Policy-based Message Scheduling Using FlexRay

Philipp Mundhenk, Florian Sagstetter,
Sebastian Steinhorst, Martin Lukasiewycz
TUM CREATE, Singapore
<firstname.lastname>@tum-create.edu.sg

Samarjit Chakraborty
TU Munich, Germany
samarjit@tum.de

ABSTRACT

This paper proposes a virtual communication layer for time-triggered networks, enabling a policy-based message scheduling as well as preemption which in turn simplifies real-time verification. The introduced layer is particularly advantageous in the automotive domain since it reduces the complexity of scheduling time-triggered communication systems and simplifies incremental changes of existing schedules. We propose a framework to schedule event-triggered messages based on a predefined policy in time-triggered communication slots, improving the network utilization while logically separating messages from different devices. Furthermore, we enhance the versatility of the system, allowing to transmit data that exceeds the size of one time-triggered slot. The proposed policy-based scheduling with fixed priorities enables the integration of mixed criticality applications in time-triggered networks, while ensuring hard deadline constraints. A prototypical implementation is provided for FlexRay, complying with the existing protocol and, thus, making it possible to coexist with the standard transmission scheme. Our experimental results demonstrate the advantages of the virtual layer, showing an increase in flexibility and significantly lower message latencies in case of asynchronous communication.

Categories and Subject Descriptors: C.3 [Special-purpose and application-based systems]: Real-time and embedded systems
General Terms: Algorithms, Design
Keywords: FlexRay, scheduling

1. INTRODUCTION

The amount of software and Electronic Control Units (ECUs) in cars is continuously increasing. New Advanced Driver Assistance Systems (ADASs) are one of the major selling points of vehicles and rely heavily on the use of software to achieve the required functionality. These applications are not implemented on one single ECU, but distributed over multiple ECUs. To achieve the desired functionality, communication between the ECUs is required. When defining a communication system for this, either an event-triggered or a time-triggered architecture needs to be selected.

In time-triggered, or Time Division Multiple Access (TDMA), systems, each message is assigned a time slot in which it is transmitted. The assignment of messages to time slots is called a schedule. This schedule is usually repeated infinitely. TDMA allows easy calculation of the Worst Case Response Times (WCRTs), as the schedule is fixed at time of implementation and usually not changed at runtime. These WCRTs rely on the assignment of messages to time slots. A message always has to wait until the assigned time slot repeats, before it can be sent. This can lead to high WCRTs, if a message misses its slot. TDMA schedules do not allow preemption. A TDMA system might lead to oversampling, when a message cycle time is not equal to, or a multiple of, the schedule duration. Additionally, as the sizes of time slots are fixed, it is usually not possible to transmit messages longer than one slot length.

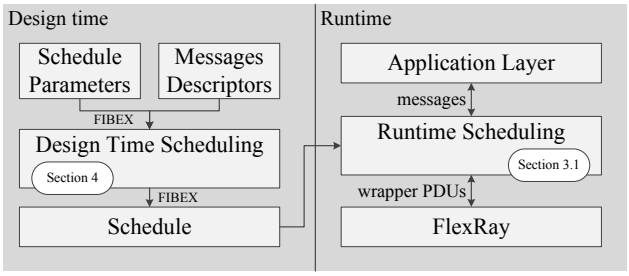

Figure 1: Framework for policy-based scheduling in time-triggered networks. A design time scheduling approach determines a slot to ECU assignment for a set of messages. During runtime our scheduler then assigns the event-triggered messages created in the application layer to communication slots depending on their priority.

By contrast, event-triggered systems transmit messages as required, allowing more flexibility in message transmissions and a higher utilization of the underlying bus system. An arbitration process is used to avoid collisions of messages transmitted on the bus. Due to the option to transmit messages at any point in time and the arbitration process of an event-triggered system, it can be difficult to calculate the WCRTs. For example, the generally accepted WCRT calculations for the widely applied Controller Area Network (CAN) bus have been shown to be incomplete only after 13 years [1, 2].

This paper proposes a virtual communication layer which enables policy-based scheduling in time-triggered networks. Figure 1 presents our framework, consisting of a design time scheduling determining a slot to ECU assignment, and a priority-based runtime scheduler, supporting message preemption. It enables incremental changes and significantly decreases the complexity of time-triggered systems. As the introduced virtual communication layer complies with the underlying time-triggered transmission scheme, it enables mixed criticality systems supporting concurrent time- and event-triggered communication.

FlexRay. We have implemented our framework for the time-

This work was financially supported in part by the Singapore National Research Foundation under its Campus for Research Excellence And Technological Enterprise (CREATE) programme.

ESWEEK'14 October 12 - 17 2014, New Delhi, India

http://dx.doi.org/10.1145/2656075.2656094.

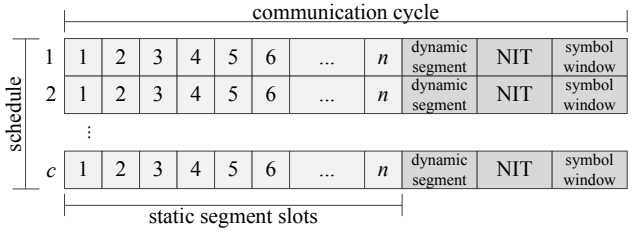

Figure 2: FlexRay schedule with n slots in the static segment, dynamic segment, Network Idle Time (NIT), symbol window and repetitive schedule containing c communication cycles.

triggered segment of the FlexRay bus. FlexRay is an automotive communication system with a bandwidth of up to 10 Mbit/s, incorporating TDMA and an event-triggered segment into one schedule. In FlexRay, TDMA is called the static segment and the event-triggered segment is called the dynamic segment. A FlexRay schedule is organized into a fixed number of cycles c. Each cycle is divided into static and dynamic segment, as well as synchronization segments (see Figure 2). Over the runtime of a FlexRay system, the schedule is repeated infinitely.

Each cycle of the static segment is further subdivided into time slots. The time-triggered communication in the static segment allows for a straight-forward calculation of response times. The priority-based arbitration in the dynamic segment, however, makes it difficult to calculate response times [3]. By contrast, policy-based scheduling ensures strict priorities per ECU and easy to compute WCRTs.

Contributions. This work proposes (1) an architecture for policy-based scheduling of event-triggered messages in time-triggered communication systems, (2) a set of scheduling algorithms to schedule policy-based messages to time-triggered slots, and (3) a prototypical implementation of this scheduling approach for FlexRay.

Policy-based scheduling is achieved by adding a virtual event-triggered layer on top of a time-triggered communication system. This virtual communication layer allows a higher flexibility in message sizes and lower latencies than the underlying time-triggered system through message preemption. Application layer messages can be scheduled event-triggered, based on predefined policies such as priority-based scheduling, and guarantees for message deliveries can be given. Worst-case response times can be calculated for all messages and the schedulability of the system can be easily verified.

The proposed approach is fully compatible with time-triggered communication systems and can be used side-by-side on the same physical bus, without interference. Changes to and addition of new messages to the communication system are simple, as recomputation of the time-triggered schedule can be avoided.

Policy-based scheduling has been implemented for the time-triggered FlexRay static segment. This is achieved by assigning slots to ECUs, instead of messages. Messages are then scheduled per ECU at run-time, based on priority and arrival time. To integrate the policy-based scheduling with existing FlexRay toolchains, standardized FIBEX input and output is employed. Finally, the developed framework has been evaluated regarding bandwidth utilization and WCRTs.

Organization of this paper. This paper is organized as follows. Section 2 gives an overview over existing literature on the combination of time- and event-triggered systems, as well as scheduling for the FlexRay static and dynamic segment. Furthermore, the differences of policy-based scheduling to all existing approaches are discussed. Section 3 introduces the architecture of policy-based scheduling and discusses possible runtime scheduling algorithms. In Section 4, a heuristic algorithm and an Integer Linear Program (ILP) approach to policy-based scheduling for FlexRay are proposed. Section 5 presents several tests to evaluate the performance of policy-based scheduling on FlexRay, in comparison to conventional scheduling algorithms. Section 6 summarizes the proposed approach and concludes.

2. RELATED WORK

In the following, an overview of existing time-triggered architectures and their approaches to scheduling, is given. Additionally, timing analysis and scheduling algorithms for the static and dynamic segment of conventional FlexRay are discussed. Other approaches for event-triggered scheduling in TDMA systems are illustrated and differences to the proposed approach are clarified.

Time-triggered architectures. A basic time-triggered architecture is presented in [4]. Based on this time-triggered architecture, scheduling algorithms have been developed, allowing reliable and predictable transmission of data. Some of these approaches consider the message transmission itself [5, 6], while others follow a holistic approach and view the message transmissions in context with the message generating tasks [7]. All of these approaches employ fixed timeslots, as described in [4], and are thus limited in message lengths and asynchronous scheduling capabilities.

FlexRay static segment. The analysis of time-triggered systems concerning response times and scheduling has also been conducted for the time-triggered static segment of FlexRay. Multiple approaches have been proposed to schedule messages in the conventional FlexRay static segment. Based on the AUTOSAR standard, [8] proposes a heuristic approach for scheduling messages, maximizing the bandwidth utilization. This assumes, however, that all message lengths equal the slot length and only one message is sent in every frame. The problem of message fragmentation is addressed in [9]. There, a heuristic and an ILP are used to optimize the bandwidth utilization in the form of a bin packing problem.

The optimization of message jitter and the number of frames used in the system is the primary target in [10] and [11]. As a measurement for the efficiency of the approach, jitter and bandwidth utilization are employed. For optimization of bandwidth, a Non-linear Integer Program (NIP) is employed, while the jitter is optimized with an ILP.

A combination of task and message scheduling is addressed in [12]. As a measure for efficiency, the control performance of an example system is used, incorporating latencies and jitters.

FlexRay dynamic segment. In addition to the analysis of the FlexRay static segment, scheduling in the dynamic segment has been researched. The calculation of the WCRT for the FlexRay dynamic segment, as proposed in [13], is highly complex. A scheduling approach for the FlexRay dynamic segment is proposed in [3]. There, the required number of frame IDs and the message jitter is to be minimized. An additional analysis of the FlexRay dynamic segment has been presented in [14]. It considers cycle multiplexing as a method to further reduce the number of required frame IDs. In contrast to the approaches above, this paper focuses on the static segment and uses preemption for lower response times, as well as introduces possibilities for multi-mode messages.

Event-based messages over TDMA. In [15], an event-

triggered layer that is virtually placed on top of the time-triggered architecture from [4] is described. The approach divides every time slot into two segments for time- and event-triggered messages. This achieves a slightly higher flexibility than a conventional time-triggered system, but assumes fixed message lengths. The approach proposed in [16] adds arbitration points to every slot of the time-triggered protocol in [17]. Based on these approaches, response time evaluations have been proposed in [18]. There, a set of parameters is proposed, allowing a good estimation of the worst- and best-case response times for an event-triggered communication layer in a time-triggered system.

An approach to integrate event-triggered communication with the FlexRay static segment is shown in [19]. However, this approach assumes one message per frame and a message length equal to the frame size. Additionally, it is assumed that the message deadline is always equal to the interarrival time of the message. Interarrival times smaller than the length of the schedule are not considered. This dependency leads to an inversion in design. The static segment needs to be modeled after the requirements of the event-triggered layer, as both layers are not independent. In case the parameters of the event-triggered layer are changed, the time-triggered layer needs to be regenerated, or guarantees might be lost.

Hierarchical Scheduling. With the rising complexity of real-time systems, compositional design becomes increasingly important. By dividing a system into subsystems and arranging these in a scheduling hierarchy, the scheduling of such complex systems can be simplified. An approach for hierarchical scheduling has been proposed in [20] and a comparison of different hierarchical scheduling techniques is presented in [21]. In comparison to hierarchical scheduling, policy-based scheduling does not require a scheduling hierarchy. The given communication system is scheduled in the traditional, flat scheduling manner. In policy-based scheduling, the time-triggered schedule is generated at design time, but filled with event-triggered messages at runtime.

Our work. In this paper we present an approach to flexibly schedule messages in a virtual event-triggered layer on top of a time-triggered communication infrastructure. The virtual event-triggered layer allows to transmit messages flexibly, while the underlying time-triggered communication system allows us to guarantee message deadlines. In contrast to the work above, we support messages of any length, as well as multiple messages per slot. Furthermore, interarrival times of messages can be chosen freely and multiple messages can be sent per schedule repetition. Oversampling of messages is reduced and preemption can be utilized for faster response times. The proposed approach offers flexibility, as messages can be changed easily, without the need for reconfiguration of the complete system. The FlexRay static segment has been used to implement the approach and to verify its feasibility. Additionally, the system proposed in this paper can be used in parallel with a conventional FlexRay system, sharing the same physical bus, without interference.

3. ARCHITECTURE

This section outlines how time- and event-triggered systems are integrated in policy-based scheduling. While policy-based scheduling is applicable to time-triggered systems in general, we use FlexRay terminology to describe our system. No definitions and parameters from the FlexRay standard have been altered, thus achieving full compatibility to conventional FlexRay systems. Instead, we reserve time slots in the static segment and assign these to ECUs. We call the messages used to reserve a time slot in the static segment wrapper Protocol Data Units (PDUs). In the proposed approach, a wrapper PDU always fills the static slot in its entirety. These wrapper PDUs are coordinated by the communication layer introduced for policy-based scheduling. This additional layer is logically placed between the application layer and the conventional FlexRay communication layer of every ECU (see Figure 1).

At runtime, messages from the application layer can arrive at any point in time, i.e. event-triggered, and the policy-based communication layer is sorting these messages into wrapper PDUs and thus FlexRay static slots. This way, a virtual event-triggered layer is created on top of the time-triggered FlexRay static segment. To reserve sufficient time slots for all messages to be transmitted, the time-triggered schedule needs to be calculated, based on the message parameters of the application layer. This calculation is described in Section 4. With this calculation and the correct mapping of event-triggered messages to time slots, it can be ensured that every deadline in the system is satisfied.

3.1 Runtime Scheduling Algorithm

In the following, the scheduling of messages into timeslots at runtime is explained. The additional communication layer is required to add application layer messages to slots of the static segment. This is performed according to a fixed algorithm on the ECU at runtime. Many suitable scheduling algorithms are available in literature, such as First Come First Serve (FCFS), Round Robin (RR), Shortest Message First (SMF), Priority-based, Multi-Level Queues and Multi-Level Feedback Queues. Based on the requirements existing in vehicles, a suitable scheduling algorithm needs to be found.

In vehicles, some messages always have a higher priority than others. Consider an Anti-lock Braking System (ABS) system versus a park distance sensor. Even when parking, the vehicle brakes need to be able to react reliably. This makes schemes like FCFS, RR and SMF unsuitable. These schemes would not represent the priorities in vehicles accurately. As multi-level feedback queues are also changing this priority behavior, they do not fulfill the requirements for vehicles. A multi-level queue could be used if the individual queues are served in a strict priority-based fashion. This, however, equals priority-based scheduling. Thus, the scheduling of event-triggered messages to timeslots is chosen to be strictly priority-based with FCFS, if two messages of the same priority are queued. The priorities for messages can be determined in any fashion, as required by the user.

For the application shown in the following, priorities are generated based on Earliest Deadline First (EDF) scheduling, with longer messages having higher priority, if two deadlines are equal. The strict adherence to priorities in the policy-based scheduling allows for mixed criticality applications to be implemented and to guarantee the delivery of all messages within their required deadline.

3.2 Multi-Mode Applications

The proposed framework is designed such that different priorities per message can be considered at various times. This allows to integrate multi-mode applications for different situations the vehicle experiences very efficiently. For example, the automatic parking assistant has a higher priority when parking the vehicle, than when driving on the highway. By contrast, adaptive cruise control has a higher importance on the highway than in parking situations. The proposed approach allows to define multiple modes for different situations the vehicle experiences. These modes are

required to be exclusive and non-overlapping. The algorithm automatically selects the minimum set of messages required to transmit all data without deadline violations. Due to the inherent flexibility of policy-based scheduling, this results in a smaller amount of messages for the overall schedule than in a conventional scheduling approach where oversampling is required.

3.3 Wrapper PDUs

When placing application layer messages into wrapper PDUs and, consequently, into time slots, the borders of time slots are factored out. The sum of all wrapper PDUs for one ECU appears as one continuous communication channel to the application layer. This allows the transmission of messages longer than the length of one static slot, e.g. for diagnosis or programming of ECUs, by continuing messages in following timeslots. However, the loss of time slot boundaries also leads to two issues. First, receiving ECUs can not identify separate messages anymore, as there are no time slots or other start/end indicators for messages inside a wrapper PDU (Segmentation). Second, the implicit addressing scheme which is inherent to TDMA slots is lost (Addressing). Receiving ECUs have no possibility to identify which type of message is transmitted or to which ECU a message is addressed. To solve these two issues, an additional header is introduced to be transmitted with every application layer message. As this header introduces overhead into the system, it needs to be as small as possible.

In the following, the issues are addressed and the fields of the message header are introduced:

Segmentation. To distinguish two messages, a length field is introduced into the header. This 8 bits long field allows to specify the length of the following payload up to a maximum of 255 bytes. This header can be placed in multiple positions, such as in the beginning of every wrapper PDU, allowing a higher bandwidth efficiency by combining the headers of all included messages. This placement, however, reduces flexibility, as all messages transmitted in one wrapper PDU need to be known at the start of the wrapper PDU. Messages arriving late could not be added. Placing the header in front of every application layer message (see Figure 3) allows a higher flexibility, as high priority messages arriving late can still be added while the slot transmission is in progress.

Addressing. In a time-triggered system, the assignment of messages to time slots also contains an implicit addressing scheme. Due to the placement of application layer messages into the continuous communication channel generated by wrapper PDUs, this implicit addressing information is lost. In TDMA, every transmitting ECU knows where each message is to be transmitted and each receiving ECU knows how to interpret a received message in a timeslot, as only one type of message is allowed for transmission at any one position in the schedule. To compensate for this, a message type field is introduced into the header of every application layer message. This is a 16-bit field, describing the type of the message. The addressing is similar to the approach as it is done in the FlexRay dynamic segment. A message type field of 16-bit allows 65536 different message types. This amount is sufficient for most FlexRay communication. It is the same value as used in the FlexRay dynamic segment. With the introduction of this field, the addressing information is restored and the receiving ECUs are able to process the message.

In addition to the header in front of every application layer message, a header of 8 bits length, called preemption indicator (PI), is added to every wrapper PDU. This header allows to implement preemption in a time-triggered system. In case a high priority message arrives, while a low priority message is transmitting, the high priority message may interrupt the lower priority message at the beginning of the next timeslot. The preemption indicator is used to signal the number of preempting messages at the beginning of a timeslot. The preempting messages are transmitted first, afterwards, the lower priority message is continued as illustrated in Figure 3. This work does not consider non-preemptive scheduling. As preemptive scheduling yields significantly lower latencies at the cost of only minimal overheads in the waiting queues of ECUs and slightly less net bandwidth on the bus, the non-preemptive case does not add any advantages.

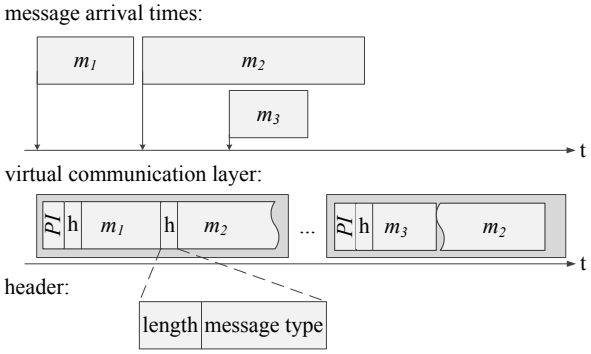

Figure 3: Preemptive scheduling of application layer messages for the proposed approach with continuous virtual communication channel. The arrows mark arrivals of messages. Message m_3 arrives after message m_2, but the priority of m_3 is greater than the priority of m_2. The messages do not need to be placed in consecutive slots.

4. DESIGN-TIME SCHEDULING

In the following, a scheduling algorithm is developed which allows the transmission of application layer messages in event-triggered fashion. The scheduling algorithm creates a time-triggered schedule for the policy-based communication layer. This is based on the application layer messages to be scheduled in the policy-based approach.

For the time-triggered layer, a schedule needs to be generated, incorporating all wrapper PDUs. This scheduling is performed at time of implementation. Two approaches have been developed for this contribution, a heuristic and an ILP. While it is well established to use a heuristic for FlexRay scheduling, the developed ILP shall serve as a benchmark for the heuristic, showing the optimal solution to the given problem. Both approaches are described in the following.

For both approaches, heuristic and ILP, minor differences exist for the different FlexRay versions. While FlexRay 2.1A requires that each slot in every cycle contains the same message type, FlexRay 3.0.1 allows to assign different message types to slots in different cycles (see Figure 4). The developed algorithms account for these differences.

In the following, the terms message and application layer message refer to a message descriptor, including length, period and deadline, not a specific message with content to be sent on the bus.

The expected input to the scheduler is a set of application layer messages to be scheduled. Additionally, the parameters of the schedule need to be defined. These include the number of static slots, the slot size, the number of cycles and the cycle length, among others. A reservation for prede-

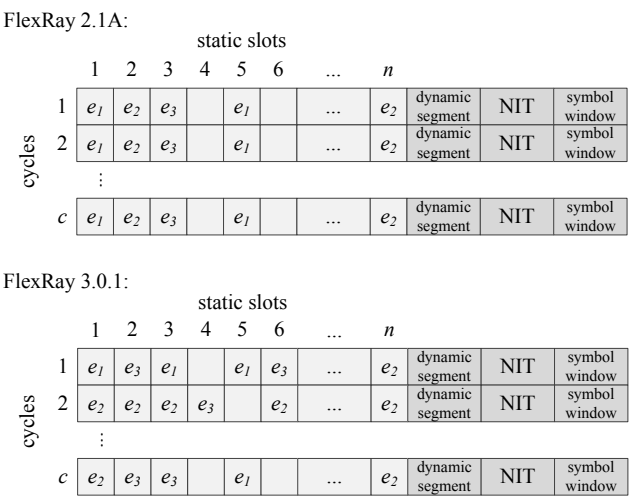

Figure 4: Comparison of FlexRay 2.1A and 3.0.1 scheduling. For FlexRay 2.1A we only consider one cycle to assign an ECU. For FlexRay 3.0.1 the whole schedule is considered, as multiple ECUs can share one slot. The basis of assignment is one frame.

fined time-triggered messages, which are transmitted within the conventional FlexRay static segment can be added as well. The slots occupied by these messages will not be used to schedule wrapper PDUs, thus achieving coexistence with conventional FlexRay.

The deadline of every message is used as a priority indicator, with a shorter deadline equaling a higher priority. This ensures that all deadlines are satisfied, as the message queues are ordered along message deadlines.

The set M of application layer messages m to be scheduled is described by the following tuple:

$$m = (N, s, R, l^m, t_p^m, t_d^m), m \in M \tag{1}$$

with

- N: name of m, a unique string in the FlexRay cluster,
- s: sender, name of the ECU sending m,
- R: receivers, list of receiving ECUs,
- l^m: length of m in bytes,
- t_p^m: period of m in milliseconds,
- t_d^m: deadline of m in milliseconds.

All messages are sent and received by an ECU e, with e being part of the set of all ECUs in the cluster, $e \in E$.

A schedule is defined by the following parameters:

- c: number of cycles, required,
- n: number of static slots per cycle, required,
- t_{dc}: duration of cycle in milliseconds, required,
- t_{ds}: duration of static slot in milliseconds, required,
- b: length of one static slot in bytes, required,
- M_p: set of predefined messages, optional.

Based on these inputs, both scheduling algorithms, heuristic and ILP, described in the following, calculate an assignment of static slots to ECUs. To ensure compatibility to conventional FlexRay, a minimal set of placeholder messages is generated from these assignments that is used to reserve slots in the FlexRay schedule. These place holder messages are the described wrapper PDUs.

4.1 Heuristic

This section describes the heuristic approach to policy-based message scheduling. An algorithm has been developed, using the application layer messages and schedule parameters as input to calculate the assignment of static slots to ECUs. The required number of slots per cycle and frames for the complete schedule are determined for FlexRay 2.1A and FlexRay 3.0.1, respectively. The heuristic processes the input data in two steps:

1. slot distance calculation
2. slot allocation

In the first step, the maximum distance between two slots for every application layer message of an ECU is calculated, such that the deadline for every message is met. The second step allocates all slots into the schedule with the given parameters, adjusting message and slot distances, such that all messages fit the schedule.

1) Slot distance calculation. The first part of the heuristic calculates the required number of slots and frames for the schedule, based on the bandwidth requirements of each ECU. The bandwidth is determined by the periods, sizes and lengths of messages, as well as their deadlines. A value dist_e is calculated for every ECU e, describing the distance two consecutive slots can be allocated apart, such that all bandwidth and deadline requirements are fulfilled.

The upper bound of the WCRT $r_{\text{dist}_e}(m)$ for a message in terms of slot lengths needs to be calculated. This calculation is based on the distance between two slots (see Equation 6) for an ECU and the WCRT $r_{\text{hp}}(m)$ of all higher priority messages on this ECU:

$$\forall e \in E, \forall m \in M_e : r_{\text{dist}_e}(m) = \text{dist}_e + (\text{dist}_e \cdot r_{\text{hp}}(m)) \tag{2}$$

The utilization of all higher priority messages describes the delay $r_{\text{hp}}(m)$ caused to a message and is based on the deadline t_d^m of message m, as well as the period $t_p^{\tilde{m}}$ and the length $l^{\tilde{m}}$ of all higher priority messages $\tilde{m}$ of message m. This delay is normalized to the slot length b:

$$r_{\text{hp}}(m) = \frac{1}{b} \cdot \left(\sum_{\tilde{m} \in \text{hp}(m)} \left(\frac{t_d^m}{t_p^{\tilde{m}}} \cdot l^{\tilde{m}} \right) \right) \tag{3}$$

The response time $r_{\text{dist}_e}(m)$ for every message m needs to be shorter than the deadline t_d^m and the distance dist_e between two slots needs to be lower than the maximum distance between two slots for one ECU:

$$r_{\text{dist}_e}(m) \leq t_d^m \tag{4}$$

$$\text{dist}_e \leq \text{dist}_{e,\max} \tag{5}$$

The maximum distance is computed from the available number of slots n in a cycle and the required number of slots $s_{c,e}$ for an ECU e.

$$\text{dist}_{e,\max} = \frac{n}{s_{c,e}} \tag{6}$$

The required number of slots $s_{c,e}$ is determined by the overall number of slots $slots_e$ for one ECU, divided by the

number of cycles c. As only complete slots can be allocated, this number needs to be an integer value. To ensure a sufficient number of slots in the cycle, this value is rounded to the next higher integer number:

$$s_{c,e} = \left\lceil \frac{\text{slots}_e}{c} \right\rceil \tag{7}$$

The required number of slots per ECU slots_e is based on the duration of the schedule $t_{dc} \cdot c$, the periods t_p^m of all messages $m \in M_e$ and the lengths l^m of these messages:

$$\text{slots}_e = \frac{1}{b} \cdot \sum_{m \in M_e} \frac{t_{dc} \cdot c}{t_p^m} \cdot l^m \tag{8}$$

This describes the bandwidth requirement of the ECU. The bandwidth is normalized to one slot length b.

Based on these equations, the maximum distance between two consecutive slots of one ECU is defined by the maximum distance that satisfies the deadline requirement of all messages $m \in M_e$ for this ECU:

$$\arg\max_{\text{dist}_e \in \mathbb{N}} \{ r_{\text{dist}_e}(m) \leq t_d^m \quad \forall m \in M_e \} \tag{9}$$

2) Slot allocation. After the distances between messages have been determined for each ECU, the allocation of slots is described in the following, as depicted in Figure 5 for FlexRay 2.1A. These steps are executed for all ECUs in ascending order of their previously calculated slot distance dist_e. The slot allocation starts with the first free slot in the cycle (1). Based on this slot, all other slots are allocated in distance dist_e apart (2). This placement also adjusts for the dynamic segment, Network Idle Time (NIT) and symbol window. The slot allocation completes when sufficient slots are placed throughout the complete schedule in distance dist_e (3). This also considers the distance between the last slot and the first slot of every schedule to adjust for the final dynamic segment, NIT and symbol window. Should any of these steps fail for any ECU, because the shortest available slot distance is too long, a deadline violation may occur and the system is considered not schedulable (4). In case all steps succeed, the algorithm terminates with the correctly allocated results (5).

The slot allocation for FlexRay 3.0.1 is performed similarly to the calculations for FlexRay 2.1A. In contrast to FlexRay 2.1A, FlexRay 3.0.1 supports the assignment of identical slots in different cycles to different ECUs (see Figure 4). This allows more flexibility in allocating the slots. To accommodate this flexibility, the algorithm has been adjusted to calculate the slots over one complete schedule ($c \cdot n$ slots), instead of one cycle (n slots). This results in a few additional checks to cover the dynamic segment, NIT and symbol window at the end of every cycle. As FlexRay 3.0.1 does not require the assignment of a slot in every cycle, dist_e might be longer than the shortest deadline for an ECU. To accommodate this, the algorithm continuously checks the deadline constraint when allocating slots. Additionally, the end condition (see Figure 5 (3)) has been adjusted to incorporate a full schedule instead of one cycle.

4.2 ILP

In the following, an ILP formulation which determines an optimal slot to sender, or frame to sender, assignment for FlexRay 2.1A or 3.0.1, respectively, is presented. While an ILP allows to find the optimal solution to a given problem,

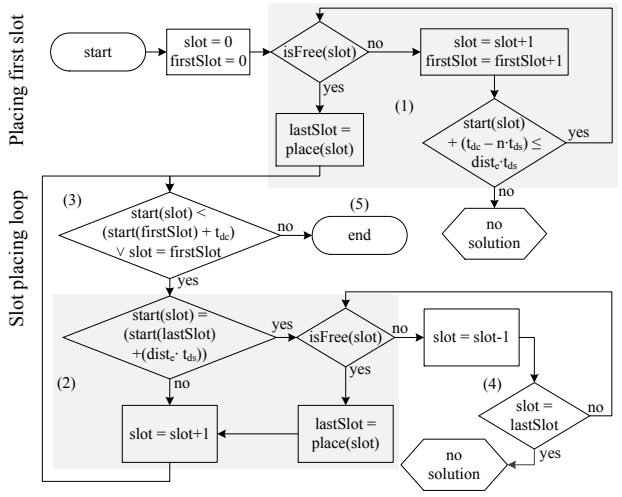

Figure 5: FlexRay 2.1A scheduling algorithm for slot allocation. Based on the initial slot found in the first part of the algorithm, other slots are allocated a maximum of dist_e slots apart. The function isFree(x) determines if slot x is in use by another ECU and the function start(x) returns the start time of slot x. These calculations are performed for every ECU.

it does not scale well. Thus, the proposed ILP shall be used as a benchmark, evaluating the performance of the heuristic, wherever possible.

The ILP determines a schedule with minimal bandwidth utilization. The ILP approach is based on the following constants and variables:

- $n_{\text{all}} = \left\lceil \frac{t_{dc}}{t_{ds}} \right\rceil$: theoretical number of slots fitting in one cycle including the dynamic segment.
- P_e: set containing all message periods t_p^m for an ECU e.
- $\mathbf{x}_{k,e}$: binary variable indicating if slot/frame k is occupied by ECU e (1) or not (0).

To support both FlexRay 2.1A and 3.0.1, the constant n_{tot} is introduced, defining the number of frames the ILP considers. For FlexRay 2.1A n_{tot} equals the number of slots for one cycle, as slots might not be shared between senders:

$$n_{\text{tot}} = n_{\text{all}} \tag{10}$$

For FlexRay 3.0.1 n_{tot} equals the number of frames for all cycles allowing to divide the slot among multiple ECUs:

$$n_{\text{tot}} = c \cdot n_{\text{all}} \tag{11}$$

The objective of the ILP formulation is to minimize the number of occupied frames:

$$\min \sum_{\forall e \in E} \sum_{k \in \{0,1,..,n_{\text{tot}}-1\}} \mathbf{x}_{k,e} \tag{12}$$

For a certain message period t_p^m, a minimal number of slots has to be assigned to fulfill the bandwidth and deadline requirements of all messages m with periods of t_p^m. This is considered for every message m of every ECU e in Equation 13. The number of slots required is determined by the length of the message l^m, normalized to one slot length b. The

right-hand side also considers preemption of messages m by higher priority messages $\tilde{m}$ with a deadline $t_p^{\tilde{m}}$ shorter than the message deadline t_p^m. The deadline requirement is ensured on the left-hand side by placing at least one slot within every period t_p^m. To ensure that the first and the last frame also keep the maximal distance, the modulo operation % ensures that also the first element in the schedule is considered:

$$\forall e \in E, i \in \{0,1,..,n_{\text{tot}}-1\}, t_p^m \in P_e\text{:}$$
$$\sum_{k \in \{i,\ldots,i+\left\lfloor \frac{t_p^m}{t_{ds}} \right\rfloor -1\}} \mathbf{x}_{k\%n_{\text{tot}},e} \geq \left\lceil \sum_{\tilde{m} \in \{\tilde{m} | t_p^{\tilde{m}} \leq t_p^m, \tilde{m} \in M_e\}} \frac{t_p^m}{t_p^{\tilde{m}}} \cdot \frac{l^m}{b} \right\rceil \tag{13}$$

For all ECUs, no frames shall be placed outside the static segment:

$$\forall e \in E, k \in \{0,1,..,n_{\text{tot}}-1\}, \exists k\%n_{\text{all}} > n\text{:}$$
$$\mathbf{x}_{k,e} = 0 \tag{14}$$

For every frame in the schedule, it needs to be ensured that two ECUs cannot occupy the same frame:

$$k \in \{0,1,..,n_{\text{tot}}-1\}, \exists k\%n_{\text{all}} < n\text{:}$$
$$\sum_{\forall e \in E} \mathbf{x}_{k,e} = 1 \tag{15}$$

The solution of the ILP delivers the assignment of slots and frames to ECUs for FlexRay 2.1A and FlexRay 3.0.1, respectively. This is used to reserve timeslots in the FlexRay static segment, in the form of wrapper PDUs.

5. EXPERIMENTAL RESULTS

In this section, the policy-based scheduling is evaluated and compared against the conventional approach from [9]. For this purpose, a FlexRay scheduling framework, including the policy-based heuristic and ILP scheduling, has been implemented in Java, allowing to compare the scheduling algorithms to the framework from [9]. The developed scheduling framework is connected to the development toolchain via Field Bus Exchange Format (FIBEX). FIBEX is a standardized data format for the exchange of data related to bus systems between tools. It is based on eXtensible Markup Language (XML) and is the standard data exchange format for FlexRay systems. The proposed system supports import and export for FIBEX versions 2.0.1 and 3.1.

To evaluate the performance of the developed algorithms for FlexRay, the input parameters are varied and the output is surveyed. The performance is measured in terms of bandwidth and WCRT. Additionally, the computational performance of the different approaches is evaluated. The input to all algorithms consists of externally generated and verified FlexRay parameters, as well as a set of messages to be scheduled. To accommodate for statistical variations in the message sets, multiple sets have been generated for every test. The FlexRay system is defined by a cycle duration t_{dc} of 5ms, 62 static slots per cycle (n), a slot size b of 42 bytes and 64 cycles per schedule (c). The set of messages is generated randomly from a set of given parameters. The parameters are varied for the different testcases and described below.

The metrics for comparison are the average number of slots and frames of the scheduling runs, for FlexRay 2.1A and FlexRay 3.0.1, respectively. As the number of slots and frames represent the bus utilization, a lower number of slots and frames is better. In addition to the average number of slots and frames, error margins are given in the form of the standard deviation from the average value over the set of generated schedules.

It is important to note that the conventional scheduling approach assumes an AUTOSAR architecture, requiring one byte at the beginning of every slot for administrative information and is thus working on a slot length of $b = 41$ bytes. Similarly, the policy-based approach requires one byte as preemption indicator and thus also works with $b = 41$ bytes available for payload.

For all testcases, multi-mode messages have been employed. The messages have been generated such that 50% of all messages in a testcase have two modes. For all multi-mode messages, the period and size of a message may vary for the modes. However, it is guaranteed that a longer or equal period always contains a longer or equal message size.

5.1 Size Variations

To determine the performance of policy-based message scheduling under different message lengths, the average size of all messages in the system is varied. This is achieved by adjusting the distribution of message sizes, starting from the distribution used in [9]. The test is repeated multiple times, as the set of messages is generated in a statistical process. The results for this test are shown in Figure 6. The size distributions of all messages in the system are varied and sorted by increasing average message size. The last two sets of messages include messages with length longer than one slot. This can be used for future applications, such as a combination with Ethernet and Audio Video Bridging (AVB)/Time-Sensitive Networking (TSN). The message length used here is 64 bytes, the minimum length of an Ethernet frame.

As it can be seen from Figure 6, the conventional scheduling fails to process messages longer than one slot length, thus failing to schedule the given sets of messages. All shown testcases with size variations are based on 58 different periods. Policy-based scheduling can schedule any message length, independent from the slot length in the static segment.

FlexRay 2.1A. As can be seen in Figure 6a, for smaller message sizes, the policy-based scheduling performance about matches the performance of conventional scheduling. For larger message sizes of more than one slot length, the conventional scheduling fails to perform its task, while the policy-based approach can continue to schedule messages. The minimum difference between the two approaches is 2%, while the maximum difference is 8.7%. However, for larger message sizes, a comparison is not possible, as the conventional algorithm fails to schedule these messages. This is due to the fact that in a conventional FlexRay scheduling algorithm, the maximum allowed message size is equal to the slot length. By contrast, the policy-based message scheduling can schedule messages of arbitrary lengths. While the FlexRay system is not fully utilized, the policy-based scheduling will always be able to accommodate messages, even if the longest message length is larger than one slot length.

FlexRay 3.0.1. Figure 6b shows the scheduling results for FlexRay 3.0.1. For smaller message sizes, the policy-based scheduling performance almost matches the conventional scheduling. For larger message sizes of more than one slot length, the conventional scheduling fails to find a solution, while the policy-based approach can continue to schedule messages.

Due to the structure of FlexRay 3.0.1, being able to as-

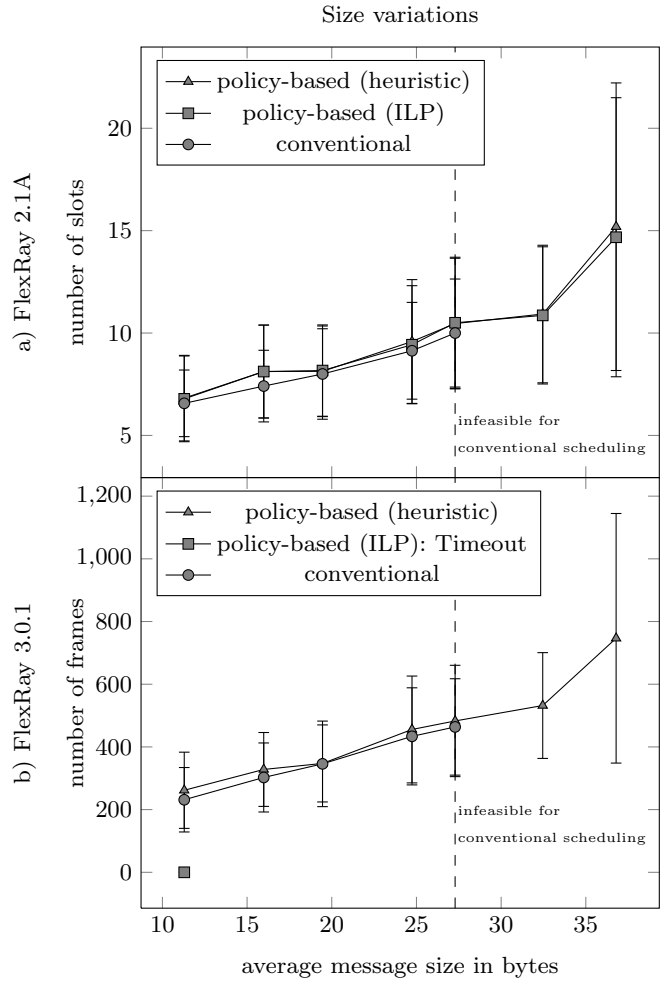

Figure 6: Number of slots and frames required for scheduling with FlexRay 2.1A and FlexRay 3.0.1 for different average message sizes. The system contains 100 different messages and 5 ECUs.

sign frames instead of slots, the complexity of the problem size increases significantly. The increase in complexity of the problem leads to an exponential growth of the constraints and variables in the ILP. Thus, the comparison for FlexRay 3.0.1 is based on the heuristic approach for conventional and policy-based FlexRay, as the ILPs for both approaches can not be calculated with reasonable resources (see Section 5.4). For small messages, the performance difference between the two approaches varies between 0.4% and 13%. Additionally, like in FlexRay 2.1A, the conventional approach is not capable of scheduling messages larger than one slot size, leaving the larger message sets as not schedulable. Again, the policy-based approach can utilize its continuous communication channel to also schedule messages with length longer than one slot and thus is able to schedule all given message sets.

5.2 Latency

Besides the higher flexibility in message lengths, the policy-based scheduling also allows lower WCRTs for all messages in the system than a purely time-triggered system. To verify policy-based scheduling with regard to deadlines and to calculate WCRTs, we employ Real-Time Calculus (RTC) [22]. RTC is an extension of network calculus, tailored to real-time systems [23]. Similar to network calculus, real-time calculus allows to compute the arrival and service curves for a communication system. Based on the Modular Performance Analysis [24], implemented in the RTC Toolbox, the proposed approach is analyzed, to verify that the deadlines of all messages are met. The service and arrival curves are generated from the slot and frame assignment determined per ECU by the proposed algorithm and the set of messages supplied by the user, respectively. Based on the slot and frame assignment from the proposed algorithm, RTC calculates the WCRTs for all messages on all ECUs. These WCRTs are compared to the message deadlines. This way, it can be ensured that all messages meet their deadlines at any point in time.

Additionally, the WCRTs are compared to those of a conventional FlexRay system. The results for FlexRay 2.1A and FlexRay 3.0.1 are shown in Figure 7. In conventional FlexRay scheduling, a message can only be transmitted at a fixed place in the TDMA scheme. Thus, the WCRT of any message is equal to its period, or the oversampling required by the scheduling algorithm. As we are using a set of periods which does not require oversampling, the WCRTs for conventional FlexRay are equal to the message periods, which in turn, are equal to the message deadlines, for both heuristic and ILP. From Figure 7a, it can be seen that in policy-based scheduling, the latencies for all messages are significantly shorter than the deadlines achievable in the conventional scheduling approach. This holds for all messages on all ECUs, regardless of their size or deadline. This is due to the flexible EDF scheduling and the implemented preemption capabilities. As messages are not bound to timeslots, far lower worst-case latencies can be achieved. Messages with short deadlines always have a higher priority and can be transmitted first, while messages with longer deadlines are queued. As slots can be allocated more flexibly in FlexRay 3.0.1, compared to FlexRay 2.1A, the differences in latency between policy-based scheduling and conventional scheduling are smaller (see Figure 7b). However, in absolute terms, improvements of policy-based scheduling over conventional FlexRay are still significant, especially for longer message deadlines. Figure 7 omits the ILP approach to policy-based scheduling. The results are in the same range as the results for the heuristic approach.

5.3 Period Variations

The second relevant parameter of the message distributions is the set of periods used for the message generation. A higher set of periods allows more flexibility, as the application developer does not require to adjust his application to the communication system, but can freely schedule the application, as required. This can make control algorithms and networks more efficient, as less oversampling and messages are required to reach stable operation. The size distribution used for this testcase is taken from [9]. The number of periods has been increased for different message sets, to verify the influence of this parameter on the performance of policy-based scheduling, compared to conventional scheduling.

Conventional FlexRay is optimized for a set of periods defined in the FlexRay standard. This set has been used as a starting point and further periods have been added for other sets. For a cycle time t_{dc} of 5ms, the basic set of periods is defined as 5ms, 10ms, 20ms, 40ms, 80ms, 160ms, 320ms. All periods are integer multiples of one cycle length, as the conventional scheduling algorithm, in contrast to the policy-based scheduling, is not capable of scheduling fraction pe-

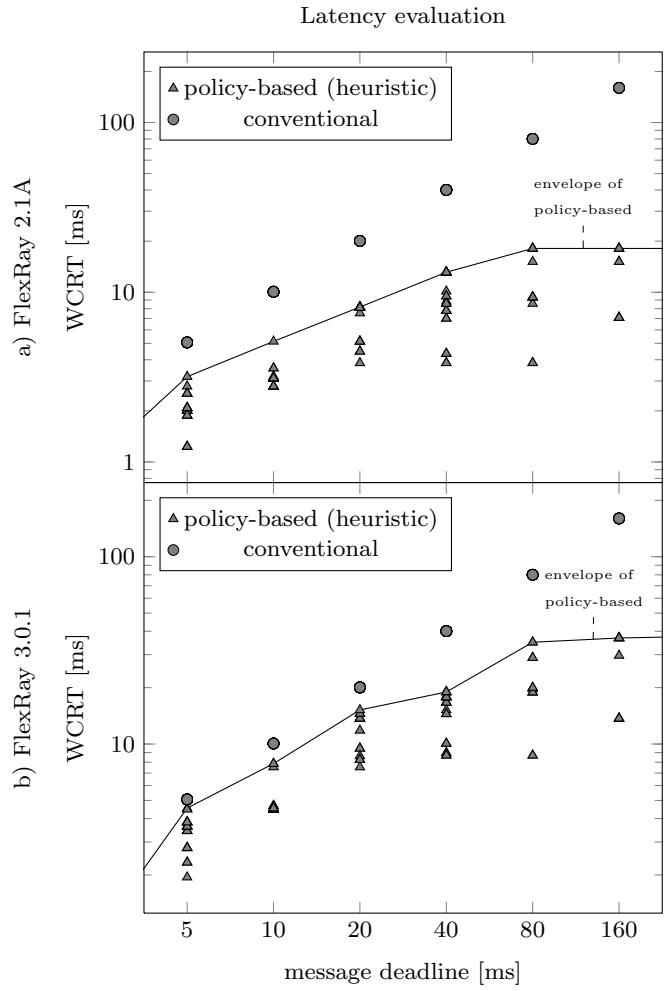

Figure 7: Exemplary worst-case response time (WCRT) analysis for FlexRay 2.1A and FlexRay 3.0.1 for heuristic scheduling of policy-based approach and conventional scheduling. The policy-based scheduling shows a significant improvement in latencies for both FlexRay versions and all messages in the system.

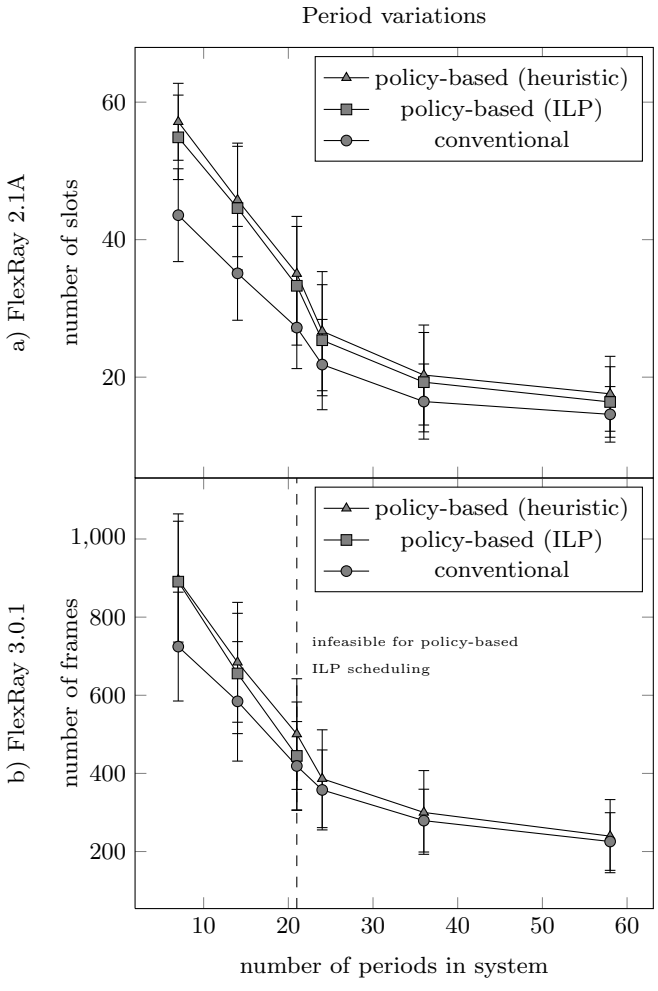

Figure 8: Number of slots and frames required for scheduling with FlexRay 2.1A and FlexRay 3.0.1 for different numbers of periods. Based on the shown performance differences, the typical use cases for the policy-based scheduling would start at 30 different periods and beyond.

riods. The results for this test are shown in Figure 8 and discussed in the following.

FlexRay 2.1A. When comparing the policy-based approach to the conventional FlexRay scheduling, the performance for a low number of periods is expectedly lower, as the conventional algorithm is optimized for this application (see Figure 8). However, when crossing a threshold of 30 different periods in the system, the policy-based approach performs only slightly worse than the conventional algorithm. Although having a smaller net bandwidth, due to overhead in the message headers, performance is nearly equal for a high amount of different periods. It is to note that this testcase is limited to integer periods, as fraction periods cannot be processed by the conventional scheduling algorithm and thus have no basis for comparison. This limits the policy-based approach in its performance. As it is common in combined time- and event-triggered systems, the policy-based approach is a trade-off between the bus utilization on the one hand and lower latencies and increased flexibility, manifesting e.g. in larger messages, on the other.

FlexRay 3.0.1. Due to the complexity of computations, this comparison focuses on the heuristic approaches of conventional and policy-based scheduling. The ILP for policy-based scheduling could only be performed for the first three sets of messages, which have a lower count of periods.

Figure 8b shows the comparison between the two approaches for FlexRay 3.0.1. The more flexible slot assignment in FlexRay 3.0.1 allows the policy-based scheduling an increase in performance, compared to FlexRay 2.1A. The difference in performance between the conventional and policy-based scheduling falls from 19.1% for the set of periods optimized for conventional FlexRay to 5.7% for a larger selection of periods. These testcases show the potential of the policy-based algorithm. With policy-based scheduling, the user can select any set of periods and is not limited by the communication system. Even non-integer periods are supported by policy-based scheduling, which is not shown here for the sake of comparison to conventional FlexRay.

The flexibility gained by employing the policy-based scheduling approach reduces the design effort for a communication system significantly, as it can be adjusted to the applications. Additionally, the latency is lowered and larger messages sizes can be transmitted. Policy-based scheduling

can also be used in prototyping, when messages in a system change often, as it is not required to always generate a new TDMA schedule, whenever one single message changes.

5.4 Computational Performance

As the runtime scheduling of application layer messages to the virtual communication layer is limited to a priority queue, the computational and memory overhead on the ECUs at runtime is minimal. By contrast, the design time scheduling solves the algorithms presented in the previous sections and exhibits varying performance, depending on the FlexRay version and testcase involved. The heuristic approach to policy-based scheduling achieves results in a short amount of time, with the given number of messages per cluster. A FlexRay 2.1A cluster, as shown in this section, can be calculated in about 10 milliseconds, a FlexRay 3.0.1 cluster requires 570 milliseconds on average. The computations have been performed on a workstation with an Intel Xeon quad-core processor with 3.2 GHz and a 64-Bit Java Virtual Machine.

As solver for all policy-based ILPs, SAT4J has been employed [25]. The runtime of the ILP has been limited to 15 minutes for one FlexRay cluster. In case no solution is found within this time frame, the results have been discarded. Due to the higher complexity and exponentially higher number of variables in the ILP for FlexRay 3.0.1 computations, the requirements, especially on the available main memory, are much higher. A high performance computing cluster has been employed to solve these computations, including 4 nodes with 2 Intel Xeon octa-core processors with 2.6 GHz each. The nodes are networked over a 60 Gbps InfiniBand connection and each node contains 192 GB of main memory, allowing to parallelize the computations and boost the main memory available per ILP. We allowed up to 22 GB of main memory for each ILP to achieve the results in this section. This sets the bound for scheduling at 21 periods, as shown in Figure 8. To schedule a larger set of periods, a larger amount of main memory per ILP process is required. While the performance requirement for the ILP seems high, it is to note that the ILP is only used as a benchmark in this paper. As shown, the performance of the developed heuristic almost matches the performance of the ILP. Thus, in a production environment, the heuristic would be used, being able to process FlexRay clusters on standard computer equipment within milliseconds.

6. CONCLUSION

This paper proposes a policy-based scheduling architecture to integrate a virtual event-triggered communication layer into time-triggered communication systems. The policy-based layer is designed to provide asynchronous message transmissions with preemption. In this context, simplified verification of worst-case response times (WCRT) for application layer messages is enabled. A set of algorithms is proposed to schedule policy-based messages in time-triggered systems while ensuring coexistence with existing TDMA communication. An implementation to integrate the proposed approach into the FlexRay static segment has been developed. Based on this implementation, multiple tests have been carried out to evaluate the performance of policy-based FlexRay. The results illustrate a higher flexibility in message size and period selection as well as lower end-to-end latencies with an acceptable trade-off in bandwidth utilization.

Future work comprises the integration of AVB and TSN into our framework for seamless combination of different bus systems in future in-vehicle communication architectures.

7. REFERENCES

[1] K. Tindell, A. Burns, and A. J. Wellings. Calculating controller area network (CAN) message response times. *Control Engineering Practice*, 1995.

[2] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien. Controller area network (CAN) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*, 35(3):239–272, 2007.

[3] E. G. Schmidt and K. Schmidt. Schedulability analysis and message schedule computation for the dynamic segment of FlexRay. *Proc. of VTC*, page 1–5, 2010.

[4] H. Kopetz and G. Grunsteidl. TTP - a protocol for fault-tolerant real-time systems. *Computer*, 27(1):14 –23, 1994.

[5] M. Hamdaoui and P. Ramanathan. A service policy for real-time customers with (m, k) firm deadlines. *Proc. of FTCS*, page 196–205, 1994.

[6] H.-Y. Tyan and J. C. Hou. A rate-based message scheduling paradigm. In *Proc. of WORDS*, page 203–215, 1999.

[7] G. Manimaran, M. Shashidhar, A. Manikutty, and C. S. R. Murthy. Integrated scheduling of tasks and messages in distributed real-time systems. *Proc. of WPDRTS*, page 64–71, 1997.

[8] M. Grenier, L. Havet, and N. Navet. Configuring the communication on FlexRay-the case of the static segment. *Proc. of ERTS 2008*, 2008.

[9] M. Lukasiewycz, M. Glaß, P. Milbredt, and J. Teich. Flexray schedule optimization of the static segment. In *Proc. of CODES+ISSS*, volume 9, page 363–372, 2009.

[10] K. Schmidt and E. G. Schmidt. Message scheduling for the FlexRay protocol : The static segment. *IEEE Transactions on Vehicular Technology*, 58(5):2170–2179, 2009.

[11] K. Schmidt and E. G. Schmidt. Optimal message scheduling for the static segment of FlexRay. *Proc. of VTC*, 2010.

[12] H. Zeng, M. Di Natale, A. Ghosal, and A. Sangiovanni-Vincentelli. Schedule optimization of Time-Triggered systems communicating over the FlexRay static segment. *IEEE Transactions on Industrial Informatics*, 7(1):1–17, February 2011.

[13] H. Zeng, A. Ghosal, and M. Di Natale. Timing analysis and optimization of FlexRay dynamic segment. *Proc. of ICCIT*, page 1932–1939, June 2010.

[14] M. Neukirchner, M. Negrean, R. Ernst, and T.T. Bone. Response-time analysis of the flexray dynamic segment under consideration of slot-multiplexing. In *Proc. of SIES*, pages 21–30, June 2012.

[15] R. Maier. Event-triggered communication on top of time-triggered architecture. In *Proc. of DASC*, volume 2, page 13C5–1 – 13C5–9 vol.2, 2002.

[16] V. Claesson and N. Suri. TTET: event-triggered channels on a time-triggered base. *Proc. of ICECCS*, 2004.

[17] H. Kopetz and G. Bauer. The Time-Triggered architecture. *Proceedings of the IEEE*, 91(1):112 – 126, 2003.

[18] R. Obermaisser. End-to-End delays of Event-Triggered overlay networks in a Time-Triggered architecture. *Proc. of INDIN*, page 541–546, July 2007.

[19] R. Lange and F. Vasques. Guaranteeing real-time message deadlines in the FlexRay static segment using a on-line scheduling approach. *Proc. of WFCS*, 2012.

[20] A. Easwaran, I. Shin, O. Sokolsky, and I. Lee. Incremental schedulability analysis of hierarchical real-time components. In *Proceedings of the 6th ACM & IEEE International Conference on Embedded Software*, EMSOFT '06, page 272–281, New York, NY, USA, 2006. ACM.

[21] M. Anand, S. Fischmeister, and I. Lee. A comparison of compositional schedulability analysis techniques for hierarchical real-time systems. *ACM Trans. Embed. Comput. Syst.*, 13(1):2:1–2:37, September 2013.

[22] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *Proc. of ISCAS*, volume 4, pages 101–104 vol.4, 2000.

[23] J.-Y. Le Boudec and P. Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. 2001.

[24] E. Wandeler, L. Thiele, M. Verhoef, and P. Lieverse. System architecture evaluation using modular performance analysis: A case study. *International Journal on Software Tools for Technology Transfer*, 8(6):649–667, October 2006.

[25] D. Le Berre and A. Parrain. The SAT4J library, release 2.2, system description. *Journal on Satisfiability, Boolean Modeling and Computation*, 7:59–64, 2010.