Scheduling of Safety-Critical Time-Constrained Traffic with F-shaped Messages

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

Zdenek Hanzalek
DCE FEE and CIIRC
Czech Technical University in Prague
Prague, Czech Republic

Premysl Sucha
DCE FEE
Czech Technical University in Prague
Prague, Czech Republic

***Abstract*—The rapid improvement of surrounding systems such as automotive vehicles brings new challenges for system designers and manufacturers. New functionalities such as advanced driver assistants leverage a part of responsibilities from the driver to an autonomous system. Being able to provide such functionalities requires a safety certification the system, namely the reliability of the communication backbone. In complex systems, the safety certification is becoming a hard problem, especially in the Event-Triggered environments. On the other hand, Time-Triggered communications are well-known for their determinism, reliability, and ease of certification but lack the flexibility that is required e.g. for message retransmissions.**

To support safety-critical applications, we improve the reliability of Time-Triggered communications even more. We build schedules that account for retransmissions of lost messages. They are robust static schedules that encapsulate all the possible alternative execution scenarios arising from the uncertainty of transmission outcomes. However, being too robust can be very costly. Therefore, our schedules compensate retransmissions by a possibility of rejecting the transmission of less critical messages to achieve a trade-off between the safety and efficient usage of resources. To solve this complex problem, we present a novel two-stage decomposition algorithm for the synthesis of static schedules accounting for alternative execution scenarios with non-preemptive messages that are constrained by release times and deadlines. We show that our method attains solutions within 6–7% from a lower bound even for large problem instances.

I. Introduction

The communication backbone in modern vehicles is responsible for carrying safety-critical traffic. ECUs (*Electronic Control Units*) are implementing a broad range of functionalities including advanced driver assistants and safety-critical applications like drive-by-wire. Teslas Autopilot continuously streams high-frame rate, high-resolution video to the central vision processing unit to support lane-keeping assistants and autonomous driving. Moreover, the current trend in the automotive industry shows that requirements on the throughput and determinism of the communication channel will grow.

Nowadays, in addition to existing Event-Triggered communication protocols such as CAN (*Controller Area Network*) or CAN FD (*Flexible Data Rate*) car manufacturers start to embed high-throughput buses like Time-Triggered extensions of Ethernet [9] to handle data-intensive applications. The rapid improvement in computer vision algorithms in recent time has enabled the self-driving functionalities. However, the underlying hardware platform needs to catch-up in its dependability. It needs to provide an affordable safety certification critical systems (e.g. SIL – *System Integrity Levels* classification [3]). Moreover, it needs to deal with problems arising from the co-existence of activities of different criticalities — granting resources to a low critical activity should not prevent more critical ones from being successfully completed.

Time-Triggered communication protocols have a significant advantage. The certification, usually done by means of the *response time analysis*, is easy for them. In Time-Triggered environments, the network nodes have synchronized clocks and messages are transmitted at the moments defined by the static schedule. Before the construction of the schedule, a designer imposes a set of constraints (such as message release time and deadline) on the communication that are met in every feasible solution to the scheduling problem. Therefore, the certification is achieved by showing the feasibility of the produced communication schedule.

One of the disadvantages of Time-Triggered protocols is their low flexibility. To account for message retransmission in a static schedule, one has to either allocate more resource time for unforeseen retransmissions or, construct a schedule with alternative execution paths. The first option is not desired since in an average runtime case the communication resource is idle for a significant amount of time. The biggest obstacle for the second option is that all the alternative execution paths need to be precomputed in advance, leading to complex schedules that are difficult to synthesize.

In this paper, we keep the advantages of Time-Triggered protocols while introducing more flexibility to the static schedules by modeling messages and their retransmissions with an abstraction called *F-shapes* [11]. It allows us to construct schedules that are compact, efficient and that account for retransmissions to a certain degree. We compensate the retransmission by a possibility of rejecting a low critical message. However, the Time-Triggered segment is typically accompanied with the Event-Triggered segment that is used for non-critical communication (e.g. Time-Triggered Ethernet, FlexRay). Thus, one minimizes the length of Time-Triggered schedule.

For this problem, we propose an efficient scheduling algorithm that synthesizes communication schedule for mes-

sages while ensuring their successful transmission with the assurance given by the corresponding function criticality. To accommodate modeling of real-life problems, the algorithm works with message release times and deadlines to represent communications with complex precedence relations and strict deadlines.

II. Related Work

The study of systems with activities of different safety requirements sharing common resources is traditionally clustered around the research on mixed-criticality [4]. This research is typically 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 a seminal paper [14] 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. The F-shape abstraction follows naturally the proposition of increasing execution times with increasing system criticality. A simplifaction of Vestal's model was done by Burns [4]. However, it is known to be strictly less expressive while preserving the same the worst case computational complexity [2]. Theis et al. [13] argued that mixed-criticality shall be pursued in Time-Triggered systems. Baruah and Fohler's [1] approach in the Time-Triggered environment assumed preemptive tasks with up to two criticality levels. However, the application concerning message retransmission requires a non-preemptive model. Hanzalek et al. [15] presented the problem of time-triggered non-preemptive mixed-critical scheduling. However, their work did not address how to obtain schedules for large problem instances, i.e. no efficient algorithm is given there.

A large amount of work is done on message scheduling in the Time-Triggered domain. There, a static schedule is computed offline and then it is fully deterministic. [7] solved the scheduling on Profinet IO IRT as the Resource Constrained Project Scheduling with temporal constraints. Scheduling in Time-Sensitive Networks TSN IEEE 802.1Qbv was done by [5]. They focused on creating schedules for the timed-gates of scheduled queues by a Time-Triggered schedule. However, their approach does not account statically in advance for message retransmissions. Dvorak et al. [6] developed an efficient algorithm for scheduling periodical messages in the static segment of FlexRay bus. However, they do not consider retransmissions, and they assume only release times and deadlines given as a multiple of a basic period. Novak et al. [11] studied a related problem of message retransmissions in dyadic periodical schedules. However, their aim is not to generate compact schedules, but rather optimize for low jitter. Furthermore, they do not consider general deadlines for messages.

To the best of our knowledge, this is the first work that addresses the challenge of combining strict timing constraints and message retransmission in Time-Triggered mixed-critical environments while introducing an efficient scheduling method. The resulting static schedules are robust, easily certifiable but yet still efficient.

A. Contribution and Outline

In this work, we present a novel scheduling algorithm to solve the problem of message retransmission in the Time-Triggered segment of communications, where messages are of different criticalities and are constrained by release times and deadlines. We aim to obtain the shortest Time-Triggered feasible schedule, to maximize space for non-critical Event-Triggered traffic. In such a problem, the key difficulty lies in obtaining a compact schedule of so-called *F-shapes* to achieve a good utilization of resources. To ease the problem, we propose a two-stage decomposition that allows us to address opposite goals separately — ensuring the feasibility and obtaining a compact schedule. Moreover, we proposed a method how to express safety requirement of a message as an F-shape.

The rest of the paper is structured as follows. In Section III we introduce a model of F-shaped tasks that models message retransmission taking safety requirements into account, and we describe the runtime execution of static schedules with alternatives. In Section IV we state the scheduling of non-preemptive F-shaped tasks in a Time-Triggered environment as a single resource scheduling problem with release times and deadlines. We give a *Mixed-Integer Linear Programming* formulation of the corresponding optimization problem. In Section V we describe a novel two-stage decomposition algorithm for the synthesis of static schedules with retransmissions. A comprehensive evaluation of computational results in Section VI shows that our method achieves near-optimal solutions. Finally, a conclusion is given in Section VII.

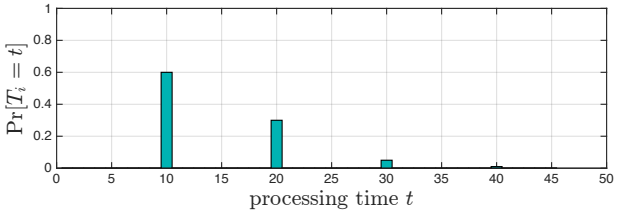
III. Message Criticality Model

A. Static Schedules with F-shaped Messages

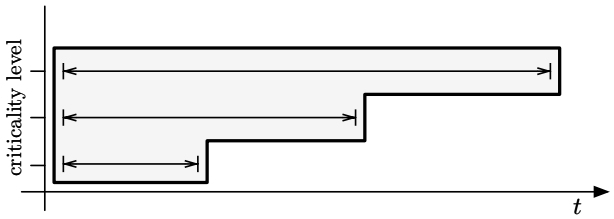
In the considered model of communication, we assume that each message carries out a functionality of a given safety requirements. For example, IEC 61508 standard [3] defines so-called *Safety Integrity Levels* (SIL). This classification distinguishes four levels of functionality (SIL-1 to SIL-4) with respect to their tolerable dangerous failures. In particular, SIL-1 is associated with a probability of dangerous failure 10^{-5} while SIL-4 requires this probability to be at most 10^{-9}.

We assume that messages are transferred through the network at moments defined by a static schedule. However, the communication is often not reliable; therefore, to meet the requirements of a given SIL, it may be necessary to build schedules that account for message retransmission to improve the reliability. The basic unit of static schedules that account for message retransmission are non-preemptive mixed-critical messages called F-shapes [15]:

Definition 1: The F-shaped task T_i is 4-tuple $(r_i, \tilde{d}_i, \mathcal{X}_i, \mathbf{P}_i)$ where $r_i \in \mathbb{N}_0$ is the release time, $\tilde{d}_i \in \mathbb{N}_0$ is the deadline, $\mathcal{X}_i \in \mathbb{N}$ is the criticality and $\mathbf{P}_i = \left(p_i^{(1)}, p_i^{(2)}, \ldots, p_i^{(\mathcal{X}_i)}\right) \in$



(a) PDF of message retransmission.



(b) The non-preemptive mixed-critical message T_i with $\mathcal{X}_i = 3$ depicted as an F-shape.

Fig. 1: Stochastic characteristic of a message and its representation as an F-shape.

$\mathbb{N}^{\mathcal{X}_i}$ is a vector of distinct transmission times such that $0 < p_i^{(1)} < p_i^{(2)} < \ldots < p_i^{(\mathcal{X}_i)}$.

Now we describe how to model the messages of different SILs using F-shapes. Each F-shaped message has a release time, deadline, a set of alternative transmission times and a criticality assigned. Release time and deadlines are set e.g. according to transmission precedences in the system or according to the frequency of a control loop. The criticality is the number of allowed retransmissions. This number is set such that the requirements of a given SIL are met. Consider an illustrative example of the transmission a message with *probability distribution function* (PDF) in Figure 1a. There we see that one transmission takes ten time units while being the most probable scenario. A less probable scenario that the transmission takes twenty time units corresponds to that the second transmission attempt was successful. More attempts have an increasingly smaller probability of happening.

If the message T_i carries a functionality of SIL-λ that corresponds to the tolerable probability of dangerous failure at most f_λ, then the criticality of T_i is given as

$$\mathcal{X}_i = \min\left\{ \ell \;\middle|\; \sum_{k=1}^{\ell} \Pr\left[T_i \text{ succeeds at } k\text{-th attempt}\right] \geq 1 - f_\lambda \right\}$$

In Figure 1b there is an F-shape corresponding to PDF in Figure 1a with tolerable probability failure $f_\lambda = 0.05$. The transmission times $\mathbf{P}_i$ are $(10, 20, 30)$. Furthermore, we say that $p_i^{(\ell)}$ as the ℓ-th element of $\mathbf{P}_i$ is the transmission time of T_i at criticality level ℓ. In this example it is set to the multiple of the basic transmission length; however, Definition 1 admits an arbitrary increasing vector $\mathbf{P}_i$; thus the communication overhead between the consecutive transmissions can be included. Since F-shapes are non-preemptive and contain different transmission times that are increasing with respect to criticality levels, we display them as F-like shapes in the Gantt chart as shown in Figure 1b.

Offline scheduling of communication consists of building a static schedule with F-shapes. We distinguish between the static schedule of F-shapes (see Figure 2a) and a *realized schedule* (see Figure 2b). A static schedule is an assignment of F-shapes to start times. The message retransmissions are realized during the runtime as needed. That is, during the runtime execution, if a message is not transmitted successfully at the first attempt, then we allow retransmissions up to its criticality. However, retransmissions may lead to conflicts in resource usage. To avoid conflicts, instead of shifting start times of following messages, we compensate prolongations by skipping some. This gives us an advantage in the analysis of the schedule – it remains static, and hence, the analysis is carried out just by inspecting the schedule.

In the next section, we describe how the static schedules are executed with respect to uncertain transmission outcomes that are observed only during the runtime execution.

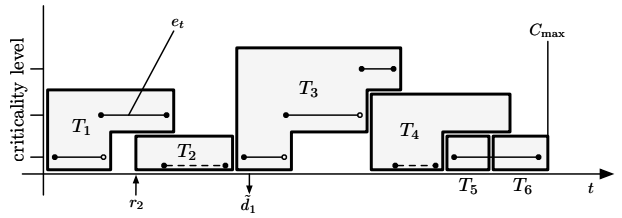
B. Runtime Execution

The runtime evaluation of the schedule can be described in terms of the execution level of the schedule that defines the performed alternative. An alternative is being picked online based on the observed events, i.e. failed transmissions.

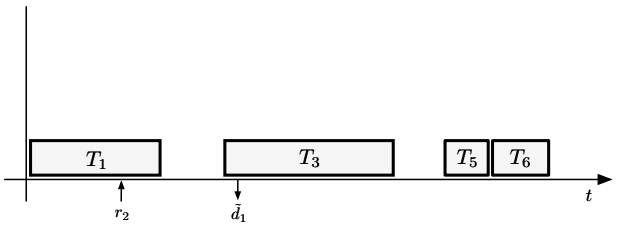
The execution level of the schedule is a function e_t defined for each time instance t. The execution starts at $t = 0$ at zeroth execution level e_t, i.e. $e_0 = 0$. Message T_i is transmitted at time $t = s_i$ if and only if $e_{s_i} = 0$, i.e. the resource is available. By the assumption, if the message is transmitted at s_i, then it is delivered no later than $s_i + p_i^{(\mathcal{X}_i)}$ (i.e. the $\mathcal{X}_i$-th level represents its the worst case execution time). The execution level e_t of the schedule is raised, if the transmitted message T_i is not delivered at $s_i + p_i^{(e_t)}$. After upon a message is delivered at one of its levels, the execution level is set back 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 are not executed. We say that the message T_j is *covered* by T_i at level e_t if and only if $s_i + p_i^{(1)} \leq s_j < s_i + p_i^{(e_t)}$.

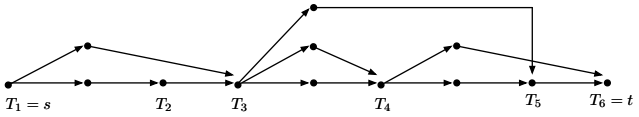
An example of the schedule with F-shaped messages is illustrated in Figure 2a. It represents a static schedule with six F-shapes. The realized runtime execution scenario for this schedule is depicted by the black line. In this scenario, the T_1 was prolonged, therefore the execution level e_t was raised. The execution policy states that messages, covered by a retransmitted message, are skipped, i.e. T_2 and consecutively T_4 in this case. After a message is transmitted, the execution matches up with the schedule at the lowest criticality. Therefore, in this example, after T_3 finishes, T_5 is up next. Naturally, there might be a different runtime scenario. In fact, there is an exponential number of other runtime execution scenarios.



(a) A feasible schedule of F-shapes and a runtime execution scenario e_t.



(b) Realized schedule at runtime execution scenario e_t.



(c) The execution graph of all alternative scenarios.

Fig. 2: A static schedule with F-shapes and one of the possible realized scenarios.

The execution policy gives rise to the intuitive notion of *the execution graph*. Given the schedule with F-shaped messages, it is a directed acyclic graph that guides the schedule execution (see Figure 2c). It captures the topology of possible match-ups that follow from coverages of messages in the static schedule. All possible execution alternatives arising from the static schedule from Figure 2a are given by all $s{-}t$ paths in the execution graph in Figure 2c. It encodes exponentially many alternative realized schedules, yet it just takes a polynomial-sized space. Thus, the schedules with F-shaped messages are compact despite their large flexibility.

In the next section, we define the problem of time-constrained message retransmission as a scheduling problem with F-shaped messages, show its computational complexity and describe it with a *Mixed-Integer Linear Program* (MILP).

IV. Scheduling Problem Statement

A. Problem Definition

The problem of time-constrained message retransmission can be stated as the scheduling problem denoted in the three-field Graham-Blazewicz notation as $1|r_i, \tilde{d}_i, mc = \mathcal{L}|C_{\max}$. The first field denotes the scheduling on a single resource, r_i and $\tilde{d}_i$ denote the presence of release times and deadlines, $mc = \mathcal{L}$ stands for the mixed-criticality aspect of messages with maximal criticality $\mathcal{L}$ and $C_{\max}$ stands for the minimization of the maximum completion time. We say that messages are mixed-critical since their criticalities may vary (as described in Section III).

The input to the problem is a set of F-shapes corresponding to the message set to be scheduled. We aim to construct a feasible static schedule of F-shaped messages that is defined as follows:

Definition 2 (Feasible Schedule): A schedule for a set of F-shapes $I_{\mathcal{MC}} = \{T_1, T_2, \ldots, T_n\}$ is an assignment of start times $(s_1, s_2, \ldots, s_n) \in {\mathbb{N}_0}^n$. We say that the schedule $(s_1, s_2, \ldots, s_n)$ for $I_{\mathcal{MC}}$ is feasible if and only if

1) $\forall i \in \{1, \ldots, n\} : r_i \leq s_i \leq s_i + p_i^{(\mathcal{X}_i)} \leq \tilde{d}_i$

2) $\forall i, j \in \{1, \ldots, n\}, i \neq j :$
$\left(s_i + p_i^{(\min\{\mathcal{X}_i, \mathcal{X}_j\})} \leq s_j\right) \vee \left(s_j + p_j^{(\min\{\mathcal{X}_i, \mathcal{X}_j\})} \leq s_i\right)$

The problem accepts a set $I_{\mathcal{MC}}$ of non-preemptive mixed-critical messages with unrestricted criticalities that are subject to release times and deadlines. The messages are scheduled on a single resource while the criterion is to minimize the maximal completion time. The completion time of the message T_i is given as $C_i = s_i + p_i^{(\mathcal{X}_i)}$, i.e. its start time plus the processing time at the highest criticality level. Hence, the maximal completion time $C_{\max}$ is given as $\max_i C_i$. The solution is a schedule with F-shapes that can be depicted in a single Gantt chart, as shown in Figure 2a.

Conditions in Definition 2 enforce that any feasible schedule guarantees that all the messages' deadlines and release times are met while accounting for retransmissions. The retransmissions have the feature that in any case, the retransmission of less critical message cannot consume resource allocated for a more critical one; however, the converse is true. The optimization criterion favors shorter schedules, i.e. minimizing the length of the Time-Triggered segment.

This problem is known to be $\mathcal{NP}$-hard in the strong sense when relaxing on mixed-criticality (i.e. $1|r_i, \tilde{d}_i|C_{\max}$) [10] or when relaxing on release times and deadlines while keeping just $mc = 2$ two criticality levels (i.e. $1|mc = 2|C_{\max}$) [15].

B. Relative-Order MILP Formulation

The search space of feasible schedules can be searched through using the following *Mixed-Integer Linear Program* (MILP). It can be described by $\mathcal{O}(n)$ continuous and $\mathcal{O}(n^2)$ binary variables together with $\mathcal{O}(n^2)$ constraints:

$$\min C_{\max} \tag{1}$$

subject to

$$r_i \leq s_i \leq s_i + p_i^{(\mathcal{X}_i)} \leq \tilde{d}_i \quad \forall i \in I_{\mathcal{MC}} \tag{2}$$

$$s_i + p_i^{(\mathcal{X}_i)} \leq C_{\max} \quad \forall i \in I_{\mathcal{MC}} \tag{3}$$

$$s_i + p_i^{(\min\{\mathcal{X}_i, \mathcal{X}_j\})} \leq s_j + M x_{ij}$$

$$\forall i,j \in I_{\mathcal{MC}} : i > j \tag{4}$$
$$s_j + p_j^{(\min\{\mathcal{X}_i,\mathcal{X}_j\})} \leq s_i + M(1 - x_{ij})$$
$$\forall i,j \in I_{\mathcal{MC}} : i > j \tag{5}$$

where

$$C_{\max} \geq 0 \tag{6}$$
$$s_i \geq 0 \quad \forall i \in I_{\mathcal{MC}} \tag{7}$$
$$x_{ij} \in \{0,1\} \quad \forall i,j \in I_{\mathcal{MC}} : i > j \tag{8}$$

Constraint (2) ensures that messages are scheduled within release times and deadlines while (3) expresses the makespan of the schedule. The most important constraints are (4) – (5) which ensure that transmissions are not overlapping on any criticality level and that messages are scheduled without preemption. Those constraints implement necessary condition for schedule feasibility. Binary variable x_{ij} is used to decide on the relative order of each pair of messages. The M is a positive constant as large as the sum of all the worst case transmission times of all messages.

The purpose of the MILP formulation is two-fold — the first is that one can take the formulation and use a standard MILP solver to solve the problem without implementing an algorithm. The second one is a rigorous definition of the problem constraints and the criterion. However, the stated model can deal with small problem instances only, being far from the practical usage. Therefore, in the next section, we introduce a heuristic algorithm capable of solving large problem instances.

Even tough MILP solvers cannot solve large problems they proved themselves as a powerful and mature technology for smaller-scale problems; thus, we employ the model described above as a part of our heuristic algorithm. The trick is that it is being used just for a quick optimization over a small neighborhood defined by the mathematical structure of the problem, as will be shown in the next section.

V. Two-stage Decomposition Algorithm

The key idea of the algorithm is to split the solution of the problem into two steps. The purpose of the stage separation is that the determination of the schedulability of $1|r_i, \tilde{d}_i, mc = \mathcal{L}|C_{\max}$ problem is $\mathcal{NP}$-hard in the strong sense. The MILP from Section IV-B is not powerful enough to find any feasible solution even for medium-sized instances. Therefore, we leverage the finding an initial solution to a heuristic.

In the first stage of the algorithm, an initial feasible solution is obtained. The first stage is inspired by NEH heuristics [8] extended by our local search for reducing infeasibility. In the second stage, the solution is iteratively reoptimized using *Large Neighborhood Search* technique [12]. It is a local search method that iteratively explores the local neighborhood by the mathematical program (1) – (8). The best solution over each neighborhood is taken and adopted as a starting solution for another iteration. The choice of the neighborhood is problem-dependent and utilizing the structure of the problem is crucial. The description of neighborhoods and their rationale will be described in Section V-B.

The both stages that are described in the following pages optimize the schedule over the space of permutations of messages. A feasible schedule is defined in terms of the start times of messages (according to Definition 2); however, an equivalent way to represent it is by a permutation that defines the order of messages. It can be shown that for the $1|r_i, \tilde{d}_i, mc = \mathcal{L}|C_{\max}$ problem the left-shifted schedule is dominant. Given the permutation π of messages, the $C_{\max}$ is minimized by shifting all messages to the left as possible. Hence, we denote $C_{\max}(\pi)$ as the makespan of the left-shifted schedule of permutation π. For example, the schedule in Figure 2a can be represented with the permutation $\pi = (T_1, T_2, T_3, T_4, T_5, T_6)$.

A. Initial Stage

The aim of the first step of the algorithm is to obtain a feasible starting solution. Since determining schedulability of $1|r_i, \tilde{d}_i, mc = \mathcal{L}|C_{\max}$ is $\mathcal{NP}$-hard in the strong sense, we use a heuristic algorithm to solve it. The algorithm is a constructive heuristics with a destructive operator. The constructive heuristics guide the search and destructive operator helps to escape from infeasible parts of the search space.

The algorithm of the initial stage is described in Algorithm 1. It iteratively creates a left-shifted schedule by scheduling messages one by one in a priority order. Each message is scheduled at the position in the permutation π where it increases the makespan of the currently scheduled messages by the smallest possible amount (lines 6–11).

The message is inserted in such a way, that its release time and deadline are always satisfied. However, the next message that is inserted (at position j in line 9), might shift start times of previously scheduled messages such that for some of the already scheduled messages the deadline would not be longer satisfied. Therefore, the resulting schedule might be infeasible. If it is the case, then the destructive operator is applied (lines 12–23), i.e. all messages whose deadline is not met ($nogood$ set) are removed from the schedule (line 22). All the removed messages are sorted again in a priority order and sequentially scheduled into the positions where they increase the makespan by the smallest possible amount. This rescheduling can give us a different schedule, where some of the reinserted messages will now satisfy the deadline. However, other messages might be shifted again, and their deadline might be violated now; those are taken out again, and by the same procedure, the algorithm attempts to fit them back.

During the iterations of removing and inserting back, it can happen that either all messages' deadlines will be met or a subset of them that needs to be rescheduled is the same as the one previously obtained at some point in the past (line 14). If it is detected, then the algorithm looks at all unsatisfied sets observed in the past ($unsat$ set), pick the smallest one (denoted as $\hat{\Delta}$), and set the $violated_i$ flag for all messages in $\hat{\Delta}$. Then the constructive step is restarted from the schedule corresponding to this set (line 17).

Algorithm 1 Initial Stage

1: $priority \leftarrow$ PRIORITY-SORT($I_{\mathcal{MC}}$)
2: $\boldsymbol{s} \leftarrow (-\infty, -\infty, \ldots, -\infty)$
3: $unsat \leftarrow \emptyset$, $unsched \leftarrow \emptyset$, $\pi \leftarrow ()$
4: $violate_i \leftarrow$ TRUE $\quad \forall i \in I_{\mathcal{MC}}$
5: **while** the stopping condition is not met **do**
6: **while** $\exists T_i$ such that $s_i = -\infty$ **or** $iter \leq n$ **do**
7: $T_i \leftarrow \arg\max_{j \in I_{\mathcal{MC}}: s_j = -\infty} priority_j$
8: $j \leftarrow \arg\min_j C_{\max}$ subject to $violate$
9: $\pi \leftarrow (\pi(1), \ldots, \pi(j-1), T_i, \pi(j+1), \ldots)$
10: $\boldsymbol{s} \leftarrow$ LEFT-SHIFTED(π)
11: **end while**
12: **if** $\boldsymbol{s}$ is not feasible **then**
13: $nogood \leftarrow \left\{ T_i \,|\, s_i + p_i^{(\mathcal{X}_i)} > \tilde{d}_i \text{ in } \boldsymbol{s} \right\}$
14: **if** $nogood \in unsat$ **then**
15: $\hat{\Delta} \leftarrow \arg\min_{\Delta \in unsat} |\Delta|$
16: $violate_i \leftarrow$ FALSE $\quad \forall i \in \hat{\Delta}$
17: $\boldsymbol{s} \leftarrow unsched_{\hat{\Delta}}$
18: $unsat \leftarrow \emptyset$, $unsched \leftarrow \emptyset$
19: **else**
20: $unsat \leftarrow unsat \cup nogood$
21: $unsched_{nogood} \leftarrow \boldsymbol{s}$
22: $s_i \leftarrow -\infty \quad \forall i \in nogood$
23: **end if**
24: **else**
25: **return** feasible $\boldsymbol{s}$
26: **end if**
27: **end while**
28: **return** infeasible

If a set of messages is flagged with $violated_i$ flag, then it means, that those cannot never be shifted by other messages such that their deadline would be violated (line 8). Therefore, it prevents the algorithm from infinite looping and implements a sort of branching. The process continues until we find a feasible schedule or until the stopping condition is not met.

As a priority rule in the algorithm, we have used MCF (*the most critical first*) with ties broken by EDF (*the earliest deadline first*). The stopping condition of the algorithm is defined in terms of the number of top-level loop iterations.

B. Large Neighborhood Search

In the second stage, the algorithm improves the initial solution by applying local improvements to the permutation. It is a two-step process — a neighborhood is chosen first, and then the permutation is optimized over the neighborhood by a MILP solver. The neighborhood specifies the set of all variables determining the relative order in the permutation. Assume that $L \subseteq I_{\mathcal{MC}}$ is the selected subset of messages. Then the neighborhood with respect to the set L is given as

$$\mathcal{N}_L = \{x_{ij} \,|\, \forall i, j \in I_{\mathcal{MC}} : i > j,\, i \in L \,\vee\, j \in L\}\,.$$

$\mathcal{N}_L$ is a set of all relative-order variables for messages in L. The algorithm iteratively optimizes the MILP from Section IV-B over the $\mathcal{N}_L$. The binary variables outside of neighborhood $\mathcal{N}_L$ are fixed according to the previous solution. The pseudocode of this procedure can be seen in Algorithm 2. The choice of the neighborhood is left to be defined. Below,

Algorithm 2 Large Neighborhood Search

while the stopping condition is not met **do**
  $L \leftarrow$ NEIGHBORHOOD(π)
  $\pi^* \leftarrow \arg\min C_{\max}$ over $\mathcal{N}_L$
  if $C_{\max}(\pi^*) \leq C_{\max}(\pi)$ **then**
    $\pi \leftarrow \pi^*$
  end if
end while
return LEFT-SHIFTED(π)

we propose two neighborhood selection strategies. The first one is based on the concept of the *critical path*. Informally said, in given the schedule it is the set of messages that causes the achieved makespan. By rescheduling those, the makespan can be reduced. The other neighborhood is the *sequence neighborhood*. It takes the given permutation and randomly selects a continuous sequence of k messages in it.

C. Critical Path Neighborhood

The *critical path neighborhood* aims to rearrange messages that prevent makespan from being decreased. A *critical path* in the schedule is defined as follows:

Definition 3 (Critical Path): For any given feasible schedule, the critical path $\mathcal{CP} \subseteq I_{\mathcal{MC}} \times \{1, \ldots, \mathcal{L}\}$ is a maximal set of messages with associated criticality levels such that $\forall (i, \ell) \in \mathcal{CP}$ holds that if the processing time $p_i^{(\ell)}$ is increased by arbitrarily small $\epsilon > 0$, then the makespan of the augmented schedule is also increased by ϵ.

We illustrate the critical path on the example from Figure 2a. In this schedule, the critical path consists of messages T_2, T_3, T_4, T_5 and T_6. The *critical path neighborhood* is then defined as

$$\begin{aligned}\mathcal{CP}\text{-NEIGHBORHOOD}(\pi, k) = \{&\text{choose randomly } k \text{ messages at}\\ &\text{the critical path of the}\\ &\text{left-shifted schedule of } \pi\}\end{aligned}$$

In general, the critical path might not be unique. In that case, an arbitrary critical path is chosen. If the critical path contains less than k messages, then every message at the critical path is taken. The feature of $\mathcal{CP}$-NEIGHBORHOOD is that it prevents from an unnecessary optimization over messages that cannot improve the current solution. Indeed, rearranging only messages that are out of a critical path cannot improve the solution as they can enter a critical path, hence increasing the objective.

D. Sequence Neighborhood

Another neighborhood we use is the *sequence neighborhood*. It aims to reoptimize local segments of the permutation of messages. That is, given the permutation π, we randomly

select continuous segment of k messages from π. More specifically, the sequence neighborhood is defined as follows:

$$\mathcal{SEQ}\text{-NEIGHBORHOOD}(\pi, k) = \{\pi(i), \pi(i+1), \ldots, \pi(i+k) \text{ for some random } i \leq |\pi| - k\}$$

The $\mathcal{SEQ}$-NEIGHBORHOOD allows to reschedule messages that are not part of a critical path in order to free up space for other messages that are currently on it. Such situations can arise for example due to the presence of release times.

E. Implementation Details

In each iteration of Large Neighborhood Search, the choice of the neighborhood is uniformly random. The size of the neighborhood k was chosen to $k \in \{13, 20\}$ in separate runs. The value of this parameter is set according to the time required by the MILP solver to find an improving solution the size of the neighborhood. As a stopping condition for the second stage of the algorithm, we have used that either one of the followings holds:

- the number of iterations of the Large Neighborhood Search reached 25
- the solution has not improved in last 6 iterations
- the gap from a lower bound is less than 2%.

The number of the top-level loop in the first stage was set to $15n$, where n is the number of messages in the instance. The algorithm is implemented in Python 2.7 and executed with PyPy 5.6 interpreter supporting *Just-in-Time* (JIT) compilation to obtain near-native performance. The code was executed on two Intel Xeon E5-2620 v2 @ 2.10 GHz processors. As a MILP solver, the Gurobi Optimizer 7.0 was used.

VI. Experimental Results

We have tested our algorithm on instances with up to the 300 messages. We generated 20 instances for each $n \in \{50, 100, \ldots, 300\}$. In each test instance, the criticalities of messages are distributed $\sim$ POISSON(3) distribution. The prolongation of the processing time at criticality level ℓ is randomly sampled from the uniform distribution $\mathcal{U}(\ell, \ell + 6)$. Release times are distributed $\sim \Gamma(2, 4)$, where $\Gamma(a, b)$ is the Gamma distribution and the deadline for each message is given as $\tilde{d}_i \sim r_i + p_i^{(\mathcal{X}_i)} + \lceil \beta \rceil$, where $\beta \sim \mathcal{U}(D/2, D) \cdot 10n$, n is the number of messages and D is a random variable distributed $D \sim \mathcal{U}(0, 5)$. This set of parameters represents a message set, where the release times are distributed around the beginning of the schedule and deadlines are distributed equally along the second half of the schedule. In this way, the order of messages is not fully determined by the deadlines, but they are not loose enough to ignore them completely. In our experience, the combination of the given criticalities and transmissions times yields to a challenging message set to schedule.

To show the trade-off between running time and solution quality, we run the algorithm with limits $l \in \{5, 15, 25\}$ s for the MILP solver at each iteration of the Large Neighborhood Search. The results for each instance is averaged over 5 independent runs of the algorithm.

From 120 instances, the algorithm was able to schedule 81.6% of instances without violating a deadline. In the Table I, the results are displayed. The first column denotes the number of messages in each instance within the set. The column *avg time* denotes the mean and the standard deviation of the total running time. The column *avg gap* denotes the mean the standard deviation of the optimality gap. The gap is computed as $gap = 100 \cdot \frac{ub - lb}{ub}$ where the ub is the objective value of the best solution found, and the lb is a lower bound on the objective. It denotes the relative distance from the provably optimal results, i.e. the shortest possible schedule. In our case, we have computed the lower bound by relaxing the problem $1|r_i, \tilde{d}_i, mc = \mathcal{L}|C_{\max}$ into the $\mathcal{L}$ independent problems of $1|r_i|C_{\max}$. The ℓ-th problem instance contains messages with processing times $p_i = p_i^{(\ell)}$ for all messages with $\mathcal{X}_i \geq \ell$ in the original problem instance. Each of these problems is solved up to the optimality. The maximum of objective values for all $\mathcal{L}$ problems is a lower bound on the objective of the original problem. The column *avg time* denotes the mean solving time combined from both stages.

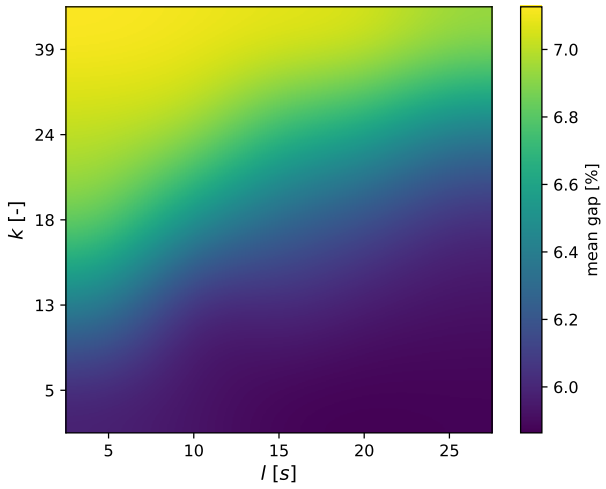


Fig. 3: The sensitivity to k and l parameters values. The depicted mean gap is for instances with $n = 100$ messages.

As it can be seen from Table I, increasing neighborhood size k without increasing time limit l might be harmful (see results for $k = 13$ versus $k = 20$), since too large neighborhoods often cannot be searched in the given time limit. Therefore, less improving solutions are found and the stopping condition of the second stage is reached faster (see *avg time* column). However, the gap is consistently smaller with increasing time limit l. Although the standard deviations of the gap may seem to be quite large, one has to realize that these gaps are averaged over the whole set of instances with the same n. Therefore, the deviations will be always non-zero even though the algorithm performs consistently.

The analysis of the sensitivity to neighborhood size k and the time l to search it through can be seen in Figure 3. Moreover, demonstrate that the trade-off between the solution

TABLE I: Mean optimality gaps and running times of the algorithm for different time limits l.

		$l = 5$ s		$l = 15$ s		$l = 25$ s	
	n messages	avg gap [%]	avg time [s]	avg gap [%]	avg time [s]	avg gap [%]	avg time [s]
$k = 13$	50	6.29 ($\pm$3.29)	66.4 ($\pm$25.4)	6.17 ($\pm$3.11)	181.7 ($\pm$70.8)	6.17 ($\pm$3.11)	285.7 ($\pm$107.4)
	100	6.40 ($\pm$1.87)	86.3 ($\pm$37.9)	5.97 ($\pm$1.75)	269.4 ($\pm$98.2)	5.93 ($\pm$1.74)	429.0 ($\pm$156.6)
	150	7.37 ($\pm$1.91)	87.5 ($\pm$42.0)	6.75 ($\pm$2.02)	304.4 ($\pm$105.4)	6.51 ($\pm$1.98)	509.1 ($\pm$164.3)
	200	6.13 ($\pm$1.80)	89.4 ($\pm$46.5)	6.00 ($\pm$1.75)	217.2 ($\pm$107.6)	5.60 ($\pm$1.48)	434.7 ($\pm$195.8)
	250	6.81 ($\pm$1.30)	94.6 ($\pm$27.3)	6.65 ($\pm$1.21)	229.3 ($\pm$109.9)	6.46 ($\pm$1.14)	416.6 ($\pm$205.6)
	300	8.08 ($\pm$1.16)	121.8 ($\pm$30.4)	7.82 ($\pm$1.08)	288.3 ($\pm$135.9)	7.68 ($\pm$1.03)	458.5 ($\pm$212.5)
$k = 20$	50	6.35 ($\pm$3.19)	72.9 ($\pm$27.8)	6.27 ($\pm$3.19)	189.1 ($\pm$72.2)	6.21 ($\pm$3.15)	294.8 ($\pm$109.4)
	100	6.88 ($\pm$1.84)	58.7 ($\pm$26.9)	6.46 ($\pm$1.77)	209.1 ($\pm$93.3)	6.23 ($\pm$1.84)	385.0 ($\pm$164.2)
	150	7.84 ($\pm$1.94)	60.5 ($\pm$20.0)	7.52 ($\pm$2.02)	191.2 ($\pm$86.1)	7.20 ($\pm$1.99)	367.9 ($\pm$178.1)
	200	6.36 ($\pm$2.02)	64.0 ($\pm$6.1)	6.16 ($\pm$1.82)	173.3 ($\pm$79.6)	6.04 ($\pm$1.67)	276.4 ($\pm$130.9)
	250	6.85 ($\pm$1.30)	82.4 ($\pm$7.8)	6.81 ($\pm$1.28)	165.8 ($\pm$38.9)	6.77 ($\pm$1.25)	272.0 ($\pm$111.3)
	300	8.09 ($\pm$1.17)	109.5 ($\pm$10.8)	8.08 ($\pm$1.17)	186.2 ($\pm$35.0)	8.00 ($\pm$1.15)	289.6 ($\pm$113.7)

quality and running time can be controlled by l parameter very well. We can see that having too large neighborhood with small time limit yields worse solutions since the neighborhood is not searched completely. Having the size of neighborhood $k \approx 13$ with time limit $l \approx 25$ s performs on average the best. Furthermore, by the analysis of the computed schedules we found out, that the schedules with F-shapes are on average 67% shorter than schedules without them ensuring the same level of commitment to retransmissions. Therefore, the significant part of the bandwidth is saved.

VII. Conclusion

In this paper, we have presented a scheduling model with F-shapes that improves the reliability of Time-Triggered communications by message retransmissions while preserving the efficient use of resources. It enables us to construct static schedules, which encapsulate all possible alternative execution scenarios that are being observed during the runtime execution.

We defined a procedure how to construct F-shapes from messages of a given SIL to met their safety requirements. Those messages are scheduled in a static schedule to facilitate the safety certification. The communication schedules are synthesized by a novel two-stage decomposition algorithm such that release times and deadlines of messages are always satisfied.

We have demonstrated the ability of our algorithm to solve large problem instances within a few percents from the optimality. The results on a comprehensive set of synthetic benchmark instances suggest that the proposed method allows smooth control between the quality of schedules and the computation time.

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 and Zhishan Guo. Mixed-criticality job models: a comparison. In *Proc. WMC, RTSS*, pages 5–9, 2015.

[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] Silviu S. Craciunas, Ramon Serna Oliver, Martin Chmelík, and Wilfried Steiner. Scheduling real-time communication in IEEE 802.1Qbv time sensitive networks. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, RTNS '16, pages 183–192, New York, NY, USA, 2016. ACM.

[6] J. Dvorak and Z. Hanzalek. FlexRay static segment scheduling on two independent channels with gateway. In *2015 IEEE World Conference on Factory Communication Systems (WFCS)*, pages 1–4, May 2015.

[7] Z. Hanzalek, P. Burget, and P. Sucha. Profinet IO IRT message scheduling with temporal constraints. *IEEE Transactions on Industrial Informatics*, 6(3):369–380, Aug 2010.

[8] Pawel Jan Kalczynski and Jerzy Kamburowski. On the NEH heuristic for minimizing the makespan in permutation flow shops. *Omega*, 35(1):53 – 60, 2007.

[9] 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.

[10] J.K. Lenstra, A.H.G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. In *Studies in Integer Programming*, volume 1 of *Annals of Discrete Mathematics*, pages 343 – 362. Elsevier, 1977.

[11] Antonin Novak, Premysl Sucha, and Zdenek Hanzalek. Efficient algorithm for jitter minimization in time-triggered periodic mixed-criticality message scheduling problem. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, RTNS '16, pages 23–31, New York, NY, USA, 2016. ACM.

[12] David Pisinger and Stefan Ropke. Large neighborhood search. In *Handbook of metaheuristics*, pages 399–419. Springer, 2010.

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

[14] 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.

[15] 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.