for the real-time guarantees of S such that $1 \leq P_2(S) \leq 0$, the extensibility index $E(S)$ denotes the grade of extensibility $P_1(S)$ of slot S weighted according its quality rating. In other words, $E(S)$ represents a joint metric that indicates the extensibility of a certain slot S with respect to the number of schedules it provides for future messages in conjunction with the slots' real-time guarantees.

Running example. Figure 8 depicts the extensibility index $E(S), \ \forall S \in \mathcal{S}$ for design iteration $I = 1$. As the extensibility index represents the grade of extensibility $P_1(S)$ weighted by the quality rating $P_2(S)$ for every $S \in \mathcal{S}$, points along $P_2(S)$ indicate empty slots, namely full availability of schedules. As illustrated in the figure, $E(S)$ dramatically decreases for high slot numbers $S \rightarrow 258$ even if all schedules are available in a slot S. This accounts for the low-priority slots where future messages might experience high interference from other messages and hence will less likely satisfy real-time guarantees. Thus, extensibility in these slots is weighted less according to the quality rating $P_2(S)$. On the other hand, Figure 8 reflects a high grade of extensibility for low slot numbers, indicating that these slots might easily accommodate high-priority messages. Note that $E(S) = 0, \forall S \in \mathcal{R} = \{1, \ldots, 7\}$ because we do not consider reserved slots in our extensibility analysis according to $P_2(S)$. Further, $E(S) = 1, \forall S \in \mathcal{S} \backslash \mathcal{R}$ as no messages have been scheduled in the ST segment and hence $E(S) = P_1(S) = 1$. In essence, a visual reading of the extensibility index graph in Figure 8 reveals the following.

—$E(S_i) = 0$. Slots that are not extensible at all are denoted by points on the x-axis. This follows from the definition of the quality rating function where slots that cannot be assigned any schedule evaluate to $P_1(S) = 0$ and hence, according to (14), $E(S) = 0$.

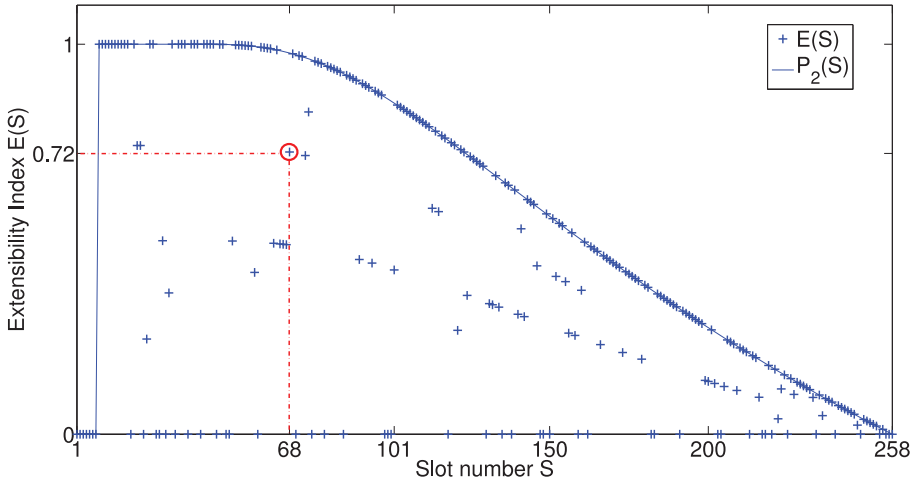

Fig. 8. Extensibility index for I = 1.

—$E(S) = P_2(S)$. Slots that provide maximum extensibility, namely slots that are not assigned any schedules, are denoted by the points located at the top of the graph that coincide with $P_2(S)$ (see continuous line in the figure). Here, the grade of extensibility $P_1(S) = 1$ is maximum as all schedules are available at each of the slots and hence $E(S) = P_2(S)$. Note that extensible slots with high priorities are located close to $E(S) = 1$ according to the quality rating function.
—$0 < E(S) < P_2(S)$. Slots that are already utilized by schedules but not assigned the slot in every cycle are denoted by points between $0 < E(S) < P_2(S)$. Depending on the number of available schedules, the points are either located: (i) close to $P_2(S)$, that is, many schedules are available as, for example, in slot $S = 68$, where $E(68) = 0.72$; or (ii) in the lower half of the graph that indicates the availability of only few schedules.

Figure 9 shows the updated extensibility index for iteration $I = 2$ where 100 additional messages have been scheduled in the DYN segment. Note that the overall extensibility decreases as many points have dropped from the reference curve $P_2(S)$ compared to $I = 1$ (see Figure 8). Observe that many high-priority slots not assigned previously have now been assigned to the new messages, for example, in Figure 9 at points $S \in \{18, \ldots, 50\}$, where $E(S) < P_2(S)$ compared to Figure 8 where $E(S) = P_2(S)$. Compared to Figure 8, many more points superimpose the x-axis, implying that many slots are now completely allocated. Similarly, slots that have been partially allocated have now accumulated along a second curve segment around at $0.5 \cdot E(S)$ as illustrated in Figure 9. Further, the extensibility index $E(68)$ decreased from 0.72 to 0.48 between the two iterations, that is, a new message has been scheduled for $I = 2$ using slot multiplexing, and thus $E(68)$ decreased.

5.4. Effective Network Extensibility

Our motivation for introducing a single metric to encapsulate the extensibility of *all* slots instead of a slot-by-slot metric is the following. If one metric can capture the behavior of the FlexRay network in a holistic fashion, it can be used by optimization frameworks that synthesize schedules subject to extensibility. In the next section, we will propose one such scheme. In contrast, a quantified value per slot would lead to ease of visual interpretation (as shown previously in this section) and can be easily adapted by system designers in the industry for human-intervened schedule design

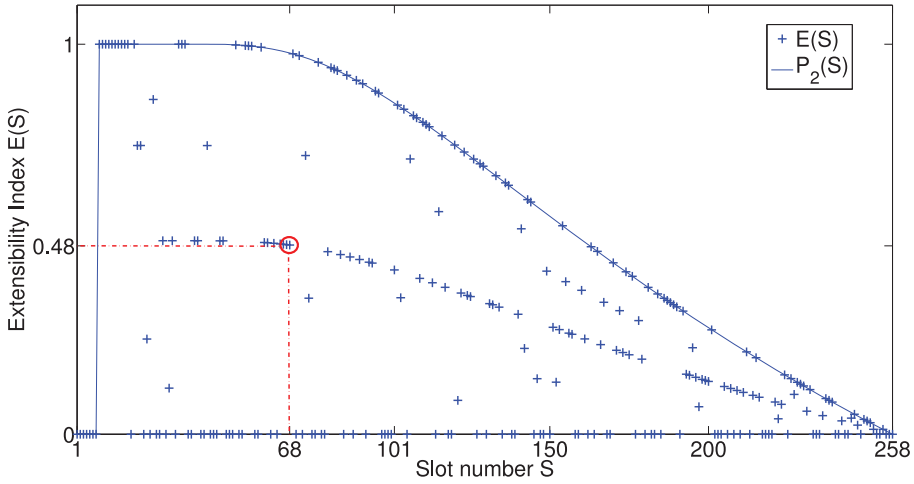

Fig. 9. Updated extensibility index for I = 2.

as well. Towards introducing a single holistic metric, we first separately consider the ST and the DYN segment extensibility and condense them into one metric. The total extensibility of the ST and DYN segment may be computed as

$$E_{ST} = \frac{1}{N}\sum_{S=1}^{N} E(S),\ E_{DYN} = \frac{1}{M}\sum_{S=(N+1)}^{N+M} E(S), \tag{15}$$

where N is the number of static slots and M the number of minislots in the DYN segment. The extensibility of the entire FlexRay network is computed as

$$E_{FR} = \frac{1}{N+M}\sum_{S=1}^{N+M} E(S). \tag{16}$$

The previous formula essentially denotes the summation of the extensibility index $E(S)$ over all slots in the FlexRay network ranging from the first slot $S = 1$ to the last slot $S = N + M$ in the DYN segment normalized by the total number of slots $N + M$. While the aforesaid metric quantifies the extensibility of FlexRay schedules, it does not consider the actual schedulability of messages for any synthesis algorithm. However, the schedulability has a significant impact on the extensibility. To account for this, we introduce the concept of *effective network extensibility* that we explain in what follows.

Let us assume we want to schedule five new messages $\mathcal{M} \in \{m_1, m_2, m_3, m_4, m_5\}$ on top of an existing network using two different schedulers Algorithm 1 and Algorithm 2. We are interested in studying which scheduling algorithm generates more extensible schedules. Let us assume that Algorithm 1 assigns a feasible schedule to all five messages, which results in a certain network extensibility according to (16), such as $E_{FR}^{alg1} = 0.48$. Further assume that Algorithm 2 is only able to schedule three out of the five messages resulting in a higher network extensibility of $E_{FR}^{alg2} = 0.49$ than in the case of Algorithm 1 and hence indicates a better extensibility. This is expected as we only accommodated three new messages, whereas in the case of Algorithm 1 we scheduled all five messages. This obviously requires more resources as we accommodated two more messages. Thus, to consider the schedulability in the extensibility

analysis, we introduce the *effective network extensibility* E_{eff} defined as

$$E_{eff} = \max\left(E_{FR} - \frac{\kappa(|\mathcal{M}| - |\mathcal{L}|)}{(N+M)}, 0\right), \tag{17}$$

where E_{FR} denotes the network extensibility as defined in (16), $\mathcal{M}$ denotes the set of all messages to be scheduled, and $\mathcal{L} \subseteq \mathcal{M}$ is the subset of all messages that actually have been assigned a feasible schedule. The term $(|\mathcal{M}| - |\mathcal{L}|)$ represents the number of messages that could not be assigned feasible schedules, therefore these messages should not contribute to increasing the overall network extensibility E_{FR} but rather deteriorate the extensibility. To account for this, we introduce a penalty, denoted by κ, that represents the worst-case decrease in extensibility for any message $m_i \notin \mathcal{L}$. The worst-case extensibility decrease κ is derived by the maximum possible extensibility index in the ST or DYN segment, namely $\kappa = \max_S(E(S))$ with $S \in \{1, \ldots, N\}$ for the ST segment and $S \in \{N+1, \ldots, N+M\}$ for the DYN segment. In the preceding example, we have $(|\mathcal{M}| - |\mathcal{L}|) = 2$ for Algorithm 2 as two out of five messages could not have been scheduled because $|\mathcal{M}| = 5$ and $|\mathcal{L}| = 3$. Further, assume that the maximum possible extensibility index is $\kappa = \max_S(E(S))$, such as $\kappa = 1$. Consequently, for Algorithm 2 and 100 slots, we get $E_{eff}^{alg2} = \max(0.49 - \frac{1 \cdot 2}{100}, 0) = 0.47$ which is smaller than $E_{FR}^{alg2} = 0.49$. For Algorithm 1 we get $E_{eff}^{alg1} = E_{FR}^{alg1} = 0.48$ as $(|\mathcal{M}| - |\mathcal{L}|) = 0$ and hence $E_{eff}^{alg1} > E_{eff}^{alg2}$. Note that, in case all messages are schedulable with any algorithm, the term $(|\mathcal{M}| - |\mathcal{L}|)$ in (16) evaluates to zero which finally yields $E_{eff} = E_{FR}$.

Section 7.1 illustrates how the effective network extensibility is used to compare the extensibility performance of different algorithms.

5.5. Extensions to Other Delay Models

The presented concepts of extensibility introduced in Sections 5.1, 5.2, and 5.3 are generic in nature as they solely depend on the FlexRay-specific schedule parameters R_i, B_i, and S_i. Hence, the extensibility analysis and the associated notions EXT1 (*grade of extensibility*), EXT2 (*quality rating*), and EXT3 (*extensibility index*) do not depend on the underlying delay model. This will be discussed in what follows.

First, recall that EXT1 indicates the number of available schedules that may be assigned to any messages in a particular slot S. This corresponds to the availability of schedules Θ_f for future messages according to (12), which is computed based on the number of available base cycles per repetition rates in a particular slot. Hence, the output of (12) does not consider any timing properties of the messages but rather the availability of FlexRay schedules.

Second, EXT2 quantifies the real-time capabilities of a slot S_i for future messages. That is, slots with high priorities (small slot numbers) have higher-quality rating compared to slots with low priorities (high slot numbers). The quality rating function in (13) only depends on the parameters N and M, the number of static slots and number of minislots, respectively, and k that is a user-defined constant. Therefore, there is no direct relation between the actual message delays D_i and the quality rating function and thus (13) does not depend on the delay model under consideration.

Further, the extensibility index EXT3 is also independent of the delay model as it is defined as the product of EXT1 and EXT2 according to (14).

Finally, the concept of effective network extensibility presented in Section 5.4 requires as an input the set of messages $\mathcal{L}$ that have been assigned feasible schedules, in order to evaluate Eq. (17). Naturally, the schedulability of a message depends on its delay D_i and its associated deadline d_i. Consequently, the number of messages $|\mathcal{L}|$ that have been assigned feasible schedules depends on the worst-case delay estimates computed using a dedicated delay model. Although the feasibility check for any schedule

depends on the message delay, the proposed concept of effective network extensibility in (17) is not affected. This is because (17) just requires the *number* of messages that have been assigned feasible schedules. Note that it is not necessary to modify the preceding notion of effective network extensibility when using a different delay model, however, the derived results for network extensibility may be affected. For instance, results derived with the delay model of (3) may be more conservative (resulting in a lower value of E_{eff}) compared to the value of E_{eff} using one of the models that allow for computing tighter worst-case delays (e.g., Tanasa et al. [2012] and Neukirchner et al. [2012]).

6. SCHEDULE SYNTHESIS

In this section, we are interested in synthesizing extensible schedules according to the proposed metrics. For n messages, the possible set of schedules is exponential in the order of C^n in the worst case with C a constant. Hence, any approach based on enumerating all possible schedules does not scale and therefore we propose heuristics to synthesize schedules following the metrics introduced earlier. We show that the introduced heuristics have reasonable runtimes (in the order of 10–15s) for real-life industry-strength problems.

6.1. Real-Time Constraints

Two key parameters for a FlexRay schedule that affect message delays are the repetition rate R_i and the slot number S_i, where R_i determines the number of communication cycles a message is blocked in case it missed its slot in a cycle. Hence, the choice of both parameters is crucial to guarantee real-time constraints of messages. Given a repetition rate R_i, the FlexRay bus potentially provides the assigned slot S_i to transmit a message m_i every R_i cycles, namely within a time interval of $R_i \cdot T_{bus}$. We define the sampling period of the FlexRay bus for any message m_i as the interval length $R_i \cdot T_{bus}$. Note that the value of $R_i \cdot T_{bus}$ must: (i) satisfy the minimum interarrival distance p_i of two consecutive data items processed by the sender task, and (ii) be upper bounded by the message deadline d_i. In such a scenario, $p_i = R_i \cdot T_{bus}$ is a sufficient condition to guarantee the transmission of every instance of m_i. Hence, we upper bound the repetition rate by $R_i \leq \frac{\min(p_i,d_i)}{T_{bus}}$. As the repetition rate is only defined by a power of two and is limited to 64 (see Section 2), the maximum repetition rate $R_{max,i}$ is computed according to Schneider et al. [2010] as

$$R_{max,i} = \min\left(2^{\left\lfloor \log_2\left(\frac{\min(p_i,d_i)}{T_{bus}}\right)\right\rfloor}, 64\right). \tag{18}$$

In contrast to the ST segment, the locations of the communication slots in the DYN segment may vary from cycle to cycle as they depend on the actual transmission of messages that are assigned higher priorities, namely smaller slot numbers (see Section 2). Thus, in the DYN segment, the precise time at which a message is ready for transmission can be before or after the corresponding dynamic communication slot was ready. Consequently, a bus sampling rate equal to the maximum message rate does not provide safe transmission points. In the worst case, data cannot be transmitted on the bus before a new value is produced by the application. This is demonstrated in the example depicted in Figure 10(a). We consider message m_5 with $p_5 = 2T_{bus}$ and $d_i = p_i$. Using (18) yields $R_5 = 2$. Then, in the worst case, m_5 just missed its slot S_5 in cycle 0 at t_0, and at $t_1 = t_0 + p_5$ the next slot S_5 is not yet ready as messages having higher priority than m_5 are transmitted first (see the figure). Thus, the first instance of m_5 is overwritten by the second instance of m_5 that finally gets transmitted on the bus in cycle 2. To reflect the sampling constraints in the DYN segment, we require

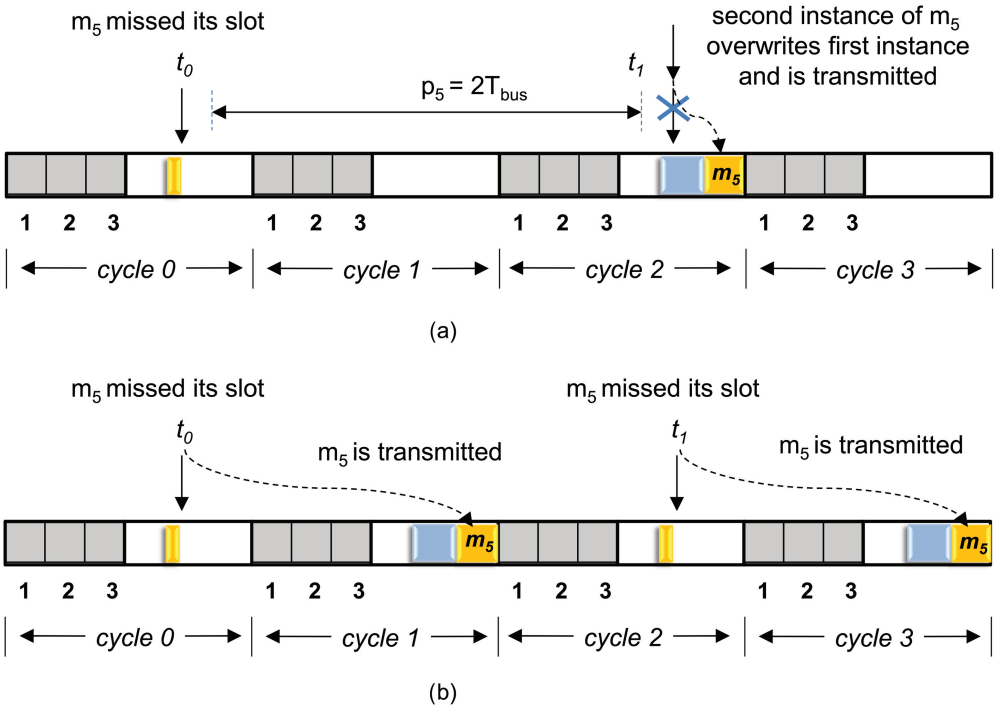

Fig. 10. Scenarios for message transmissions with different repetition rates. In scenario (a) m_5 is assigned repetition rate $R_5 = 2$. In scenario (b) m_5 is assigned repetition rate $R_5 = 1$.

that the FlexRay sampling rate has to be at least *twice* the message activation rate. Consequently, (18) yields

$$R_{max,i} = \min\left(2^{\left\lfloor \log_2\left(\frac{\min(p_i, d_i)}{2T_{bus}}\right)\right\rfloor}, 64\right) \tag{19}$$

with $B_{max,i} < R_{max,i}$ for the maximum base cycle $B_{max,i}$. Using (19), we now get $R_5 = 1$ as illustrated in Figure 10(b). Since m_5 is now assigned a slot in every cycle, it can be safely transmitted on the bus before the next instance of m_5 is computed and written to the transmit buffer of the FlexRay controller. Consequently, (19) provides a safe bound on $R_{max,i}$ considering the interference of messages with higher priorities in the DYN segment.

6.2. Schedule Synthesis Scheme

We present three heuristics for schedule synthesis: (i) Max-slack, (ii) First-hit, and (iii) Max-E. The latter implements the presented metrics for schedule synthesis, whereas Max-slack and First-hit use intuitive approaches to generate extensible schedules. In Section 7.1 we compare the performance of the heuristics and show that extensibility is best achieved using the Max-E approach.

Max-slack heuristic. First, we introduce a heuristic that only optimizes schedules for slack, defined as $s_i = d_i - D_i$ (difference between deadline and message delay). This means a small slack for m_i may lead to deadline violations if interfering messages having a higher priority are scheduled in future iterations. Note that the slack not only depends on the slot number S_i but also on the interference due to higher-priority

messages in every cycle $\gamma \in \Gamma_i$ of S_i and the bus blocking time $R_i \cdot T_{bus}$. Maximizing slack is an intuitive approach to realize compatibility without actually implementing the compatibility test (see (9)). The Max-slack heuristic involves the following steps.

—Sort the messages m_i to be scheduled according to deadlines in ascending order, that is, messages with small deadlines are scheduled first (denoting a high priority).
—For each m_i iterate over all feasible schedules $\Theta_i = \{S_i, R_i, B_i\}$ in the DYN segment, namely schedules that do not interfere with existing messages, with: (i) slots $(N+1) \leq S_i \leq (N + M)$, (ii) repetition rates $R_i \in \{1, 2, 4, 8, 16, 32, 64\}$, and (iii) base cycles $B_i < R_i$.
—Check for schedulability (see (5)); in case the schedulability test evaluates to *true*, then $\Theta_i = \{S_i, B_i, R_i\}$ is a valid schedule and we compute the slack s_i, otherwise discard Θ_i. If there exists no feasible Θ_i for a particular message m_i, then the message is referred as *unschedulable* and no schedule is assigned m_i.
—Finally, return schedule $\overline{\Theta}_i$ with maximum slack.

First-hit heuristic. As discussed in Section 5, the higher the repetition rates R_i of the existing messages, the more cycles are available for future messages to be scheduled in the same slot and hence the higher the grade of extensibility $P_1(S)$. Further, according to the proposed quality rating function $P_2(S)$, the priority of a slot is maximum and constant in the ST segment and decreases with increasing slot numbers in the DYN segment. Hence, our goal is to assign the maximum possible repetition rate $R_{max,i}$ and the highest feasible slot number S_i, namely the slot with lowest priority, where Θ_i is compatible. The First-hit schedule fulfills both the constraints. Note that, compared to the Max-slack heuristic, the First-hit heuristic actually implements the compatibility test defined in (9) to synthesize the schedules. This involves the following.

—Compute $R_{max,i}$ for every m_i according to (19) and sort the messages m_i to be scheduled in ascending order according to $R_{max,i}$. The smaller the $R_{max,i}$, the more resources (cycles) are necessary to schedule m_i and the fewer resources are available to schedule future messages. Therefore we start the scheduling engine with those messages having the tightest resource requirements, namely the smallest $R_{max,i}$, and hence we assign $R_i = R_{max,i}$.
—Iterate over all slots in the dynamic segment in descending order starting with the highest available slot number $S_i = N + M$ until reaching the first dynamic slot $S_i = N + 1$. Note that we need not iterate over all repetition rates as in the case of the Max-slack heuristic. Instead, we directly compute $R_{max,i}$ and select $B_i < R_{max,i}$ and hence reduce the design space.
—Check for compatibility (see (9)); in case the compatibility test evaluates to *true*, then return $\Theta_i = \{S_i, B_i, R_i\}$ and repeat the procedure with the next message. In case no compatible schedule can be found or the protocol constraints are violated, discard Θ_i. Note that this heuristic does not explicitly make use of the extensibility index $E(S_i)$ but intuitively realizes an extensibility objective by assigning slots as late as possible and repetition rates as high as possible.

Max-E heuristic. Finally, we present an algorithm that implements both the concepts, the compatibility test and the extensibility index, to synthesize the schedules (see Algorithm 1). We first compute the maximum available repetition rates $R_{max,i}$ and sort the message indices $i \in I$ in ascending order according to $R_{max,i}$ (line 1 and 2). Next we evaluate the slots in ascending order starting with the first dynamic slot $S_i = N + 1$ (line 4) and compute the actual extensibility index $E(S_i)$ of slot S_i before assigning the message m_i to the slot (line 6). Next, we check if the available repetition rates bounded by $R_i < R_{max,i}$ and base-cycles constraint by $B_i < R_i$ pass the compatibility test and satisfy the protocol constraints (lines 7, 8, 9). In case the test evaluates to *true*

Table II. Overview of Scheduling Algorithms

heuristic	*Max-slack*	*First-hit*	*Max-E*
schedulability test	x	x	x
compatibility test	-	x	x
extensibility index	-	-	x

ALGORITHM 1: Max-E heuristic.

Require: FlexRay configuration, set of messages $\mathcal{M}$
1: $R_{max,i}$ = ComputeRmax();
2: I = SortByRmax(); //in ascending order
3: **for all** $i \in I$ **do**
4: **for all** $(N+1) \leq S_i \leq (N+M)$ **do**
5: $sus = False$;
6: $E_i(S_i)$ = ComputeExtensibility(S_i);
7: **for all** $R_i \leq R_{max,i}$ **do**
8: **for all** $B_i < R_i$ **do**
9: **if** ((CompatibilityTest() $\wedge$ ProtocolCheck()) == $True$) **then**
10: $sus = True$;
11: $\Theta_i := (S_i, B_i, R_i)$;
12: $E_{i,new}(S_i)$ = ComputeExtensibility(S_i, Θ_i);
13: $E_{i,\Delta}(S_i) = E_i(S_i) - E_{i,new}(S_i)$;
14: **else**
15: //discard schedule
16: **end if**
17: **end for**
18: **end for**
19: **if** (($sus == False$) $\wedge$ (SlotIsEmpty(S_i) == $True$) **then**
20: break;
21: **else**
22: //go to line 4 and increment slot number S_i
23: **end if**
24: **end for**
25: Θ_i = MaxExtensibility($E_{i,\Delta}$);
26: return Θ_i;
27: **end for**

(line 9), we assign the schedule Θ_i and compute the new extensibility index $E_{i,new}(S_i)$ considering that m_i is assigned the slot S_i according to Θ_i. Subsequently, we compute and store the difference $E_{i,\Delta}(S_i) = E_i(S_i) - E_{i,new}(S_i)$, that is, we compute how the extensibility index decreased for slot S_i after scheduling m_i using Θ_i. In case the test in line 9 evaluates to *false*, the schedule Θ_i is discarded (line (15)). We repeat this procedure until we reach an empty slot $\hat{S}$, namely any slot where no cycle is assigned to any message, where no compatible schedule can be found (lines 19, 20). This implies that no compatible schedule can be found for any schedule Θ_i with $S_i > \hat{S}$ because the future delay $\bar{D}_i$ naturally increases with increasing slot numbers and all further schedules will violate the compatibility test as well. For this reason there is no need to evaluate the remaining slots $\hat{S} < S_i \leq M + N$. Finally, we return the schedule Θ_i with the minimum $E_{i,\Delta}(S_i)$, that is, the schedule that realizes the minimum decrease in extensibility (lines 25, 26).

Overview of scheduling algorithms. Table II gives an overview of the presented heuristics depicting the implemented mechanisms for schedule synthesis. The Max-slack

heuristic neither implements the compatibility test nor makes use of any extensibility metric. Schedules are generated subject to deadline satisfaction and slack maximization. Both the First-hit heuristic and the Max-E heuristic implement the proposed compatibility test for schedule synthesis. Note that the compatibility test computes a tighter bound on the worst-case delay considering the impact of messages from future iterations. This means a successful compatibility test also implies a successful schedulability test. The First-hit heuristic uses a simple approach to provide extensibility by assigning the repetition rate R_{max}, however, it does not use the concept of the extensibility index. In contrast, the Max-E heuristic evaluates the extensibility index to synthesize the schedules. In the next section we will study and compare the performance of the proposed scheduling algorithms with respect to runtimes, schedulability, and extensibility.

7. EXPERIMENTAL RESULTS

In this section, we present a case study motivated by the automotive design paradigm to demonstrate the applicability of our scheme. Further, we present a parametrized competitive analysis to compare the performance of our proposed scheduling algorithms.

7.1. Case Study: Iterative Development Cycle

For the purpose of experiments we present a case study where we consider a vehicle development cycle that consists of an existing legacy implementation and five design iterations. In general, evolution in an automotive electric/electronic architecture occurs when new features and functions are added and refined. For instance, plugging new ECUs involves enhancing both new and existing functions by stepwise refinement during several design iterations. Towards this aim, new applications like driver assistance systems (such as Adaptive Cruise Control (ACC) or lane keeping) are highly distributed over the network. As a consequence, such applications impose communication requirements that need to be satisfied by the bus schedules.

System description. Motivated by the preceding, we consider five design iterations $I = \{1, \ldots, 5\}$ where, in each iteration I, $|\mathcal{M}_I|$ new messages are added and should be scheduled. Typically, at first the legacy network is refined and adapted according to the new product-line requirements. Subsequently, major functions are added in the earlier design stages whereas in the later iterations only minor refinements and bug fixes are performed, reflected in a lower number of messages being added. The design iterations are characterized as follows.

—*legacy.* This iteration considers a base architecture where $|\mathcal{M}_{legacy}| = 50$. The schedules for the legacy design are available and should be maintained throughout incremental additions and refinements.
—$I = 1$. This includes refinement of legacy features: $|\mathcal{M}_1| = 10$.
—$I = 2$. New major functions and ECUs are created: $|\mathcal{M}_2| = 40$.
—$I = 3$. In this iteration new functions and features are created: $|\mathcal{M}_3| = 30$.
—$I = 4$. This includes refinements and bug fixes: $|\mathcal{M}_4| = 15$.
—$I = 5$. Minor refinements and bug fixes are made in this iteration: $|\mathcal{M}_5| = 10$.

Each message $m_i \in \mathcal{M}_I$ is defined by the tuple (p_i, d_i, n_i). The parameters selected for this case study are based on real automotive configurations.

—We selected message periods p_i in the range of typical chassis domain applications [Grenier et al. 2008] between $10ms$ and $500ms$ with deadlines $d_i \leq p_i$.

Table III. FlexRay Bus Configuration Parameters

Parameter	Configuaration
bus speed [Mbit/s]	10
cycle length $T_{bus}[ms]$	5
#static slots N	17, i.e., $\mathcal{S}_{ST} = \{1, \ldots, 17\}$
static slot length $\Delta[ms]$	0.060
#minislots M	241, i.e., $\mathcal{S}_{DYN} = \{18, \ldots, 258\}$
minislot length $\delta[ms]$	0.015
pLatestTx	238

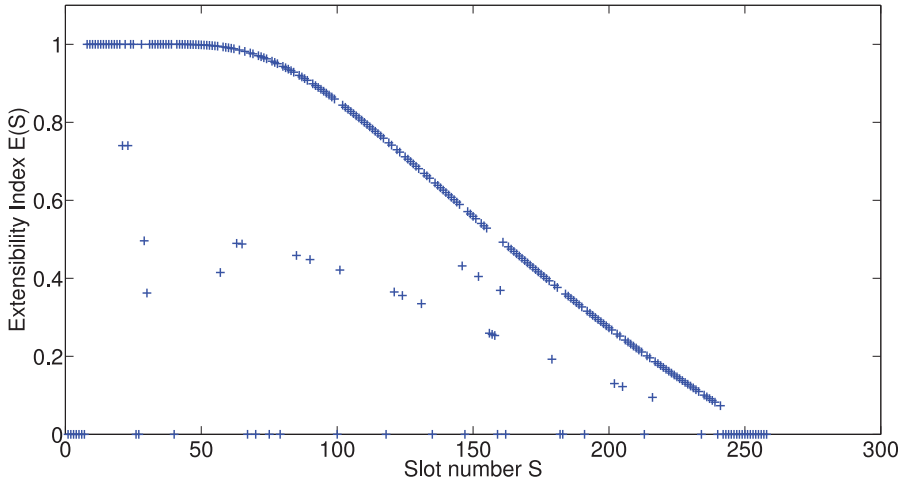

Fig. 11. Extensibility index for legacy schedules, $E_{FR} = 51.7$.

—As payloads, we considered sizes in the range $n_i = [2bytes, 40bytes]$ based on the FlexRay payload sizes of a BMW car reported in Lim et al. [2011][3].
—We considered a FlexRay configuration as depicted in Table III and a quality rating function as defined in (13) with $k = 1$. We assume a cycle length $T_{bus} = 5ms$ in line with the design choice made by BMW [Schedl 2007].

We selected all FlexRay parameters to be compliant with the specification using SIM-TOOLS [2013] configuration software. As mentioned in Section 1 we mainly focus on the DYN segment. Therefore, for the sake of illustration, we have configured a bus cycle T_{bus} where most of the slots are assigned to the DYN segment (see $N = 17$ slots in the ST segment and 241 slots in the DYN segment in Table III).

Figure 11 illustrates the extensibility index $E(S)$ of the legacy network for all slots $S \in \mathcal{S}$. The figure shows that most of the slots are fully available as most of the points coincide with the quality rating function and only few points are located at the x-axis where $E(S) = 0$. Figures 12, 13, and 14 illustrate the extensibility index $E(S)$ for design iteration $I = 2$ for the Max-slack, First-hit, and Max-E heuristics. Tables IV, V, and VI show the results for the three approaches as follows.

—$|\mathcal{M}_I|$ is the total number of messages to be scheduled for a design iteration I (see preceding).

[3]The details of all message and schedule parameters that we used for the experiments can be found in the Appendix.

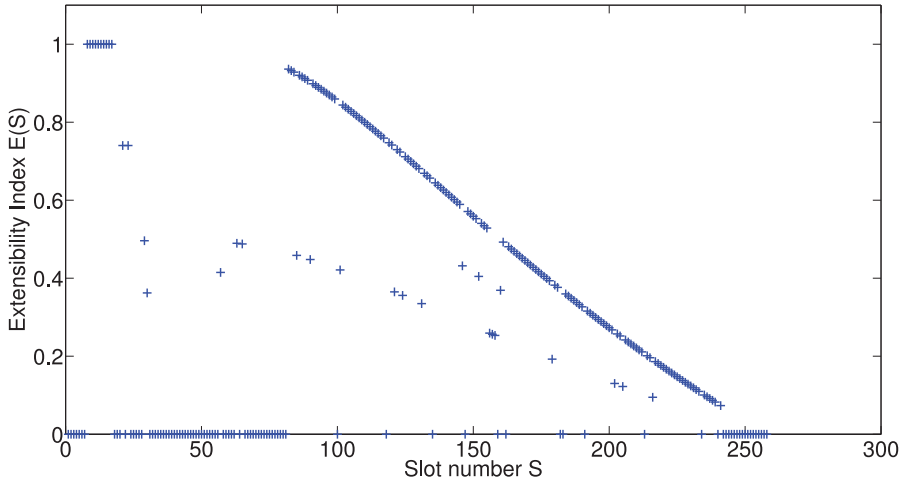

Fig. 12. Max-slack for I = 2: $\xi = 100\%$, $E_{FR} = 0.325$, $E_{eff} = 0.325$.

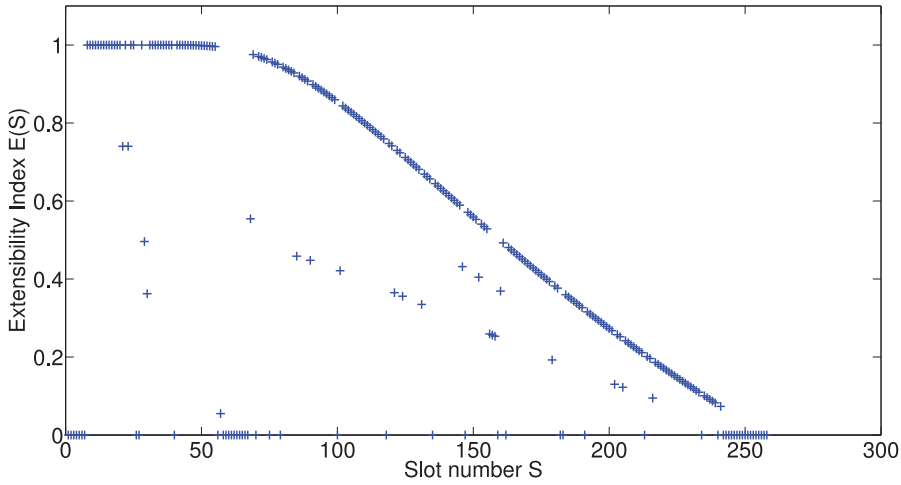

Fig. 13. First-hit for I = 2: $\xi = 65\%$, $E_{FR} = 0.479$, $E_{eff} = 0.424$.

—τ [in sec] is the runtime of the scheduling algorithm for design iteration I. All experiments have been carried out on a dual-core 1.8. GHz processor with 3GB RAM.
—ξ_I [in %] is the compatibility index $\xi_I = \frac{|\mathcal{Q}_I|}{|\mathcal{M}_I|}$, with $\mathcal{Q}_I \subseteq \mathcal{M}_I$ denoting the number of messages assigned with compatible schedules (see (10)).
—$E_{FR,I}$ is the network extensibility for design iteration I.
—$E_{eff,I}$ is the effective network extensibility according to (17). Recall that $E_{eff,I}$ considers all the messages $m_i \notin \mathcal{L}_I$ that could not have been scheduled and hence do not contribute to the extensibility index $E_{FR,I}$. To account for this, $E_{eff,I}$ introduces a penalty for all the messages $m_i \notin \mathcal{L}$ assuming the worst-case extensibility.

Observations and discussions. The Max-slack heuristic synthesizes compatible schedules for the first two design iterations as $\xi_1 = \xi_2 = 100\%$, whereas the First-hit scheduler only achieves $\xi_2 = 65\%$ in the second iteration (see Table V). Here, it is interesting

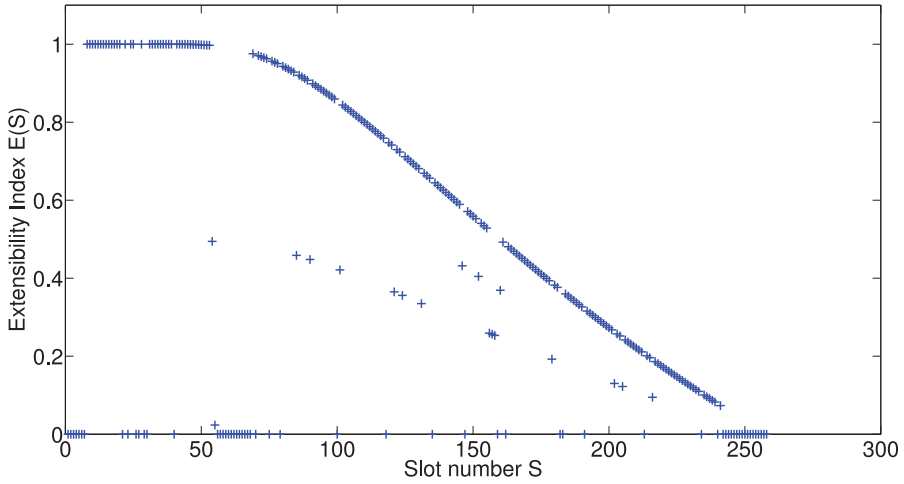

Fig. 14. Max-E for I = 2: $\xi = 100\%$, $E_{FR} = 0.462$, $E_{eff} = 0.462$.

Table IV. Max-Slack Heuristic

Iteration I	$\|\mathcal{M}_I\|$	τ [in sec]	ξ_I [in %]	$E_{FR,I}$	$E_{eff,I}$
legacy	50	-	-	0.517	0.517
1	10	32	100	0.478	0.478
2	40	59	100	0.325	0.325
3	30	39	33	0.312	0.234
4	15	21	0	0.312	0.176
5	10	15	0	0.312	0.137

to observe how compatibility is related to extensibility over several design iterations. Although the Max-slack heuristic synthesized compatible schedules for design iterations $I = \{1, 2\}$, the compatibility performance of the algorithm decreases dramatically in the next iterations where $\xi_3 = 33\%$ and $\xi_4 = \xi_5 = 0\%$, that is, where no compatible schedule could be found. This is because the purely slack-optimized algorithm assigns schedules with low repetition rates and low slot numbers to achieve minimum delays, namely maximum slack. In other words, slack not only depends on the slot number S_i but also on the bus blocking time $R_i \cdot T_{bus}$ as the worst-case delay experienced by any message m_i increases with R_i (see (1) and (3)). Hence, for $R_i = 1$, the blocking time is minimal and the optimal schedules Θ_i are generated such that S_i and R_i are minimized. The Max-slack heuristic has two major drawbacks. First, the purely slack-optimized schedules result in a high reservation of bandwidth because a slot is assigned in as many cycles as possible for every new message m_i regardless of the actual message requirements such as periods and deadlines. Second, slots with high priorities (small slot numbers) are assigned first. This is also illustrated in Figure 12 which shows the extensibility index for $I = 2$. From the figure it is very clear that mostly small slot numbers with repetition rates $R = 1$ have been assigned to the messages as most of the points between $18 \leq S \leq 80$ have dropped to the x-axis (compared to Figure 11) where $E(S) = 0$. Consequently, there are no more schedules available for messages to be schedules in successive design iterations $I \in \{3, 4, 5\}$ that guarantee the deadlines. Therefore, the Max-slack heuristic realizes compatibility at the cost of extensibility, resulting in a significant deterioration of extensibility and schedulability in later design iterations $I = \{3, 4, 5\}$ due to the dissipative resource assignments of

Table V. First-Hit Heuristic

Iteration I	$\|\mathcal{M}_I\|$	τ [in sec]	ξ_I [in %]	$E_{FR,I}$	$E_{eff,I}$
legacy	50	-	-	0.517	0.517
1	10	7	100	0.514	0.514
2	40	11	65	0.479	0.424
3	30	8	73	0.435	0.349
4	15	6	100	0.401	0.315
5	10	5	100	0.379	0.293

Table VI. Max-E Heuristic

Iteration I	$\|\mathcal{M}_I\|$	τ [in sec]	ξ_I [in %]	$E_{FR,I}$	$E_{eff,I}$
legacy	50	-	-	0.517	0.517
1	10	12	100	0.514	0.514
2	40	13	100	0.462	0.462
3	30	6	100	0.405	0.405
4	15	4	100	0.370	0.370
5	10	4	100	0.348	0.348

slots and cycles. This is also reflected in the network extensibility $E_{FR,I}$ and the effective network extensibility $E_{eff,I}$ that dramatically decrease with I as illustrated in Table IV. If we compare $E_{FR,I}$ for design iteration $I = 1$ for the three heuristics, we have $E_{FR,1}^{Max-slack} < E_{FR,1}^{First-hit} = E_{FR,1}^{Max-E}$, where $E_{FR,1}^{Max-slack} = 0.478$ clearly realizes the worst extensibility.

The First-hit approach achieves compatible schedules for design iterations $I = \{1, 4, 5\}$ where $\xi_I = 100\%$. However, for $I = \{2, 3\}$, $\xi_2 = 65\%$ and $\xi_3 = 73\%$, meaning that only 65% of the 40 messages in $I = 2$ as well as 73% of the 30 messages in $I = 3$ have been assigned schedules that passed the compatibility test. Consequently, the effective network extensibility $E_{eff,I}$ significantly deviates from the network extensibility $E_{FR,I}$. This is depicted in Table V where $E_{FR,2} = 0.479$, whereas $E_{eff,2} = 0.424$. Similarly, for $I = 3$, $E_{FR,3} > E_{eff,3}$ with $0.435 > 0.349$. This is because the First-hit heuristic statically assigns the highest possible repetition rate $R = R_{max,i}$ that satisfies the real-time constraints according to (19). Although this approach is beneficial with respect to extensibility, the assignment of $R_{max,i}$ always results in a high worst-case delay as the maximum bus blocking time $T_{bus} \cdot R_{max,i}$ increases with R_i. For this reason, some of the schedules might not pass the compatibility test. This shows that meeting the compatibility constraint may require a decrease in extensibility, namely a repetition rate $R_i < R_{max,i}$, to synthesize compatible schedules.

The Max-E heuristic implements both the compatibility test and the extensibility index for schedule synthesis and is the only algorithm that generates compatible schedules for all the 105 messages as $\xi_I = 100\%$ for $I \in \{1, 2, 3, 4, 5\}$, and hence, $E_{FR,I} = E_{eff,I}, \forall I$ (see Table VI). Further, if we compare the effective network extensibility of all three algorithms, it turns out that $E_{eff,I}^{Max-E} = E_{eff,I}^{First-hit} > E_{eff,I}^{Max-slack}$ for $I = 1$ and $E_{eff,I}^{Max-E} > E_{eff,I}^{First-hit} > E_{eff,I}^{Max-slack}$ for $I = \{2, 3, 4, 5\}$ as depicted in the last column of Tables IV, V, and VI.

Finally, we compare the runtimes of the three heuristics. We can see from the tables that the synthesis times τ_I per iteration for the Max-slack heuristic approximately five times higher compared to the First-hit and Max-E heuristics. This is because the latter do not need to evaluate the full search space and hence allow ruling out a wide range of schedules. In contrast, the Max-slack heuristic iterates over all possible schedules and returns Θ_i where s_i is maximum.

7.2. Parametrized Competitive Analysis

In the previous section, the evolution of extensibility in an incremental automotive design scenario has been studied. In particular, a comparison of the proposed heuristics has been performed based on five design iterations and a typical automotive network specification. While the aforesaid performance evaluation has been performed based on a dedicated case study, in this section we pursue a more general and standardized approach to performance analysis. In this regard, the incremental scheduling problem that we tackle in this work corresponds to an online scheduling problem in such a way that the scheduling algorithm at each iteration does not know the messages to be scheduled in future iterations. Given an online scheduling algorithm *ALG* and a request sequence σ, specifying the set of messages to be scheduled at each iteration, the algorithm has to serve each request online. A standard technique to evaluate the performance of an online algorithm is *competitive analysis* [Sleator and Tarjan 1985]. Let $\mathcal{C}_{ALG}(\sigma)$ denote the cost of an online algorithm *ALG* on an input sequence σ and $\mathcal{C}_{OPT}(\sigma)$ denote the cost of an optimal offline algorithm OPT on σ. Then, *ALG* is referred to as *c-competitive* with respect to all finite input sequences σ if

$$\mathcal{C}_{ALG}(\sigma) \leq c \cdot \mathcal{C}_{OPT}(\sigma), \tag{20}$$

with σ being any sequence $\sigma = \sigma_1, \ldots, \sigma_i, \ldots, \sigma_n$ that must be serviced in the order of occurrence $1 \leq i \leq n$. That is, when serving the request σ_i, the algorithm does not know any request σ_j with $j > i$.

In our setting, an online algorithm $ALG \in \{Max\text{-}slack, First\text{-}hit, Max\text{-}E\}$ has to serve an input sequence σ that consists of a set of n messages with $n = \sum_{\forall I} |\mathcal{M}_I|$ denoting the total number of messages added in all iterations. In particular, *ALG* makes scheduling decisions, that is, which schedules Θ_i to assign the messages $m_i \in \mathcal{M}_I$, without having the knowledge of the messages that will be added in future iterations $I' > I$. The cost of serving σ for *ALG* is denoted by the network extensibility as defined in (16), namely $\mathcal{C}_{ALG}(\sigma) = E_{FR}$. As shown in (20) this must be compared against the cost of an optimal offline algorithm that we discuss next.

Optimal offline algorithm. The optimal offline algorithm *OPT* knows the entire message specification in advance, including the total number of messages to be scheduled at each iteration and the corresponding message characteristics, that is, sizes, periods, and deadlines. Thus, *OPT* may exploit the knowledge about the actual workloads of the future messages, namely $c_f = c_I, \forall I \in \{1, \ldots, n\}$, and thus makes a workload estimation model obsolete (see Section 4.1) that is necessary for an online algorithm $ALG \in \{First\text{-}hit, Max\text{-}E\}$. Similarly, since the entire input is known, there is no need for a compatibility test and a conventional schedulability test as presented in (5) is sufficient to guarantee that messages meet their corresponding deadlines. Essentially, *OPT* is a *clairvoyant* scheduling algorithm that schedules the entire set of n messages according to DM while maximizing the overall network extensibility E_{eff} as defined in (16).

Adversary request sequence. Since, for a competitive algorithm we require (20) to hold for any input sequence σ, we may assume that σ is generated by a malicious adversary that selects a sequence resulting in a worst-case scenario for the online algorithm. In particular, we consider an *oblivious adversary* [Ben-David et al. 1990] that knows the strategy of the online algorithm *ALG* and generates and presents the complete input sequence σ to the algorithm such that the ratio $\frac{\mathcal{C}_{ALG}(\sigma)}{\mathcal{C}_{OPT}(\sigma)}$ is minimized. For this purpose, we generate σ as follows.

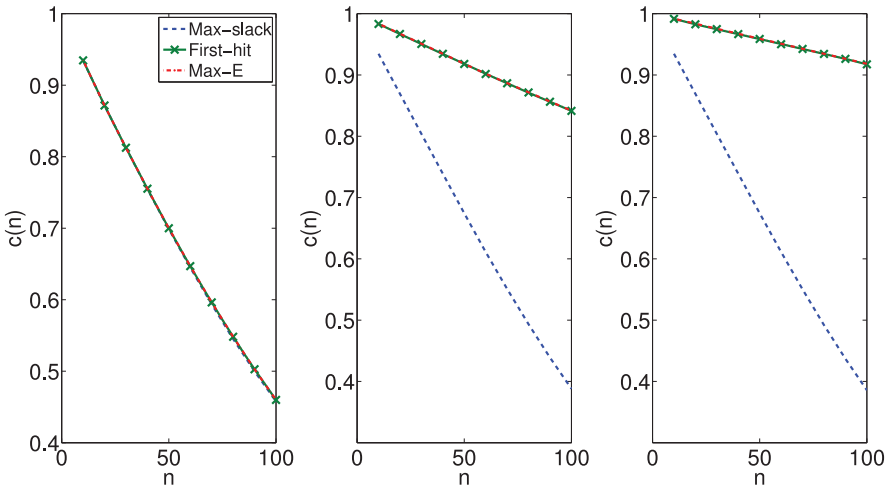

Fig. 15. Competitive ratio $c(n)$ for: (a) $p_I = 2T_{bus}$ (left); (b) $p_I = 10T_{bus}$ (middle); and (c) $p_I = 20T_{bus}$ (right).

—Each iteration I contains only one message such that $|\mathcal{M}_I| = 1, \forall I \in \{1, \ldots, n\}$. This corresponds to a request sequence $\sigma = \sigma_1, \ldots, \sigma_I, \ldots, \sigma_n$ with $1 \leq I \leq n$, and $\sigma_I := m_I$. As a result, the information provided to the online algorithm at each iteration is minimum, resulting in a maximum uncertainty of the number and types of the future messages.

—The *actual* number of minislots consumed by any message is minimum. That is, for all m_I, $c_I = c_{min}$ with $c_{min} = f(n_{min})$ and $n_{min} = 2bytes$ denotes the minimum payload size defined by the FlexRay protocol. On the contrary, the *expected* number of minislots consumed by any message is maximum. That is, $c_f = c_{max}$ with $c_{max} = f(n_{max})$ and $n_{max} = 254bytes$ denotes the maximum payload size defined by the FlexRay protocol. Note that c_f is used to estimate the future workload for the compatibility test described in Section 4.1. Hence, assuming $c_I = c_{min}$ and $c_f = c_{max}$ results in a maximum misprediction of future message sizes that in turn leads to a low range of available, namely compatible, slots, therefore reducing the network extensibility.

Results. We performed a parametrized competitive analysis for which we computed the competitive ratios $c(n)$ according to (20) for different problem sizes $n = \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$ and message periods $p_I = \{2T_{bus}, 10T_{bus}, 20T_{bus}\}$ with $d_I = p_I$. The results are depicted in Figure 15. The first important observation is that the competitive ratio $c(n)$ of all algorithms heavily depends on the problem size n. In particular, $c(n)$ decreases with n. In other words, the performance of the algorithms deteriorates with an increasing number of messages being scheduled. This is expected and can be explained as follows. First, the definition of extensibility (and thus the cost from which the ratio $c(n)$ is computed) depends on both the number of existing messages and the number of future messages to be added. Hence, the performance of scheduling algorithms that optimize for extensibility depends on the total number of messages n to be scheduled. Second, the extensibility achieved with the online algorithms $ALG \in \{First\text{-}hit, Max\text{-}E\}$ depends on the workload estimation of the future messages. The more messages are being added in the future, the higher the impact of the workload estimation on the overall extensibility. Since the adversary request sequence generates the inputs such that the misprediction of the future message sizes is maximum, the extensibility performance deteriorates with every new message that

is being scheduled. The performance of Max-slack intrinsically depends on the number of messages as maximizing the slack generally comes with assigning low repetition rates (see Section 7.1), thus resulting in a severe decrease in extensibility for every new message being scheduled. The different plots of Figure 15 are described in more detail in what follows.

—*Case a.* The generated input sequence models the worst-case behavior of all three online algorithms $ALG \in \{Max\text{-}slack, First\text{-}hit, Max\text{-}E\}$. The competitive ratios $c(n)$ nearly linearly decrease with $\frac{\Delta c(n)}{\Delta n} = 0.0063$, which is the worst decrease among all cases. This is because we consider the minimum feasible message periods $p_I = 2T_{bus}$ that results in the bound $R_{max} = 1$ (see (19)) for all the messages. Since $R_I = 1$, a complete slot is assigned to any message m_I and hence the grade of extensibility $P_1(S_I) = 0$ for all the slots $S_I, I \in \{1, \ldots, n\}$. Thus, the extensibility index is dominated by the choice of the slot numbers S_I reflected in P_2. Since, in the worst-case scenario, the choice of S_I does not significantly differ for the three algorithms, the competitive ratios $c(n)$ of all three algorithms show similar results.
—*Case b.* In this case, we generated the input sequence such that $p_I = 10T_{bus}$. It can be observed that for $ALG = \{First\text{-}hit, Max\text{-}E\}$, $c(n)$ significantly improved compared to case (a) that is reflected in a higher competitive ratio for all n. This is because as $p_I = 10T_{bus}$, the maximum repetition rate is computed as $R_{max} = 4$. According to (12), for any slot S_I we have $P_1(S_I) = 0$ with $R_I = 1$ and $P_1(S_I) = 0.2441$ with $R_I = 4$. Hence, the corresponding increase in $P_1(S_I)$ is 24.41% compared to case (a) and the extensibility now depends on both the choice of the slot and repetition rate. This is also visible in the gradient as $\frac{\Delta c(n)}{\Delta n} = 0.0015 < 0.0063$. Since Max-slack solely optimizes for slack, we again have $R_I = 1$ for all m_I, resulting in a similar competitive ratio $c(n)$ as in case (a).
—*Case c.* Similarly to case (b), the competitive ratios improve for $ALG = \{First\text{-}hit, Max\text{-}E\}$ because for $p_I = 20T_{bus}$ we have $R_{max} = 8$, whereas Max-slack does not improve as we again have $R_I = 1$ for all m_I.

In summary, we can observe that the competitive ratios for $ALG \in \{First\text{-}hit, Max\text{-}E\}$ are very close to each other, with Max-E showing a slight improvement over First-hit (but not visible in the graphs). This is because First-hit realizes extensibility by assigning slots as late as possible and repetition rates as high as possible. However, it does not make use of the extensibility index implemented in Max-E. Further, $c(n)$ increases with p_I and hence with R_{max}. This is because the contribution of $P_1(S)$ to the overall extensibility prevails over the impact of $P_2(S)$ that depends on the choice of slots and on the workload prediction, respectively. Note that the competitive ratio is a worst-case performance measure. Even if First-hit and Max-E have very similar competitive ratios on average, as shown in the experiments of Section 7.1, Max-E results in significantly better extensibility compared to First-hit as it exploits the extensibility index in the optimization procedure. Max-slack clearly shows the worst competitive ratio, mainly due to the slack optimization policy leading to very low repetition rates and hence poor extensibility.

8. OUTLOOK AND CONCLUDING REMARKS

The metrics and design methodologies presented in this article aim to efficiently strike a balance between the naturally conflicting objectives of: (i) saving bandwidth (extensibility) and (ii) guaranteeing timing constraints (compatibility). In this discussion, we would like to give an outlook on how the proposed concepts can be efficiently applied and integrated in a modular fashion into the development process to further consider the characteristics of the automotive design process. As already described in the case

study, the major functionalities (and hence most of the new messages) are usually added in the earlier design stages whereas refinements and bug fixes are performed in the later iterations. This is reflected in a lower number of messages being added. In this light, the extensibility metrics being used to evaluate the communication design may be applied in a stepwise manner, too. That is, the metrics may be adapted in different phases corresponding to the maturity of the development process. Towards this end, we propose three phases described as follows.

—*Base design evaluation.* In practice, the base (legacy) implementation may be evaluated using a simple measure for extensibility solely based on the grade of extensibility $P_1(S)$. As a result, the network extensibility of (16) for the legacy implementation evaluates to $E_{FR}^{legacy} = \frac{1}{N+M} \sum_{S=1}^{N+M} P_1(S)$. As the grade of extensibility quantifies the pure schedule extensibility, E_{FR}^{legacy} may be used to identify a suitable, that is, extensible, base design that may be used for a new product line. In other words, applying E_{FR}^{legacy} may help to answer the question: Which base design is best suited to accommodate the expected communication requirements of a new product line?

—*Foresightful schedule design.* At the early design phases, the implementation of functionalities must be flexible to meet requirements of several vehicle projects in the product line. During this phase, most of the functionalities are added, which typically involves a lot of changes in communication, that is, many messages are added, removed, or changed with respect to their timing constraints.

Hence, in addition to extensibility, assuring timing guarantees with respect to expected enhancements in later iterations (i.e., compatibility) becomes especially important in the first few iterations. As a result, the full extensibility index considering both quality rating and grade of extensibility may be applied to foresightfully schedule messages with respect to extensibility.

—*Design assurance.* In the later iterations, close to the completion of the development cycle for a specific product in the product line, minor refinements and bug fixes are usually performed. Given that only a few messages are expected to be added in this phase, it is more important to assure the timing guarantees of the existing schedules (compatibility) than to provide extensible schedules for the few messages yet to be expected. As a consequence, the schedules may be designed focusing on only compatibility with the goal to guarantee the timing properties of the existing schedules until completion of the development cycle. To account for this, the extensibility index may be reduced to the quality rating, namely $E(S) = P_2(S)$. This will accordingly be reflected in the schedule synthesis.

In summary, the central question that we try to answer in this article is: What is an appropriate notion of extensibility for FlexRay? Concomitantly, we also define a notion of compatibility for FlexRay schedules and we show that compatibility is necessary to achieve extensibility. We would also like to mention that there are many possible ways to define and apply notions of extensibility. In this work, our goal was to define a notion of network extensibility that: (i) reflects the iterative design process of the automotive industry, (ii) allows for visualization and hence easy interpretation by engineers, and (iii) captures all the protocol-specific details of FlexRay such as cycle multiplexing. Along these lines, we developed analysis techniques to identify extensible schedules and we also presented a scheduling framework to automatically synthesize extensible FlexRay schedules based on the presented notions. In particular, our analysis has been devoted to the dynamic segment of FlexRay where the evaluation of extensibility is especially challenging due to the dynamic nature of the protocol. Further, we performed comprehensive experiments with industrial-size case studies and compared the results

of different schedule synthesis algorithms implementing different stages of the defined metrics to automatically synthesize schedules. We also discussed the compatibility of the presented concepts with other FlexRay delay models and gave insights on how the analysis might be extended to integrate other timing analysis techniques. For future work we envisage to extend the concept of extensibility to other domains such as the packing of signals and messages to slots, or extensibility for ECU schedules.

REFERENCES

M. Anand and I. Lee. 2008. Robust and sustainable schedulability analysis of embedded software. In *Proceedings of the ACM SIGPLAN-SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'08).* 61–70.

E. Armengaud, A. Steininger, and M. Horauer. 2008. Towards a systematic test for embedded automotive communication systems. *Industr Inf.* 4, 3, 146–155.

Autosar. 2013. AUTomotive open system architecture. www.autosar.org.

Autosarxcp. 2013. AUTOSAR. Specification of module XCP. www.autosar.org.

T. P. Baker and S. K. Baruah. 2009. Sustainable multiprocessor scheduling of sporadic task systems. In *Proceedings of the 21st Euromicro Conference on Real-Time Systems (ECRTS'09).* 141–150.

S. K. Baruah and A. Burns. 2006. Sustainable scheduling analysis. In *Proceedings of the 27th IEEE International Real-Time Systems Symposium (RTSS'06).* 159–168.

S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Wigderson. 1990. On the power of randomization in online algorithms. *Algorithmica* 11, 1, 2–14.

U. D. Bordoloi, B. Tanasa, P. Eles, and Z. Peng. 2012. On the timing analysis of the dynamic segment of flexray. In *Proceedings of the 7th IEEE International Symposium on Industrial Embedded Systems (SIES'12).* 94–101.

A. Burns and S. K. Baruah. 2008. Sustainability in real-time scheduling. *J. Comput. Sci. Engin.* 2, 1, 74–97.

FlexRay. 2013. The FlexRay communications system specifications. www.flexray.com.

E. Fuchs. 2010. FlexRay-Beyond the consortium phase. http://www.simtools.at/doc/flexray_consortium.pdf.

A. Ghosal, H. Zeng, M. D. Natale, and Y. Ben-Haim. 2010. Computing robustness of FlexRay schedules to uncertainties in design parameters. In *Proceedings of the Design, Automation, and Test in Europe Conference and Exhibition (DATE'10).* 550–555.

M. Grenier, L. Havet, and N. Navet. 2008. Pushing the limits of CAN - Scheduling frames with offsets provides a major performance boost. In *Proceedings of the 4th European Congress on Embedded Real Time Software (ERTS'08).*

A. Hagiescu, U. D. Bordoloi, S. Chakraborty, P. Sampath, P. V. V. Ganesan, and S. Ramesh. 2007. Performance analysis of FlexRay-based ecu networks. In *Proceedings of the 44th Annual Design Automation Conference (DAC'07).* 284–289.

H.-T. Lim, K. Weckemann, and D. Herrscher. 2011. Performance study of an in-car switched Ethernet network without prioritization. In *Proceedings of the 3rd International Conference on Communication Technologies for vehicles (Nets4Cars/Nets4Trains'11).* 165–175.

M. Lukasiewycz, M. Glaß, P. Milbredt, and J. Teich. 2009. FlexRay schedule optimization of the static segment. In *Proceedings of the 7th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'09).* 363–372.

M. Neukirchner, M. Negrean, R. Ernst, and T. Bone. 2012. Response-time analysis of the FlexRay dynamic segment under consideration of slot-multiplexing. In *Proceedings of the 7th IEEE International Symposium on Industrial Embedded Systems (SIES'12).* 21–30.

W.-C. Poon and A. K. Mok. 2010. Necessary and sufficient conditions for non-preemptive robustness. In *Proceedings of the 16th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'10).* 349–354.

P. Pop, P. Eles, Z. Peng, and T. Pop. 2004. Scheduling and mapping in an incremental design methodology for distributed real-time embedded systems. *VLSI Syst.* 12, 8, 793–811.

T. Pop, P. Pop, P. Eles, Z. Peng, and A. Andrei. 2006. Timing analysis of the FlexRay communication protocol. In *Proceedings of the 18th Euromicro Conference on Real-Time Systems (ECRTS'06).*

M. Rausch. 2008. *FlexRay: Grundlagen, Funktionsweise, Anwendung*. Carl Hanser Verlag GmbH and CO. KG.

A. Sangiovanni-Vincentelli, L. Alberto, and M. Di Natale. 2009. Challenges and solutions in the development of automotive systems. *Comput.-Aided Des. Integr. Circ. Syst.* 28, 7, 937–940.

A. Sangiovanni-Vincentelli and M. Di Natale. 2007. Embedded system design for automotive applications. *Comput.* 40, 10, 42–51.

A. Schedl. 2007. Goals and architecture of FlexRay at BMW. http://vector.com/portal/medien/cmc/speeches/FlexRay_Symposium_2007/FRS07_02_Schedl.pdf.

F. Scheler and W. Schroeder-Preikschat. 2006. Time-triggered vs. event-triggered: A matter of configuration? In *Proceedings of the GI/ITG Workshop on Non-Functional Properties of Embedded Systems*.

S. Schliecker, J. Rox, M. Negrean, K. Richter, M. Jersak, and R. Ernst. 2009. System level performance analysis for real-time automotive multicore and network architectures. *Comput.-Aided Des. Integr. Circ. Syst.* 28, 7, 979–992.

E. G. Schmidt and K. Schmidt. 2009. Message scheduling for the FlexRay protocol: The dynamic segment. *Vehic. Technol.* 58, 5, 2160–2169.

K. Schmidt and E. G. Schmidt. 2010. Schedulability analysis and message schedule computation for the dynamic segment of FlexRay. In *Proceedings of the 72nd IEEE Vehicular Technology Fall Conference (VTC'10)*. 1–5.

R. Schneider, U. Bordoloi, D. Goswami, and S. Chakraborty. 2010. Optimized schedule synthesis under real-time constraints for the dynamic segment of FlexRay. In *Proceedings of the IEEE/IFIP International Conference on Embedded and Ubiquitous Computing (EUC'10)*. 31–38.

Simtools. 2013. SIMTOOLS. www.simtools.at.

D. Sleator and R. Tarjan. 1985. Amortized efficiency of list update and paging rules. *Comm. ACM* 28, 2, 202–208.

B. Tanasa, U. Bordoloi, P. Eles, and Z. Peng. 2010. Scheduling for fault-tolerant communication on the static segment of FlexRay. In *Proceedings of the 31st IEEE Real-Time Systems Symposium (RTSS'10)*. 385–394.

B. Tanasa, U. D. Bordoloi, S. Kosuch, P. Eles, and Z. Peng. 2012. Schedulability analysis for the dynamic segment of FlexRay: A generalization to slot multiplexing. In *Proceedings of the 18th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS'12)*. 185–194.

H. Zeng, A. Ghosal, and M. Di Natale. 2010. Timing analysis and optimization of FlexRay dynamic segment. In *Proceedings of the 10th IEEE International Conference on Computer and Information Technology (CIT'10)*. 1932–1939.

H. Zeng, W. Zheng, M. Di Natale, A. Ghosal, P. Giusto, and A. Sangiovanni-Vincentelli. 2009. Scheduling the FlexRay bus using optimization techniques. In *Proceedings of the 46th Annual Design Automation Conference (DAC'09)*. 874–877.

W. Zheng, J. Chong, C. Pinello, S. Kanajan, and A. Sangiovanni-Vincentelli. 2005. Extensible and scalable time triggered scheduling. In *Proceedings of the 5th International Conference on Application of Concurrency to System Design (ACSD'05)*. 132–141.

Received April 2013; revised March 2014; accepted March 2014