Efficient Algorithm for Jitter Minimization in Time-Triggered Periodic Mixed-Criticality Message Scheduling Problem

Antonin Novak
DCE FEE and CIIRC
Czech Technical University in Prague
Czech Republic
antonin.novak@cvut.cz

Premysl Sucha
DCE FEE
Czech Technical University in Prague
Czech Republic
suchap@fel.cvut.cz

Zdenek Hanzalek
DCE FEE and CIIRC
Czech Technical University in Prague
Czech Republic
hanzalek@fel.cvut.cz

ABSTRACT

The current research in real-time scheduling focuses mostly on the certification of functionalities with respect to safety requirements under conservative assumptions or to achieve efficient resource utilization but with optimistic assumptions. With growing system complexity, the safety certification is becoming hard, especially in event-triggered environments. In time-triggered environments, the network nodes are synchronized by clocks and follow a static schedule hence they are easily certifiable. However, the time-triggered paradigm has two disadvantages. The first one is its general non-flexibility (e.g. message retransmission, efficient resource usage) and the second one is the need for an efficient scheduling algorithm producing the schedule.

In this paper, we propose a solution to both of these issues. To address the first disadvantage, we propose a method for non-preemptive message retransmission in time-triggered environments while preserving the efficient use of resources. Based on the message criticality we allow a certain number of retransmissions. The observed prolongation of the processing time of a highly critical message is compensated by skipping transmission of less critical messages. Static schedules then contain all alternatives caused by the retransmissions that can occur during a run time execution. Schedules conform with certification requirements imposed on the highly critical messages while preserving the efficient use of resources. To address the second disadvantage, we propose a novel heuristic scheduling algorithm with an unscheduling step for solving large instances of periodic message scheduling problem. The message periodicity is assumed to be a power of two and the objective is to minimize the maximal jitter. The efficiency of the approach is demonstrated on problem instances with up to 2000 messages.

RTNS '16, October 19-21, 2016, Brest, France

ACM ISBN .

DOI: http://dx.doi.org/10.1145/2997465.2997481

CCS Concepts

•**Theory of computation** → **Mathematical optimization;** •**Computer systems organization** → **Real-time systems;** *Embedded and cyber-physical systems;* •**Computing methodologies** → *Planning and scheduling;*

Keywords

Mixed-Criticality, Message Scheduling, Offline Scheduling

1. INTRODUCTION

Modern vehicles consist of many ECUs (*Electronic Control Units*) that communicate together over communication buses. ECUs are implementing a broad range of functionalities including advanced driver assistants and safety-critical applications like x-by-wire systems (e.g. drive-by-wire). The current trend in the automotive industry shows that requirements on the throughput and determinism of the communication channel will grow.

Traditionally, event-triggered communication protocols such as CAN *(Controller Area Network)* [5] are commonly used. Due to the improving capabilities of driver assistance systems, the amount of data transferred through the network in a vehicle is growing. For example, it is often the case that modern car contains more than one camera generating high data traffic. Furthermore, the traffic generated by ECUs is safety-critical in many cases. An example is a lane-keeping assistant which is a critical functionality since undesirable delays in communication or lost messages may result in malfunctions and dangerous failures. Messages generated by ECUs are typically periodic, and low jitter is required to guarantee the quality of control.

Traditional protocols like CAN were not designed for a high data throughput; therefore their usage in modern cars is not suited for driver assistant systems. CAN FD (*Flexible Data Rate*) was designed to support data transfer with speeds >1 Mbit/s. However, it still cannot handle data-intensive applications. Moreover, since the response time analysis in real-life event-triggered communication systems including gateways and precedence relations (between messages on different segments) is very complex problem, safety certification of systems utilizing event-triggered environment is a tremendously difficult task. In time-triggered communication protocols, the response time analysis is easy. There, the network nodes have synchronized clocks and messages

are transmitted at the moments defined by the static schedule. When constructing the schedule, a designer imposes a set of constraints (such as message release and deadline) on the communication that are met in every feasible solution to the scheduling problem. Therefore, the certification of the system can be achieved via showing the feasibility of the produced communication schedule. New protocols were designed to combine both time-triggered and event-triggered communication. For example, FlexRay bus is used nowadays in the automotive industry (e.g. Porsche Panamera, Nissan Infinity Q50). The scheduling of FlexRay static segment can be solved very efficiently due to Dvorak et al. [6]. However, FlexRay's bandwidth is only 10 Mbit/s which is not sufficient for applications like autonomous driving. Therefore, Ethernet-based protocols containing time-triggered paradigm [11] are considered as a successor of FlexRay, at least in the automotive industry. However, the support for the time-triggered paradigm in the Ethernet needs to be contained in the network elements [9].

One of the disadvantages of time-triggered protocols is their non-flexibility. For example, the static schedule does not take into consideration the message retransmission. When a critical message is not delivered due to a disturbance or electromagnetic noise, it may be necessary to repeat its transmission. If one wants to take it into account, then the static schedule has to allocate more of the resource time for these retransmissions. However, message retransmissions are not that frequent thus the average resource utilization may be low. Nevertheless, retransmissions in time-triggered protocols can be resolved in a similar manner like in event-triggered ones. If a more critical message is retransmitted, then it consumes the resource time of a less critical message, which is skipped. However, bringing this dynamic behavior into static time-triggered schedules is a very challenging problem. The schedule has to assume alternative schedules based on the observed run time scenario. In this paper, we introduce this flexibility to the static schedules while modifying the traditional time-triggered scheduling approach. However, the general idea is simple – schedule critical messages apart from each other and interleave them with less critical ones.

1.1 Contribution and Outline

In this paper, we propose a solution to the scheduling of periodic non-preemptive mixed-criticality messages with different processing times in time-triggered communication protocols. We introduce a new scheduling model that constructs static schedules encapsulating alternative run time execution scenarios. Feasible schedules ensure robustness with respect to the processing time prolongation and safety-critical aspects while improving the utilization of the communication resource. Furthermore, we introduce a novel constructive heuristic with an unscheduling step efficient at solving feasibility problems. The aim is to find a low-jitter periodic schedule for messages that takes into consideration the message retransmission. The message periodicity is assumed to be given as a power of two. We solve a successive sequence of feasibility problems in order to minimize the maximal jitter in the schedule. Furthermore, the proposed algorithm can be easily extended to include message release dates, deadlines, and precedence relations.

The rest of the paper is structured as follows. In Section 2 we review scheduling in time-triggered networks, mixed-criticality scheduling and the work on jitter minimization. In Section 3 we describe the Non-Preemptive Mixed-Criticality Match-Up scheduling model which we use to deal with different processing times in time-triggered protocols. In Section 4 we define the problem of minimizing jitter of periodic messages with different processing times. We provide a MIP (*Mixed-Integer Programming*) model, and we show the problem complexity. Finally, Section 5 presents the proposed algorithm based on iterative scheduling with an unscheduling step for mixed-criticality messages. The ability to solve large problems is demonstrated with instances containing up to 2000 messages in Section 6. The conclusions are drawn in Section 7.

2. RELATED WORK

The exhaustive survey on mixed-criticality in real-time systems is presented by Burns and Davis [4]. This research is traditionally concentrated around event-triggered approach to scheduling. It aims towards the construction of scheduling policies that help mixed-criticality systems to achieve a certification and that make efficient use of the shared resources.

In the seminal paper [16] Vestal proposed a method that assumes different WCETs (*the worst-case execution time*) obtained for discrete levels of assurance. Apart from this proposition, the paper presents modified preemptive fixed priority schedulability analysis algorithms. Although we adopted his proposition of different processing times for different levels of assurance, the preemptive model is not suitable for communication protocols, and it significantly changes the scheduling problem.

Baruah et al. [2] formulated the basic model of mixed-criticality systems. They study MC schedulability problem under special restrictive cases in the event-triggered environment. Theis at al. [15] argued that mixed-criticality shall be pursued in time-triggered systems. Baruah's and Fohler's [1] approach in the time-triggered environment assumed preemptive tasks with up to two criticality levels. It makes it unsuitable for communication protocols since the preemption would be costly. In this paper, we follow the model presented by Hanzalek et al. [17]. The paper formulated the scheduling problem of non-preemptive mixed-criticality tasks in time-triggered systems with different processing times. They constructed static schedules with F-shaped tasks allowing them to switch between schedules at different execution levels and match-up back with the basic schedule. Unfortunately, the paper [17] did not address two aspects important for real-life applications: periodic schedules and the actual efficient algorithm solving instances with hundreds of messages.

Schedules with large jitter values are undesired in embedded systems. Output jitter brings difficulties for control loops, and the computational load is not balanced across the hyperperiod. Therefore, we minimize jitter in constructed schedules. Our problem is related to periodic scheduling with constraints on timing properties of tasks. The optimization of FlexRay static segment was done by Lukasiewycz et al. [12], Dvorak et al. [6]. They have transformed periodic scheduling problem in the time-triggered domain to bin-packing problem in order to find a zero-jitter schedule. Taewoong et al. [10] reduced output jitter of real-time tasks under EDF scheduling. Approximation and hardness results for harmonic periods are known due to Eisen-

brand et al. [7].

Therefore, to the best to our knowledge, the retransmission of non-preemptive mixed-criticality messages in periodic static time-triggered schedules was not addressed before. We formulate the problem as a periodic scheduling problem with F-shaped tasks. We express it as a Mixed-Integer Program. Further, we propose an efficient heuristic algorithm solving problem instances with thousands of messages.

3. TIME-TRIGGERED MIXED-CRITICALITY MATCH-UP SCHEDULING

The problem of the non-preemptive mixed-criticality match-up scheduling can be represented by F-shapes [17]. Each F-shaped task represents one message. An F-shaped task is given by its criticality and a set of alternative processing times as follows:

Definition 1. The F-shaped task T_i is a pair $(\mathcal{X}_i, \mathbf{P}_i)$ where $\mathcal{X}_i \in \{1, \ldots, \mathcal{L}\}$ is the task criticality and $\mathbf{P}_i = (p_i^{(1)}, p_i^{(2)}, \ldots, p_i^{(\mathcal{X}_i)}) \in \mathbb{N}^{\mathcal{X}_i}$ is the vector of processing times such that $p_i^{(1)} < p_i^{(2)} < \ldots < p_i^{(\mathcal{X}_i)}$. We say that $p_i^{(\ell)}$ is the processing time of T_i at criticality level ℓ.

Each F-shape represents an approximation of the distribution function describing the uncertainty about its actual processing time. If we consider the processing time of a task T_i as a random variable $\mathcal{T}_i$, then we can relate the vector of its processing times $\mathbf{P}_i$ to the approximation of its distribution function of the processing time. For each *criticality level* $\ell \in \{1, \ldots, \mathcal{X}_i\}$, we define the associated processing time at ℓ as $p_i^{(\ell)} = \mathcal{F}_i^{-1}(c_\ell)$, where $\mathcal{F}_i^{-1}$ is the quantile function of $\mathcal{T}_i$ and c_ℓ is a parameter given by the application requirement for level ℓ. For example, if we identify criticality levels with IEC 61508 SIL (*Safety Integrity Levels*) standard [3], then the task criticality $\mathcal{X}_i$ is given by the SIL of the functionality carried out by the message content and c_ℓ is defined as $1-$*probability of failure* defined by SIL ℓ.

For each task (message) T_i, the vector $\mathbf{P}_i$ then serves as a finite-sized approximation of its CDF (*cumulative distribution function*) of the processing time. The approximation simplifies the scheduling problem and allows us to solve it more efficiently than with the consideration of full CDF. We display the finite-sized vector of increasing processing times in the Gantt chart vertically into layers as the single task (see Figure 1).

Notice, that the F-shapes do not only assume the existence of a probability distribution on the processing time of a task. The basic case of the message retransmission without additional overhead can also be represented. In this case, the F-shapes would have a constant prolongation at each level, and their height would be determined by the number of allowed retransmissions. Therefore, if messages have defined the inner structure that cannot be changed, then processing time prolongation corresponds to the complete message retransmission.

An example of the schedule with F-shaped messages is depicted in Figure 1. There, T_2 and T_3 have criticality 1 and no retransmission is allowed, tasks T_1 and T_5 have criticality 2 and can be retransmitted once. T_4 has criticality 3 and thus total three transmissions are allowed.

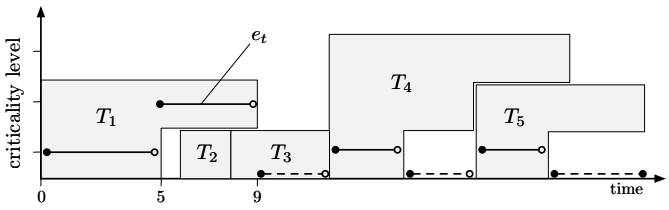

Figure 1: A static schedule with F-shaped tasks with one particular execution scenario e_t.

A solution of the scheduling problem is given by the schedule that switches to the higher criticality level when a prolongation of a message occurs. After its successful completion, it matches up with the original schedule.

By the schedule for a set of F-shaped messages $I_{\mathcal{MC}} = \{T_1, \ldots, T_n\}$ we refer to an assignment $(s_1, \ldots, s_n) \in \mathbb{N}^n$. If no timing requirements on start times of messages are imposed, then the schedule is feasible if and only if messages are not overlapping on any criticality level. Therefore, if T_i precedes T_j in a feasible schedule, then $s_j \geq s_i + p_i^{(\min\{\mathcal{X}_i, \mathcal{X}_j\})}$, where $\min\{\mathcal{X}_i, \mathcal{X}_j\}$ is the maximum common criticality level of T_i and T_j. The inequality ensures that messages are not overlapping on any criticality level and, therefore, only higher critical messages are *covering* less critical ones. The notion of covering influences the run time schedule evaluation governed by the execution policy.

3.1 Execution Policy

A feasible schedule with F-shaped messages describes alternative schedules for any actual realization of the processing time of messages. Observed prolongations of more critical messages are compensated by skipping execution of less critical messages. This behavior can be described in terms of the execution level of the schedule that defines the performed schedule alternative.

Assuming discrete time, let $e_t \in \{0, \ldots, \mathcal{L}\}$ be the execution level of the schedule $(s_1, s_2, \ldots, s_n)$ during the run time execution at time t. Let $H \in \mathbb{N}$ be the length of the hyperperiod (scheduling horizon). The execution policy defines the execution level of the schedule based on processing time prolongations observed during a run time execution according to rules described in Algorithm 1.

Algorithm 1 Execution Policy

1: $e_0 \leftarrow 0$
2: **for** $t \leftarrow 0$ **to** H **do**
3: **if** $\exists T_i : t = s_i$ **and** $e_t = 0$ **then**
4: $e_t \leftarrow 1$
5: execute T_i
6: **end if**
7: **if** $\exists T_i$ that is being executed at t **then**
8: $e_{t+1} \leftarrow \begin{cases} e_t & \text{if } t+1 < s_i + p_i^{(e_t)} \\ e_t + 1 & \text{if } t+1 \geq s_i + p_i^{(e_t)} \text{ and} \\ & T_i \text{ is not completed} \\ 0 & \text{otherwise} \end{cases}$
9: **else**
10: $e_{t+1} \leftarrow 0$
11: **end if**
12: **end for**

The execution starts at $t = 0$ at zero level. Message T_i is executed at time $t = s_i$ if and only if $e_{s_i} = 0$ (i.e. the resource is available). By the assumption, if the message is executed at s_i, then it is completed no later than $s_i + p_i^{(\mathcal{X}_i)}$ (i.e. the $\mathcal{X}_i$ level represents its WCET). The execution level e_t of the schedule is raised, if the executed message T_i is not completed at $s_i + p_i^{(e_t)}$. After upon a message is completed at one of its levels, the execution level is decreased to 0 and stays there until the start time of the next message. Therefore, if the execution level e_t is raised above 1 during the execution of T_i (i.e. T_i is prolonged), then messages that are *covered* by T_i at the level e_t, i.e. $\forall j \in I_{\mathcal{MC}} : s_i + p_i^{(1)} \leq s_j < s_i + p_i^{(e_t)}$, are not executed.

An example of a schedule with F-shaped messages is illustrated in Figure 1. There are five F-shaped messages forming a static schedule. The realized run time execution scenario for this schedule is depicted by the black line. While following the schedule, the actual processing time of T_1 was 9 instead of 5 due to a disturbance. The execution policy states that messages, covered by the prolonged message, are skipped, i.e. T_2 and T_3. After a message is transmitted, the execution matches up with the schedule at the lowest criticality. Therefore, in this example, after T_1 finishes, T_4 is up next. In another possible run time execution scenario, where the processing time of T_1 would be 5, T_2 would follow.

4. PROBLEM STATEMENT

We solve the periodic message problem in time-triggered communications. The aim is to find a static periodic schedule for a given set of communication messages that are of mixed-criticality. Let the length of the basic period be a parameter $\tau \in \mathbb{N}$. For each message T_i, its *periodicity* $R_i = 2^{n_i}, n_i \in \mathbb{N}_0$ is given. If the message has periodicity R_i, then it has to be scheduled once in each τR_i time units, i.e. its period is equal to τR_i.

This assumption is not limiting in practice since the data from our automotive industrial partner shows that messages between ECUs have periodicity given by a power of two [6]. Also in Vestal's paper [16] a processor workload data from avionics system have periodicity given as a power of two as well.

The length of the hyperperiod is defined as $H = \tau \cdot \max R_i$. It is considered as the feasibility interval, since if a message would be scheduled outside the hyperperiod, then the schedule cannot be repeated, hence it would not be periodic. A message T_i has multiple occurrences in the hyperperiod if $R_i \neq \max_k R_k$. We say that the jitter between a pair of two consecutive occurrences r and $r+1$ of the message T_i is the absolute value of the difference of start times relative to message period

$$J_{i,r} = |s_{i,r} + \tau R_i - s_{i,r+1}| \qquad \text{(JIT)}$$

where $s_{i,r}$ is the start time of r-th occurrence of message T_i. Similarly, the jitter is also calculated between the first and the last occurrence of each message. For messages with $R_i = \max_k R_k$ the jitter is not defined.

The solution to the problem is a feasible periodic schedule of mixed-critical messages that minimizes the maximal jitter, i.e. $\min \max_{i,r} J_{i,r}$. We call this problem of minimizing jitter in periodic mixed-criticality match-up scheduling as JPMC.

In Figure 2 we can see a solution to the problem. There

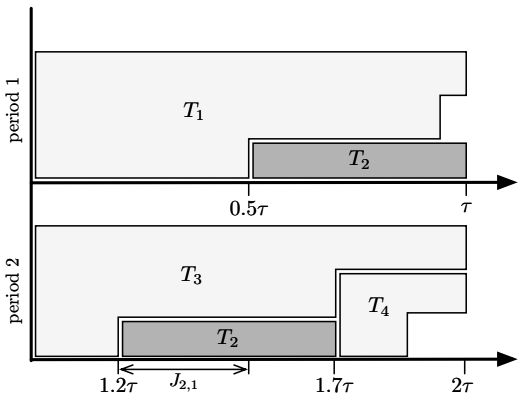

Figure 2: An example of a schedule with messages where only the non-zero jitter schedule is feasible.

we have four messages, where all of them have periodicity 2 except the message T_2 with $R_2 = 1$. In this schedule, there is a positive jitter $J_{2,1} = 0.3\tau$ between the first and the second occurrence of T_2. It can also be seen that no other feasible schedule can achieve lower jitter.

4.1 MIP Formulation

The JPMC Problem can be formulated as a Mixed-Integer Linear Program using the relative-order model. For better clarity, let us define for any $m \in \mathbb{N}_0$, the expression $[m]$, which denotes the index set $\{1, \ldots, m\}$ and it an empty set when $m < 1$. The $I_{\mathcal{MC}}$ denotes the set of all messages. For each message $T_i \in I_{\mathcal{MC}}$, we view its occurrences given by its periodicity R_i as individual messages. For such an occurrence r we seek its start time $s_{i,r}$. The start time is bounded by the time window given by the message occurrence's period. Furthermore, for any pair of occurrences r and $\overline{r}$, we define the variable $x_{i,j,r,\overline{r}}$ that takes value 1 if the occurrence r of the message T_i is scheduled before the occurrence $\overline{r}$ of the message T_j. For such a pair of occurrences, the maximal common criticality level is determined, and equations (9) and (10) then ensure that messages will not overlap. This is achieved with so-called big-M constant that can be used for turning constraints on and off depending on the value of a variable. Since $x_{i,j,r,\overline{r}}$ determinates the relative order of occurrences r and $\overline{r}$ we require exactly one of (9) and (10) constraints to be satisfied for each pair of occurrences.

The time window for each message occurrence is given by constraints (2) and (3). Notice, that in the equation (3) we require that the message occurrence is scheduled within its period including its highest criticality level $\mathcal{X}_i$ (i.e. its WCET). The maximum jitter $J_{\max}$ is determined by (4)–(8), where the jitter between occurrences in successive periods is constrained by (4)–(5). This is the linear way of expressing the equation (JIT). The jitter between different hyperperiods is given by (6)–(7).

$$\min J_{\max} \qquad (1)$$

subject to

$$(r-1)\tau R_i \leq s_{i,r} \quad \forall i \in I_{\mathcal{MC}}, \forall r \in \left[\tfrac{H}{\tau R_i}\right] \qquad (2)$$

$$s_{i,r} + p_i^{(\mathcal{X}_i)} \leq r\tau R_i \quad \forall i \in I_{\mathcal{MC}}, \forall r \in \left[\tfrac{H}{\tau R_i}\right] \qquad (3)$$

$$J_{i,r} \geq s_{i,r} + \tau R_i - s_{i,r+1} \quad \forall i \in I_{\mathcal{MC}}, \forall r \in \left[\frac{H}{\tau R_i} - 1\right] \quad (4)$$

$$J_{i,r} \geq -s_{i,r} - \tau R_i + s_{i,r+1} \forall i \in I_{\mathcal{MC}}, \forall r \in \left[\frac{H}{\tau R_i} - 1\right] \quad (5)$$

$$J_{i,\frac{H}{\tau R_i}} \geq s_{i,1} - s_{i,\frac{H}{\tau R_i}} + H - \tau R_i$$
$$\forall i \in I_{\mathcal{MC}} : \tau R_i < H \quad (6)$$

$$J_{i,\frac{H}{\tau R_i}} \geq -s_{i,1} + s_{i,\frac{H}{\tau R_i}} - H + \tau R_i$$
$$\forall i \in I_{\mathcal{MC}} : \tau R_i < H \quad (7)$$

$$J_{\max} \geq J_{i,r} \quad \forall i \in I_{\mathcal{MC}}, \forall r \in \left[\frac{H}{\tau R_i}\right] \quad (8)$$

$$s_{i,r} \geq s_{j,\overline{r}} + p_j^{(\min\{\mathcal{X}_i, \mathcal{X}_j\})} - M x_{i,j,r,\overline{r}}$$
$$\forall i,j \in I_{\mathcal{MC}} : i > j\,, \forall r \in \left[\frac{H}{\tau R_i}\right], \forall \overline{r} \in \left[\frac{H}{\tau R_j}\right] \quad (9)$$

$$s_{j,\overline{r}} \geq s_{i,r} + p_i^{(\min\{\mathcal{X}_i, \mathcal{X}_j\})} - M(1 - x_{i,j,r,\overline{r}})$$
$$\forall i,j \in I_{\mathcal{MC}} : i > j\,, \forall r \in \left[\frac{H}{\tau R_i}\right], \forall \overline{r} \in \left[\frac{H}{\tau R_j}\right] \quad (10)$$

where

$$s_{i,r} \geq 0, J_{i,r} \geq 0 \quad \forall i \in I_{\mathcal{MC}}, \forall r \in \left[\frac{H}{\tau R_i}\right] \quad (11)$$

$$J_{\max} \geq 0 \quad (12)$$

$$x_{i,j,r,\overline{r}} \in \{0,1\} \quad \forall i,j \in I_{\mathcal{MC}} : i > j\,,$$
$$\forall r \in \left[\frac{H}{\tau R_i}\right], \forall \overline{r} \in \left[\frac{H}{\tau R_j}\right] \quad (13)$$

Two things about the MIP model shall be noted. First of all, for practical purposes we can fix some of the $x_{i,j,r,\overline{r}}$ variables if for occurrences r, $\overline{r}$ and tasks T_i, T_j the following holds

$$\langle (r-1)\tau R_i, r\tau R_i \rangle \cap \langle (\overline{r}-1)\tau R_j, \overline{r}\tau R_j \rangle = \emptyset$$

i.e. they are scheduled in disjunctive time windows. For those, the relative order is given by the definition of individual occurrences; hence we can reduce the number of variables in the model. Moreover, the model allows to incorporate more difficult constraints such as release dates, deadlines, and precedences with only minor modifications. Since the start times of message occurrences are expressed as $s_{i,r}$ variables, imposing constraints on them is straightforward. However, the disadvantage of this model is the usage of big-M constant inside constraints since it degrades linear relaxation strength, which typically negatively affects the solver performance.

The scheduling of F-shaped messages brings additional complexity to the problem. It can be showed that given an input instance of JPMC, deciding, whether just messages with periodicity $R_i = 1$ can be scheduled inside one period is $\mathcal{NP}$-hard in the strong sense [17] since it represents a hard partitioning problem. Therefore, the exact solution of JPMC Problem is intractable for practical problem sizes. In the following pages, we propose an efficient heuristic algorithm to solve the problem.

5. ITERATIVE SCHEDULING ALGORITHM FOR JITTER MINIMIZATION

We propose the Iterative Scheduling Algorithm that solves JPMC Problem efficiently. The presented algorithm is a constructive heuristic with an unscheduling step. It is based on the algorithm efficient at solving Resource Constrained Project Scheduling Problem [8] and a heuristic for a scheduling problem with mixed-criticality tasks with time lags [13].

A constructive algorithm starts from an empty solution and gradually constructs parts of it towards to the complete solution, usually based on some kind of priority rule. Their advantage is the speed. However, they often achieve worse solution quality since they get easily trapped in local optima which they cannot escape. Moreover, constructive algorithms are only rarely suitable for problems where producing a feasible solution is computationally hard [14].

On the other hand, local search algorithms like *Simulated Annealing* or *Genetic Algorithms* start with any feasible solution and successively apply a neighborhood operators to escape from a local optimum and to explore a space of feasible solutions. Local search algorithms are generally able to find solutions with reasonable quality, but their running time is longer. However, the basic problem, i.e., obtaining an initial feasible solution, remains. Our algorithm combines advantages of both approaches. It constructs the schedule following a priority rule until it can. If the priority rule cannot be applied further, i.e. some constraint(s) would be violated, then an unscheduling operator is applied. The unscheduling operator removes a conflicting part of the partial solution. Then, it continues to follow the priority rule again and repeats the process until the feasibility of the complete solution is achieved. The mix of the constructive approach and an unscheduling operator makes the algorithm fast and efficient at solving hard feasibility problems. The algorithm is used iteratively to solve a sequence of feasibility problems obtained by the binary search on the value $J_{\max}$.

5.1 Temporal Graph

We use a directed graph to represent messages and constraints between them efficiently. As we will show, it allows us to perform quick update manipulations and lets us express even more general constraints. Let us define the temporal graph $G_{\text{temp}} = (V, E)$. Each vertex represents a message occurrence in the schedule

$$V = \{T_{i,r} \,|\, \forall i \in I_{\mathcal{MC}}, \forall r \in \left[\frac{H}{\tau R_i}\right]\} \cup \{T_0\}$$

together with a dummy vertex T_0. For each message occurrence, a feasible assignment of the start time in the schedule is described by temporal constraints represented by the edges of G_{temp}. An edge $(k, l) \in E$ is directed and valued with $l_{kl} \in \mathbb{Z}$. Each edge represents a temporal constraint

$$s_k + l_{kl} \leq s_l$$

where s_k, s_l are start times of occurrences $k \in V$ and $l \in V$ in the schedule. In JPMC Problem we have two kinds of temporal constraints:

- message occurrences are scheduled in their respective periods
- the maximal allowed jitter $J_{i,r}$ for each consecutive occurrence pair of the message (see Algorithm 2, where the maximal jitter is narrowed by the binary search)

The dummy vertex T_0 with $R_0 = \max R_i$ is used to express the first kind of constraints. T_0 is always scheduled at $s_0 = 0$. The pair of edges going in both directions from/to T_0 express the feasible start window for each message occurrence, i.e. the constraints (2) and (3) from MIP model in Section 4.1.

The other kind of constraints limits the maximal jitter in the schedule. Therefore, the jitter for every consecutive

pair of message occurrences is constrained. Once again, with two directed edges we limit the jitter $J_{i,r}$ expressed as an absolute value like in the equation (JIT). We replace absolute value according to (4) and (5). Then, in a feasibility problem, where maximal jitter is set to $\hat{J}_{\max}$, the temporal constraints are given as

$$s_{i,r} + \tau R_i - \hat{J}_{\max} \leq s_{i,r+1}$$
$$s_{i,r+1} - \tau R_i - \hat{J}_{\max} \leq s_{i,r}$$

The value $\hat{J}_{\max}$ is iteratively updated by the binary search in order to find the lowest possible jitter. Figure 3 shows a temporal graph for the problem instance from Figure 2. In this example, the maximal jitter $\hat{J}_{\max}$ was set to 0. Therefore, the pair of edges between $T_{2,1}$ and $T_{2,2}$ forces these occurrences to be scheduled exactly one period $\tau = 100$ apart. The value of the edge $(T_{1,1}, T_0)$ determinates the latest possible start time for the first (and the only) occurrence of T_1 message. Since it has periodicity $R_1 = 2$ and its processing time at the highest criticality level is $p_1^{(3)} = \tau$, the temporal constraint is valued with $-(2\tau - p_1^{(3)}) = -(200 - 100) = -100$.

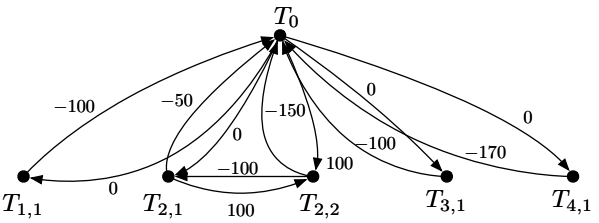

Figure 3: The temporal graph corresponding to the instance in Figure 2 assuming zero-jitter with period length $\tau = 100$.

Finally, let us denote d_{kl} as the longest path between k and l in G_{temp}. Notice, that since we have described constraints by a temporal graph, our algorithm can also solve problems where the messages are subject to release dates, deadlines or precedence constraints. These additional constraints can be easily incorporated by introducing appropriate edges to the G_{temp}.

5.2 Iterative Scheduling Algorithm

The main idea of the algorithm is to schedule messages into the free time windows. If there are no free windows, then a part of the schedule is canceled, and the message is inserted into the newly created free window. See the description of its main loop in Algorithm 2. The binary search on $J_{\max}$ value is realized by lines 6–14. Obviously, a lower bound on $J_{\max}$ is 0, and an upper bound is a half of scheduling hyperperiod, i.e. $H/2$. The interval $\langle lb, ub \rangle$ is consecutively narrowed.

A single feasibility problem parameterized by $\hat{J}_{\max} = \lceil \frac{lb+ub}{2} \rceil$ obtained by the binary search is solved by the procedure *findSchedule*. It accepts a list of priorities to guide the search and a budget to limit time spent in this procedure. The budget is an integer that states the maximum number of constructive and unscheduling steps combined for solving each feasibility problem. It is proportional to the total number of messages including all occurrences. The larger values improve the capability of solving more difficult feasibility problems but increase the algorithm's running time.

The list of priorities specifies the order in which messages are scheduled in the constructive step. In our experiments, we have used the strategy that gives more priority to the messages with smaller values of periodicity R_i.

Algorithm 2 Iterative Scheduling Algorithm

```
1: lb ← 0
2: ub ← H/2
3: priority_i ← 1/R_i   ∀i ∈ V
4: budget ← budgetRatio · |V|
5: Ĵ_max ← lb
6: while lb ≤ ub do
7:     S ← findSchedule(Ĵ_max, priority, budget)
8:     if S is feasible then
9:         ub ← J_max(S) − 1
10:    else
11:        lb ← Ĵ_max + 1
12:    end if
13:    Ĵ_max ← ⌈(lb+ub)/2⌉
14: end while
```

```
1: function findSchedule(Ĵ_max, priority, budget)
2: scheduled ← ∅
3: s_i ← −∞   ∀i ∈ V
4: update G_temp with a new Ĵ_max bound
5: while budget > 0 and |scheduled| < |V| do
6:     i ← arg max_{j∈V: j∉scheduled} {priority_j}
7:     ES_i ← max_{j∈V: j∈scheduled} {s_j + d_ji}
8:     LS_i ← min_{j∈V: j∈scheduled} {s_j − d_ij}
9:     s_i ← findTimeSlot(i, ES_i, LS_i)
10:    if s_i was found then
11:        scheduleMessage(i, s_i, scheduled)
12:    else
13:        scheduleMessageViolently(i, s_i, scheduled)
14:    end if
15:    budget ← budget − 1
16: end while
17: return (s_1, ..., s_n)
```

Function *findSchedule* attempts to find a feasible solution for the given maximum jitter value $\hat{J}_{\max}$. First, temporal constraints bounding the maximal jitter in G_{temp} are updated with respect to $\hat{J}_{\max}$ value. Then it attempts to build the schedule by following the priority rule. For each yet unscheduled message occurrence i, its earliest ES_i and latest LS_i start times are determined by lines 7–8. The procedure *findTimeSlot* attempts to find the start time s_i in the interval $\langle ES_i, LS_i \rangle$ in such a way, that it is not overlapping with any other scheduled message on any criticality level. If such a feasible assignment exists, then the message is scheduled at s_i. If not, then the message i is inserted violently by *scheduleMessageViolently* into the schedule at position $s_i \leftarrow s_i^{\text{prev}} + 1$, where s_i^{prev} is the start time assigned in the previous attempt. If the message i is scheduled for the first time, then it is scheduled at its earliest start time, i.e. $s_i \leftarrow ES_i$. Furthermore, scheduled messages conflicting with the new assignment s_i are unscheduled and removed from the *scheduled* set.

The process is repeated until all messages are scheduled or the budget is depleted. If the returned schedule in Algorithm 2 at line 7 is feasible, then the upper bound is updated. If not, then the lower bound on $J_{\max}$ is increased.

Table 1: Computational results for instances with $R_{\max} = 8$

	$budgetRatio = 2$			$budgetRatio = 20$			$budgetRatio = 200$		
n	$J_{\max}$	J_{avg}	t [s]	$J_{\max}$	J_{avg}	t [s]	$J_{\max}$	J_{avg}	t [s]
100	0.000 (7)	0.000	0.01	0.044 (5)	14.208	0.06	0.045 (5)	14.208	0.35
200	0.017 (5)	12.778	0.02	0.053 (2)	36.450	0.16	0.053 (2)	36.450	1.04
500	0.041 (8)	99.798	0.17	0.139 (0)	161.194	1.38	0.139 (0)	161.194	9.79
700	0.000 (8)	0.000	0.30	0.060 (5)	153.761	4.96	0.070 (4)	179.630	41.23
900	0.014 (6)	43.084	0.39	0.051 (3)	132.366	4.66	0.051 (3)	132.366	37.74
1000	0.000 (6)	0.000	0.48	0.089 (0)	208.224	3.68	0.089 (0)	208.224	27.11
1200	0.000 (4)	0.000	0.48	0.071 (0)	225.125	1.90	0.071 (0)	225.125	15.90
1400	0.000 (6)	0.000	1.00	0.059 (3)	219.416	11.52	0.059 (3)	219.416	103.62
1600	0.000 (5)	0.000	1.13	0.076 (1)	270.117	8.37	0.076 (1)	270.117	65.04
1800	0.000 (5)	0.000	1.37	0.067 (0)	383.653	10.27	0.067 (0)	383.653	75.87
2000	0.000 (5)	0.000	1.76	0.051 (1)	370.402	13.17	0.051 (1)	370.402	128.32

Table 2: Computational results for instances with $R_{\max} = 16$

	$budgetRatio = 2$			$budgetRatio = 20$			$budgetRatio = 200$		
n	$J_{\max}$	J_{avg}	t [s]	$J_{\max}$	J_{avg}	t [s]	$J_{\max}$	J_{avg}	t [s]
100	0.000 (6)	0.000	0.01	0.021 (5)	8.778	0.03	0.021 (5)	8.778	0.06
200	0.000 (10)	0.000	0.03	0.106 (5)	56.999	0.23	0.109 (5)	60.170	0.75
500	0.000 (4)	0.000	0.06	0.029 (2)	47.346	0.72	0.029 (2)	47.346	4.75
700	0.000 (8)	0.000	0.23	0.104 (4)	168.436	2.60	0.104 (4)	168.436	16.05
900	0.000 (6)	0.000	0.32	0.113 (0)	240.851	2.36	0.113 (0)	240.851	13.97
1000	0.000 (5)	0.000	0.31	0.033 (3)	92.599	4.07	0.033 (3)	92.599	28.14
1200	0.000 (6)	0.000	0.58	0.055 (3)	150.932	6.71	0.055 (3)	150.932	44.73
1400	0.000 (5)	0.000	0.64	0.101 (0)	339.956	4.84	0.093 (0)	319.244	30.35
1600	0.000 (7)	0.000	1.20	0.053 (5)	198.025	17.62	0.053 (5)	198.025	136.20
1800	0.000 (5)	0.000	1.18	0.097 (0)	405.662	9.63	0.097 (0)	405.662	57.54
2000	0.000 (7)	0.000	1.98	0.093 (2)	408.145	12.95	0.093 (2)	408.145	96.06

Table 3: Computational results for instances with $R_{\max} = 32$

	$budgetRatio = 2$			$budgetRatio = 20$			$budgetRatio = 200$		
n	$J_{\max}$	J_{avg}	t [s]	$J_{\max}$	J_{avg}	t [s]	$J_{\max}$	J_{avg}	t [s]
100	0.038 (5)	9.203	0.00	0.016 (4)	4.971	0.01	0.016 (4)	4.971	0.02
200	0.032 (7)	18.374	0.01	0.031 (5)	19.442	0.04	0.031 (5)	19.442	0.12
500	0.000 (7)	0.000	0.04	0.040 (2)	56.239	0.21	0.040 (2)	56.239	0.34
700	0.000 (3)	0.000	0.04	0.009 (1)	19.157	0.51	0.009 (1)	19.157	0.54
900	0.000 (8)	0.000	0.13	0.066 (3)	126.969	1.31	0.066 (3)	126.969	2.22
1000	0.011 (2)	37.324	0.06	0.023 (0)	59.830	0.34	0.023 (0)	59.830	0.53
1200	0.027 (9)	63.475	0.26	0.045 (6)	143.388	3.54	0.045 (6)	143.388	9.89
1400	0.000 (9)	0.000	0.34	0.078 (5)	211.616	5.01	0.078 (5)	211.616	12.86
1600	0.023 (5)	79.745	0.32	0.037 (3)	142.691	3.47	0.037 (3)	142.691	9.58
1800	0.000 (4)	0.000	0.33	0.009 (3)	18.461	4.73	0.009 (3)	18.461	21.18
2000	0.026 (8)	176.555	0.76	0.108 (3)	435.951	6.83	0.108 (3)	435.951	25.84

6. EXPERIMENTAL RESULTS

We have tested our algorithm on a set of randomly generated message data. Since we assume that messages have periodicity given by a power of two, we have generated three sets with maximum periodicity $R_{\max} \in \{8, 16, 32\}$. For each $R_{\max}$ we have a set of instances with n messages, $n \in \{100, 200, \ldots, 2000\}$. For each n, we have a set of 20 instances.

Since messages are of mixed-criticality, we need to generate a set of different processing times for each one. For each message, the criticality is distributed according to the $\mathcal{X}_i \sim \text{Poisson}(2)$ distribution with $\lambda = 2$. Then, for each criticality level $\ell \in \{1, \ldots, \mathcal{X}_i\}$ the processing time prolongation relative to the previous level is sampled from the uniform distribution $\mathcal{U}(\ell, \ell + 6)$. This choice of parameters results in a set of F-shaped messages representing a challenging packing problem.

We have three data sets with a different number of periods. For the dataset with $R_{\max} = 8$, the message periodicity $R_i \sim 2^{\text{Poisson}(2)}$ is distributed according to the 2 to a power drawn from the Poisson distribution with $\lambda = 2$. For each n (i.e. the number of messages) the period length τ is distributed according to $\mathcal{U}(n/0.29, n/0.36)$. For $R_{\max} = 16$, the periodicity is distributed according $R_i \sim 2^{\text{Poisson}(4)}$ and the period length $\tau \sim \mathcal{U}(n/0.74, n/0.9)$. For $R_{\max} = 32$, R_i is distributed according $R_i \sim 2^{\text{Poisson}(8)}$ and the period length $\tau \sim \mathcal{U}(n/2.8, n/3.4)$. Values of R_i above $R_{\max}$ were rounded down to $R_{\max}$. We observed that these values make gener-

ated instances tight. The choice of means of R_i for different instance sets represents a broad range of instances with different properties. The generated set of instances contains both feasible and infeasible instances.

The results are summarized in Tables 1–3. In each table, there are three columns each corresponding to the run of the algorithm with different *budgetRatio* value. For each n, the $J_{\max}$ column denotes the mean of the relative maximal jitter for solved instances, i.e. the ratio between the maximal jitter and hyperperiod length. The value in parenthesis denotes the number of instances that were not solved by the algorithm. The J_{avg} denotes mean of the absolute jitter averaged over all messages for solved instances. The column t denotes the mean of the algorithm's running time measured in seconds. The algorithm is implemented in C# programming language and running times are obtained with Intel Core i5 5300U@2.3GHz processor and 8 GB RAM using a single core.

In the first column in Tables 1–3 we can see that the algorithm with $budgetRatio = 2$ was mostly able to solve only instances, where the zero-jitter schedule is feasible (i.e. easier ones). With $budgetRatio = 20$ the algorithm solved a much larger number of instances, including more the challenging ones. In the third column, the algorithm with even more budget was able to solve one more instance, but overall with significantly higher time.

Out of 660 instances, 12.5% were not solved by the algorithm with $budgetRatio = 200$. We attempted to solve every of these instances by the MIP model introduced in Section 4.1. We invested computational effort of more than 10 CPU hours for solving each instance with Gurobi MIP solver. In just four cases, a feasible solution was found. For the rest, no feasible solution was found within the time limit. Therefore, these instances are likely to be infeasible.

The results show that setting *budgetRatio* to 20 offers a good balance between the algorithm's running time and the solution quality (the number of solved instances).

7. CONCLUSION

In this paper, we proposed a method for the message retransmission in time-triggered systems. We construct static schedules where each message has a given number of allowed retransmissions based on its criticality. The prolongation of the processing time observed during the run time is compensated by skipping less critical messages. Static schedules encapsulate all alternative execution scenarios, and therefore, the response time analysis can be carried out trivially since for each alternative we have a static schedule. We model the problem with messages with different processing times by F-shaped tasks.

Further, we introduced a novel heuristic algorithm that minimizes the maximal jitter in the periodic message scheduling problem with messages having different possible processing times. The proposed algorithm is an iterative-based algorithm with an unscheduling step efficient at solving hard feasibility problems. We showed its capability to solve instances with up to the 2000 messages in a matter of seconds. Furthermore, since we expressed the problem constraints via the temporal graph, the algorithm is easily extendable to solve an even more complex problem, including message release dates, deadlines, and precedences. Moreover, we proposed a mathematical program for solving the JPMC Problem to the optimality. It can be used for solving smaller instances, and we use it also for tuning and the verification of the proposed heuristic algorithm.

8. ACKNOWLEDGMENTS

This work was supported by the US Department of the Navy Grant N62909-15-1-N094 SALTT issued by Office Naval Research Global. The United States Government has a royalty-free license throughout the world in all copyrightable material contained herein. Furthermore, this work was supported by the Grant Agency of the Czech Technical University in Prague, grant No. SGS16/233/OHK3/3T/13.

9. REFERENCES

[1] Sanjoy Baruah and Gerhard Fohler. Certification-cognizant time-triggered scheduling of mixed-criticality systems. In *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*, pages 3–12. IEEE, 2011.

[2] Sanjoy Baruah, Haohan Li, and Leen Stougie. Towards the design of certifiable mixed-criticality systems. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2010 16th IEEE*, pages 13–22. IEEE, 2010.

[3] Ron Bell. Introduction to IEC 61508. In *Proceedings of the 10th Australian workshop on Safety critical systems and software-Volume 55*, pages 3–12. Australian Computer Society, Inc., 2006.

[4] Alan Burns and Rob Davis. Mixed criticality systems-a review. *Department of Computer Science, University of York, Tech. Rep*, 2013.

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

[6] J. Dvorak and Z. Hanzalek. Using two independent channels with gateway for FlexRay static segment scheduling. *IEEE Transactions on Industrial Informatics*, article in press, 2016.

[7] Friedrich Eisenbrand, Nicolai Hähnle, Martin Niemeier, Martin Skutella, José Verschae, and Andreas Wiese. Scheduling periodic tasks in a hard real-time environment. In *International Colloquium on Automata, Languages, and Programming*, pages 299–311. Springer, 2010.

[8] Zdeněk Hanzálek and Přemysl Šůcha. Time symmetry of resource constrained project scheduling with general temporal constraints and take-give resources. *Annals of Operations Research*, pages 1–29, 2016.

[9] Kaisrlik J. and Sojka M. Time-triggered switch for mixed-criticality applications. Technical report, CTU in Prague, DCE, 2016.

[10] Taewoong Kim, Heonshik Shin, and Naehyuck Chang. Deadline assignment to reduce output jitter of real-time tasks. In *16th IFAC Workshop on Distributed Computer Control Systems*, pages 67–72. Citeseer, 2000.

[11] Hermann Kopetz, Astrit Ademaj, Petr Grillinger, and Klaus Steinhammer. The time-triggered Ethernet (TTE) design. In *Object-Oriented Real-Time Distributed Computing, 2005. ISORC 2005. Eighth IEEE International Symposium on*, pages 22–33. IEEE, 2005.

[12] Martin Lukasiewycz, Michael Glaß, Jürgen Teich, and Paul Milbredt. FlexRay schedule optimization of the static segment. In *Proceedings of the 7th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, pages 363–372. ACM, 2009.

[13] Cincibus P. Algorithms for mixed-criticality scheduling with positive and negative time lags. Master's thesis, CTU in Prague, 2015.

[14] El-Ghazali Talbi. *Metaheuristics: from design to implementation*, volume 74. John Wiley & Sons, 2009.

[15] Jens Theis, Gerhard Fohler, and Sanjoy Baruah. Schedule table generation for time-triggered mixed criticality systems. *Proc. WMC, RTSS*, pages 79–84, 2013.

[16] Steve Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, pages 239–243. IEEE, 2007.

[17] Hanzalek Z., Tunys T., and Sucha P. An analysis of the non-preemptive mixed-criticality match-up scheduling problem. *Journal of Scheduling, doi: 10.1007/s10951-016-0468-y*, 2016.