Property-Based Timing Analysis of Distributed Real-Time Systems

Dissertation

zur Erlangung des Grades eines

D o k t o r s   d e r   N a t u r w i s s e n s c h a f t e n

der Technischen Universität Dortmund
an der Fakultät für Informatik

von

Mario Günzel

Dortmund

2024

Tag der mündlichen Prüfung:	11. November 2024
Dekan / Dekanin:	Prof. Dr. Jens Teubner
Gutachter / Gutachterinnen:	Prof. Dr. Jian-Jia Chen
	(TU Dortmund University)
	Prof. Dr. Sanjoy Baruah
	(Washington University in St. Louis)

Abstract

For real-time systems, timing requirements must be met to avoid catastrophic outcomes. While the behavior of classical real-time systems is already studied extensively in the literature, the trend towards distributed systems raises new challenges. In this dissertation, we focus on the following two challenges: (i) distributed execution of jobs, resulting in self-suspending task behavior, and (ii) interplay of several distributed tasks, requiring to shift the real-time requirements from a single task to a sequence of tasks, the so-called end-to-end latency of cause-effect chains. Although the study of cause-effect chains and self-suspending tasks has led to a series of results, some of them suffer from missing assumptions or use intuitive extension of classical results on real-time systems, resulting in flawed analyses. We tackle this issue by providing careful analysis of self-suspending tasks and cause-effect chains based on fundamental properties of possible evolutions of the system.

The dissertation is structured into five main chapters. In Chapter 2, real-time systems and classical task models used in the literature are introduced. In this dissertation, we pursue a fundamental view on real-time systems, starting from possible system evolutions and schedules, and abstracting task properties afterwards. This is different to the typical procedure of defining the task model first and deriving possible schedules based on the task model. Our approach has the benefit that a task can be observed in different abstraction levels without translating the task into a different task model. Furthermore, in Chapter 2, we introduce scheduling algorithms as a systematic approach to derive task schedules from system evolutions based on the information of the task abstraction, and we discuss analytical literature results for the adherence to real-time requirements.

Chapter 3 introduces the challenges that arise from distributed systems. The first challenge is the distributed execution of jobs. This is enabled by refining tasks into smaller blocks which can be distributed on different computing elements. The impact of this refinement on the generation of schedules is discussed and different refined task models are presented. One of them, the so-called self-suspending task model, is in focus for this dissertation. The second challenge is the interplay of distributed tasks. More specifically, if a functionality is not described by a single task but by a sequence of tasks (a so-called cause-effect chain), then the timing requirement must be shifted from the single task to the cause-effect chain. The resulting timing metric for cause-effect chains is the end-to-end latency.

Chapter 4 emphasizes the need for careful, property-based analysis. That is, three counterexamples for literature results are provided. Two of them are related to self-suspending tasks, closing the unresolved issues discussed in a review paper on self-suspending tasks in 2019. The third one disproves a basic result for real-time systems in the context of probabilistic setups.

Chapter 5 examines self-suspending tasks. Self-suspending tasks can voluntarily suspend their execution. Such behavior can cause timing anomalies, i.e., the worst-case behavior cannot be observed with maximal execution and suspension time. Therefore, self-suspending tasks require careful analysis and treatment. We provide novel analytical approaches for the Worst-Case Response Time (WCRT) of self-suspending tasks under different scheduling algorithms, namely, (i) Task-level Fixed-Priority (T-FP), (ii) Earliest-Deadline-First (EDF) and (iii) EDF-Like (EL), all outperforming the state of the art. Furthermore, we present mechanisms to avoid the analytical pessimism that is necessary to tolerate timing anomalies, namely, segment release time enforcement and segment priority modification.

Chapter 6 examines cause-effect chains. While for typical task models the real-time requirement is given on the task-level, for cause-effect chains the end-to-end latency is considered. We prove fundamental properties of the end-to-end latency, namely the compositional property and the equivalence of typical metrics. Furthermore, we provide novel analyses of the worst-case and probabilistic end-to-end latency. For the worst-case end-to-end latency, we focus on a property-based approach, uncovering fundamental principles of literature results and using our insights from the fundamental properties to derive novel solutions. For the probabilistic end-to-end latency, we are one of the first to define a metric. Therefore, we focus on potential pitfalls and conduct careful analysis.

Publications

The majority of the ideas and findings presented in this dissertation have been published in the following peer-reviewed articles that appeared in international journals and proceedings of international conferences:

[GBC20] Mario Günzel, Georg von der Brüggen, and Jian-Jia Chen. “Suspension-Aware Earliest-Deadline-First Scheduling Analysis.” In: *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 39.11 (2020), pp. 4205–4216.

[GC20] Mario Günzel and Jian-Jia Chen. “Correspondence Article: Counterexample for suspension-aware schedulability analysis of EDF scheduling.” In: *Real Time Syst.* 56.4 (2020), pp. 490–493.

[GC21] Mario Günzel and Jian-Jia Chen. “A note on slack enforcement mechanisms for self-suspending tasks.” In: *Real Time Syst.* 57.4 (2021), pp. 387–396.

[Gün+21a] Mario Günzel, Kuan-Hsun Chen, Niklas Ueter, Georg von der Brüggen, Marco Dürr, and Jian-Jia Chen. “Timing Analysis of Asynchronized Distributed Cause-Effect Chains.” In: *RTAS.* IEEE, 2021, pp. 40–52.

[Gün+21b] Mario Günzel, Harun Teper, Kuan-Hsun Chen, Georg von der Brüggen, and Jian-Jia Chen. “Work-in-Progress: Evaluation Framework for Self-Suspending Schedulability Tests.” In: *RTSS.* IEEE, 2021, pp. 532–535.

[GUC21] Mario Günzel, Niklas Ueter, and Jian-Jia Chen. “Suspension-Aware Fixed-Priority Schedulability Test with Arbitrary Deadlines and Arrival Curves.” In: *RTSS.* IEEE, 2021, pp. 418–430.

[Che+22] Kuan-Hsun Chen, Mario Günzel, Georg von der Brüggen, and Jian-Jia Chen. “Critical Instant for Probabilistic Timing Guarantees: Refuted and Revisited.” In: *RTSS.* IEEE, 2022, pp. 145–157.

[Gün+22] Mario Günzel, Georg von der Brüggen, Kuan-Hsun Chen, and Jian-Jia Chen. “EDF-Like Scheduling for Self-Suspending Real-Time Tasks.” In: *RTSS.* IEEE, 2022, pp. 172–184.

[Gün+23a] Mario Günzel, Kuan-Hsun Chen, Niklas Ueter, Georg von der Brüggen, Marco Dürr, and Jian-Jia Chen. “Compositional Timing Analysis of Asynchronized Distributed Cause-effect Chains.” In: *ACM Trans. Embed. Comput. Syst.* 22.4 (2023), 63:1–63:34.

[Gün+23b] Mario Günzel, Harun Teper, Kuan-Hsun Chen, Georg von der Brüggen, and Jian-Jia Chen. "On the Equivalence of Maximum Reaction Time and Maximum Data Age for Cause-Effect Chains." In: *ECRTS.* Vol. 262. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, 10:1–10:22.

[Gün+23c] Mario Günzel, Niklas Ueter, Kuan-Hsun Chen, Georg von der Brüggen, and Jian-Jia Chen. "Probabilistic Reaction Time Analysis." In: *ACM Trans. Embed. Comput. Syst.* 22.5s (2023), 143:1–143:22.

[Gün+23d] Mario Günzel, Niklas Ueter, Kuan-Hsun Chen, and Jian-Jia Chen. "Timing Analysis of Cause-Effect Chains with Heterogeneous Communication Mechanisms." In: *RTNS.* ACM, 2023, pp. 224–234.

[Lin+23] Ching-Chi Lin, Mario Günzel, Junjie Shi, Tristan Taylan Seidl, Kuan-Hsun Chen, and Jian-Jia Chen. "Scheduling Periodic Segmented Self-Suspending Tasks without Timing Anomalies." In: *RTAS.* IEEE, 2023, pp. 161–173.

[Gün+24] Mario Günzel, Harun Teper, Georg von der Brueggen, and Jian-Jia Chen. "End-To-End Latency of Cause-Effect Chains: A Tutorial." In: *ACM Trans. Embed. Comput. Syst.* (2024). Just Accepted.

As part of my research, I also contributed to the following peer-reviewed articles that appeared in international journals and proceedings of international conferences but are not part of this dissertation:

[Bus+21] Sebastian Buschjäger, Jian-Jia Chen, Kuan-Hsun Chen, Mario Günzel, Christian Hakert, Katharina Morik, Rodion Novkin, Lukas Pfahler, and Mikail Yayla. "Margin-Maximization in Binarized Neural Networks for Optimizing Bit Error Tolerance." In: *DATE.* IEEE, 2021, pp. 673–678.

[Gün+21a] Mario Günzel, Christian Hakert, Kuan-Hsun Chen, and Jian-Jia Chen. "HEART: Hybrid Memory and Energy-Aware Real-Time Scheduling for Multi-Processor Systems." In: *ACM Trans. Embed. Comput. Syst.* 20.5s (2021), 88:1–88:23.

[Gün+21b] Mario Günzel, Niklas Ueter, Kuan-Hsun Chen, Georg von der Brüggen, Junjie Shi, and Jian-Jia Chen. "End-to-End Processing Chain Analysis." In: *RTSS Industrial Challenge.* 2021.

[Uet+21] Niklas Ueter, Mario Günzel, Georg von der Brüggen, and Jian-Jia Chen. "Hard Real-Time Stationary GANG-Scheduling." In: *ECRTS.* Vol. 196. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 10:1–10:19.

[UGC21] Niklas Ueter, Mario Günzel, and Jian-Jia Chen. "Response-Time Analysis and Optimization for Probabilistic Conditional Parallel DAG Tasks." In: *RTSS.* IEEE, 2021, pp. 380–392.

[Che+22a] Kuan-Hsun Chen, Mario Günzel, Boguslaw Jablkowski, Markus Buschhoff, and Jian-Jia Chen. "Unikernel-Based Real-Time Virtualization Under Deferrable Servers: Analysis and Realization." In: *ECRTS*. Vol. 231. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 6:1–6:22.

[Che+22b] Kuan-Hsun Chen, Mario Günzel, Boguslaw Jablkowski, Markus Buschhoff, and Jian-Jia Chen. "Unikernel-Based Real-Time Virtualization Under Deferrable Servers: Analysis and Realization (Artifact)." In: *Dagstuhl Artifacts Ser.* 8.1 (2022), 02:1–02:2.

[Tep+22] Harun Teper, Mario Günzel, Niklas Ueter, Georg von der Brüggen, and Jian-Jia Chen. "End-To-End Timing Analysis in ROS2." In: *RTSS*. IEEE, 2022, pp. 53–65.

[Che+23] Jian-Jia Chen, Niklas Ueter, Mario Günzel, Georg von der Brüggen, and Tei-Wei Kuo. "Property-Based Timing Analysis and Optimization for Complex Cyber-Physical Real-Time Systems." In: *DAC*. IEEE, 2023, pp. 1–2.

[GC23] Mario Günzel and Jian-Jia Chen. "Parameter Optimization for EDF-Like Scheduling of Self-Suspending Tasks." In: *RTCSA*. IEEE, 2023, pp. 261–262.

[Lin+23] Ching-Chi Lin, Junjie Shi, Niklas Ueter, Mario Günzel, Jan Reineke, and Jian-Jia Chen. "Type-Aware Federated Scheduling for Typed DAG Tasks on Heterogeneous Multicore Platforms." In: *IEEE Trans. Computers* 72.5 (2023), pp. 1286–1300.

[Uet+23] Niklas Ueter, Mario Günzel, Georg von der Brüggen, and Jian-Jia Chen. "Parallel Path Progression DAG Scheduling." In: *IEEE Trans. Computers* 72.10 (2023), pp. 3002–3016.

[GBC24] Mario Günzel, Georg von der Brüggen, and Jian-Jia Chen. "Tighter Worst-Case Response Time Bounds for Jitter-Based Self-Suspension Analysis." In: *ECRTS*. Vol. 298. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024, 4:1–4:24.

[Kuh+24] Daniel Kuhse, Nils Hölscher, Mario Günzel, Harun Teper, Georg von der Brüggen, Jian-Jia Chen, and Ching-Chi Lin. "Sync or Sink? The Robustness of Sensor Fusion Against Temporal Misalignment." In: *RTAS*. IEEE, 2024, pp. 122–134.

[Mar+24] Filip Marković, Georg von der Brüggen, Mario Günzel, Jian-Jia Chen, and Björn Brandenburg. "A Distribution-Agnostic and Correlation-Aware Analysis of Periodic Tasks." In: *RTSS*. Accepted for publication. 2024.

[Shi+24] Junjie Shi, Mario Günzel, Niklas Ueter, Georg von der Brüggen, and Jian-Jia Chen. "DAG Scheduling with Execution Groups." In: *RTAS*. IEEE, 2024, pp. 149–160.

[SGN24] Srinidhi Srinivasan, Mario Günzel, and Geoffrey Nelissen. "Response Time Analysis for Limited-Preemptive Self-Suspending and Event-Driven Delay-Induced Tasks." In: *RTSS*. Accepted for publication. 2024.

[Tep+24a] Harun Teper, Tobias Betz, Mario Günzel, Dominic Ebner, Georg von der Brüggen, Johannes Betz, and Jian-Jia Chen. “End-To-End Timing Analysis and Optimization of Multi-Executor ROS 2 Systems.” In: *RTAS*. IEEE, 2024, pp. 212–224.

[Tep+24b] Harun Teper, Daniel Kuhse, Mario Günzel, Georg von der Brüggen, Falk Howar, and Jian-Jia Chen. “Thread Carefully: Preventing Starvation in the ROS 2 Multithreaded Executor.” In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43.11 (2024), pp. 3588–3599.

Acknowledgments

I would like to take this opportunity to express my gratitude to the people who helped and supported me during this significant chapter of my life.

First, I thank my supervisor Jian-Jia Chen for believing in me from the very beginning and for giving me the opportunity to pursue my doctorate in a discipline that was initially unfamiliar to me. I am grateful for his guidance and advice to shape my academic growth and for his support in building an academic network. I would also like to thank Sanjoy Baruah, Mario Botsch, and Ben Hermann for being part of the examination committee and supporting me through the examination process. Especially, Sanjoy's dedication in carefully reviewing the results of this work has been crucial for its quality.

I would like to acknowledge all current and past members of the DAES group of LS12, especially Kuan-Hsun Chen, Georg von der Brüggen, Yun-Chih Chen, Ching-Chi Lin, Junjie Shi, Christian Hakert, Nils Hölscher, Daniel Kuhse, Simon Kurz, Vahidreza Moghaddas, Tristan Seidl, Noura Sleibi, Harun Teper, Zahra Valipour, Lars Willemsen, Kay Heider, Daniel Biebert, Mikail Yayla, Claudia Graute, and Lars Dröge. The time with the group has been interesting and enjoyable, and I really appreciated the deep discussions we shared on our research topics. Special thanks to Claudia for managing the chaos within the group and ensuring that everything ran smoothly, to Christian for sharing the early morning hours in the office with me, and to Georg for always challenging my writing style.

I am grateful to Nils, Niklas, Junjie, and Kuan for becoming not just colleagues but also good friends. The moments we shared, especially outside the workplace, have made this journey so enjoyable. A special thank you goes to Nils, for sharing the office with me. He ensured that the office hours were never boring by providing me with much-needed breaks and laughter.

My deepest gratitude goes to my parents for giving me the freedom to focus on my education from a young age. Their encouragement and support have been very reassuring.

Finally, I would like to thank all my friends, family, and everyone else who supported me. Especially, the constant reminders on the importance of life beyond work have helped me to maintain a healthy lifestyle throughout this journey.

My work has been supported by the Deutsche Forschungsgemeinschaft (DFG), as part of Sus-Aware (project no. 398602212) and by the project PropRT that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement no. 865170).

My coauthors, who have contributed to the results presented in this dissertation, have been (partially) funded by the Federal Ministry of Education and Research (BMBF) in the course of the project 6GEM under the funding reference 16KISK038, the project PropRT that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement no. 865170), and by Deutsche Forschungsgemeinschaft (DFG), as part of the Collaborative Research Center SFB 876 (project no. 124020371) and Sus-Aware (project no. 398602212).

Contents

1 Introduction

Nowadays, computing systems have become prevalent in our everyday life. Besides consumer electronics like TV sets, multimedia phones and game consoles, computing systems can also be found in many safety-critical applications, like the automotive electronics or avionics [Mar21]. For such safety-critical systems, incorrect system behavior can lead to a catastrophic outcome. Hence, it is crucial to guarantee correct system behavior. While in general computing systems the correctness depends only on the logical result, i.e., the correct output is computed for any input, for some systems the correctness also depends on the physical time when the result is produced [KS22]. Such systems are called real-time systems.

With the trend towards more complex safety-critical systems, more computational power is required. While deploying more powerful hardware is costly, an alternative to achieve computational power is to distribute workload on several components to be executed concurrently. Distribution can be done within a task, by distributing the components of a task, or by deploying several tasks that cooperate to achieve a functionality. Such distributed systems require tasks (or their components) to communicate with each other.

This dissertation studies the challenges that arise from distributed real-time systems. The primary focus is on self-suspending tasks, which are common when the distribution of functionalities is achieved by partitioning task components, and on cause-effect chains, which are usually considered when a functionality is achieved by cooperating tasks. Although the study of cause-effect chains and self-suspending tasks has led to a series of results, some of these results suffer from missing assumptions or use intuition to extend classical results on real-time systems, leading to flawed analyses. This dissertation tackles this issue by providing careful analysis based on fundamental properties of possible evolutions of the system.

This chapter outlines the context of this dissertation. To that end, we introduce distributed real-time systems in Section 1.1, and we classify typical mechanisms for scheduling and communication within the context of distributed real-time systems in Section 1.2. The contribution of this dissertation is presented in Section 1.3, and the organization of this dissertation is detailed in Section 1.4. The author's contribution to this dissertation is specified in Section 1.5. We summarize mathematical notation and convention that is utilized in this dissertation in the appendix, in Chapter A.

1.1 Distributed Real-Time Systems

Real-time systems are commonly found in cyber-physical systems [Mar21]. Cyber-physical systems are computing systems integrated into physical systems, such as automotive electronics, avionic systems, robotics, and medical devices. Many cyber-physical systems, especially in the context of safety-critical systems, require timing correctness. That is, computation is required to be completed within a certain time. For a real-time system, an instant when certain computation must be completed is called deadline. We distinguish hard, firm and soft deadlines [SR94]. That is, missing a hard deadline has catastrophic consequences, while missing a firm deadline leads to useless results, and missing a soft deadline only degrades the usefulness of the result. If a real-time system must meet at least one hard deadline, then we call it hard real-time system. Otherwise, if no hard deadlines exist, we call it a soft real-time system [KS22].

Real-time systems usually encompass several concurrent tasks sharing hardware resources [TGY22]. Each task is recurrently activated for (countably) infinitely many times, and every activation releases a task instance (called job). A scheduling algorithm decides which jobs are executed on which hardware resources. Usually, the real-time requirement for a task is given in the form of a relative deadline. That is, the time between job release and job completion must not exceed the relative deadline of the respective task. For real-time systems, it is critical to guarantee that all deadlines are met. This can be done using schedulability analysis (also called schedulability tests), based on task properties like the time between job releases (called period if the time between job releases is fixed, and minimum inter-arrival time if the time is lower bounded) and the maximum amount of time a job can be executed (called Worst-Case Execution Time (WCET)).

A distributed system is defined as "a collection of several autonomous computing elements that appears to its users as a single coherent system" [vT17]. Specifically, a distributed system has two characteristic features: First, the computing elements can behave independently of each other. That is, the computing elements apply scheduling algorithms that make decisions independent of each other, and they might have different properties that affect the execution of tasks. Second, the computing elements appear to be a single system, which means that they cooperate to achieve functionalities. An automotive system is a typical example of a distributed system. Specifically, a typical car consists of up to one hundred Electronic Control Units (ECUs) connected by Controller Area Network (CAN) buses [KS22]. As running example for the introduction, we consider an automotive system as depicted in Figure 1.1. The system encompasses three ECUs, and we want to deploy three tasks on the system, specifically, *Get Camera Image*, *Image Recognition*, and *Danger Analysis*, all relevant for an automatic brake system. Executing tasks with real-time behavior on distributed systems raises new challenges. In this dissertation, we focus on two challenges: (i) distributed execution of jobs, and (ii) interplay of distributed tasks.

For (i), we refine a job into smaller components (so-called *blocks*) that can be executed on different ECUs. An example is illustrated in Figure 1.2. Specifically, we consider the task *Image Recognition* from the running example in Figure 1.1. We assume that

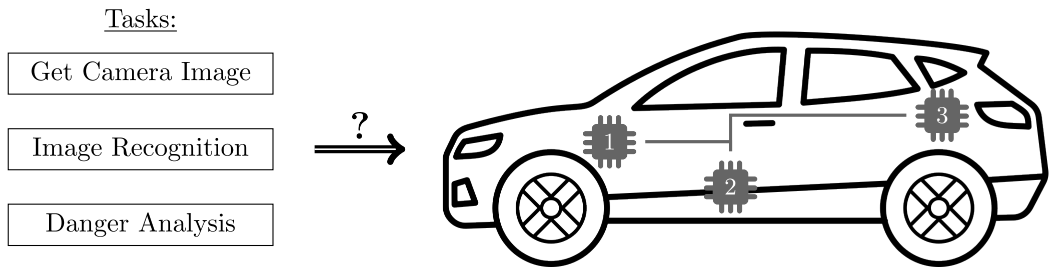


Figure 1.1.: Distributed system from the automotive domain.

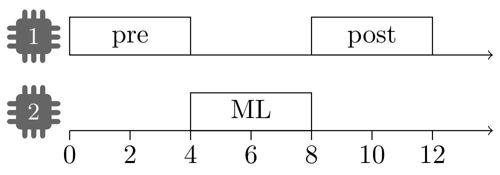


Figure 1.2.: Distributing the task *Image Recognition* on two ECUs. From time 0 to 4 the preprocessing (pre) block is executed on ECU 1, from time 4 to 8 the machine learning model (ML) is executed on ECU 2, and from time 8 to 12 the post-processing (post) block is executed on ECU 1.

the *Image Recognition* task is refined into three blocks: preprocessing, application of a machine learning model, and post-processing. If the machine learning model can be executed more efficiently on ECU 2, then we assign the block to that ECU. From the perspective of ECU 1, the task voluntarily suspends itself from being executed in the time interval $[4, 8]$. Such behavior is called *self-suspension* in the literature [Che+19b].

For (ii), multiple tasks executed on different ECUs are involved to achieve a functionality. In that case, the communication between tasks has to be analyzed. Moreover, it is not sufficient to specify timing requirements on the task-level by relative deadlines. Rather, the timing requirement has to be lifted to the level of the functionality. For the running example, all three tasks are involved in the functionality of an automatic brake system. Specifically, if a dangerous situation occurs, first the camera image has to be taken, then image recognition algorithms have to be applied, and last danger analysis gives the result to trigger the brake. The situation is depicted in Figure 1.3. A sequence of tasks that describes a functionality is denoted as a cause-effect chain in the literature. We are interested in the timing behavior from one end to the other end of the cause-effect chain. This is denoted as the end-to-end latency of cause-effect chains. In the literature, there are different formalizations of the end-to-end latency, and the most prominent ones are the Maximum Data Age (MDA) and Maximum Reaction Time (MRT). Timing requirements are usually specified on such.

In this dissertation, we use the notion of ECU when we examine the end-to-end analysis, and we use the notion of processors when examining timing requirements for classical or self-suspending tasks. This is due to their historical development. However, results of

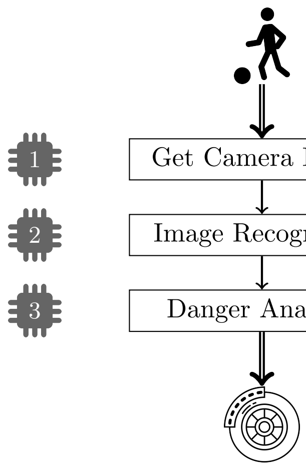


Figure 1.3.: Tasks describing an automatic brake system. The whole sequence from external activity (occurrence of dangerous situation) to actuation (decision to activate the brake) is depicted.

this dissertation are not limited to those specific computing elements.

In the literature, there are several results analyzing whether timing requirements can be met for both self-suspending tasks and cause-effect chains. The typical procedure in all these results is to start at the abstraction level of task models and only concretize possible evolutions of the system when necessary. However, this abstracted view favors to overlook missing assumptions and to introduce unnecessary properties. This impacted the research landscape as follows:

- A large body of results on self-suspending tasks has been shown to be flawed.
- Different versions of end-to-end latency have been analyzed without a clear picture of their relation.
- Similar concepts and ideas to analyze end-to-end latency have been developed independently only for specific models.

Contrarily, in this dissertation we aim for more robust definitions and proof strategies. To that end, we do not start at the level of task models, but rather consider concrete system evolutions and abstract useful properties about the tasks. This bottom-up strategy is more error-proof than the typical top-down strategy because it allows to carefully check necessary assumptions. Moreover, the bottom-up strategy avoids introducing unnecessary properties, provoking a more unified view on the examined concepts.

1.2 Scheduling and Communication

In real-time systems, scheduling algorithms are used to decide which job to be executed on which processor at which time. A typical approach is to make the scheduling decision based on job priorities. That is, whenever the scheduling algorithm makes a decision it chooses the available job with the highest priority to be executed. We distinguish partitioned scheduling, where each task is tied to a specific processor, and global scheduling, where tasks are allowed to migrate freely between processors. Furthermore, we distinguish preemptive scheduling, where the execution of a job can be interrupted at any time to execute another job with higher priority instead, and non-preemptive scheduling, where a job is always executed until it completes without being interrupted. This dissertation focuses on preemptive, partitioned scheduling.

Scheduling algorithms are classified in Task-level Fixed-Priority (T-FP) and Task-level Dynamic-Priority (T-DP). In T-FP scheduling, all jobs of the same task have the same priority, with typical examples being Deadline Monotonic (DM) [LW82] or Rate Monotonic (RM) [LL73] scheduling. In T-DP scheduling, jobs of the same task can have different priorities. A typical example is the Earliest-Deadline-First (EDF) scheduling algorithm, where the job with the earliest deadline has the highest priority. EDF-Like (EL) scheduling generalizes EDF by using priority points instead of deadlines to determine job priorities.

Besides the scheduling of tasks, communication between tasks is another important aspect of this dissertation to analyze the interplay of (potentially distributed) tasks. To that end, the communication is modeled via a shared resource. More specifically, jobs communicate by receiving (reading) their input data from a shared resource and handing over (writing) their output data to a shared resource. We distinguish explicit communication, implicit communication and communication under Logical Execution Time (LET). With implicit communication, data is always read and written when the job starts and finishes its execution, respectively. Contrarily, with explicit communication, read and write operations can occur at any time during execution. While implicit and explicit communication depend on the evolution of the system, LET abstracts the communication from the system evolution by ensuring that read operations occur before the job starts, and write operations occur after the job finishes. To achieve that, under LET jobs typically read data when they are released and write data at their deadline. Therefore, under LET the communication between tasks is independent of the choice of the scheduling algorithm as long as it is ensured that no deadlines are missed.

1.3 Contribution of this Dissertation

In this dissertation, we study the challenges that arise from distributed systems. Specifically, the contributions are focused on the analysis of self-suspending tasks and of the end-to-end analysis of cause-effect chains. To that end, we follow an approach that abstracts properties of the system in a bottom-up strategy. In the following, the contribution of this dissertation to each of these topics is detailed.

1.3.1 Property-Based Analysis

Contrarily to the typical top-down approach that concretizes task models only where necessary, in this dissertation we follow a bottom-up strategy that abstracts properties from possible evolutions of the system. This procedure allows a property-based view on real-time systems, as introduced in Chapter 2. To the best of our knowledge, this dissertation is the first work introducing classical task models and formalization of scheduling principles using the bottom-up approach.

The typical top-down strategy favors to overlook missing assumptions and to introduce unnecessary properties. As a result, a large body of results on self-suspending tasks has been flawed and different versions of end-to-end latency have been analyzed without a clear picture of their relation. This dissertation emphasizes the importance of careful, property-based analysis by providing three counterexamples for well-established literature results in Chapter 4. In particular, the counterexamples close the unresolved issues of the review paper by Chen et al. [Che+19b] on self-suspending tasks.

Specifically, the **contributions** of the dissertation to property-based analysis are as follows:

- In Chapter 2, this dissertation uses a novel bottom-up approach to introduce task models and to formalize scheduling.
- In Section 4.1, we show that slack enforcement mechanisms [LR10] can provoke deadline misses. This result is published in [GC21].
- In Section 4.2, we provide a counterexample for the schedulability test from Devi [Dev03] for periodic task sets scheduled by EDF. This result is published in [GC20].
- In Section 4.3, we show that the extension of the classical Critical Instant Theorem (CIT) to probabilistic scenarios by Maxim and Cucu-Grosjean [MC13] is flawed. Furthermore, we provide two possible solutions to extend the CIT. This result is published in [Che+22a].

1.3.2 Self-Suspension

Regarding self-suspending tasks, the contributions of this dissertation are in Chapter 5. To ensure that timing requirements of tasks are met, it must be checked whether all jobs complete their execution until their deadline. This can be done using schedulability tests. The maximal time from release to completion among all jobs of a task is the Worst-Case Response Time (WCRT). Hence, most approaches of this dissertation focus on analyzing the WCRT to verify timing requirements. In Section 5.1, we provide novel schedulability tests of self-suspending tasks for different scheduling algorithms. That is, we analyze schedulability under T-FP scheduling in Section 5.1.1, under EDF scheduling in Section 5.1.2, and under EL scheduling in Section 5.1.3. Moreover, we enhance an evaluation framework that collects schedulability tests for self-suspending tasks in Section 5.1.4.

Under T-FP scheduling, the state-of-the-art analysis for sporadic tasks (i.e., tasks with a minimum inter-arrival time) is given by Chen et al. [CNH16]. However, their method is limited to constrained-deadline task sets (i.e., the relative deadline must be no more than the minimum inter-arrival time). In Section 5.1.1, we present an analysis of the WCRT of tasks described by arrival curves (which is a generalization of the sporadic task model) with arbitrary deadlines (i.e., the deadline can be higher than the minimum inter-arrival time). Our analysis is a natural extension of the method in [CNH16], in the sense that they coincide for constrained-deadline sporadic tasks.

Under EDF scheduling, there are only few analytical results for self-suspending tasks in the literature. Besides the trivial suspension-oblivious approach, that considers suspension as additional execution time, there are two analyses for global EDF with multiple processors [DL16; LA13]. However, no analysis achieves to analytically dominate the trivial suspension-oblivious approach. In Section 5.1.2, we present two novel analyses. First, we provide an analysis of the WCRT which outperforms the two analyses for global EDF. Second, we present a schedulability test for periodic tasks which analytically dominates the trivial suspension-oblivious approach.

EL scheduling generalizes T-FP and EDF using priority points. While the state of the art was limited to either T-FP or EDF, we present the first unifying schedulability test that is applicable to all EL scheduling algorithms in Section 5.1.3 with arbitrary-deadline tasks. Our schedulability test outperforms the state of the art under EDF, and performs only slightly worse than [CNH16] under T-FP with constrained-deadline tasks.

To evaluate the performance of different schedulability tests on synthesized task sets, von der Brüggen et al. [von+19] presented an evaluation framework for self-suspending task sets in 2019. In Section 5.1.4, we enhance this framework by implementing state-of-the-art analyses for self-suspending tasks scheduled on a single processor. Moreover, we provide additional features to improve its usability, e.g., Python 3 support, an improved Graphical User Interface (GUI), multiprocessing, and evaluation of external task sets.

While for tasks without self-suspension the WCRT is achieved when all jobs execute their WCET, this is not the case for self-suspending tasks. In particular, a reduction of execution or suspension time of some jobs can enlarge the response time of other jobs. This phenomenon is called a timing anomaly, and existing schedulability analyses need to account for timing anomalies by over-approximation. Another approach is to apply a mechanism that avoids timing anomalies. However, the treatments in the literature [HC16; LR10; Raj91; SL96] may increase the WCRT compared to the WCRT without any treatment. In Section 5.2 we propose two treatments that do not increase the WCRT. More specifically, we propose *segment release time enforcement* in Section 5.2.2 and *segment priority modification* in Section 5.2.3.

Specifically, the **contributions** of the dissertation to self-suspension are as follows:

- In Section 5.1.1, we provide a WCRT analysis of arbitrary-deadline tasks described by arrival curves under T-FP scheduling. This result is published in [GUC21].
- In Section 5.1.2, we propose two schedulability tests under EDF. This result is published in [GBC20].
- In Section 5.1.3, we provide a unifying WCRT analysis under EL scheduling. This result is published in [Gün+22].
- In Section 5.1.4, we present an enhanced evaluation framework for self-suspending tasks. This result is published in [Gün+21b].
- In Section 5.2, we propose two treatments to avoid timing anomalies without enlarging the WCRT. This result is published in [Lin+23].

1.3.3 End-to-End Analysis

Regarding end-to-end latency, the contributions of this dissertation are in Chapter 6. The contributions are divided in three parts. Specifically, in Section 6.1 fundamental properties of the end-to-end latency are examined, in Section 6.2 several analytical bounds on the end-to-end latency are proposed, and in Section 6.3 the probabilistic behavior of the end-to-end latency is analyzed.

Two fundamental properties are presented in this dissertation. The first is a *compositional property* that allows to cut cause-effect chains into smaller segments and analyze them separately, as detailed in Section 6.1.1. This property can be exploited to reduce the analysis of cause-effect chains to the analysis of subchains. The second fundamental property is the *equivalence* between different metrics of the end-to-end latency, as detailed in Section 6.1.2. This has an impact on the applicability of analytical results, as well as on the specification and verification of timing constraints. In the proof of the equivalence, we also introduce a novel description of the end-to-end latency using p-partitioned job chains. This description can be used to develop novel analytical bounds on the end-to-end latency. Both properties are fundamental in the sense that they hold for a very general system model, encompassing a variety of different communication, release and scheduling mechanisms.

While analytical bounds on the end-to-end latency of cause-effect chains are discussed in the literature since 2007 [Dav+07], and more frequently since 2016 [Bec+16b; Bec+17a; Bi+22; Dür+19; GNV22; Ham+17; KBS18; Klo+22; KT20; MSB20; PM22; Tan+23a], they are usually limited to a specific task model and communication model, building on a dedicated proof structure. In Section 6.2.1, we present a formal analysis framework, uncovering the principles of abovementioned analytical results. We show that classical analytical bounds from the literature can be proven using this analysis framework. In

Sections 6.2.2 and 6.2.3, we provide two novel bounds on the end-to-end latency of periodic tasks. The bounds are built on new objects to describe the end-to-end latency, specifically, *abstract integer representations* and *p-partitioned job chains* (introduced to prove the second fundamental property). While in Sections 6.2.1–6.2.3 we consider cause-effect chains which are *local*, i.e., comprised of tasks on one ECU, and *homogeneous*, i.e., comprised of tasks of the same communication and release policy, we relax these assumptions by exploiting the compositional property in Sections 6.2.4 and 6.2.5. More specifically, in Section 6.2.4, we extend bounds for local cause-effect chains to *interconnected* cause-effect chains, i.e., comprised of tasks of potentially different ECUs. In Section 6.2.5, we deal with *heterogeneous* cause-effect chains, i.e., tasks of the same chain can have different release and communication policies.

The state of the art is mostly concerned with worst-case analysis of the end-to-end latency. However, a recent survey-based study on industry practice in real-time systems [Ake+20] indicates that soft real-time requirements are common in industry practice as well. Specifically, the study shows that 45 % of respondents can tolerate some deadline misses. This dissertation contributes in providing the first probabilistic analysis of the end-to-end latency for cause-effect chains comprised of sporadic tasks in Section 6.3. Besides the analytical advancement, this dissertation defines probabilistic end-to-end latency metrics and discusses potential pitfalls of the definition.

Specifically, the **contributions** of the dissertation to end-to-end analysis are as follows:

- In Section 6.1.1, we present a *compositional property*. This result is published in [Gün+21a] and its journal extension [Gün+23a].
- In Section 6.1.2, we prove the *equivalence* of end-to-end latency metrics. This result is published in [Gün+23b].
- In Section 6.2.1, we present a formal analysis framework uncovering the principles of classical analytical bounds on the end-to-end latency. This result is published in [Gün+23d].
- In Section 6.2.2, we develop an analytical upper bound on the end-to-end latency for periodic tasks under implicit communication using abstract integer representations. This result is published in [Gün+21a] and its journal extension [Gün+23a].
- In Section 6.2.3, we provide an analytical upper bound on the end-to-end latency for periodic tasks under LET using partitioned job chains. This result is published in [Gün+23b].
- In Section 6.2.4, we extend bounds for local cause-effect chains to interconnected cause-effect chains. This result is published in [Gün+21a] and its journal extension [Gün+23a].

- In Section 6.2.5, we develop bounds for heterogeneous cause-effect chains. This result is published in [Gün+23d].
- In Section 6.3, we provide a probabilistic analysis of the end-to-end latency of cause-effect chains comprised of sporadic tasks. This result is published in [Gün+23c].

1.4 Organization of this Dissertation

This dissertation is organized as follows:

- In Chapter 2, we introduce tasks, jobs and schedules. We derive task properties resulting in classical task models, and explain how task properties are utilized in scheduling algorithms. Furthermore, we recap classical analytical results from the literature.
- In Chapter 3, we describe the focus of this dissertation. Specifically, we discuss two challenges: distributed execution of jobs, in Section 3.1, and interplay of distributed tasks, in Section 3.2. In the context of these challenges, we introduce the self-suspension task model and cause-effect chains, which are predominantly studied in this dissertation. The state of the art for self-suspending tasks and cause-effect chains is reviewed in Section 3.3.
- In Chapter 4, we emphasize the need for careful, property-based analysis by providing three counterexamples for well-established analytical results from the literature.
- In Chapter 5, we discuss our contributions with respect to self-suspending tasks. The chapter is divided in three parts. We present analyses of the schedulability of self-suspending tasks in Section 5.1, and we discuss mechanisms to avoid timing anomalies for self-suspending tasks in Section 5.2. In Section 5.3, we summarize our results on self-suspending tasks and detail open problems.
- In Chapter 6, we discuss our contributions with respect to end-to-end analysis of cause-effect chains. This chapter is divided in four parts. We present fundamental properties of the end-to-end analysis in Section 6.1. Subsequently, we analyze the worst-case end-to-end latency in Section 6.2, and the probabilistic end-to-end latency in Section 6.3. In Section 6.4, we summarize our results on end-to-end analysis and detail open problems.
- In Chapter 7, we summarize the key results of this dissertation, and detail current limitations and opportunities for further research.

In Appendix A, we give an overview over mathematical notation and convention that this dissertation is subject to. Appendix B contains detailed proofs for some results of Chapter 4 to improve the reading flow of this dissertation.

1.5 Author's Contribution to this Dissertation

According to §10(2) of the doctoral regulations of the Department of Computer Science at TU Dortmund University, dated August 29, 2011 (including the first amendment dated January 8, 2013) [TU 13], this section details the author's contribution to scientific results utilized in this dissertation that were obtained in cooperation with other parties. The research results are categorized in results on self-suspension and results on end-to-end analysis of cause-effect chains. All authors of each research result had significant contribution by active participation in discussions and by giving valuable feedback to written content. All authors helped in polishing the work to improve its general appearance and readability. Furthermore, discussion of the related work was a joint effort of all authors involved.

The author's contribution to research results on self-suspension is as follows:

- For the work on slack enforcement mechanisms of self-suspending tasks [GC21], utilized in Section 4.1, I was the principal author. While I developed the counterexample presented in this work, Jian-Jia Chen helped with the organization of the paper, especially with the structuring of the proofs.

- For the work on EDF analysis with suspension as blocking [GC21], utilized in Section 4.2, I was the principal author. The counterexample was developed by me, and Jian-Jia Chen helped to organize the paper and put the work into context.

- For the work on the CIT in probabilistic scenarios [Che+22a][1], utilized in Section 4.3, Kuan-Hsun Chen and I were the principal authors. The counterexample of this work was entirely developed by me, and I was the main contributor to the design of the two safe bounds presented. Kuan-Hsun Chen was coordinator of the paper and main contributor to detailing the impact of the counterexample on related work. Moreover, Kuan-Hsun Chen evaluated the proposed approaches and examined how to efficiently calculate the analytical bounds using the Chernoff bound, which is not part of this dissertation.

- For the work on suspension-aware analysis under T-FP scheduling [GUC21][1], utilized in Section 5.1.1, I was the principal author. The analysis and implementation were provided by me. Niklas Ueter was the principal author of the evaluation section.

- For the work on suspension-aware analysis under EDF scheduling [GBC20][1], utilized in Section 5.1.2, I was the principal author. I provided the analysis and implementation of that paper. Georg von der Brüggen and Jian-Jia Chen improved the presentation and the paper flow significantly, and put the paper into context of related work.

[1]The copyright of this work has been transferred to IEEE for publication, © 2020–2023 IEEE.

- For the work on suspension-aware analysis under EL scheduling [Gün+22][1], utilized in Section 5.1.3, I was the principal author. I provided the analysis and implementation of that paper, with support by Kuan-Hsun Chen regarding the implementation. Georg von der Brüggen, Kuan-Hsun Chen and Jian-Jia Chen improved the presentation and the paper flow significantly, and put the paper into context of related work.

- For the work on the evaluation framework for self-suspending task schedulability tests [Gün+21b][1], utilized in Section 5.1.4, I was the principal author. I coordinated the paper writing process and organized the paper structure. Jian-Jia Chen and I contacted the authors and provided implementation for some schedulability tests. Harun Teper did the integration of all schedulability tests into the framework and also implemented some schedulability tests on his own. He improved the framework significantly under my guidance. The contents of the paper were written in tight collaboration, mainly by Harun Teper and me.

- For the work on preventing timing anomalies for self-suspending tasks [Lin+23][1], utilized in Section 5.2, Ching-Chi Lin was the principal author. He coordinated the paper writing process. The idea and the analysis, including proofs, were provided in tight collaboration between Ching-Chi Lin and me, supported by the other authors of that work. Junjie Shi evaluated the proposed approaches. Tristan Seidl implemented one of the two approaches in RTEMS, an open source Real-Time Operating System (RTOS). The implementation in RTEMS is not part of this dissertation.

The author's contribution to research results on end-to-end analysis of cause-effect chains is as follows:

- For the tutorial on cause-effect chains [Gün+24], utilized to introduce end-to-end latency of cause-effect chains Section 3.2, to summarize the state-of-the-art for cause-effect chains in Section 3.3.2, and to structure Chapter 6, I was the principal author. I coordinated the paper writing process and laid out the paper structure. The paper content was written by all authors in tight collaboration.

- For the work on compositional timing analysis of cause-effect chains [Gün+21a][1], utilized in Sections 6.1.1, 6.2.2 and 6.2.4, I was the principal author. The idea of the local analysis was provided by Jian-Jia Chen. The proofs for the local analysis, the interconnected analysis, and the compositional property were provided by me. The implementation was started by Marco Dürr and finalized by me. The paper was put into context of related work by the coauthors Kuan-Hsun Chen, Niklas Ueter, Georg von der Brüggen and Jian-Jia Chen. The paper was extended into a journal version [Gün+23a] mainly by me. While the coauthors helped to improve the general clarity of the paper, I developed the extended analysis and the extended implementation.

- For the work on the equivalence of end-to-end latencies [Gün+23b], utilized in Sections 6.1.2 and 6.2.3, I was the principal author. I provided the idea and the proof of the equivalence using partitioned job chains. The idea of applying partitioned job chains to periodic tasks was by me. I provided the analysis and implementation of this application of partitioned job chains. Harun Teper contributed in providing a case study in Robot Operating System 2 (ROS2). All the coauthors, especially Georg von der Brüggen, improved the clarity and readability of this work significantly.

- For the work on cause-effect chains with heterogeneous communication mechanisms [Gün+23d], utilized in Sections 6.2.1 and 6.2.5, I was the principal author. The analysis and the implementation of this work were provided by me. The coauthors supported the paper writing process and improved the overall quality of the paper significantly. Furthermore, they helped to structure the paper and to make the proofs more intuitive.

- For the work on probabilistic end-to-end latency [Gün+23c], utilized in Section 6.3, I was the principal author. The analysis and the implementation of this work were provided by me. The coauthors supported the paper writing process and improved the overall quality of the paper. In particular, they helped in detailing the independence assumptions that this work is subject to.

2 Real-Time Systems

While for general-purpose computing systems functional correctness is sufficient, many safety-critical systems additionally require *temporal correctness*. That is, certain functionalities of the system are subject to timing requirements. Such computing systems that require functional as well as temporal correctness are called *real-time systems* [KS22]. Functionalities are achieved by deploying *tasks* on the real-time system. Each task releases countably infinitely many task instances (called jobs) recurrently, and a scheduling algorithm decides which jobs are executed on which hardware resource. Usually, the timing requirement is given in the form of deadlines. That is, each job must finish until its deadline is reached.

Typically, in the literature, real-time systems are introduced by defining task models, and possible evolutions of the system are only derived if necessary. This top-down view on real-time systems favors overlooking missing assumptions and introducing unnecessary properties. Historically, this resulted in a large body of flawed results, especially for self-suspending tasks [Che+19b]. Furthermore, concepts and ideas were developed independently for different specific models without a clear picture of their relation, resulting in a lack of generalization. An example for this behavior is the area of end-to-end latency of cause-effect chain, where similar concepts are used to analyze different communication and release policies (see Section 6.2.1), or where different metrics were analyzed side-by-side although being equivalent (see Section 6.1.2). In this chapter, we follow a bottom-up strategy by defining possible evolutions of the system first and abstracting properties where necessary. This bottom-up strategy is more error-proof than the typical top-down strategy because it allows to carefully check necessary assumption. To the best of our knowledge, this dissertation is the first work introducing classical task models and formalization of scheduling principles using this bottom-up approach.

This chapter is organized as follows. In Section 2.1, we introduce the terminology used to describe tasks and jobs. Section 2.2 discusses the concept of system evolutions as aggregation of release times and execution demands. Section 2.3 formally defines schedules that can be derived from one system evolution. In Section 2.4, we extend our view to account for all possible system evolutions. To that end, we abstract properties that are exploited throughout this dissertation, in Sections 2.4.1 and 2.4.2. Moreover, we explain the mechanism to derive a schedule based on the abstracted information in Section 2.4.3. In particular, Section 2.4 defines standard task models like the periodic or the sporadic task model. We conclude this chapter by a review of analytical literature results in Section 2.5.

2.1 Tasks and Jobs

The set of tasks deployed on the system is denoted by $\mathbb{T}$. We assume that $\mathbb{T}$ is finite. Each task $\tau \in \mathbb{T}$ releases countably many jobs $\tau(j)$, $j \in \mathbb{N}$. The set of all jobs is denoted by $\mathbb{J} = \{\tau(j) \,|\, \tau \in \mathbb{T}, j \in \mathbb{N}\}$. Dependent on the context, we use $J \in \mathbb{J}$ or $\tau(j) \in \mathbb{J}$ to denote a job, for the sake of readability.

The release time of a job $\tau(j) \in \mathbb{J}$ is denoted by $r_{\tau(j)} \in \mathbb{R}$. We assume that the enumeration $(\tau(j))_{j\in\mathbb{N}}$ respects release times, that is $r_{\tau(j)} \leq r_{\tau(j+1)}$ for all $j \in \mathbb{N}$. Considering a job $\tau(j) \in \mathbb{J}$, we call $\tau(j+1)$ the subsequent job of τ, and $\tau(j-1)$, if it exists, the preceding job of τ.

The execution demand of a job $J \in \mathbb{J}$ is denoted by $c_J \in \mathbb{R}_{>0}$. That is, after the job is released, it must be executed for c_J time units to be completed. Sometimes, for convenience in the proofs, we also consider degenerated jobs with $c_J = 0$. We assume that such jobs are completed as soon as they are released.

With respect to real-time systems, the timing requirement of a task is usually given in the form of a *relative deadline* $D_\tau \in \mathbb{R}_{>0}$. If such a relative deadline is set, a job $\tau(j)$ must be completed until its *absolute deadline* $d_{\tau(j)} \coloneqq r_{\tau(j)} + D_\tau$ is reached to meet the timing requirement.

2.2 System Evolutions

Given a task set $\mathbb{T}$ and the corresponding set of jobs $\mathbb{J}$, the aggregation of releases times and execution demands of all jobs describes a specific evolution of the system. More specifically, a system evolution ω is given by

$$\omega = ((r_J)_{J\in\mathbb{J}}, (c_J)_{J\in\mathbb{J}}). \tag{2.1}$$

Although, from the design perspective, jobs of the same task serve the same purpose, their execution demand depends on various parameters, e.g., input data, cache behavior or bitflips. Similarly, a job release may depend on the outside environment, or on previous scheduling decisions (e.g., if tasks are allowed to trigger job releases of another task, then the decision when to schedule a job might have an impact on the release time of another job). These uncertainties result in different system evolutions. We denote the set of all system evolutions as Ω and a specific system evolution as

$$\omega = ((r_J^\omega)_{J\in\mathbb{J}}, (c_J^\omega)_{J\in\mathbb{J}}) \in \Omega. \tag{2.2}$$

The concept of systems evolutions can also be found in the literature within the context of probabilistic analysis of scheduling behavior, e.g., [Mar+23, Section V], where Ω is equipped with a probability measure. For most parts of this dissertation, we ignore the probability measure and are more concerned with the worst-case behavior. If the system evolution under consideration is clear within the context, we sometimes omit the index ω for r_J and c_J, as done in Section 2.3 to introduce schedules. Subsequently, in Section 2.4, we discuss how to deal with several system evolutions by abstracting task properties.

2.3 Schedules

A schedule $\mathcal{S}$ specifies which jobs are executed at which time on each processing unit. In this section, we provide a formal description of schedules assuming a specific system evolution ω is processed. To that end, we first focus on systems with only one processor, so-called uniprocessor systems, in Section 2.3.1. Afterwards, we extend the formalization of schedules to the multiprocessor case in Section 2.3.2. The generation of schedules using scheduling algorithms is described in Section 2.4.3.

2.3.1 Uniprocessor Schedules

We start by considering systems where only one core is processing the jobs $\mathbb{J}$ under a specific system evolution ω. The uniprocessor schedule $\mathcal{S}$ can be described by a function from the time domain to the set of jobs $\mathbb{J}$. Intuitively, if the processor executes a job $J \in \mathbb{J}$ at time t, then $\mathcal{S}(t) = J$. To allow the processor to idle, i.e., no job is being executed at time t, we expand the job set by the symbol $\bot$. That is, the processor idles at time t if $\mathcal{S}(t) = \bot$.

In the literature, two time domains are predominantly studied: the discrete time domain and the continuous time domain. Since processors are discrete systems, the discrete time domain is usually more realistic. However, if the discretization is fine enough, the continuous time model approximates the real world sufficiently and additionally shows some analytical advantages. In the remainder of this section we discuss both—discrete schedules and continuous schedules.

Discrete Schedules Inherently, processors are discrete systems. That is, time granularity is given by the length of one clock cycle $t_{cycle} \in \mathbb{R}_{>0}$. Hence, the description of the scheduling behavior should also be conducted in a discrete fashion. In the discrete model, the values r_J and c_J are integer multiples of the clockcycle t_{cycle}. A schedule $\mathcal{S}$ maps each cycle to a corresponding job $J \in \mathbb{J}$ to be executed or to $\bot$ if no job is executed during the cycle, i.e., the processor *idles*.

Definition 2.1 (Discrete Schedule). *Given a set of jobs $\mathbb{J}$, then each function*

$$\mathcal{S} \colon \mathbb{Z} \to \mathbb{J} \cup \{\bot\}, t \mapsto \mathcal{S}(t) \tag{2.3}$$

defines a discrete schedule. The set of all discrete schedules is defined as $\mathbb{S}^{discr}(\mathbb{J})$.

If $\mathcal{S}(t) = J$, then job J is executed at the t-th cycle, and if $\mathcal{S}(t) = \bot$, then no job is executed in the t-th cycle. Given a system evolution $\omega \in \Omega$, we can specify when a job *starts*, i.e., the first time that the job is executed after its release. To do that, we use the preimage $\mathcal{S}^{-1}(\{J\})$ to describe at which cycles job J is executed. In particular, $\inf \mathcal{S}^{-1}(\{J\}) \cdot t_{cycle}$ is the earliest time that J can be executed. Since J cannot be executed before its release time r_J, taking the maximum $\max(\inf \mathcal{S}^{-1}(\{J\}) \cdot t_{cycle}, r_J)$ gives us the start of job J. Similarly, we specify the *finish* of a job J, i.e., the time that J completes its execution demand. Specifically, J must be executed for $\frac{c_J}{t_{cycle}}$ clock cycles, and for $t \in \mathbb{Z}$ we can calculate the number of clock cycles that J is executed until the t-th cycle by $|\mathcal{S}^{-1}(\{J\}) \cap \left\{\frac{r_J}{t_{cycle}}, \dots, t\right\}|$. Finding the minimal $t \in \mathbb{Z}$ such that the number of clock cycles that $|\mathcal{S}^{-1}(\{J\}) \cap \left\{\frac{r_J}{t_{cycle}}, \dots, t\right\}| \geq \frac{c_J}{t_{cycle}}$ gives us the index of the cycle at which J finishes. Since J can still be executed during the whole cycle, the finishing time of job J is $(t+1) \cdot t_{cycle}$.

Definition 2.2 (Start and Finish). *Let $\mathcal{S} \in \mathbb{S}^{discr}(\mathbb{J})$ and $\omega = ((r_J)_{J\in\mathbb{J}}, (c_J)_{J\in\mathbb{J}}) \in \Omega$ be a schedule and a system evolution of the job set $\mathbb{J}$, respectively. We define the* start *of a job $J \in \mathbb{J}$ as*

$$s_J^{\mathcal{S},\omega} := \max\left(\inf \mathcal{S}^{-1}(\{J\}) \cdot t_{cycle}, r_J\right), \tag{2.4}$$

where $\mathcal{S}^{-1}(\{J\})$ is the preimage of $\{J\}$. We define the finish *of a job $J \in \mathbb{J}$ as*

$$f_J^{\mathcal{S},\omega} := \left(\inf\left\{t \in \mathbb{Z} \,\middle|\, \left|\mathcal{S}^{-1}(\{J\}) \cap \left\{\frac{r_J}{t_{cycle}}, \dots, t\right\}\right| \geq \frac{c_J}{t_{cycle}}\right\} + 1\right) \cdot t_{cycle}. \tag{2.5}$$

If we consider jobs with execution demand $c_J = 0$ (e.g., for convenience in the proofs), then we assume that such degenerated jobs finish as soon as they are released, i.e., $s_J^{\mathcal{S},\omega} = f_J^{\mathcal{S},\omega} = r_J$.

If a job is not assigned enough processor cycles to be executed, then it can never complete its execution demand. We call such a behavior *starvation.*

Definition 2.3 (Starvation). *In case a job J can never be completed, we have $f_J^{\mathcal{S},\omega} = \infty$, and we say that the job J* starves. *If $f_J^{\mathcal{S},\omega} < \infty$ for all $J \in \mathbb{J}$, we say that the schedule $\mathcal{S}$ is* starvation free *with respect to ω.*

To meet timing requirements of the real-time system, all jobs J must be completed until their respective deadline d_J^ω.

Definition 2.4 (Feasible). *Let $\mathcal{S} \in \mathbb{S}^{discr}(\mathbb{J})$ and $\omega = ((r_J)_{J\in\mathbb{J}}, (c_J)_{J\in\mathbb{J}}) \in \Omega$ be a schedule and a system evolution of the job set $\mathbb{J}$, respectively. We say that $\mathcal{S}$ is a* feasible *schedule for the system evolution ω, if $f_J^{\mathcal{S},\omega} \leq d_J^\omega$ for all jobs $J \in \mathbb{J}$ for which a deadline d_J^ω exists.*

Usually, the cycle length is very small compared to the job execution demand. Hence, analyzing scheduling behavior on the cycle-level is often intractable. As an alternative, we approximate the discrete behavior by a continuous model.

Continuous Schedules A continuous schedule is defined by a function on the continuous time domain $\mathcal{S}\colon \mathbb{R} \to \mathbb{J} \cup \{\bot\}$. Intuitively, we want to define $\mathcal{S}(t) = J$ if J is executed at time t. However, the execution at a specific time point t is not clearly defined. For example, consider a job J executed from time t_1 to t_2 and another job J' executed from time t_2 to t_3. At time t_2, is $\mathcal{S}(t_2) = J$, or J', or even $\bot$? Also, does it make a difference for the scheduling behavior if the schedule changes only at a few time points? In the literature, there are two approaches to solve such problems: Either it is assumed that even changing a single value of the scheduling function creates a new schedule (e.g., [BRH90; Uet+20]), or a convention is used to assign those unclear time points (e.g., by assuming that the scheduling function is right-continuous [BLV09]). While the former solution allows different schedules to describe the same execution behavior, the latter limits the generalizability of research results that abide to specific conventions. In this dissertation, we solve these two shortcomings of current research praxis, by using the Lebesgue-measure λ to identify equivalent scheduling functions.

We call a function $\mathcal{S}\colon \mathbb{R} \to \mathbb{J} \cup \{\bot\}$ measurable, if $\mathcal{S}^{-1}(\{\bot\})$ and $\mathcal{S}^{-1}(\{J\})$ are Lebesgue-measurable for all $J \in \mathbb{J}$. Under a measurable function $\mathcal{S}\colon \mathbb{R} \to \mathbb{J} \cup \{\bot\}$, we say that $J \in \mathbb{J}$ is executed for $\lambda(\mathcal{S}^{-1}(\{J\}) \cap (t_1, t_2))$ time units during a time interval $(t_1, t_2) \subseteq \mathbb{R}$. Given a job J and an interval (t_1, t_2), then if two different functions $\mathcal{S}$ and $\mathcal{S}'$ coincide *almost everywhere*[1], then $\lambda(\mathcal{S}^{-1}(\{J\}) \cap (t_1, t_2)) = \lambda(\mathcal{S}'^{-1}(\{J\}) \cap (t_1, t_2))$, i.e., they describe the same execution behavior of J. We define an equivalence relation $\mathcal{S} \sim \mathcal{S}'$ if and only if $\mathcal{S}$ and $\mathcal{S}'$ coincide almost everywhere.

Definition 2.5 (Continuous Schedule)**.** *Given a set of jobs $\mathbb{J}$, then we define the set of all continuous schedules as the quotient set:*

$$\mathbb{S}^{cont}(\mathbb{J}) = \{\mathcal{S}\colon \mathbb{R} \to \mathbb{J} \cup \{\bot\} \text{ Lebesgue-measurable}\} \big/ {\sim} \tag{2.6}$$

That is, $\mathbb{S}^{cont}(\mathbb{J})$ is the set of all equivalence classes $[\mathcal{S}]$ with $\mathcal{S}\colon \mathbb{R} \to \mathbb{J} \cup \{\bot\}$ Lebesgue-measurable, and two equivalence classes $[\mathcal{S}]$ and $[\mathcal{S}']$ are equal in $\mathbb{S}^{cont}(\mathbb{J})$ if $\mathcal{S} \sim \mathcal{S}'$. An equivalence class $[\mathcal{S}] \in \mathbb{S}^{cont}(\mathbb{J})$ is called a continuous schedule.

We note that, in the continuous time domain, we distinguish between a *scheduling function* $\mathcal{S}$ and a *schedule* $[\mathcal{S}]$. Every scheduling function defines a schedule, but different scheduling functions may define the same schedule.

Based on a continuous schedule $[\mathcal{S}] \in \mathbb{S}^{cont}(\mathbb{J})$, we specify the execution behavior of jobs. More specifically, during an interval $(t_1, t_2) \subseteq \mathbb{R}$, the processor idles for

$$\mathrm{idle}^{[\mathcal{S}]}(t_1, t_2) \coloneqq \lambda(\mathcal{S}^{-1}(\{\bot\}) \cap (t_1, t_2)) \tag{2.7}$$

and executes a job $J \in \mathbb{J}$ for

$$\mathrm{exec}_J^{[\mathcal{S}]}(t_1, t_2) \coloneqq \lambda(\mathcal{S}^{-1}(\{J\}) \cap (t_1, t_2)) \tag{2.8}$$

[1]Almost everywhere means everywhere up to a set of measure 0.

time units. Moreover, we denote by $\operatorname{exec}_\tau^{[\mathcal{S}]}(t_1,t_2) = \sum_{j=1}^{\infty} \operatorname{exec}_{\tau(j)}^{[\mathcal{S}]}(t_1,t_2)$ the time that τ is executed during the interval (t_1,t_2), and we denote by $\operatorname{exec}^{[\mathcal{S}]}(t_1,t_2) = \sum_{\tau\in\mathbb{T}} \operatorname{exec}_\tau^{[\mathcal{S}]}(t_1,t_2)$ the time that any task is executed during the interval (t_1,t_2). We say that J is *exclusively executed* during the bounded interval (t_1,t_2) if $\operatorname{exec}_J^{[\mathcal{S}]}(t_1,t_2) = t_2 - t_1$. Similarly, τ is *exclusively executed* if $\operatorname{exec}_\tau^{[\mathcal{S}]}(t_1,t_2) = t_2 - t_1$. We say the processor *exclusively idles* during the bounded interval (t_1,t_2) if $\operatorname{idle}^{[\mathcal{S}]}(t_1,t_2) = t_2 - t_1$.

We note that $\operatorname{idle}^{[\mathcal{S}]}$, $\operatorname{exec}_J^{[\mathcal{S}]}$, $\operatorname{exec}_\tau^{[\mathcal{S}]}$ and $\operatorname{exec}^{[\mathcal{S}]}$ are well-defined, in the sense that they are independent of the choice of the representative $\mathcal{S}$. More specifically, consider two representatives $\mathcal{S}$ and $\mathcal{S}'$ of the equivalence class $[\mathcal{S}] = [\mathcal{S}']$. Then $\mathcal{S}$ and $\mathcal{S}'$ coincide almost everywhere. Therefore, also $\mathcal{S}^{-1}(\{\xi\}) \cap (t_1,t_2)$ and $\mathcal{S}'^{-1}(\{\xi\}) \cap (t_1,t_2)$ coincide almost everywhere, for all $\xi \in \mathbb{J} \cup \{\perp\}$ and $(t_1,t_2) \subseteq \mathbb{R}$. We obtain

$$\operatorname{idle}^{[\mathcal{S}]}(t_1,t_2) = \lambda(\mathcal{S}^{-1}(\{\perp\}) \cap (t_1,t_2)) = \lambda(\mathcal{S}'^{-1}(\{\perp\}) \cap (t_1,t_2)) = \operatorname{idle}^{[\mathcal{S}']}(t_1,t_2) \tag{2.9}$$

$$\operatorname{exec}_J^{[\mathcal{S}]}(t_1,t_2) = \lambda(\mathcal{S}^{-1}(\{J\}) \cap (t_1,t_2)) = \lambda(\mathcal{S}'^{-1}(\{J\}) \cap (t_1,t_2)) = \operatorname{exec}_J^{[\mathcal{S}']}(t_1,t_2). \tag{2.10}$$

Moreover, $\operatorname{exec}_\tau^{[\mathcal{S}]}(t_1,t_2) = \sum_{j=1}^{\infty} \operatorname{exec}_{\tau(j)}^{[\mathcal{S}]}(t_1,t_2) = \sum_{j=1}^{\infty} \operatorname{exec}_{\tau(j)}^{[\mathcal{S}']}(t_1,t_2) = \operatorname{exec}_\tau^{[\mathcal{S}']}(t_1,t_2)$ by definition of exec_τ, and $\operatorname{exec}^{[\mathcal{S}]}(t_1,t_2) = \operatorname{exec}^{[\mathcal{S}']}(t_1,t_2)$.

Definition 2.6 (Start and Finish)**.** *Let $[\mathcal{S}] \in \mathbb{S}^{cont}(\mathbb{J})$ and $\omega = ((r_J)_{J\in\mathbb{J}}, (c_J)_{J\in\mathbb{J}}) \in \Omega$ be a schedule and a system evolution of the job set $\mathbb{J}$, respectively. We define the start of a job $J \in \mathbb{J}$ as*

$$s_J^{[\mathcal{S}],\omega} := \max\left(\sup\left\{t \,\middle|\, \operatorname{exec}_J^{[\mathcal{S}]}(-\infty,t) = 0\right\}, r_J\right). \tag{2.11}$$

We define the finish of a job $J \in \mathbb{J}$ as

$$f_J^{[\mathcal{S}],\omega} := \inf\left\{t \,\middle|\, \operatorname{exec}_J^{[\mathcal{S}]}(r_J,t) \geq c_J\right\}. \tag{2.12}$$

Similar to continuous schedules, if jobs with execution demand $c_J = 0$ are considered, then we assume that such degenerated jobs finish as soon as they are released, i.e., $s_J^{[\mathcal{S}],\omega} = f_J^{[\mathcal{S}],\omega} = r_J$.

Both, $s_J^{[\mathcal{S}],\omega}$ and $f_J^{[\mathcal{S}],\omega}$ are well-defined with respect to the choice of $\mathcal{S}$ since $\operatorname{exec}_J^{[\mathcal{S}]}$ is well-defined. As for discrete schedules, we define starvation and starvation free.

Definition 2.7 (Starvation)**.** *In case a job J can never be completed, we have $f_J^{[\mathcal{S}],\omega} = \infty$, and we say that the job J* starves*. If $f_J^{[\mathcal{S}],\omega} < \infty$ for all $J \in \mathbb{J}$, we say that the schedule $[\mathcal{S}]$ is* starvation free *with respect to ω.*

The following definition specifies whether all timing requirements of the real-time system are met.

Definition 2.8 (Feasible)**.** *Let $[\mathcal{S}] \in \mathbb{S}^{cont}(\mathbb{J})$ and $\omega = ((r_J)_{J\in\mathbb{J}}, (c_J)_{J\in\mathbb{J}}) \in \Omega$ be a schedule and a system evolution of the job set $\mathbb{J}$, respectively. We say that $[\mathcal{S}]$ is a feasible schedule for the system evolution ω if $f_J^{[\mathcal{S}],\omega} \leq d_J^\omega$ for all jobs $J \in \mathbb{J}$ for which a deadline d_J^ω exists.*

In this dissertation, continuous schedules are designed to be an extension of discrete schedules. That is, every discrete schedule can be considered as a continuous schedule.

Remark 2.9. *The set of continuous schedules can be considered as an extension of the set of discrete schedules, in the sense that there exists an embedding*

$$\mathbb{S}^{discr} \hookrightarrow \mathbb{S}^{cont}, \mathcal{S} \mapsto \left[t \mapsto \mathcal{S}\left(\left\lfloor \frac{t}{t_{cycle}} \right\rfloor \right) \right] \tag{2.13}$$

This map preserves start *and* finish *of jobs, as well as* feasibility *of the schedule.*

In this dissertation, the focus is on the continuous time domain. That is, the contributions in this work consider continuous schedules. However, Remark 2.9 demonstrates that the contributions are relevant to the discrete model as well.

2.3.2 Multiprocessor Schedules

If several cores $P_1, \dots, P_M$ are involved in processing the system evolution ω of the jobs $\mathbb{J}$, then the notion of schedules from the previous section has to be adjusted. That is, one schedule function $\mathcal{S}_j$ for each core P_j describes the schedule on that specific core. Hence, the complete multiprocessor scheduling function is defined as

$$\mathcal{S}^M := \mathcal{S}_1 \times \cdots \times \mathcal{S}_M. \tag{2.14}$$

The set of all schedules can be described as $\prod_{j=1}^M \mathbb{S}^{discr}(\mathbb{J})$ in the discrete time domain, and as $\prod_{j=1}^M \mathbb{S}^{cont}(\mathbb{J})$ in the continuous time domain. In particular, two continuous schedules $[\mathcal{S}^M]$ and $[\mathcal{S}'^M]$ defined by functions $\mathcal{S}^M = \mathcal{S}_1 \times \cdots \times \mathcal{S}_M$ and $\mathcal{S}'^M = \mathcal{S}'_1 \times \cdots \times \mathcal{S}'_M$ coincide if and only if $[\mathcal{S}_j] = [\mathcal{S}'_j]$ for all $j = 1, \dots, M$. Since the results of this dissertation focus on the model with continuous time domain, we limit to that model for the further discussion of multiprocessor schedules. However, the discussion can be done similarly for the discrete time domain.

First, we observe that the execution time of jobs and of tasks can be determined from the execution time on each core:

$$\text{exec}_J^{[\mathcal{S}^M]}(t_1, t_2) := \sum_{j=1}^{M} \text{exec}_J^{[\mathcal{S}_j]}(t_1, t_2), \quad \text{exec}_\tau^{[\mathcal{S}^M]}(t_1, t_2) := \sum_{\xi=1}^{\infty} \text{exec}_{\tau(\xi)}^{[\mathcal{S}^M]}(t_1, t_2) \tag{2.15}$$

We note that usually only schedules are considered where jobs cannot be executed on several cores simultaneously, i.e., $\text{exec}_J^{\mathcal{S}^M}(t_1, t_2) \leq t_2 - t_1$ for all $t_1 \leq t_2$. The system idles if all cores idle at the same time, that is:

$$\text{idle}^{[\mathcal{S}^M]}(t_1, t_2) := \lambda \left(\bigcap_{j=1}^{M} \mathcal{S}_j^{-1}(\{\perp\}) \cap (t_1, t_2) \right) \tag{2.16}$$

Next, we formally specify the start and finish of jobs in multiprocessor schedules. This is done similar to Definition 2.6, utilizing the execution function for multiprocessor schedules instead.

Definition 2.10 (Start and Finish). *Let $[\mathcal{S}^M] \in \prod_{j=1}^M \mathbb{S}^{cont}(\mathbb{J})$ be a multiprocessor schedule, and let $\omega = ((r_J)_{J\in\mathbb{J}}, (c_J)_{J\in\mathbb{J}}) \in \Omega$ be a system evolution of the job set $\mathbb{J}$. We define the start of a job $J \in \mathbb{J}$ as*

$$s_J^{[\mathcal{S}^M],\omega} := \max\left(\sup\left\{t \,\middle|\, \text{exec}_J^{[\mathcal{S}^M]}(-\infty, t) = 0\right\}, r_J\right). \tag{2.17}$$

Moreover, we define the finish of a job $J \in \mathbb{J}$ as

$$f_J^{[\mathcal{S}^M],\omega} := \inf\left\{t \,\middle|\, \text{exec}_J^{[\mathcal{S}^M]}(r_J, t) \geq c_J\right\}. \tag{2.18}$$

We say that a job $J \in \mathbb{J}$ starves *if $f_J^{[\mathcal{S}^M],\omega} = \infty$, and we say that the schedule $[\mathcal{S}^M]$ is starvation free with respect to ω if $f_J^{[\mathcal{S}^M],\omega} < \infty$ for all $J \in \mathbb{J}$.*

By definition of start, $s_J^{[\mathcal{S}^M],\omega} := \min_{j=1,\dots,M} s_J^{[\mathcal{S}_j],\omega}$ holds. However, a similar statement for the finish by taking the maximum finish among all processors is not possible, since the finish on each processor accounts for the whole execution demand of job J. As in the uniprocessor case, all timing requirements can be met if all jobs finish until their respective deadline.

Definition 2.11 (Feasible). *Let $[\mathcal{S}^M] \in \prod_{j=1}^M \mathbb{S}^{cont}(\mathbb{J})$ be a multiprocessor schedule, and let $\omega = ((r_J)_{J\in\mathbb{J}}, (c_J)_{J\in\mathbb{J}}) \in \Omega$ be a system evolution of the job set $\mathbb{J}$. We say that $[\mathcal{S}^M]$ is a feasible schedule for the system evolution ω if $f_J^{[\mathcal{S}^M],\omega} \leq d_J^\omega$ for all jobs $J \in \mathbb{J}$ for which a deadline d_J^ω exists.*

When considering multiprocessor scheduling, we usually distinguish between partitioned and global scheduling. Under *partitioned scheduling*, each task is assigned to one specific core. That is, the task set is partitioned as $\mathbb{T} = \mathbb{T}_1 \sqcup \cdots \sqcup \mathbb{T}_M$, and tasks $\mathbb{T}_j$ are assigned to core P_j. All jobs of the same task can only be executed on the core they are assigned to. Under *global scheduling*, tasks are not fixed to a specific core. In that model, several cores can be involved in the execution of the same job. Although both approaches are inherently different, we do not need to distinguish between those mechanisms for the extension of uniprocessor schedules to multiprocessor schedules. The only difference is that under partitioned scheduling, a smaller set of scheduling functions is available. That is, let $\mathbb{J}_j$ be the jobs obtained by tasks $\mathbb{T}_j$, then under partitioned scheduling only the schedules $\prod_{j=1}^M \mathbb{S}^{cont}(\mathbb{J}_j) \subseteq \prod_{j=1}^M \mathbb{S}^{cont}(\mathbb{J})$ can be obtained. The definitions of this section simplify accordingly for partitioned scheduling.

Generating a feasible schedule given a system evolution is challenging, since the system evolution may not be clear from the beginning. That is, job releases may not be predefined, and the job demand is only known when the job finishes. Moreover, the uncertainty in the environment and behavior outside the model has an impact, provoking different system evolutions. Therefore, to make a reasonable scheduling decision, tasks must be analyzed and designed to give useful information about the system evolution beforehand. The subsequent section considers different system evolutions, and discusses abstractions of the systems evolutions to be utilized for the analysis of scheduling mechanisms.

2.4 Uncertainties and Abstraction Levels

When the system runs, it is not clear which system evolution $\omega \in \Omega$ the system deals with. Due to the absence of information it is not clear (i) if a *feasible* schedule of the system evolution always exists and (ii) how the feasible schedule can be generated. This section discusses the abstraction of properties of the set of system evolutions Ω that can be utilized to make scheduling decisions and to analyze the feasibility of schedules under those scheduling decisions. The abstraction of a task is given in the form of

$$\tau \in \text{Property}_1(\textit{values}_1) \cap \text{Property}_2(\textit{values}_2) \cap \dots \cap \text{Property}_\xi(\textit{values}_\xi) \tag{2.19}$$

if task τ satisfies all ξ properties with the corresponding values.

2.4.1 Abstraction of Execution Demand

To describe the execution demand of jobs under several system evolutions, we abstract its analytical properties. That is, given a job $\tau(j)$, its execution demand can take different values $\left\{ c^{\omega}_{\tau(j)} \,\middle|\, \omega \in \Omega \right\}$. In case Ω is equipped with a probability measure $\mathbb{P}$, then $c^{\bullet}_{\tau(j)}$ is a random variable and its behavior can be abstracted by a probability distribution. Otherwise, only upper and lower bounds on the execution demand can be provided, so-called Worst-Case Execution Time (WCET) and Best-Case Execution Time (BCET), respectively. Usually, it is assumed that jobs released by the same task adhere to a similar behavior. Therefore, abstracting all jobs of a task by the same information (bounds or distribution) is sufficient. However, some abstractions also specify execution demand patterns [MC97]. In the following, we specify abstractions of the execution demand which are relevant to this dissertation.

Worst-Case Execution Time (WCET) The WCET of a task is an upper bound on the execution demand of all jobs under all system evolutions. More specifically, we denote by

$$\tau \in \text{WCET}(C_\tau) \tag{2.20}$$

the case that $c^{\omega}_{\tau(j)} \leq C_\tau$ for all $j \in \mathbb{N}$ and for all $\omega \in \Omega$.

Best-Case Execution Time (BCET) The BCET of a task is a lower bound on the execution demand of all jobs under all system evolutions. More specifically, we denote by

$$\tau \in \text{BCET}(C^{\min}_\tau) \tag{2.21}$$

the case that $c^{\omega}_{\tau(j)} \geq C_\tau$ for all $j \in \mathbb{N}$ and for all $\omega \in \Omega$. Naturally, the demand of any job is lower bounded by 0, which is why every task τ is in BCET(0).

If the BCET is given, the WCET is also denoted by $C^{\max}_\tau$ instead of C_τ for the sake of uniformity. That is, if BCET and WCET are given, we denote that case by

$$\tau \in \text{BCET}(C^{\min}_\tau) \cap \text{WCET}(C^{\max}_\tau). \tag{2.22}$$

Probabilistic Execution Time (pET) Assume that Ω is equipped with a probability measure $\mathbb{P}$. If there exists a cumulative distribution function $\mathcal{D}_\tau \colon \mathbb{R} \to [0,1]$ such that the execution demand $c^\omega_{\tau(j)}$ is distributed according to $\mathcal{D}_\tau$ for all $j \in \mathbb{N}$, then we say that τ has pET with distribution $\mathcal{D}_\tau$. More specifically, we denote by

$$\tau \in \mathrm{pET}(\mathcal{D}_\tau) \tag{2.23}$$

the case that $\mathbb{P}\left(\left\{\omega \in \Omega \,\middle|\, c^\omega_{\tau(j)} \leq t\right\}\right) = \mathcal{D}_\tau(t)$ for all $j \in \mathbb{N}$ and $t \in \mathbb{R}$. Of particular interest, from an analytical perspective, is the case that the random variables $c^\bullet_{\tau(j)}$ are independent for all jobs $\tau(j)$ with $j \in \mathbb{N}$. More specifically, we denote by

$$\tau \in \mathrm{pET}^{\mathrm{iid}}(\mathcal{D}_\tau) \tag{2.24}$$

the case that $(c^\bullet_{\tau(j)})_{j\in\mathbb{N}}$ are independent and identically distributed (iid) variables with $\tau \in \mathrm{pET}(\mathcal{D}_\tau)$.

Probabilistic Worst-Case Execution Time (pWCET) If Ω is equipped with a probability measure $\mathbb{P}$ but the execution demand of jobs of task τ cannot all be described by the same probability distribution, then we cannot use the pET task model above to abstract the execution demand. However, we can use a cumulative distribution function $\mathcal{D}_\tau$ to abstract the behavior of upper bounds on the execution demands. Specifically, we denote by

$$\tau \in \mathrm{pWCET}(\mathcal{D}_\tau) \tag{2.25}$$

the case that there exist upper bounds $\bar{c}^\omega_{\tau(j)} \geq c^\omega_{\tau(j)}$ for all $\omega \in \Omega$ and $j \in \mathbb{N}$ such that $\mathbb{P}\left(\left\{\omega \in \Omega \,\middle|\, \bar{c}^\omega_{\tau(j)} \leq t\right\}\right) = \mathcal{D}_\tau(t)$ for all $j \in \mathbb{N}$ and $t \in \mathbb{R}$. To exploit independence analytically, for the pWCET it is sufficient that the upper bounds $\bar{c}^\bullet_{\tau(j)}$ are independent for all jobs $\tau(j)$ with $j \in \mathbb{N}$. More specifically, we denote by

$$\tau \in \mathrm{pWCET}^{\mathrm{iid}}(\mathcal{D}_\tau) \tag{2.26}$$

the case that $\tau \in \mathrm{pWCET}(\mathcal{D}_\tau)$ such that the upper bounds $(\bar{c}^\bullet_{\tau(j)})_{j\in\mathbb{N}}$ are iid variables.

Remark 2.12 (Comparison of Abstraction Levels). *The different abstraction levels of the execution demand allow a trade-off between more general solutions and more specific solutions [Brü+22]. Specifically, an abstraction of pETs allows to derive the BCET and WCET as follows (assuming that $\mathbb{P}(\{\omega\}) > 0$ for all $\omega \in \Omega$):*

$$\mathrm{pET}(\mathcal{D}_\tau) \subseteq \mathrm{BCET}(\sup \mathcal{D}_\tau^{-1}(\{0\})) \cap \mathrm{WCET}(\inf \mathcal{D}_\tau^{-1}(\{1\})) \tag{2.27}$$

Similarly, if the execution demand is described as pET, then it can be described as pWCET with the same cumulative distribution function, and the pWCET can be translated into a WCET (again, assuming that $\mathbb{P}(\{\omega\}) > 0$ for all $\omega \in \Omega$):

$$\mathrm{pET}(\mathcal{D}_\tau) \subseteq \mathrm{pWCET}(\mathcal{D}_\tau) \subseteq \mathrm{WCET}(\inf \mathcal{D}_\tau^{-1}(\{1\})) \tag{2.28}$$

Different abstraction levels for the execution demand can be chosen, depending on the capabilities of the analysis and the information available from the application.

2.4.2 Abstraction of Job Releases

For job releases, the literature provides a large variety of different abstraction approaches [Aud+93; BW94; LL73; Mok83; TCN00]. In the following, we discuss abstractions of job releases which are relevant to this dissertation. Subsequently, we compare the different abstraction levels and introduce definitions based on the abstraction of releases.

Fixed Arrival Pattern When considering classical time-triggered systems, job arrivals occur at a "predetermined tick of a clock" [KS22]. That is, jobs $\tau(j)$ are released at the same time $r_{\tau(j)} \in \mathbb{R}$ independent of the system evolution $\omega \in \Omega$. We say that a task has a fixed arrival pattern if all its jobs are released at predefined time points $(r_{\tau(j)})_{j\in\mathbb{N}}$. More specifically, we denote by

$$\tau \in \text{FixRel}((r_{\tau(j)})_{j\in\mathbb{N}}) \tag{2.29}$$

the case that $r^{\omega}_{\tau(j)} = r_{\tau(j)}$ for all $j \in \mathbb{N}$ and $\omega \in \Omega$.

Periodic With Offset The standard procedure to obtain a fixed arrival pattern is by equipping a task with a timer. Whenever the timer runs out, the task releases a new job and resets the timer. If the first job is released at time $\phi_\tau \in \mathbb{R}$ and then periodically every $T_\tau \in \mathbb{R}_{>0}$ time units, we say that task τ is periodic with period T_τ and offset ϕ_τ, and we denote that case by

$$\tau \in \text{PerOff}(T_\tau, \phi_\tau). \tag{2.30}$$

More specifically, $r^{\omega}_{\tau(j)} = (j-1)\cdot T_\tau + \phi_\tau$ for all $j \in \mathbb{N}$ and for all $\omega \in \Omega$. If all tasks $\tau \in \mathbb{T}$ are periodic with offset $\phi_\tau = 0$, then we call the task set *synchronous*.

Periodic In some cases, the offset of a periodic task is not deterministic, in the sense that the offset may depend on the system evolution. As an example, in the implementation of RTEMS [RTE23], a well-established Real-Time Operating System (RTOS), the timer is initiated at the start of the first job, which depends on the execution time of the higher-priority tasks. Without specifying an offset we still say that a task is periodic with period $T_\tau \in \mathbb{R}_{>0}$ if the release of two subsequent jobs of τ is exactly T_τ time units apart. More specifically, we denote by

$$\tau \in \text{Per}(T_\tau) \tag{2.31}$$

the case that $r^{\omega}_{\tau(j)} + T_\tau = r^{\omega}_{\tau(j+1)}$ for all $j \in \mathbb{N}$ and for all $\omega \in \Omega$.

Sporadic A more relaxed abstraction of job releases can be described by the sporadic task model. That is, the task releases jobs according to a *minimum inter-arrival time* $T_\tau \in \mathbb{R}_{>0}$. More specifically,

$$\tau \in \text{Spor}(T_\tau) \tag{2.32}$$

if $r^{\omega}_{\tau(j)} + T_\tau \le r^{\omega}_{\tau(j+1)}$ for all $j \in \mathbb{N}$ and $\omega \in \Omega$. The sporadic task model is also relevant in the context of event-triggered tasks, where job releases are triggered by events, often external events outside the system, for which only a lower bound on the inter-arrival time can be guaranteed. If upper and lower bounds on the inter-arrival time are abstracted, we denote by

$$\tau \in \mathrm{SporMinMax}(T_\tau^{\min}, T_\tau^{\max}) \tag{2.33}$$

the case that $r^\omega_{\tau(j)} + T_\tau^{\min} \leq r^\omega_{\tau(j+1)} \leq r^\omega_{\tau(j)} + T_\tau^{\max}$.

Arrival Curves A more descriptive abstraction of the inter-arrival time can be achieved by *arrival curves*, known from Real-Time Calculus, Network Calculus [BT01; TCN00], and Compositional Performance Analysis (CPA) [Hen+05; HAE17]. That is, for each task τ, an upper arrival curve $\alpha_\tau \colon \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is given, which provides by $\alpha_\tau(\Delta)$ an upper bound on the number of job releases inside an interval of length Δ. More specifically, we have

$$\alpha_\tau(\Delta) \geq \sup_t \left(\left| \left\{ r^\omega_{\tau(j)} \,\middle|\, j \in \mathbb{N} \right\} \cap [t, t+\Delta) \right| \right) \tag{2.34}$$

for all $\omega \in \Omega$ and $\Delta \geq 0$. We denote by

$$\tau \in \mathrm{Arr}(\alpha_\tau) \tag{2.35}$$

the case that job releases of τ can be described by the arrival curve α_τ. The power of the arrival curve model is that it can handle the case that the minimum inter-arrival time T_τ is 0. I.e., it is possible to release two (or more) jobs at the same time. Modeling such a task with a typical sporadic task $\tau \in \mathrm{Spor}(0)$ allows infinite bursts and is therefore infeasible.

We assume that the given upper arrival curve function α_τ is sub-additive [BT01], i.e., $\alpha_\tau(\Delta_1 + \Delta_2) \leq \alpha_\tau(\Delta_1) + \alpha_\tau(\Delta_2)$ for all $\Delta_1, \Delta_2 \in \mathbb{R}_{\geq 0}$. If the given curve is not sub-additive, then it can be tightened by applying the sub-additive closure to achieve the sub-additivity property, i.e., by defining a new upper arrival curve

$$\tilde{\alpha}_\tau(\Delta) \coloneqq \inf \left\{ \sum_{\xi=1}^{A} \alpha_\tau(\Delta_\xi) \,\middle|\, A \in \mathbb{N},\ \Delta_1, \ldots, \Delta_A \in \mathbb{R}_{\geq 0},\ \sum_{\xi=1}^{A} \Delta_\xi = \Delta \right\}. \tag{2.36}$$

Due to the sub-additivity property, we have that for all $\Delta \in \mathbb{R}$:

(i) $\alpha_\tau(\Delta) \leq \alpha_\tau(\Delta + \delta)$ for all $\delta \geq 0$ (α_τ monotonically increasing); otherwise, the existence of $\delta \geq 0$ with $\alpha_\tau(\Delta) > \alpha_\tau(\Delta + \delta)$ contradicts the sub-additivity property $\alpha_\tau(\Delta + \delta) \leq \alpha_\tau(\Delta) + \alpha_\tau(\delta) \leq \alpha_\tau(\Delta)$.

(ii) $\alpha_\tau(\Delta) \leq \alpha_\tau(\Delta - T_\tau) + 1$ if the minimum inter-arrival time of jobs is $T_\tau \geq 0$. This is due to the sub-additivity property $\alpha_\tau(\Delta) \leq \alpha_\tau(\Delta - T_\tau) + \alpha_\tau(T_\tau) = \alpha_\tau(\Delta - T_\tau) + 1$ when $T_\tau > 0$ and due to the fact that $\alpha_\tau(\Delta) \leq \alpha_\tau(\Delta) + 1$ when T_τ is 0.

Remark 2.13 (Comparison of Abstraction Levels)**.** *The sporadic task model can be considered as* behavior relaxation *[Brü+22] of the periodic task model, in the sense that it allows additional behavior. In particular, we have inclusions:*

$$\mathrm{PerOff}(T_\tau, \phi_\tau) \subseteq \mathrm{Per}(T_\tau) \subseteq \mathrm{Spor}(T_\tau) \tag{2.37}$$

That is, every periodic task with offsets can be considered as periodic task without offset, and every periodic task without offset can be considered as sporadic task. While an analytical result for sporadic tasks can be applied to periodic tasks as well, a dedicated analysis for periodic tasks might achieve tighter results because it is more restrictive for the task behavior.

The task models using fixed arrival patterns and arrival curves provide more detailed descriptions of periodic tasks with offset and of sporadic tasks, respectively. That is,

$$\mathrm{PerOff}(T_\tau, \phi_\tau) = \mathrm{FixRel}((\phi_\tau + (j-1)T_\tau)_{j\in\mathbb{N}}) \tag{2.38}$$

$$\mathrm{Spor}(T_\tau) = \mathrm{Arr}\left(\Delta \mapsto \left\lceil \frac{\Delta}{T_\tau} \right\rceil\right) \tag{2.39}$$

but not every task that can be described using a fixed release pattern or an arrival curve can be described as a periodic task with offset or as a sporadic task, respectively. In the work by von der Brüggen et al. [Brü+22], such generalization, that allows to describe the more simple instance of the problem precisely, is denoted as abstraction refinement*.*

In industry systems, periodic and sporadic tasks play an important role. That is, as reported by Akesson et al. [Ake+20; Ake+22] for industrial real-time systems, periodic and sporadic task activations are widely used in industry practice; specifically, in 82 % and 47 %, respectively, of the investigated systems. Therefore, these are also very popular in the research community.

In the following, we define common properties of periodic and sporadic tasks. Please note that the definitions of properties for sporadic tasks $\tau \in \mathrm{Spor}(T_\tau)$, especially Definitions 2.14, 2.15 and 2.16, also apply to periodic tasks and periodic tasks with offsets due to behavior relaxation (Equation (2.37)).

Definition 2.14 (Utilization)**.** *If a task τ is $\tau \in \mathrm{Spor}(T_\tau) \cap \mathrm{WCET}(C_\tau)$, then its* utilization *is defined by*

$$U_\tau := \frac{C_\tau}{T_\tau}. \tag{2.40}$$

Furthermore, the total utilization *of the task set $\mathbb{T}$ is defined by*

$$U_\mathbb{T} = \sum_{\tau\in\mathbb{T}} U_\tau. \tag{2.41}$$

Definition 2.15 (Harmonic Tasks)**.** *We call a set of periodic tasks $\mathbb{T}$ with $\tau \in \mathrm{Per}(T_\tau)$ for all $\tau \in \mathbb{T}$* harmonic*, if*

$$\frac{T_\tau}{T_{\tau'}} \in \mathbb{N} \qquad \textit{or} \qquad \frac{T_{\tau'}}{T_\tau} \in \mathbb{N} \tag{2.42}$$

for all $\tau, \tau' \in \mathbb{T}$. A typical example of a harmonic task set is if the periods T_τ are

$$T_\tau \in \{1, 2, 10, 20, 100, 200, 1\,000\} \tag{2.43}$$

for all $\tau \in \mathbb{T}$. Besides harmonic task sets, industry systems often deploy task sets that are almost *harmonic. Such task sets are called* semi-harmonic*. For instance, periods*

$$T_\tau \in \{1, 2, 5, 10, 20, 50, 100, 200, 500, 1\,000\} \tag{2.44}$$

for all $\tau \in \mathbb{T}$ are common in automotive systems [KZH15].

Definition 2.16 (Specification of Deadlines)**.** *Consider a set of sporadic tasks $\mathbb{T}$ with $\tau \in \mathrm{Spor}(T_\tau)$ for all $\tau \in \mathbb{T}$ in which every task has a relative deadline $D_\tau \in \mathbb{R}_{>0}$. We call $\mathbb{T}$ an* implicit-deadline *task set if $D_\tau = T_\tau$ for all $\tau \in \mathbb{T}$. Furthermore, we call $\mathbb{T}$ a* constrained-deadline *task set if $D_\tau \leq T_\tau$ for all $\tau \in \mathbb{T}$. In the general case, i.e., $D_\tau \leq T_\tau$ as well as $D_\tau > T_\tau$ is allowed for every task $\tau \in \mathbb{T}$, we call $\mathbb{T}$ an* arbitrary-deadline *task set. By definition, every implicit-deadline task set is a constrained-deadline task set, and every constrained-deadline task set is an arbitrary-deadline task set.*

Definition 2.17 (Hyperperiod)**.** *Consider a set of tasks $\mathbb{T}$ abstracted as periodic tasks $\tau \in \mathrm{Per}(T_\tau)$ for all $\tau \in \mathbb{T}$. The* hyperperiod *of $\mathbb{T}$ is the least common multiple (lcm) of all task periods of $\mathbb{T}$. More specifically,*

$$H(\mathbb{T}) \coloneqq \mathrm{lcm}(\{T_\tau \mid \tau \in \mathbb{T}\}) \tag{2.45}$$

defines the hyperperiod of $\mathbb{T}$. Although a similar definition could be made for tasks abstracted as sporadic, this definition restricts to periodic tasks, since the hyperperiod is usually used to in the context of repetitive release patterns, which cannot be guaranteed for sporadic tasks.

2.4.3 Scheduling Algorithms

A scheduling algorithm $\mathcal{A}$ is a systematic approach to translate the system evolution $\omega = ((r_J)_{J\in\mathbb{J}}, (c_J)_{J\in\mathbb{J}})$ into a schedule $\mathcal{A}(\omega) \in \mathbb{S}^{cont}(\mathbb{J})$. The literature provides plenty different scheduling algorithms, which can be categorized as follows.

Static schedulers create a scheduling table that allocates time slots to the tasks offline. At runtime, task instances are allocated to the processor based on these time slots and the table is repeated. Alternatively, priority-based schedulers allocate jobs to the processor based on (usually unique) priorities. This dissertation focuses on work-conserving priority-based scheduling mechanisms. That is, each job $J \in \mathbb{J}$ is assigned a priority $prio(J)$. At its release r_J, the job is moved to a *ready queue* $\mathcal{R}$, and at each point in time, the job of the ready queue with the highest priority is scheduled, i.e., the job $\arg\max_{J\in\mathcal{R}} prio(J)$ is scheduled.

We distinguish *preemptive* and *non-preemptive* scheduling. That is, under preemptive scheduling, a job that is executed may be moved back to the ready queue (i.e., *preempted*) if job priorities change or if a higher-priority job arrives at the ready queue. On the other hand, under non-preemptive scheduling, a job cannot be preempted. That is, a job that is executed runs to completion even if higher-priority jobs are in the ready queue. Each preemption is associated with a so-called *preemption delay* which is why the number of preemptions should be kept as low as possible to avoid excessive overhead.

Besides the distinction between preemptive and non-preemptive scheduling, we also distinguish different ways to set priorities. More specifically, the priority can be fixed on the task-level or on the job-level, or the priority can be set dynamically.

- **Task-level Fixed-Priority (T-FP) scheduling.** In T-FP scheduling, all jobs of the same task have the same priority level. Therefore, the priority level of the jobs is a task parameter, also referred to as the task priority. The set of higher-priority tasks is denoted by $hp(\tau)$ and the set of higher or equal priority tasks (including τ itself) is denoted by $hep(\tau)$. Despite the fact that T-FP scheduling algorithms are not optimal, in the sense that they cannot always generate a feasible schedule if one exists, they are widely supported by RTOSs for uniprocessor systems or partitioned scheduling algorithms in multiprocessor systems. This is due to an intuitive parameterization in terms of priorities that influence the response time of a task and the ability to implement such algorithms with low overhead. Typical T-FP algorithms are Deadline Monotonic (DM) [LW82] or Rate Monotonic (RM) [LL73], as defined below. An algorithm that is not T-FP is called **Task-level Dynamic-Priority (T-DP)**.

- **Job-level Fixed-Priority (J-FP) scheduling.** In J-FP scheduling, jobs of the same task may have different priority levels, but the priority of each job remains fixed throughout the schedule. Common scheduling algorithms are First In – First Out (FIFO), also called First-Come First-Served (FCFS), or Shortest Job First (SJF) [Erc19], that sort the job priorities by arrival time or execution time, respectively. Other typical examples for J-FP are Earliest-Deadline-First (EDF) [LL73] and its generalization EDF-Like (EL), defined below. Moreover, every T-FP scheduling algorithm is also J-FP. An algorithm that is not J-FP is called **Job-level Dynamic-Priority (J-DP)**. Every J-DP algorithm is also T-DP by definition. Typical examples for J-DP algorithms are Least-Laxity-First (LLF) [Leu89], also called Least Slack Time (LST), and Earliest Deadline Zero Laxity (EDZL) [Lee94].

In the following, we discuss those scheduling algorithms in more detail that are primarily considered throughout this dissertation.

DEADLINE MONOTONIC (DM) is a T-FP scheduling strategy that assigns tasks priorities according to their relative deadline. That is, the task with the shortest relative deadline D_τ has the highest priority, and the task with the largest relative deadline has the lowest priority. This scheduling strategy is discussed in detail by Leung and Whitehead [LW82].

Rate Monotonic (RM) is a T-FP scheduling strategy for sporadic tasks, i.e., $\tau \in \text{Spor}(T_\tau)$ for all $\tau \in \mathbb{T}$, that assigns tasks priorities according to their minimum inter-arrival time T_τ. That is, the task with the shortest T_τ has the highest priority, and the task with the largest T_τ has the lowest priority. This scheduling strategy is proposed by Liu and Layland [LL73]. In case of implicit-deadline tasks, DM and RM coincide.

First In – First Out (FIFO) is a T-DP/J-FP scheduling strategy that assigns the highest priority to the job with the earliest release time. FIFO has the benefit that it can easily be implemented using a single queue without the need to sort the queue. Consequently, FIFO has a very low overhead. FIFO is also called *First-Come First-Served (FCFS)*. We assume that an arbitrary but deterministic tie-braking rule is applied.

Earliest-Deadline-First (EDF) is a T-DP/J-FP scheduling strategy that assigns job priorities according to their absolute deadline [LL73]. That is, the earlier the absolute deadline $d^\omega_{\tau(j)}$ of a job $\tau(j)$ the higher is the job priority. When considering EDF scheduling, it is assumed that every task τ has a relative deadline D_τ, i.e., the absolute deadline is computed by $d^\omega_{\tau(j)} = r^\omega_{\tau(j)} + D_\tau$. Hence, the release time $r^\omega_{\tau(j)}$ of a job and the relative deadline D_τ of the corresponding task uniquely specify the job priority. We assume that an arbitrary but deterministic tie-braking rule is applied.

EDF-Like (EL) is a generalization of EDF scheduling. Specifically, EL uses a *relative priority point* $\Pi_\tau \in \mathbb{R}$ instead of the relative deadline D_τ to assign job priorities. That is, the earlier the *absolute priority point* $\pi^\omega_{\tau(j)} = r^\omega_{\tau(j)} + \Pi_\tau$ of a job $\tau(j)$, the higher is the job priority. We assume that an arbitrary but deterministic tie-braking rule is applied. The idea to model scheduling algorithms using priority points has originally been proposed by Leontyev and Anderson [LA07]. Depending on how the relative priority point is configured, EL can be modeled to behave like most J-FP schedules. For example, using relative priority points $\Pi_\tau = D_\tau$, EL behaves as EDF, and using relative priority points $\Pi_\tau = 0$, EL behaves as FIFO. In Section 5.1.3.1 we discuss the capabilities and limitations of EL scheduling in more detail, showing that EL can even imitate the behavior of T-FP scheduling algorithms.

2.5 Analytical Literature Results

Given a scheduling algorithm $\mathcal{A}$ and a task set $\mathbb{T}$, we are concerned with the question:

> Can all timing requirements be met?

Usually, if the timing requirements are given by a deadline constraint on the tasks, then we aim to ensure that the underlying scheduling algorithm always generates feasible schedules for all system evolutions $\omega \in \Omega$. In such a case, we also call the task set *schedulable under scheduling algorithm $\mathcal{A}$*. To determine if a task set is schedulable, *schedulability tests* are applied.

Since the underlying set of system evolutions Ω is unknown, schedulability tests use the information given by the task abstraction. That is, considering a specific abstraction, this covers several possible sets of system evolutions. We define by Ω^{abstr} the set of all possible system evolutions under a certain abstraction. In particular, assuming that the abstraction of Ω is correct, $\Omega \subseteq \Omega^{abstr}$ holds. We distinguish *sufficient*, *necessary* and *exact* schedulability tests:

- A sufficient test returns *True* if it guarantees that the task set is schedulable, and it returns *False* if it cannot guarantee schedulability, although the task set might still be schedulable. Specifically, a sufficient test guarantees that for all $\omega' \in \Omega^{abstr}$ the schedule $\mathcal{A}(\omega')$ is feasible.
- A necessary test returns *False* if it guarantees that the task set is *not* schedulable, and it returns *True* if it cannot guarantee that the task set is *not* schedulable. Specifically, a necessary test guarantees that there exists an $\omega' \in \Omega^{abstr}$ such that $\mathcal{A}(\omega')$ is not feasible.
- We call a schedulability test exact if is it sufficient and necessary.

An important tool for timing analysis is the Worst-Case Response Time (WCRT) of a task, i.e., the longest time from release to finish of any job of that task.

Definition 2.18 (Worst-Case Response Time (WCRT))**.** *Given a scheduling algorithm $\mathcal{A}$ and a system evolution $\omega \in \Omega$ of task set $\mathbb{T}$, then the* response time *of job $\tau(j)$ with $\tau \in \mathbb{T}$ and $j \in \mathbb{N}$ is given by*

$$R_{\tau(j)}^{\mathcal{A}(\omega),\omega} := f_{\tau(j)}^{\mathcal{A}(\omega),\omega} - r_{\tau(j)}^{\omega}. \tag{2.46}$$

The Worst-Case Response Time (WCRT) *of a task $\tau \in \mathbb{T}$ is the maximal response time of any job of τ under any system evolution, i.e.,*

$$R_{\tau}^{\mathcal{A}} := \sup_{j\in\mathbb{N}} \sup_{\omega\in\Omega} R_{\tau(j)}^{\mathcal{A}(\omega),\omega}. \tag{2.47}$$

The WCRT serves two purposes: (i) Checking if a task set is schedulable, i.e., if $R_\tau^{\mathcal{A}} \leq D_\tau$ for all $\tau \in \mathbb{T}$. (ii) Exploiting $R_\tau^{\mathcal{A}}$ in the analysis, e.g., to bound the number of interfering jobs.

In the following, we discuss classical results from the literature on real-time systems. We focus on the analysis of preemptive T-FP and J-FP scheduling algorithms, specifically RM, DM and EDF.

Critical Instant An important observation by Liu and Layland for T-FP scheduling is the Critical Instant Theorem (CIT) [LL73]: For a set of periodic tasks, i.e., $\tau \in \text{Per}(T_\tau)$ for all $\tau \in \mathbb{T}$, the WCRT of a task can be observed when all higher-priority tasks release jobs at the same time, the so-called *critical instant*. Furthermore, they also define the *critical time zone*, which is the interval where the WCRT for a task τ occurs, as the time between a critical instant and the end of the response to the corresponding request of the task. The CIT and the critical time zone are further extended to cope with the sporadic real-time task model [Mok83], and have been widely adopted as a backbone in WCRT analyses. In the following we give a short summary and explain the proof of correctness when determining the WCRT.

Definition 2.19 (Critical Instant, reworded from [LL73; Mok83])**.** *Let $\mathbb{T}$ be a task set with $\tau \in \text{Spor}(T_\tau) \cap \text{WCET}(C_\tau)$ for all $\tau \in \mathbb{T}$. The* critical instant *of a task $\tau \in \mathbb{T}$ under uniprocessor preemptive T-FP scheduling occurs at the release of a job of τ when*

(i) every higher-priority task in $hp(\tau)$ releases a job simultaneously with the job of τ,

(ii) all subsequent jobs of higher-priority tasks are released as early as possible by respecting their minimum inter-arrival times, i.e., periodically with period T_τ, and

(iii) every job is executed with its WCET C_τ.

Definition 2.20. *(Critical Time Zone, reworded from [LL73]) With Definition 2.19, the* critical time zone *of task τ is the time interval between a critical instant and the finishing time of the job of τ released at the critical instant.*

Theorem 2.21 (Critical Instant Theorem (CIT), reworded from [LL73])**.** *Under uniprocessor T-FP preemptive scheduling, if the interval length of the critical time zone of task τ is no more than T_τ, then the response time of this job of τ (i.e., the length of the critical time zone) is the WCRT of τ; otherwise, the WCRT of τ is greater than T_τ.*

Although the statement is correct, the original proof of the CIT by Liu and Layland [LL73] was incomplete. Several patches are available, e.g., in [Bri04]. For completeness, in the following we recall the proof in our notation.

Proof. Suppose that $[\mathcal{S}]$ is the preemptive T-FP schedule of a system evolution ω of jobs generated by the set $\mathbb{T}$ of sporadic tasks. In the schedule $[\mathcal{S}]$, if there is a job of task τ whose response time is greater than T_τ, let job $\tau(j)$ be the first job of them. Otherwise, let $\tau(j)$ be any arbitrary job of τ.

Interval Extension: Let t_0 be the earliest instant prior to $r^\omega_{\tau(j)}$, i.e., $t_0 \leq r^\omega_{\tau(j)}$, such that the processor only executes jobs generated by the higher-priority tasks in $hp(\tau)$ from t_0 to $r^\omega_{\tau(j)}$ in the schedule $[\mathcal{S}]$. We remove all jobs released before t_0. Let $[\mathcal{S}_1]$ be the resulting T-FP schedule of the remaining jobs. As the removal of such jobs has no impact on the schedule $[\mathcal{S}]$ between t_0 and $f^{[\mathcal{S}],\omega}_{\tau(j)}$, the two schedules $[\mathcal{S}]$ and $[\mathcal{S}_1]$ coincide (almost everywhere) from t_0 to $f^{[\mathcal{S}],\omega}_{\tau(j)}$.

Release Time Modification: According to the above definition, task τ is not executed from t_0 to $r^\omega_{\tau(j)}$. Moving the release time of $\tau(j)$ to t_0 does not change the schedule $[\mathcal{S}_1]$ but the response time of $\tau(j)$ is increased by $r^\omega_{\tau(j)} - t_0 \geq 0$.

Simultaneous Release and Periodic Interference: Since there is no job released prior to t_0 in the schedule $[\mathcal{S}_1]$, in order to achieve the maximum interference, the jobs of $\tau' \in hp(\tau)$ should be released as early as possible, starting from t_0. This implies that the first job of $\tau' \in hp(\tau)$ should be released at time t_0, followed by subsequent jobs released periodically. Furthermore, every job runs for its WCET. That is, the critical instant in Definition 2.19 is at time t_0. Let $[\mathcal{S}_2]$ be the resulting schedule and let $f^{[\mathcal{S}_2],\omega}_{\tau(j)}$ be the finishing time of $\tau(j)$ in the schedule $[\mathcal{S}_2]$. It can be proven that $f^{[\mathcal{S}_2],\omega}_{\tau(j)} \geq f^{[\mathcal{S}],\omega}_{\tau(j)}$.

Worst-Case Response Time: Therefore, the response time of the job $\tau(j)$ becomes $f^{[\mathcal{S}_2],\omega}_{\tau(j)} - t_0 \geq f^{[\mathcal{S}],\omega}_{\tau(j)} - r^{\omega}_{\tau(j)}$. This leads to the conclusion stated in Theorem 2.21. □

With the CIT, the WCRT analysis and the feasibility of the T-FP schedule can be validated by simulating the worst-case release pattern with each job executing its WCET. The CIT has been widely used in research results.

Time-Demand Analysis Based on the CIT, the Time-Demand Analysis (TDA), as proposed by Joseph and Pandya [JP86] and Lehoczky et al. [LSD89], provides exact analyses of the WCRT of tasks if they have at most one unfinished job at any time and if they are scheduled upon uniprocessor systems using T-FP scheduling. Specifically, given a task set $\mathbb{T}$ abstracted as

$$\tau \in \text{Spor}(T_\tau) \cap \text{WCET}(C_\tau) \tag{2.48}$$

with constrained deadlines $D_\tau \leq T_\tau$, and a preemptive T-FP scheduling algorithm $\mathcal{A}$ with a total priority ordering, i.e., no two tasks have the same priority. Then the *time-demand function* $W_\tau \colon \mathbb{R} \to \mathbb{R}$ of task $\tau \in \mathbb{T}$ is defined as

$$W_\tau(t) := C_\tau + \sum_{\tau' \in hp(\tau)} \left\lceil \frac{t}{T_\tau} \right\rceil \cdot C_{\tau'}, \tag{2.49}$$

where $hp(\tau)$ is the set of tasks in $\mathbb{T}$ with higher priority than τ. According to TDA, task τ is schedulable if all tasks with higher priority than τ are schedulable and there exists a $t \in (0, D_\tau]$ such that:

$$W_\tau(t) \leq t \tag{2.50}$$

Furthermore, if τ is schedulable, then any $t \in (0, D_\tau]$ fulfilling Equation (2.50) is an upper bound on the WCRT, i.e., $R^{\mathcal{A}}_\tau \leq \min\{t \in (0, D_k] \mid W_\tau(t) \leq t\}$. The minimal t to satisfy Equation (2.50) can be calculated using fixed-point iteration. To test the schedulability of $\mathbb{T}$ using TDA, we test the schedulability of each task individually, using Equation (2.50), in order of decreasing priorities. TDA forms an exact test for the sporadic abstraction, in the sense that if TDA fails to determine schedulability, then there exists a system evolution $\omega' \in \Omega^{abstr}$ such that $\mathcal{A}(\omega')$ is not feasible. In the case of arbitrary-deadline task systems in which tasks may have more than one unfinished job at each time (backlog), the TDA was extended to the busy-interval analysis by Lehoczky [Leh90].

Critical Instant in Probabilistic Scenarios While the critical instant is usually applied to ensure timeliness in a worst-case scenario (i.e., meeting the deadline under all circumstances), many embedded real-time systems can still function well even with occasional (bounded) deadline misses. Hence, providing quantification of deadline misses is of importance in practice. Specifically, safety standards such as IEC-61508 [Int10] and ISO-26262 [Int00] specify a (very) low failure probability but not necessarily a failure probability of zero. In contrast to the large body of research results regarding hard real-time systems, formal arguments and proofs for properties related to soft real-time systems with probabilistic, statistical, or deterministic guarantees remain an open research area even for simple settings.

In this regard, probabilistic timing analysis has been explored in the literature. For a probabilistic analysis, instead of having a scalar WCET C_τ of a task τ, the execution time of a job of τ is specified as a distribution of a random variable, as formalized in Section 2.4.1. The pET describes the impact of hardware effects (e.g., caches, branch prediction, pipelines, etc.) as well as different input values and paths taken in the software on the execution time. More variances and information can be found in the survey by Davis and Cucu-Grosjean [DC19].

The usage of the CIT for probabilistic timing analysis can be traced back to Tia et al. [Tia+95] in 1995 for a Probabilistic Time-Demand Analysis (PTDA). They assumed that the analyzed task set is released at the critical instant, which leads to "*an upper bound of the processor-time demand of j-th job of τ_i and all higher-priority tasks only when the deadline and maximum computation time of every task are less than its period*" [Tia+95]. In 1999, Gardner and Liu noted that the PTDA is only valid if the worst-case utilization of the task set is not more than 1, since the worst-case pattern of releases might not follow the critical instant (the concept was called an *in-phase busy-interval* in their work). They explored this aspect by a Stochastic Time-Demand Analysis (STDA) and simulated different phases. However, they did not find any cases where the probability that jobs miss their deadlines was higher for random phasing than for the simultaneous release under the critical instant.

In 2002, Diaz et al. [Día+02] noted that the worst-case scenario for a job in the first busy period following synchronous release (i.e., the classical CIT for periodic tasks) in PTDA [Tia+95] and STDA [GL99] may lead to an underestimation of the response-time distributions when the worst-case utilization of the task set exceeds 1. The main issue is the *backlog* at the end of each hyperperiod (i.e., the least common multiple of the periods of the periodic tasks). The analysis by Diaz et al. [Día+02], refined later by Lopéz et al. [Lóp+08] in 2008, considers the scenario when backlog may exist.

Backlog can be avoided by restarting the system when there is a deadline miss or by aborting a job when it misses its deadline. Under this assumption, in 2013, Maxim and Cucu-Grosjean [MC13] presented a probabilistic response-time analysis, proved the CIT from the probabilistic perspectives, and further extended the CIT to deal with tasks with inter-arrival times and relative deadlines also described by random variables. Since then, the CIT under the probabilistic setup by Maxim and Cucu-Grosjean [MC13] has been widely used in the literature [Max+17; Ren+15; Ren+19]. We revisit the probabilistic CIT [MC13] in Section 4.3, showing that their extension is flawed and discussing the impact on literature results.

Utilization Bounds While calculating bounds on the response time may be time-consuming, another approach is to test the schedulability of a task set $\mathbb{T}$ based on utilization values U_τ with $\tau \in \mathbb{T}$. Specifically, we consider a set of periodic or sporadic tasks with implicit deadlines, i.e., $\tau \in \text{Spor}(T_\tau)$ and $D_\tau = T_\tau$ for all $\tau \in \mathbb{T}$. Under RM, $\mathbb{T}$ is schedulable if $U_\mathbb{T} \leq \ln(2) \approx 0.693$, called the Liu and Layland bound [LL73]. In 2001, Bini et al. [BBB01] provided a tighter utilization bound. Specifically, they showed that $\mathbb{T}$ is schedulable if $\prod_{\tau \in \mathbb{T}}(U_\tau + 1) \leq 2$, called the hyperbolic bound. Under EDF, $\mathbb{T}$ is schedulable whenever $U_\mathbb{T} \leq 1$ as proven by Liu and Layland [LL73]. Therefore, this shows that EDF is optimal, in the sense that every task set that is schedulable can be feasibly scheduled by EDF. It is shown that EDF is even optimal for scheduling jobs with arbitrary deadlines [Uth04].

3 Focus of the Dissertation: Challenges of Distributed Real-Time Systems

With the trend towards distributed system architecture, new challenges arise for the real-time community. More specifically, to profit from a distributed system, workload must be distributed on different computing elements and data must be transmitted between those computing elements.

In this dissertation, we focus on two specific challenges.

Challenge 1: Distributed execution of jobs. A job is refined into smaller blocks which are distributed among different components. This requires more refined task models. However, extending classical results, like the Critical Instant Theorem (CIT), to the refined task models is non-trivial, as demonstrated by a series of wrong results in the literature. Of special interest for this dissertation is the **self-suspension** task model, which allows a job to pause its execution voluntarily, e.g., due to offloading certain part of its computation to another resource. Self-suspending tasks are further discussed in Section 5.

Challenge 2: Interplay of distributed tasks. If multiple tasks (on possibly different components) are involved to achieve a functionality, the interplay of these tasks has to be analyzed. That is, a sequence of tasks (a so-called cause-effect chain) is analyzed. Consequently, the timing requirement needs to be lifted from the task-level to the cause-effect-chain-level. Of special interest for this dissertation is the **end-to-end latency** of cause-effect chains which is further discussed in Section 6.

In Sections 3.1 and 3.2, we discuss the two abovementioned challenges. In particular, we introduce the self-suspension task model and cause-effect chains which are the two predominantly studied concepts of this dissertation. Subsequently, in Section 3.3, we review the state of the art for the analysis of self-suspending tasks and for the end-to-end latency of cause-effect chains.

3.1 Challenge 1: Distributed Execution of Jobs

To exploit the potential of distributed systems on the job-level, we partition the full job J in system evolution ω into finitely many blocks $\mathbb{B}_J^\omega$. In general, blocks of the same job are allowed to run on different components. Moreover, each block can have certain requirements, such as running on specific processing components or accessing a shared resource. Instead of an execution demand c_J^ω for the job J, each block $B \in \mathbb{B}_J^\omega$ has an execution demand c_B^ω.

Naturally, the system structure of a job may not allow to execute all blocks at the same time. Specifically, if a certain block $B' \in \mathbb{B}_J^\omega$ requires the result of another block $B \in \mathbb{B}_J^\omega$, then B must be finished before the execution of B' can be started. Such dependencies are modeled as *precedence constraints*. More specifically, there is a set of precedence constraints

$$\mathcal{P}_J^\omega \subseteq \mathbb{B}_J^\omega \times \mathbb{B}_J^\omega \tag{3.1}$$

on the blocks $\mathbb{B}_J^\omega$ of job J in system evolution ω. We assume that $(\mathbb{B}_J^\omega, \mathcal{P}_J^\omega)$ forms the structure of a Directed Acyclic Graph (DAG), i.e., $(\mathbb{B}_J^\omega, \mathcal{P}_J^\omega)$ is a directed graph without cycles. Due to their requirements, some blocks may need to wait to get access to, e.g., hardware accelerators (such as GPUs) or shared resources managed by a multiprocessor locking protocol. This may result in a *delay*

$$\delta_{(B,B')}^\omega \in \mathbb{R}_{\geq 0} \tag{3.2}$$

along a precedence constraint $(B, B') \in \mathcal{P}_J^\omega$.

When refining jobs into blocks, schedules can also be defined on the block-level. That is, we replace the job set $\mathbb{J}$ in the construction of the schedule by the set of all blocks $\mathbb{B}^\omega := \bigsqcup_{J \in \mathbb{J}} \mathbb{B}_J^\omega$. More specifically, every schedule[1] $[\mathcal{S}^M] = [\mathcal{S}_1 \times \cdots \times \mathcal{S}_M] \in \prod_{j=1}^M \mathbb{S}^{cont}(\mathbb{B}^\omega)$ of system evolution ω can be represented by a scheduling function

$$\mathcal{S}_1, \ldots, \mathcal{S}_M \colon \mathbb{R} \to \mathbb{B}^\omega \sqcup \{\perp\}, \tag{3.3}$$

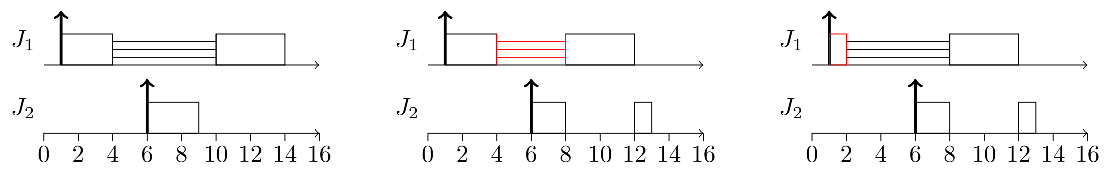
where M is the number of processors. The definitions of start $s_B^{[\mathcal{S}^M],\omega}$, finish $f_B^{[\mathcal{S}^M],\omega}$, execution $\mathrm{exec}_B^{[\mathcal{S}^M]}$ etc. are naturally extended by using blocks $B \in \mathbb{B}^\omega$ instead of jobs. We consider schedules which adhere to the precedence and delay constraints of the blocks. That is, if $(B, B') \in \mathcal{P}_J^\omega$, then $f_B^{[\mathcal{S}^M],\omega} + \delta_{(B,B')}^\omega \leq s_{B'}^{[\mathcal{S}^M],\omega}$ must hold.

Each scheduling function $\mathcal{S}_i$ of blocks gives rise to a scheduling function $\tilde{\mathcal{S}}_i$ of jobs

$$\begin{tikzcd} \mathbb{R} \arrow[r, "\mathcal{S}_i"] \arrow[rr, bend left, dashed, "\exists\ \tilde{\mathcal{S}}_i = proj \circ \mathcal{S}_i"] & \mathbb{B}^\omega \sqcup \{\perp\} \arrow[r, "proj"] & \mathbb{J} \sqcup \{\perp\} \end{tikzcd}\ , \tag{3.4}$$

where *proj* is the projection of blocks to their corresponding job. Due to the block structure, the corresponding job schedule $[\tilde{\mathcal{S}}^M] = [\tilde{S}_1 \times \cdots \times \tilde{S}_M]$ can show the following behavior:

[1]We focus on schedules in the continuous time domain for this dissertation. Similar constructions can be done for the discrete time domain.



(a) Full execution demand of all blocks and full delay. (b) Reduced delay. (c) Reduced execution demand.

Figure 3.1.: Timing anomaly: Reducing the delay/suspension or the execution demand of J_1 increases the response time of J_2.

Parallelism: The same job can be executed on different components simultaneously. This occurs if two blocks of the same job are executed at the same time in $[\mathcal{S}^M]$.

Non-Work-Conserving: Some processors may idle, although there are uncompleted jobs. This occurs due to the precedence constraints and delay between blocks.

Such effects can evoke *timing anomalies*, i.e., the reduction of execution demand or delay leads to an increased response time of other jobs. As an example we consider Figure 3.1. The example is composed of only two jobs, namely J_1 with two execution blocks $\mathbb{B}^\omega_{J_1} = \{B^1_1, B^1_2\}$, with $\mathcal{P}^\omega_{J_1} = \{(B^1_1, B^1_2)\}$, and J_2 with only one execution block $\mathbb{B}^\omega_{J_2} = \{B^2_1\}$, with $\mathcal{P}^\omega_{J_2} = \emptyset$ for all $\omega \in \Omega$. The execution demand of the blocks is specified as $c^\omega_{B^1_1} = 3$, $c^\omega_{B^1_2} = 4$, and $c^\omega_{B^2_1} = 3$, and the delay is specified as $\delta^\omega_{(B^1_1,B^1_2)} = 6$. Figure 3.1a depicts the schedule obtained when scheduling J_1 and J_2 with the Job-level Fixed-Priority (J-FP) scheduling algorithm when J_1 has higher priority than J_2. The response time of J_2 in that schedule is $9 - 6 = 3$. When considering another system evolution $\omega' \in \Omega$ where the delay $\delta^{\omega'}_{(B^1_1,B^1_2)}$ of (B^1_1, B^1_2) is reduced to 4 (in Figure 3.1b) or where the execution demand $c^{\omega'}_{B^1_1}$ of block B^1_1 is reduced to 1 (in Figure 3.1c), the response time of J_2 increases to $13 - 6 = 7$. Timing anomalies cause unintuitive behavior of the system and have led to several problems in the literature as further discussed in Chapter 5.

In the following, we consider different abstractions of the refined task model. To that end, we augment the space of system evolutions Ω to account for blocks, precedence constraints and delays, i.e.,

$$\Omega \ni \omega = \left((r^\omega_J)_{J\in\mathbb{J}}, \left((\mathbb{B}^\omega_J, \mathcal{P}^\omega_J), \left(\delta^\omega_{(B,B')}\right)_{(B,B')\in\mathcal{P}^\omega_J}, (c^\omega_B)_{B\in\mathbb{B}^\omega_J} \right)_{J\in\mathbb{J}} \right). \tag{3.5}$$

3.1.1 Event-Driven Delay-Induced Tasks

The Event-Driven Delay-Induced (EDD) task model was first proposed in 2021 by Aromolo et al. [Aro+21]. The task model assumes that all jobs of a task adhere to the same block structure and gives upper bounds on the block execution demand and on the delay of precedences.[2] We denote the common block structure by $\mathbb{B}_\tau$ with precendence constraints $\mathcal{P}_\tau$, and the upper bound on the delay by $(\Delta_{(B,B')})_{(B,B')\in\mathbb{B}_\tau}$. More specifically, we denote by

$$\tau \in \text{EDD}\left((\mathbb{B}_\tau, \mathcal{P}_\tau), \left(\Delta_{(B,B')}\right)_{(B,B')\in\mathcal{P}_\tau}, (C_B)_{B\in\mathbb{B}_\tau}\right) \tag{3.6}$$

the case that for all $j \in \mathbb{N}$ and for all system evolutions $\omega \in \Omega$ there exists a graph isomorphism[3] $f\colon (\mathbb{B}_\tau, \mathcal{P}_\tau) \xrightarrow{\sim} (\mathbb{B}^\omega_{\tau(j)}, \mathcal{P}^\omega_{\tau(j)})$ such that $c^\omega_{f(B)} \leq C_B$ for all $B \in \mathbb{B}_\tau$ and $\delta^\omega_{(f(B),f(B'))} \leq \Delta_{(B,B')}$ for all $(B, B') \in \mathcal{P}_\tau$. While this task model is very general, there have been no further analytical results beyond the approaches of the initial paper [Aro+21], where they propose a formulation of the scheduling problem as Mixed-Integer Linear Programming (MILP) and a reduction to the schedulability problem of dynamic self-suspending tasks (to be introduced in Section 3.1.3).

In the following two sections, we discuss DAG tasks and self-suspending tasks, which are two extreme cases of the EDD task model. That is, the DAG task model assumes that the delay between blocks is set to 0, while the self-suspending task model assumes that the block structure is limited to sequences.

3.1.2 Directed Acyclic Graph Tasks

The Directed Acyclic Graph (DAG) task model has its focus on the DAG structure of the blocks. That is, a common block structure $(\mathbb{B}_\tau, \mathcal{P}_\tau)$ is abstracted for all jobs of the same task, and the delay along the precedence constraints is assumed to be 0. More specifically, we denote by

$$\tau \in \text{DAG}((\mathbb{B}_\tau, \mathcal{P}_\tau), (C_B)_{B\in\mathbb{B}_\tau}) \tag{3.7}$$

the case that for all $j \in \mathbb{N}$ and for all system evolutions $\omega \in \Omega$ there exists graph isomorphism $f\colon (\mathbb{B}_\tau, \mathcal{P}_\tau) \xrightarrow{\sim} (\mathbb{B}^\omega_{\tau(j)}, \mathcal{P}^\omega_{\tau(j)})$ such that $c^\omega_{f(B)} \leq C_B$ for all $B \in \mathbb{B}_\tau$ and $\delta^\omega_{(f(B),f(B'))} = 0$ for all $(B, B') \in \mathcal{P}_\tau$. The DAG task model is a special case of the EDD task model, in the sense that

$$\text{DAG}((\mathbb{B}_\tau, \mathcal{P}_\tau), (C_B)_{B\in\mathbb{B}_\tau}) = \text{EDD}\left((\mathbb{B}_\tau, \mathcal{P}_\tau), \left(\Delta_{(B,B')} = 0\right)_{(B,B')\in\mathcal{P}_\tau}, (C_B)_{B\in\mathbb{B}_\tau}\right). \tag{3.8}$$

Therefore, the EDD task model can be considered as an *abstraction refinement* [Brü+22] of the DAG task model.

The DAG task model is well studied in the literature for both priority-based global scheduling [Bon+13; CA14; FNN17; NNB19] and partitioned scheduling [BCM19; Cas+18; Fon+16]. Besides that, different scheduling approaches have been developed to improve the performance, e.g., federated scheduling [Li+14], and hierarchical scheduling

[2]The EDD model in the work by Aromolo et al. [Aro+21] also provides lower bounds on the delay. We omit those lower bounds in the discussion of the model for simplicity.

[3]A graph isomorphism $f\colon (\mathbb{B}_\tau, \mathcal{P}_\tau) \xrightarrow{\sim} (\mathbb{B}^\omega_{\tau(j)}, \mathcal{P}^\omega_{\tau(j)})$ is a bijective function $f\colon \mathbb{B}_\tau \xrightarrow{\sim} \mathbb{B}^\omega_{\tau(j)}$ that preserves the graph structure, i.e., $f \times f\colon \mathcal{P}_\tau \xrightarrow{\sim} \mathcal{P}^\omega_{\tau(j)}$ is a bijection.

approaches [BBW10; UGC21]. A classical bound which applies to an exclusively scheduled DAG task using any work-conserving scheduling algorithms is proposed by Graham [Gra69], using only the total workload and the longest path of the DAG. The bound has been further tightened [He+22] by considering several paths in the DAG.

3.1.3 Self-Suspending Tasks

While the focus of Section 3.1.2 is on the DAG structure, for self-suspending tasks the focus is on the delay between blocks. Therefore, for self-suspending tasks we assume that all jobs adhere to an elementary block structure. That is, we assume that the directed graph $(\mathbb{B}^\omega_J, \mathcal{P}^\omega_J)$ is a sequence, i.e., isomorphic to $(1 \to 2 \to \cdots \to m)$ for some m. We denote such a graph isomorphism as $(\mathbb{B}^\omega_{\tau(j)}, \mathcal{P}^\omega_{\tau(j)}) \simeq (1 \to 2 \to \cdots \to m)$. When considering the corresponding job schedule, every delay appears as *self-suspension*. That is, the job *voluntarily* releases its processor and pauses its execution despite being incomplete. The two predominantly studied self-suspension task models are the segmented self-suspension task model and the dynamic self-suspension task model.

Segmented Self-Suspension Task Model For segmented self-suspending tasks, it is assumed that all jobs of a task adhere to the same sequence structure. That is, the number of blocks m is fixed on the task-level, and therefore denoted by m_τ. Hence, for the segmented self-suspending task model, $(\mathbb{B}^\omega_{\tau(j)}, \mathcal{P}^\omega_{\tau(j)}) \simeq (1 \to 2 \to \cdots \to m_\tau)$ for all $\omega \in \Omega$ and $j \in \mathbb{N}$. This partitions the task into *execution segments* which are the vertices $\{1, \ldots, m_\tau\}$ and *suspension segments* which are the precedence constraints $\{(i, i+1) \,|\, i \in \{1, \ldots, m_\tau - 1\}\}$. For each execution segment i, the segmented self-suspension task model provides an upper bound C^i_τ. Moreover, for each suspension segment $(i, i+1)$, the model provides an upper bound on the delay, i.e., $\delta^\omega_{(i,i+1)} \leq S^i_\tau$. We denote by

$$\tau \in \mathrm{SegSS}(C^1_\tau, S^1_\tau, C^2_\tau, \ldots, S^{m_\tau - 1}_\tau, C^{m_\tau}_\tau) \tag{3.9}$$

the case that for all $\omega \in \Omega$ and all $j \in \mathbb{N}$ there exists a graph isomorphism

$$f\colon (1 \to 2 \to \cdots \to m_\tau) \xrightarrow{\sim} (\mathbb{B}^\omega_{\tau(j)}, \mathcal{P}^\omega_{\tau(j)}) \tag{3.10}$$

such that $c^\omega_{f(i)} \leq C^i_\tau$ holds for all $i \in \{1, \ldots, m_\tau\}$ and $\delta^\omega_{(f(i), f(i+1))} \leq S^i_\tau$ holds for all $i \in \{1, \ldots, m_\tau - 1\}$. The segmented self-suspension model is a special case of the EDD task model, in the sense that

$$\mathrm{SegSS}(C^1_\tau, S^1_\tau, \ldots, C^{m_\tau}_\tau) = \mathrm{EDD}\left((1 \to \cdots \to m_\tau), \left(S^i_\tau\right)_{(i,i+1)}, \left(C^i_\tau\right)_i\right). \tag{3.11}$$

Therefore, the EDD task model can be considered as an *abstraction refinement* [Brü+22] of the segmented self-suspension model.

The segmented self-suspension model is suitable when the execution of each instance of a task always follows a specific structure. Prime examples are situations where a significant part of the computation is offloaded to hardware accelerators, which often can be modeled as a three-step process: (i) a local computation segment preparing the offloaded data, e.g., data compression, (ii) computation on the accelerator, i.e., the time the task instance is suspended, and (iii) a local computation segment for post-processing. However, such a clear structure of a task's self-suspension behavior cannot always be assumed. For instance, it has been reported that suspension-aware analysis can be applied to handle suspension-based locks for multiprocessor synchronization due to lock contention, usually termed as *remote blocking*. Specifically, when partitioned or semi-partitioned multiprocessor scheduling paradigms are applied, the problem can be modeled as self-suspending tasks in uniprocessor systems.[4] Depending on different execution paths/conditions, a task may access different mutually exclusive shared resources, and suspend when waiting for remote resource access. Therefore, for a specific task, the access pattern to the shared resources may differ for different task instances, i.e., the number of resource accesses, their order, their location, and which resources are accessed changes for individual task instances. In such cases, the system evolution shows a more dynamic block structure of individual jobs, and the dynamic self-suspension model is more suitable.

Dynamic Self-Suspension Task Model For dynamic self-suspending tasks, each job can have a different number of blocks, i.e., m is *not* a fixed task parameter. Rather, upper bounds for the *total* execution time C_τ of jobs of τ, and for the *total* suspension time S_τ that jobs of τ can experience, are provided. More specifically, we denote by

$$\tau \in \mathrm{DynSS}(C_\tau, S_\tau) \tag{3.12}$$

the case that for all $j \in \mathbb{N}$ and all system evolutions $\omega \in \Omega$ there exists an $m \in \mathbb{N}$ such that $(\mathbb{B}^\omega_{\tau(j)}, \mathcal{P}^\omega_{\tau(j)}) \simeq (1 \to 2 \to \cdots \to m)$, $\sum_{B\in\mathbb{B}^\omega_{\tau(j)}} c^\omega_B \leq C_\tau$ and $\sum_{(B,B')\in\mathcal{P}^\omega_{\tau(j)}} \delta^\omega_{(B,B')} \leq S_\tau$.

The dynamic self-suspension model is a *behavior relaxation* [Brü+22] of the segmented self-suspension model, in the sense that

$$\mathrm{SegSS}(C^1_\tau, S^1_\tau, C^2_\tau, \dots, S^{m_\tau-1}_\tau, C^{m_\tau}_\tau) \subseteq \mathrm{DynSS}\left(\sum_{i=1}^{m_\tau} C^i_\tau, \sum_{i=1}^{m_\tau-1} S^i_\tau\right). \tag{3.13}$$

However, the dynamic self-suspension model is different from the EDD task model in the sense that neither of them generalizes the other.

3.2 Challenge 2: Interplay of Distributed Tasks

To perform complex functionalities in a distributed manner, they can also be divided into several tasks, cooperating to achieve the functionalities. For example, considering modern automotive systems, functionalities such as driving assistance, electronic braking systems, and even door-locking are decomposed into tasks which are executed on the Electronic

[4]For details, please refer to [Bra19; Che+19b].

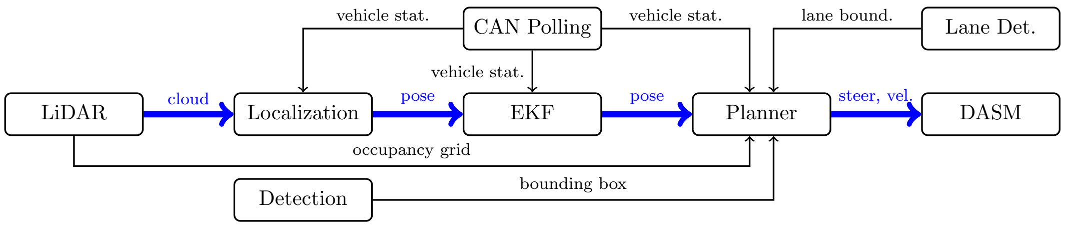


Figure 3.2.: An example application, redrawn and simplified from the WATERS challenge 2019 [Ham+19]. An exemplary chain in the application is emphasized by the bold blue arrows.

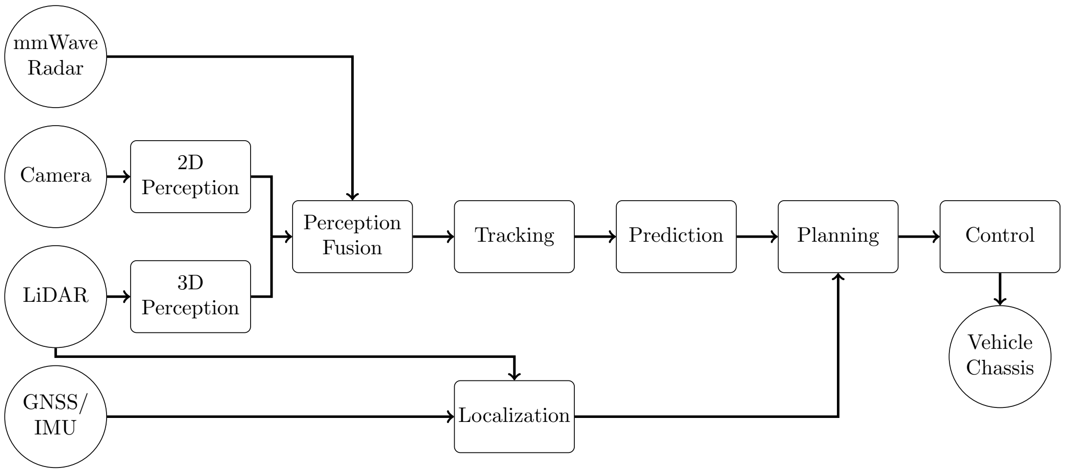


Figure 3.3.: Processing Graph of an autonomous driving system, redrawn from the RTSS 2021 Industry Challenge [Per21].

Control Units (ECUs)[5] of the automotive architecture. To demonstrate this, we consider Figure 3.2 from the WATERS 2019 challenge and Figure 3.3 from the RTSS 2021 Industry Challenge. Both illustrate the decomposition of an autonomous driving system into several tasks.

To allow the interplay of tasks, data must be transferred between them. Data dependencies, depicted by arrows in Figures 3.2 and 3.3, form the structure of a directed graph. However, the requirements on the end-to-end latency and therefore the analytical examination is usually limited to specific paths in the directed graph. Such a path in the data dependency graph is called a *cause-effect chain*, denoted by $E = (\tau_1 \to \cdots \to \tau_n)$. In Figure 3.2, a cause-effect chain is highlighted by bold blue arrows. More specifically, the cause-effect chain consists of a LiDAR grabber task, a Localization task, a sensor fusion task using an Extended Kalman Filter (EKF), a trajectory Planner task, and a control

[5] In this dissertation, we use the notion of ECU when we examine the interplay of distributed tasks, complying with the historical develop from the automotive domain. This is relevant to Section 3.2 and Chapter 6. When examining classical or self-suspending tasks, we use the more traditional notion of processor. However, the results of this dissertation are not limited to these specific computing units.

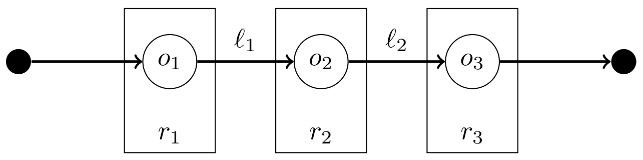


Figure 3.4.: Data flow semantic, redrawn from [Dav+07].

task (DASM) that provides the actuators with velocity and steering signals. In this example, every job of the LiDAR grabber task reads information from the LiDAR sensor, builds a point cloud, and shares that point cloud data with a job of the Localization task. Similarly, the computed data from the trajectory Planner job is shared with a job of the control task (DASM). Please note that while Figures 3.2 and 3.3 show a simplified task structure, in real applications the structure can be comprised of hundreds of tasks.

In contrast to Chapter 2, functionalities are not described by a single task but are achieved by several cooperating tasks. Hence, this poses the challenge:

> How can the timing requirements of functionalities be specified that are achieved by a set of interacting tasks?

That is, the timing requirements must be lifted from a single task to a set of tasks. To achieve this, the maximum time for the data flow between the **cause** (external activity) and the **effect** (actuation based on the external activity) is considered. Typical metrics for such end-to-end timing are:

- Maximum Reaction Time (MRT): How fast can the system react in the worst case? (This metric is also called the maximum *button-to-action delay*.)

- Maximum Data Age (MDA): How old is the data used in an actuation in the worst case? (This metric is also called the *data freshness*.)

The seminal work by Davare et al. [Dav+07] explored such end-to-end latency and optimization of *cause-effect chains* in 2007. Feiertag et al. [Fei+09] iterated upon the work of Davare et al. [Dav+07] in 2009. AUTOSAR Timing Extensions [AUT22] represent such cause-effect chains of more general functional dependency. In the following, we formally introduce cause-effect chains and the end-to-end latency.

3.2.1 Cause-Effect Chains

In general, a *cause-effect chain* is a sequence of objects that need to communicate to achieve a given functionality. Cause-effect chains are utilized to determine the flow of data within a system. One of the first models to describe data flow and the end-to-end timing behavior was provided by Davare et al. [Dav+07] in 2007. In their model, depicted in Figure 3.4, objects $\mathcal{O}$ are allocated to resources $\mathcal{R}$. A set of links $\mathcal{L}$ that each connect two objects describe the *data flow* between objects. A *path* is a finite sequence of objects

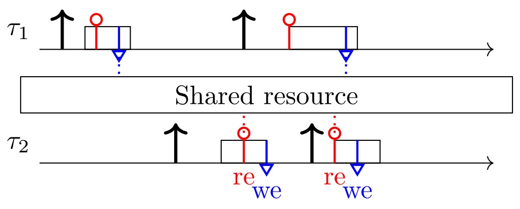


Figure 3.5.: Communication between tasks.

with links between them. Nowadays, the data flow is modeled by a cause-effect chain, i.e., a sequence of tasks

$$E = (\tau_1 \to \cdots \to \tau_n). \tag{3.14}$$

Beyond that, modern modelling approaches describe the underlying communication mechanisms more precisely. In this section, we formally introduce cause-effect chains. To that end, we first specify the communication semantics.

Communication Jobs communicate by receiving (reading) their input data from a shared resource and handing over (writing) their output data to a shared resource. Jobs of the same task write to the same resource and messages are overwritten; that is, at any time only the latest output of a task is available. Given a scheduling algorithm $\mathcal{A}$, we denote the time of the *read-event* of a job J in system evolution $\omega \in \Omega$ as $\mathrm{re}^\omega(J) \in \mathbb{R}$ and the time of the *write-event* of J in ω as $\mathrm{we}^\omega(J) \in \mathbb{R}$. The general communication semantics are depicted in Figure 3.5. We consider communication given the following two requirements:

(R1) **Proper ordering:** For each task $\tau \in \mathbb{T}$, each job $\tau(m)$ reads before it writes, i.e., $\mathrm{re}^\omega(\tau(m)) \leq \mathrm{we}^\omega(\tau(m))$ for all $m \in \mathbb{N}$. Moreover, for any two consecutive jobs the read- and write-events are ordered according to their index, that is, $\mathrm{re}^\omega(\tau(m)) < \mathrm{re}^\omega(\tau(m+1))$ and $\mathrm{we}^\omega(\tau(m)) < \mathrm{we}^\omega(\tau(m+1))$ for all $\tau \in \mathbb{T}$ and $m \in \mathbb{N}$.

(R2) **Proper distribution:** The number of read- and write-events in each bounded time interval is finite. More specifically, the two sets $\{\mathrm{re}^\omega(\tau(m)) \mid m \in \mathbb{N}\}$ and $\{\mathrm{we}^\omega(\tau(m)) \mid m \in \mathbb{N}\}$ have no accumulation point.

These two not very restrictive requirements are fulfilled by commonly considered communication semantics, e.g., for *explicit communication*, *implicit communication*, and *Logical Execution Time (LET)*, specified in the current AUTomotive Open System ARchitecture (AUTOSAR) standard [AUT22]. Implicit communication assumes that data is always read and written at the beginning and the end, respectively, of a recurrent task instance's execution. Contrarily, the explicit communication model assumes that read and write operation can happen at any time during execution, e.g., as soon as the data is produced. However, this direct, unrestricted access of explicit communication easily results in data consistency issues. Therefore, implicit communication strategies

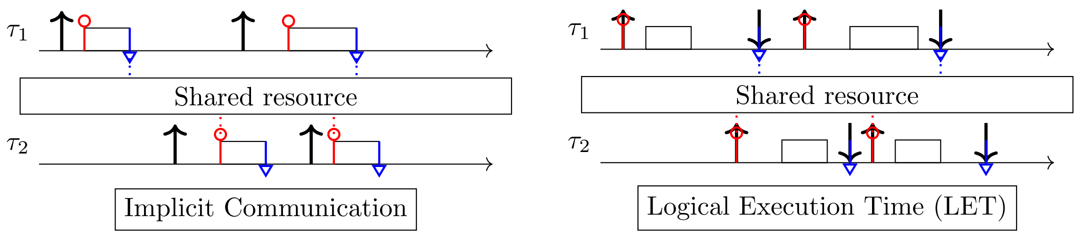


Figure 3.6.: Implicit communication (left) and LET communication (right).

are usually applied, as they guarantee the required data consistency [Ham+17]. The Logical Execution Time (LET) model, as summarized by Kirsch and Sokolova [KS12], has been introduced with the time-triggered programming language Giotto and adopted by the automotive industry as a communication mechanism that reduces jitter and improves determinism even further. LET abstracts the communication from the actual timing behavior of real-time tasks on the physical platform, i.e., the communication is independent of the actual schedule of tasks instances (called jobs). Instead, it is ensured that reads (respectively, writes) happen before (respectively, after) a job starts (respectively, finishes). This introduces a separation between functionality on the one hand and scheduling on the other hand. As reported by Ernst et al. [Ern+18], several Original Equipment Manufacturers (OEMs) and suppliers in the automotive industry have come to the conclusion that LET might be a key enabler for the design and verification of multicore systems because it allows predictability of those complex systems. Please note that "*LET [...] leads to longer latencies in event chains*" ([Ham+17]) to enable determinism. In conclusion, the following communication semantics are predominantly studied:

- **Explicit Communication:** Under explicit communication, the read-event and the write-event can occur anywhere during job execution. However, the job cannot write new data before its read-event. An example is depicted in Figure 3.5.
- **Implicit Communication:** Under implicit communication, each job's read-event is when the job starts, i.e., $\text{re}^\omega(J) = s_J^{\mathcal{A}(\omega),\omega}$, and the job's write-event is when the job finishes, i.e., $\text{we}^\omega(J) = f_J^{\mathcal{A}(\omega),\omega}$. Implicit communication is shown on the left-hand side of Figure 3.6.
- **Logical Execution Time (LET):** Under LET [KS12], each task τ is assigned an arbitrary relative deadline D_τ. The read-event of each job J of τ is set to its release time $\text{re}^\omega(J) = r_J^\omega$, and the write-event is set to its absolute deadline $\text{we}^\omega(J) = r_J^\omega + D_\tau$. Communication under LET is depicted on the right-hand side of Figure 3.6. Although LET is originally limited to systems with only a single ECU, Ernst et al. [EAG18] provide a generalization to the case with several ECUs.

We note that, for explicit communication, the system evolution ω might have to be augmented to indicate the time of read-events and write-events within the execution interval of a job. However, for implicit communication and LET the read-event and write-event can be determined by the release time and execution demand which is already encoded in the system evolution ω. Note that implicit communication and LET provide a trade-off: While implicit communication leads to shorter latencies, LET provides timing determinism [Ham+17]. In this dissertation, the focus is on implicit communication and LET, and therefore we do not augment read- and write-events in the system evolution.

Cause-effect chains. On the task-level, we model the data flow in a system by a cause-effect chain. A *cause-effect chain* $E = (\tau_1 \to \tau_2 \to \cdots \to \tau_n)$, with $n \in \mathbb{N}$, describes the data path through different programs as a finite sequence of tasks $\tau_i \in \mathbb{T}$. This definition is inspired by the *event-chains* specified in the AUTOSAR Timing Extensions [AUT22], which represent chains of more general functional dependency.

We assume that the sampling for a cause-effect chain E happens at the read-event of each job of τ_1 (i.e., implicit sampling) for convenience. Alternatively, any kind of sampling can be modeled by adding a *sampling task* τ^{sample}, where each job $\tau^{sample}(j)$ with $j \in \mathbb{N}$ reads and writes data at a time when the sampling happens. In most systems, the data dependencies are more complex and can be described by a directed graph with several sources and sinks. In this case, each cause-effect chain (i.e., path through the directed graph from a source to a sink) can be analyzed individually.

A cause-effect chain can be comprised of time-triggered and event-triggered tasks. Time-triggered tasks release jobs at predefined timestamps. Usually, the timestamps are determined using a timer with a fixed period. In such a case, the task can be abstracted as periodic task. As a generalization, a time-triggered task can also be described as sporadic task if the timestamps to not occur strictly periodic. On the other hand, event-triggered tasks release jobs when certain events occur. Most commonly, event-triggered tasks release a job whenever they receive data from another task. However, arbitrary events to trigger a job release (even outside the system model) would be possible as well. Typical applications using for example Robot Operating System 2 (ROS2) [Ope22] allow for both time-triggered and event-triggered tasks. Both triggering mechanisms result in different challenges for the analysis. While some results of this dissertation are valid for both time-triggered and event-triggered tasks, most analytical bounds are only designed for cause-effect chains comprised of time-triggered tasks.

A cause-effect chain considers the implemented functionality and describes the data path through different tasks in the systems that implements this functionality. Yet, to determine end-to-end latency, it is not sufficient to know which tasks are part of the chain. Instead, we need to consider the specific path for one data token through the task instances. We describe this specific data path as a *job chain*.

Definition 3.1 (Job chain). *For a cause-effect chain $E = (\tau_1 \to \cdots \to \tau_n)$ of the task set $\mathbb{T}$, a job chain $c = (J_1, \ldots, J_n)$ for E in system evolution ω is a sequence of jobs fulfilling the following two conditions:*

- *Job J_i is a job of task τ_i for all $i \in \{1, 2, \ldots, n\}$.*

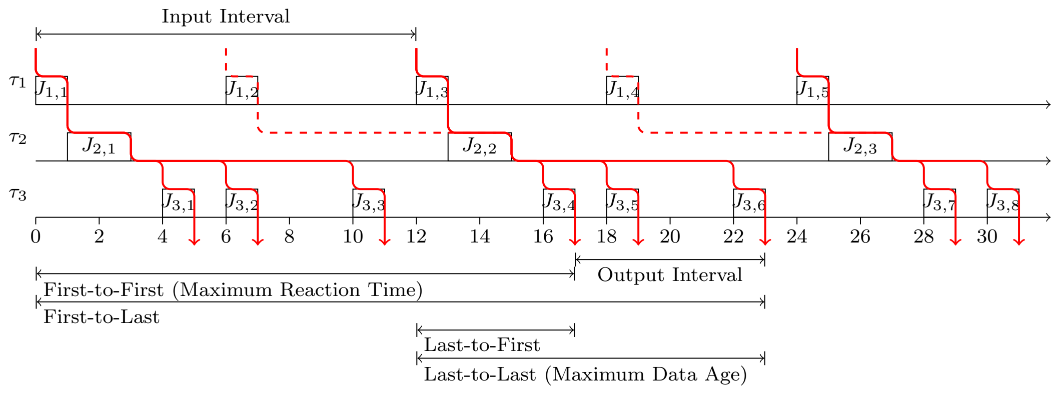


Figure 3.7.: End-to-end latency semantics, as formulated in [Fei+09].

- *Each job in the chain reads the data at the time or after it was written by the previous job in the chain. That is,* $\mathrm{we}^\omega(J_{i-1}) \leq \mathrm{re}^\omega(J_i)$ *for all* $i \in \{2, 3, \dots, n\}$*.*

Note that according to this definition one job can be part of multiple job chains. For instance, in Figure 3.7, job $J_{2,3}$ is part of the job chains $(J_{1,4}, J_{2,3}, J_{3,7})$, $(J_{1,4}, J_{2,3}, J_{3,8})$, $(J_{1,5}, J_{2,3}, J_{3,7})$, and $(J_{1,5}, J_{2,3}, J_{3,8})$. The *length* $\ell^\omega(c)$ *of a job chain* $c = (J_1, J_2, \dots, J_n)$ is the length of the time interval between the read-event of J_1 and the write-event of J_n (i.e., between the read-event of the first job in the chain and the write-event of the last job in a chain):

$$\ell^\omega(J_1, J_2, \dots, J_n) = \mathrm{we}^\omega(J_n) - \mathrm{re}^\omega(J_1). \tag{3.15}$$

3.2.2 End-to-End Latency

In this section, we introduce the notion of end-to-end latency, which describes the maximal time of a data flow in a cause-effect chain $E = (\tau_1 \to \cdots \to \tau_n)$. Specifically, we restate different timing metrics for end-to-end latency from the literature, and from our own work (cf. [Gün+23b]). Analytical results for bounding those metrics are discussed in Section 6.2.

Whereas classical response-time analysis focuses on the time between the release of a job and its completion for individual tasks, for end-to-end analysis, the focus is on the latency of the data propagation along several tasks. While some analytical results for the end-to-end latency may exploit bounds on the response time of tasks, the challenges for end-to-end latency analysis are inherently different from response-time analysis. For instance, end-to-end analysis consider over- and undersampling. That is, data may be overwritten (oversampling) or utilized by several jobs (undersampling). The problem becomes even more complex if additional communication delays need to be considered or the system consists of time-triggered as well as event-triggered tasks.

One of the first discussions of end-to-end latency was by Davare et al. [Dav+07] in 2007. They analyzed the data flow from a source node (left node in Figure 3.4) to a sink node (right node in Figure 3.4). Feiertag et al. [Fei+09] extended the work of Davare et al. [Dav+07] and refined the notion of data flow analysis. Their model focuses on the impact of over- and undersampling. They identify the MRT and the MDA as the two key metrics for end-to-end analysis.

Figure 3.7 illustrates the end-to-end latency model from Feiertag et al. [Fei+09]. For each job J of τ_n, a *source job* of τ_1 can be determined. That is, the job J' of τ_1 such that a data path (formally, a job chain) from J' to J exists without being overwritten. In Figure 3.7, the source job of $J_{3,1}$, $J_{3,2}$ and $J_{3,3}$ is $J_{1,1}$, and the source job of $J_{3,4}$, $J_{3,5}$ and $J_{3,6}$ is $J_{1,3}$. Since $J_{3,4}$, $J_{3,5}$ and $J_{3,6}$ all have the same source job, they all produce an output based on the same data. The interval that encompasses the write-events of jobs with the same source job is called the *output interval*. In Figure 3.7, the output interval is the interval $[17, 23]$.

Feiertag et al. [Fei+09] further defined the *input interval* as the interval from the read-event of the previous source job to the read-event of the current source job. Any input that can be captured by the system during this interval will be used by the system to create an output during the output interval. In Figure 3.7, the input interval for the source job $J_{1,3}$ is $[0, 12]$.

Based on these input and output intervals, Feiertag et al. [Fei+09] defined different metrics for the end-to-end latency, namely First-to-First, First-to-Last, Last-to-First, and Last-to-Last, as depicted in Figure 3.7.[6] Two key metrics are emphasized:

- **Maximum Reaction Time (MRT):** The maximum time between the beginning of the input interval and the beginning of the output interval for any source job. This is referred to as the *First-to-First* latency in [Fei+09].

- **Maximum Data Age (MDA):** The maximum time between the end of the input interval and the end of the output interval for any source job. This is referred to as the *Last-to-Last* latency in [Fei+09].

However, there is an imbalance in the definition of the input and output intervals by Feiertag et al. [Fei+09]. Specifically, the input interval is formulated as a *passive* arrival of data to the system, whereas the output interval is formulated as an *active* generation of data by the system. To solve this imbalance, we also need to consider the interval where output data that is (passively) provided by an actuator can be retrieved from the system. That is, we need to extend the output interval until the write-event that originates from different input data. When applying our solution to the scenario in Figure 3.7, the output interval is extended from $[17, 23]$ to $[17, 29]$.

In the following, we denote the output interval in the work from Feiertag et al. [Fei+09] the *reduced output interval* to distinguish from our proposed output interval. The new definitions are depicted in Figure 3.8.

Whereas MDA uses the extended output interval, we denote by MRDA the Maximum Reduced Data Age, defined in terms of the reduced output interval. For symmetry, we also define a *reduced input interval*, that is the interval of all read-events during the

[6] Some authors also refer to these latencies as First In – First Out (FIFO), First In – Last Out (FILO), Last In – First Out (LIFO), and Last In – Last Out (LILO) [MMS12a; MMS12b].

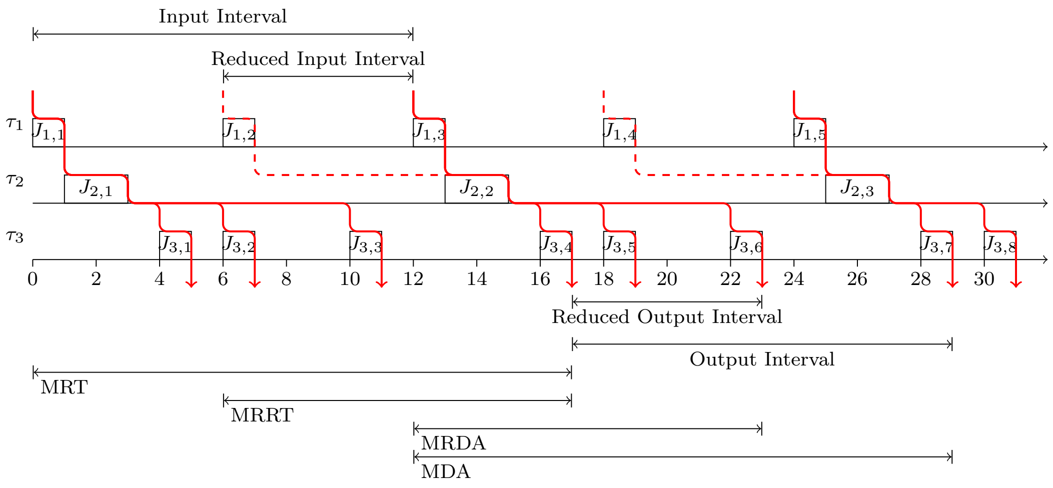


Figure 3.8.: Updated end-to-end latency definitions without imbalance in the input and output intervals.

input interval. In Figure 3.8, the reduced input interval is $[6, 12]$. Similarly, we define the Maximum Reduced Reaction Time (MRRT). Our definition removes the imbalance between the input interval and output interval, and allows the differentiation between the active and passive input and output of data. The differentiation between the MDA and MRDA, following the suggestion from [Gün+23b], can be summarized as:

- We denote the Maximum Reduced Data Age (MRDA) as the Last-to-Last latency from Feiertag et al. [Fei+09].
- We denote the Maximum Data Age (MDA) as the Last-to-Last latency with our extended output interval, covering all the time points until the write-event of a job of the last task in the cause-effect chain that originates from different input data.

Dürr et al. [Dür+19] provided an alternative definition for the MRT and MDA of sporadic tasks under implicit communication using *immediate forward* and *immediate backward* job chains. We reformulate it in terms of more general tasks with read- and write-events, similar to [Gün+23b].

Definition 3.2 (Immediate forward job chain)**.** *A job chain $c = (J_1, J_2, \ldots, J_n)$ for $E = (\tau_1 \to \cdots \to \tau_n)$ in $\omega \in \Omega$ is* immediate forward *if for all $i \in \{2, \ldots, n\}$ the job J_i is the* earliest *job of task τ_i with read-event no earlier than the write-event of J_{i-1}. That is, J_i is the earliest job that fulfills the properties from Definition 3.1.*

Definition 3.3 (Immediate backward job chain)**.** *A job chain $c = (J_1, J_2, \ldots, J_n)$ for $E = (\tau_1 \to \cdots \to \tau_n)$ in $\omega \in \Omega$ is* immediate backward *if for all $i \in \{n-1, \ldots, 1\}$ the job J_i is the* latest *job of task τ_i with write-event no later than the read-event of J_{i+1}. That is, J_i is the latest job that fulfills the properties from Definition 3.1.*

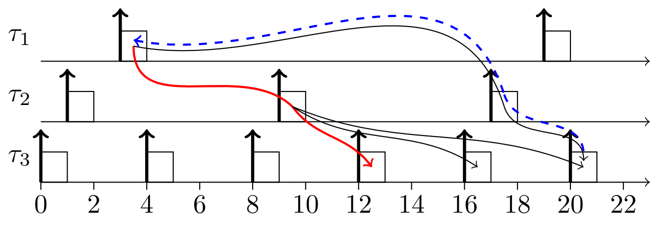


Figure 3.9.: Cause-effect chain $E = (\tau_1 \to \tau_2 \to \tau_3)$ for three tasks with implicit communication. Forward arrows mark job chains with first job $\tau_1(1)$, the *immediate forward* job chain is marked red. The dashed blue arrow marks the *immediate backward* job chain with last entry $\tau_3(6)$.

By construction, immediate forward job chains track the earliest transfer of data. Hence, they can be used to measure the time between the read-event in a reduced input interval and the first write-event in the corresponding output interval. For example, in Figure 3.8, $(J_{1,2}, J_{2,2}, J_{3,4})$ and $(J_{1,3}, J_{2,2}, J_{3,4})$ are both immediate forward job chains. Furthermore, for an example system with three tasks and implicit communication, the forward arrows in Figure 3.9 mark all job chains starting at job $\tau_1(1)$. The immediate forward job chain starting at $\tau_1(1)$ is marked red. The immediate backward job chain ending at $\tau_3(6)$ is marked with a dashed blue backwards arrow.

Comparing all immediate forward job chains yields the MRRT:

$$\sup\{\ell^\omega(c) \mid c \text{ immediate forward job chain for } E \text{ in } \omega\} \tag{3.16}$$

Similarly, immediate backward job chains track the most recent origin of data. Hence, they can be used to measure the time between the source job and a write-event in the reduced output interval. The MRDA is:

$$\sup\{\ell^\omega(c) \mid c \text{ immediate backward job chain for } E \text{ in } \omega\} \tag{3.17}$$

To cover the full time interval of the input and output interval, we extend the definition of the immediate forward and immediate backward job chains as presented in [Gün+23a; Gün+23d]. We refer to them as the *augmented* immediate forward and immediate backward job chains, respectively.

Definition 3.4 (Immediate forward augmented job chain)**.** *Let $z \in \mathbb{R}$ be a time point and $E = (\tau_1 \to \dots \to \tau_n)$ a cause-effect chain. We define the immediate forward augmented job chain for E in system evolution $\omega \in \Omega$ at z by*

$$\vec{ac}^{\omega}_{E}(z) = (z, J_1, \dots, J_n, z'), \tag{3.18}$$

where J_1 is the job of τ_1 with the earliest read-event $\mathrm{re}^\omega(J_1) \geq z$, the sequence $(J_1, \dots, J_n)$ is an immediate forward job chain for E in ω, and z' is at the write-event of J_n.

Definition 3.5 (Immediate backward augmented job chain)**.** *Let $z' \in \mathbb{R}$ be a time point and $E = (\tau_1 \to \dots \to \tau_n)$ a cause-effect chain. We define the immediate backward augmented job chain for E in system evolution $\omega \in \Omega$ at z' by*

$$\overleftarrow{ac}_E^\omega(z') = (z, J_1, \dots, J_n, z'), \tag{3.19}$$

where J_n is the job of τ_n with the latest write-event $\text{we}^\omega(J_n) \leq z'$, the sequence $(J_1, \dots, J_n)$ is an immediate backward job chain for E in ω, and z is at the read-event of J_1.

The length of an immediate forward or immediate backward augmented job chain $(z, J_1, \dots, J_n, z')$ is defined as $\ell(z, J_1, \dots, J_n, z') = z' - z$. Due to their construction, immediate forward augmented job chains can be used to measure the time between the beginning of the input and the beginning of the output interval, i.e., the MRT. Similarly, immediate backward augmented job chains can be used to measure the time between the end of the input and the end of the output interval, i.e., the MDA. Hence, the MRT is $\sup_z \ell(\overrightarrow{ac}_E^\omega(z))$, and the MDA is $\sup_{z'} \ell(\overleftarrow{ac}_E^\omega(z'))$.

However, these definitions of MRT and MDA are not complete yet, because they lack a specification of the beginning of time measurements. Specifically, if z' is chosen too small, then there is no data in the system that can be analyzed. Similarly, if z is chosen too small, the MRT approaches infinity. In fact, most applications are concerned with the system behavior only when the system is properly *warmed up*. That is, some initial data paths that do not process any relevant data should be left out. The first jobs that process data relevant to the output are those that are part of the first immediate backward job chain (that fully exists). We say that a task is warmed up as soon as the job that is part of the first immediate backward job chain is reached.

Definition 3.6 (Warm-up)**.** *Let $E = (\tau_1, \dots, \tau_n)$ be a cause-effect chain and $\omega \in \Omega$ a system evolution. Let $W_{E,\tau_i}^\omega \in \mathbb{N}$ for $i = 1, \dots, n$, such that $c = (\tau_1(W_{E,\tau_1}^\omega), \dots, \tau_n(W_{E,\tau_n}^\omega))$ is the* first *immediate backward job chain for E in ω that exists. We say that task τ_i is* warmed up *at job $\tau_i(W_{E,\tau_i}^\omega)$ for E in ω.*

For the end-to-end latency, only job chains are considered when the system is properly warmed up. More specifically, we define the following.

Definition 3.7 (End-to-End Latencies)**.** *Let $E = (\tau_1 \to \dots \to \tau_n)$ be a cause-effect chain and $\omega \in \Omega$ a system evolution. The MRT and the MDA are defined by:*

$$\text{MRT}(E, \omega) \coloneqq \sup_{z > \text{re}^\omega(\tau_1(W_{E,\tau_1}^\omega))} \ell(\overrightarrow{ac}_E^\omega(z)) \tag{3.20}$$

$$\text{MDA}(E, \omega) \coloneqq \sup_{z' > \text{we}^\omega(\tau_n(W_{E,\tau_n}^\omega))} \ell(\overleftarrow{ac}_E^\omega(z')) \tag{3.21}$$

Similarly, the MRRT and MRDA are defined by considering only the corresponding job chains without the additional z and z':

$$\text{MRRT}(E, \omega) \coloneqq \sup \left\{ \ell^\omega(J_1, \dots, J_n) \,\middle|\, \overrightarrow{ac}_E^\omega(z) = (z, J_1, \dots, J_n, z'),\ z > \text{re}^\omega(\tau_1(W_{E,\tau_1}^\omega)) \right\} \tag{3.22}$$

$$\text{MRDA}(E, \omega) \coloneqq \sup \left\{ \ell^\omega(J_1, \dots, J_n) \,\middle|\, \overleftarrow{ac}_E^\omega(z') = (z, J_1, \dots, J_n, z'),\ z' > \text{we}^\omega(\tau_n(W_{E,\tau_n}^\omega)) \right\} \tag{3.23}$$

We generalize these definitions to account for all possible system evolutions as follows:

$$\mathrm{MRT}(E) \coloneqq \sup_{\omega\in\Omega} \mathrm{MRT}(E,\omega) \qquad \mathrm{MDA}(E) \coloneqq \sup_{\omega\in\Omega} \mathrm{MDA}(E,\omega) \tag{3.24}$$

$$\mathrm{MRRT}(E) \coloneqq \sup_{\omega\in\Omega} \mathrm{MRRT}(E,\omega) \qquad \mathrm{MRDA}(E) \coloneqq \sup_{\omega\in\Omega} \mathrm{MRDA}(E,\omega) \tag{3.25}$$

We call these the overall *MRT, MDA, MRRT and MRDA.*

By definition, the difference between MRT and MRRT depends on the time between external activity and read-event of the first job in immediate forward augmented job chains. Similarly, the difference between MDA and MRDA depends on the time between actuation and write-event of the last job in immediate backward augmented job chains. We formalize this relation as follows:

Lemma 3.8. *Let $\rho^\omega_-(\tau)$ and $\rho^\omega_+(\tau)$ be the minimal and maximal time between two subsequent read-events of task τ in system evolution $\omega \in \Omega$, respectively. Furthermore, let $\psi^\omega_-(\tau)$ and $\psi^\omega_+(\tau)$ be the minimal and maximal time between two subsequent write-events of task τ in system evolution $\omega \in \Omega$, respectively. Then the following two bounds hold:*

$$\mathrm{MRT}(E,\omega) - \mathrm{MRRT}(E,\omega) \in [\rho^\omega_-(\tau_1), \rho^\omega_+(\tau_1)] \tag{3.26}$$

$$\mathrm{MDA}(E,\omega) - \mathrm{MRDA}(E,\omega) \in [\psi^\omega_-(\tau_n), \psi^\omega_+(\tau_n)] \tag{3.27}$$

Proof. By definition, we have

$$\mathrm{MRT}(E,\omega) - \mathrm{MRRT}(E,\omega) \tag{3.28}$$

$$= \sup\left\{ \mathrm{re}^\omega(J_1) - z \,\middle|\, \overrightarrow{ac}^\omega_E(z) = (z, J_1, \dots, J_n, z'),\ z > \mathrm{re}^\omega(\tau_1(W^\omega_{E,\tau_1})) \right\} \tag{3.29}$$

$$= \sup\left\{ \mathrm{re}^\omega(\tau_1(m+1)) - \mathrm{re}^\omega(\tau_1(m)) \,\middle|\, m \geq W^\omega_{E,\tau_1} \right\}. \tag{3.30}$$

Hence, $\rho^\omega_-(\tau_1) \leq \mathrm{MRT}(E,\omega) - \mathrm{MRRT}(E,\omega) \leq \rho^\omega_+(\tau_1)$. This proves Equation (3.26). We can do a similar proof for Equation (3.27). By definition, we have

$$\mathrm{MDA}(E,\omega) - \mathrm{MRDA}(E,\omega) \tag{3.31}$$

$$= \sup\left\{ z' - \mathrm{we}^\omega(J_n) \,\middle|\, \overleftarrow{ac}^\omega_E(z') = (z, J_1, \dots, J_n, z'),\ z' > \mathrm{we}^\omega(\tau_n(W^\omega_{E,\tau_n})) \right\} \tag{3.32}$$

$$= \sup\left\{ \mathrm{we}^\omega(\tau_n(m+1)) - \mathrm{we}^\omega(\tau_n(m)) \,\middle|\, m \geq W^\omega_{E,\tau_n} \right\}. \tag{3.33}$$

Hence, $\psi^\omega_-(\tau_n) \leq \mathrm{MDA}(E,\omega) - \mathrm{MRDA}(E,\omega) \leq \psi^\omega_+(\tau_n)$. This proves Equation (3.27). □

3.3 State of the Art

In the following, we review the state of the art for the analysis of self-suspending tasks and for the end-to-end latency of cause-effect chains.

3.3.1 State of the Art for Self-Suspending Tasks

Task models with self-suspension have been investigated since 1988 [RSL88]. However, attempts to extend existing techniques [Dev03; LR10] and insights such as the CIT to self-suspending task systems have been shown to be non-trivial. More specifically, under the assumption that a task may self-suspend, many "*[...] key insights underpinning the analysis of non-self-suspending tasks no longer hold*", as detailed in the self-suspension review paper by Chen et al. [Che+19b]. That is, most classical timing analysis concepts, e.g., the CIT for Task-level Fixed-Priority (T-FP) scheduling by Liu and Layland [LL73] and the demand bound function for Earliest-Deadline-First (EDF) scheduling by Baruah et al. [BMR90], assume a work-conserving behavior on the job-level and are hence invalid in presence of self-suspension. Moreover, self-suspending tasks are prone to timing anomalies, i.e., a reduction of the length of execution segments or suspension segments can increase the response time of another job. Consequently, several research results before 2015 have been found to be flawed. In the review by Chen et al. [Che+19b] the authors listed 6 categories of flaws and over 20 affected publications. Two more counterexamples of existing results on self-suspension are provided in Sections 4.1 and 4.2 of this dissertation. Correct results are summarized in the survey by Chen et al. [Che+17].

To date, dynamic and segmented (or *multi-segment*) self-suspension are the predominately studied models in the literature as explained by Chen et al. [Che+19b]. The segmented self-suspension model [BA05; Brü+16; Che+19a; CL14; GC21; HC16; Kim+13; Nel+15; PF16; Sch+18b; YNB19] and the dynamic self-suspension model [ABN22; AB04; CNH16; Dev03; GBC20; GC20; GUC21; Hua+15; LC14] differ in the allowed suspension pattern. In the segmented self-suspension model, computation and suspension segments are interleaved in a predefined manner and each segment's duration is bounded. In contrast, the dynamic self-suspension model is a generalization of the segmented model in the sense that any interleaved sequence of computation and suspension is admissible as long as the cumulative duration of computation and suspension is bounded. In addition to these two models, von der Brüggen et al. [BHC17] proposed a hybrid self-suspension model, which increases the flexibility of the segmented model and improves accuracy of the dynamic model.

The well-defined suspension structure in the segmented self-suspension model can be exploited to optimize the scheduling policies, e.g., by Fixed-Relative-Deadline (FRD) strategies [Brü+16; CL14; PF16] and by mapping to the leader-follower problem [Che+19a]. Chen [Che16] and Mohaqeqi et al. [MEY16] showed that verifying whether a set of segmented self-suspension tasks can meet their deadlines under static-priority scheduling is co$\mathcal{NP}$-hard in the strong sense. For the dynamic self-suspension model, the computational complexity of scheduler design remains an open problem [Che16; Che+17]. Under static-priority scheduling, Chen et al. [CNH16] provided a unifying response-time analysis framework, considering multiple approaches to model the self-suspension time under a dynamic self-suspension behavior, i.e., as computation, carry-in, blocking, or jitter.

There are several scenarios where self-suspension occurs [Che+19b, Section 2], which are common in distributed systems. For example, self-suspension can occur for I/O-

or memory-intensive tasks [Kan+07; Kat+11], multiprocessor synchronization [Bra22], hardware acceleration by using coprocessors and computation offloading [Liu+14; Nim+10; TC13], scheduling of parallel tasks [Fon+16; Uet+21], real-time tasks in multicore systems with shared memory [HCR16], timing analysis of deferrable servers [Che+22b; LC14], dynamic reconfigurable Field Programmable Gate Arrays (FPGAs) for real-time applications [Bio+16], and real-time communication for networks-on-chip [Uet+20]. The suspension time between two consecutive execution segments can range from a few microseconds to a few hundreds of milliseconds (or even seconds) depending on the application.

3.3.2 State of the Art for Cause-Effect Chains

The end-to-end latency semantics defined in this section have been widely studied in the literature. Most approaches in the literature that validate timing requirements of cause-effect chains can be classified into two categories: Active approaches [Bec+17b; CKK20; Gir+18; SE16], which control the release of jobs in the subsequent tasks in the chain to ensure that the data is written and read correctly, and passive approaches [Bec+16a; Bec+16b; Ben+02; Dav+07; Dür+19; Fei+09; FBP17; GNV22; Kla+18; KBS18; Raj+10; Sch+18a; Wys+13], which focus on how the data is produced and consumed among the jobs of the recurrent tasks in the cause-effect chain, provided that the release of jobs of subsequent tasks is independent of the production of data. This dissertation focuses on passive approaches. Table 3.1 provides an overview over analytical bounds for the end-to-end latency.

The analytical approach for periodic and sporadic systems is usually different. Periodic approaches mostly traverse a safe analysis window (commonly one or two hyperperiods) and quantify the length of job chains starting in that analysis window. Prime examples are the analytical results from Becker et al. [Bec+16b; Bec+17a] for MRDA and Kloda et al. [KBS18] for MRT. Recent approaches focus on more dedicated solutions [GNV22; Gün+23a; Klo+22; PM22] and on speeding-up of the computation process [Bi+22; Gün+23b]. Due to their complexity, we refer to the literature for further information on bounds for periodic tasks.

On the other hand, sporadic schedules are not repetitive. Therefore, it is impossible to define a safe analysis window. Approaches for sporadic systems rather pursue closed-form solutions with analytical over-approximation. To that end, the first analytical bound on the end-to-end latency was provided by Davare et al. [Dav+07] under implicit communication. Although the bound was originally formulated for periodic tasks, the analysis can be applied to sporadic tasks $\tau \in \text{SporMinMax}(T_\tau^{\min}, T_\tau^{\max})$ with maximum inter-arrival time T_τ^{max} as well.

Theorem 3.9 (Davare et al. [Dav+07, Section 2.1])**.** *If tasks of the cause-effect chain $E = (\tau_1 \to \cdots \to \tau_n)$ communicate via implicit communication, then the MRT is upper bounded by*

$$\text{MRT}(E) \leq \sum_{i=1}^{n} (T_{\tau_i}^{max} + R_{\tau_i}^{\mathcal{A}}) \tag{3.34}$$

Year	Paper	E2E Latency	Communication Policy	Release Policy
2007	Davare et al. [Dav+07]	MRT	implicit	sporadic[1]
2016	Becker et al. [Bec+16b]	MRDA	implicit	periodic
2017	Becker et al. [Bec+17a]	MRDA	explicit, implicit, LET	periodic
2017	Hamann et al. [Ham+17]	MRT	implicit, LET	periodic, sporadic[1]
2018	Kloda et al. [KBS18]	MRT	implicit	periodic
2019	Dürr et al. [Dür+19]	MRDA, MRT	implicit	sporadic
2020	Kordon and Tang [KT20]	MRDA	LET	periodic
2020	Martinez et al.[MSB20]	MRDA, MRT	explicit, implicit, LET	periodic
2021	Günzel et al. [Gün+21a]	MRDA, MDA, MRT	implicit	periodic
2022	Teper et al. [Tep+22]	MDA, MRT	implicit	other
2022	Gohari et al. [GNV22]	MRDA	implicit	periodic
2022	Kloda et al. [Klo+22]	MRT	implicit	periodic
2022	Pazzaglia et al. [PM22]	MRDA	LET	periodic
2022	Bi et al. [Bi+22]	MRDA	implicit	periodic
2023	Günzel et al. [Gün+23d]	MRT	implicit, LET	periodic, sporadic
2023	Tang et al. [Tan+23a]	MRT	implicit	other
2023	Günzel et al. [Gün+23b]	MRDA, MDA, MRRT, MRT	LET	periodic
2023	Günzel et al. [Gün+23a][2]	MRDA, MDA, MRT	implicit	periodic

[1] Although formulated for periodic tasks, their analysis is applicable to sporadic tasks as well.
[2] This work is a journal extension of [Gün+21a].

Table 3.1.: Analytical results for end-to-end latency in the literature.

where $R^{\mathcal{A}}_{\tau_i}$ is the Worst-Case Response Time (WCRT) of task τ_i.

The bound was further tightened and extended to the MRDA by Dürr et al. [Dür+19] in 2019. Their bound on the end-to-end latency can be summarized as follows.

Theorem 3.10 (Dürr et al. [Dür+19, Theorem 5.4])**.** *If tasks of the cause-effect chain $E = (\tau_1 \to \cdots \to \tau_n)$ communicate via implicit communication, then the MRT and MRDA are upper bounded by*

$$\mathrm{MRT}(E) \leq \sum_{i=1}^{n}(T^{max}_{\tau_i} + R^{\mathcal{A}}_{\tau_i}) - \sum_{i=1}^{n-1}[P_i] \cdot \min(R^{\mathcal{A}}_{\tau_i}, T^{max}_{\tau_{i+1}}) \tag{3.35}$$

$$\mathrm{MRDA}(E) \leq \sum_{i=1}^{n}(T^{max}_{\tau_i} + R^{\mathcal{A}}_{\tau_i}) - \sum_{i=1}^{n-1}[P_i] \cdot R^{\mathcal{A}}_{\tau_i}, \tag{3.36}$$

where $[P_i] = 1$ if τ_i and τ_{i+1} run on the same ECU and τ_i has higher priority than τ_{i+1}, and $[P_i] = 0$ otherwise.

For tasks under LET communication, Hamann et al. [Ham+17] provided an upper bound for the MRT. Although their paper is formulated for periodic tasks, this particular bound is valid for sporadic tasks as well.

Theorem 3.11 (Hamann et al. [Ham+17, Section 4.1.3])**.** *If tasks of the cause-effect chain $E = (\tau_1 \to \cdots \to \tau_n)$ communicate under LET, then the MRT is upper bounded by*

$$\mathrm{MRT}(E) \leq \sum_{i=1}^{n}(T^{max}_{\tau_i} + D_{\tau_i}). \tag{3.37}$$

Recently, the end-to-end behavior of systems with scheduling mechanisms apart from typical time-triggered T-FP scheduling have been of special interest. To that end, Teper et al. [Tep+22] provided an end-to-end analysis for systems using ROS2, which allows both event-triggered and time-triggered release mechanisms. Dasari et al. [Das+22] formally described the real-time behavior of the Adaptive Partitioning Scheduler (APS) and developed an end-to-end latency bound for event-chains under the APS. Tang et al. [Tan+23a] propose a third option to trigger the processing tasks in a chain, namely, event-triggered with data refreshing, and provide an end-to-end analysis for that case. Moreover, Tang et al. [Tan+23b] compared different communication paradigms regarding the end-to-end latency of cause-effect chains (namely, implicit communication, Dynamic Buffering Protocol (DBP), and LET), and discussed the impact of priority assignment on end-to-end latency.

Besides MRT, MRRT, MDA and MRDA, evolving metrics that are related to the end-to-end latency have been presented and discussed in 2023. Köhler et al. [Köh+23] define *robustness margins*, which guarantee that the end-to-end deadline of a cause-effect chain can still be satisfied if software extensions stay within certain bounds. Jiang et al. [Jia+23] analyze and optimize of the *worst-case time disparity* (that is, the maximum difference among the timestamps of all raw data produced by sensors that an output originates from) in cause-effect chains.

3.4 Summary

In this chapter, we identified two key challenges for distributed real-time systems, namely: distributed execution of jobs, and interplay of distributed tasks. While the former requires a refinement of jobs into smaller blocks, the latter requires specifying timing requirements that account for the cooperation of several distributed tasks. In particular, in this chapter, we defined self-suspending tasks and end-to-end latency of cause-effect chains which are subject to further investigation in the subsequent chapters.

Specifically, in Chapter 5, we examine the timing behavior of self-suspending tasks. For the examination, we focus on the schedulability analysis of dynamic self-suspending tasks, and on treatments to avoid timing anomalies for segmented self-suspending tasks. Due to the unintuitive behavior of self-suspending tasks, which led to a series of flawed results before 2015, this dissertation aims to provide sound, careful analysis of classical J-FP scheduling algorithms.

In Chapter 6, we investigate the end-to-end latency of cause-effect chains in more detail. Specifically, we examine worst-case and probabilistic guarantees for the end-to-end latency. While the main goal of Chapter 6 is the analysis of cause-effect chains distributed among (potentially asynchronized) ECUs, this dissertation has a special focus on the fundamental properties and underlying principles that data propagation is subject to, and therefore provides basic analytical discussions before extending them to the more complex scenarios. We limit our attention to the already challenging passive examination of cause-effect chains, and restrict to time-triggered tasks using implicit communication or LET for large parts of Chapter 6.

4 Importance of Careful, Property-Based Analysis

The large body of classical literature results on real-time systems might encourage the use of intuition to tackle new and emerging problems. However, relying solely on intuition to analyze real-time systems can be risky, as even minor changes to real-time systems can result in unexpected and counterintuitive behavior. For example, intuitive attempts to extend classical results, such as the Critical Instant Theorem (CIT), to self-suspending tasks have led to a series of flawed results, as shown in the review by Chen et al. [Che+19b]. Two unresolved issues highlighted in the conclusion regard the correctness of slack enforcement mechanisms by Lakshmanan and Rajkumar [LR10] and the correctness of the suspension-aware schedulability analysis by Devi [Dev03]. Another notable example for counterintuitive behavior in the domain of end-to-end analysis of cause-effect chains is that early completion can lead to increased end-to-end latency [Gün+23a, Figure 7]. Therefore, to investigate distributed real-time systems, relying on intuition is insufficient. Instead, rigorous analysis is essential, and it is crucial to check the underlying properties and assumptions of each analysis step carefully.

To emphasize the need for careful analysis further, in this chapter, we provide three counterexamples for well-established literature results. The first two counterexamples are related to self-suspension, while the third refutes the CIT in probabilistic setups. Such probabilistic setups later play a role in Section 6.3 for end-to-end analysis. In particular, the counterexamples on self-suspension close the unresolved issues identified in the review paper by Chen et al. [Che+19b]. Hence, this concludes their investigation on the literature on self-suspending tasks published before 2019.

4.1 Slack Enforcement Mechanisms for Self-Suspending Tasks

When tasks suspend themselves, this can introduce additional interference, which has to be accounted for during the analysis. To demonstrate this, we consider a system of two tasks $\mathbb{T} = \{\tau_1, \tau_2\}$, where τ_1 has higher priority than τ_2. Task τ_1 releases jobs sporadically with minimum inter-arrival time 8. Each job of τ_1 executes for 2 time units,

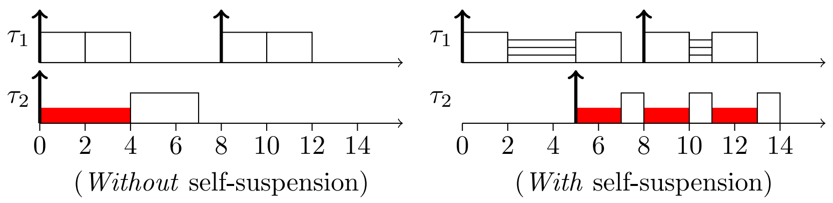


Figure 4.1.: Worst-case interference (marked red) of τ_1 on τ_2. With self-suspension, the worst-case interference increases from 4 to 6.

then suspends itself for up to 3 time units and then executes again for 3 time units. Task τ_2 is a non-self-suspending task that releases jobs sporadically with minimum inter-arrival time 20. Each job of τ_2 executes for 3 time units. This scenario can also be formulated as:

- $\tau_1 \in \mathrm{Spor}(8) \cap \mathrm{SegSS}(2, 3, 2)$
- $\tau_2 \in \mathrm{Spor}(20) \cap \mathrm{SegSS}(3)$

Figure 4.1 shows that self-suspension of τ_1 can cause interference of up to 6 time units on task τ_2. However, without self-suspension of τ_1, the maximal interference that τ_2 suffers from is 4 time units.

To mitigate the impact of self-suspending tasks on the interference of lower-priority tasks, Lakshmanan and Rajkumar [LR10] proposed *slack enforcement* in 2010. They consider sporadic segmented self-suspending tasks with two execution segments, i.e.,

$$\tau \in \mathrm{Spor}(T_\tau) \cap \mathrm{SegSS}(C^1_\tau, S^1_\tau, C^2_\tau) \tag{4.1}$$

for all $\tau \in \mathbb{T}$, and their mechanism delays only the second execution segment of jobs. In this section, we show that their slack enforcement mechanisms can provoke deadline misses. The results presented in this section appeared in the Real Time Systems Journal in 2021 [GC21].

4.1.1 Dynamic Slack Enforcement

The slack enforcement mechanism aims to *"shape the demand of a self-suspending task so that the task behaves like an ideal ordinary periodic task"* [Che+19b, Section 9.1]. Lakshmanan and Rajkumar intend to achieve this by delaying the second execution segment of a self-suspending task, to *"ensure that adequate slack is available to lower-priority tasks"* [LR10, Section IV]. The slack enforcement was originally proposed together with a critical instant by Lakshmanan and Rajkumar [LR10]. While the critical instant was shown to be flawed by Nelissen et al. [Nel+15], the dynamic slack enforcement mechanism proposed in [LR10] can still be applied when Worst-Case Response Times (WCRTs) $R^{\mathcal{A}}_\tau$ under a preemptive Task-level Fixed-Priority (T-FP) scheduling algorithm $\mathcal{A}$ are given beforehand. More specifically, the dynamic slack enforcement delays the

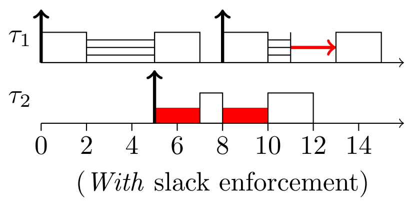


Figure 4.2.: Worst-case interference of τ_1 on τ_2 with slack enforcement is only 4.

second execution segment of each self-suspending task to the latest time such that the WCRT can still be met.

Definition 4.1 (Dynamic slack enforcement in [LR10, Section IV][1])**.** Dynamic slack enforcement *is an execution control policy that delays the release of the second segment of any job $\tau(j)$ to the latest time t, such that $\tau(j)$ can still meet its normal (non-execution-controlled) WCRT $R_\tau^\mathcal{A}$.*

The correctness of the dynamic slack enforcement algorithm is based on two properties specified in Lemmas 4 and 5 in [LR10], which we reformulate to match our notation.

- **Property P1**: If a task $\tau \in \mathbb{T}$ under preemptive T-FP scheduling has a WCRT of $R_\tau^\mathcal{A}$, applying the slack enforcement mechanism makes its WCRT always the same or shorter.

- **Property P2**: The WCRT $R_\tau^\mathcal{A}$ of τ achieved by applying the dynamic slack enforcement mechanism under preemptive T-FP scheduling is not longer than the WCRT in the corresponding scenario by considering only τ's suspension behavior and treating all higher-priority tasks as non-self-suspending tasks.

In other words, Property P1 states that the slack enforcement is superior to the original T-FP scheduler, and Property P2 implies that the suspension behavior of the higher-priority tasks can be neglected when the slack enforcement mechanism is applied.

Intuitively, the slack enforcement mechanism seems to work well as depicted in Figure 4.2. While the second execution segment of the second job of τ_1 is delayed to 13, task τ_1 still finishes within its WCRT. Moreover, the delay provides sufficient slack for τ_2 to finish its execution before the second execution segment of τ_1 start. Hence, the maximum interference, that can be experienced by τ_2 is 4 time units. This is the same maximum interference that can be achieved without suspension (cf. Figure 4.1).

Nevertheless, the review paper by Chen et al. [Che+19b] shows that the proof of the key lemma of the slack enforcement mechanisms in [LR10] is incomplete, and calls for more rigorous proofs to support the correctness of the mechanism. In the following, we show that indeed both properties P1 and P2 do *not* hold in general by providing a counterexample.

[1]The definition is reformulated to match the notation of this dissertation.

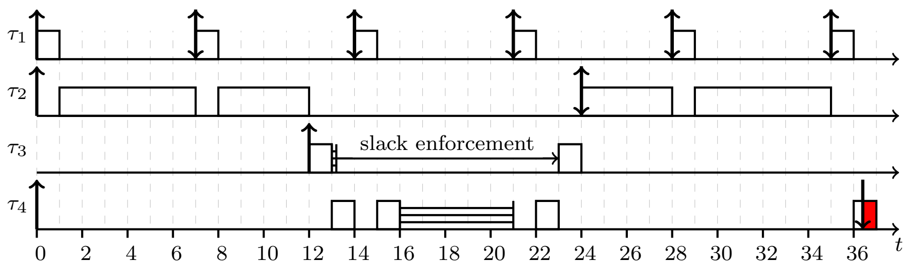


Figure 4.3.: T-FP schedule of $\mathbb{T}$ with dynamic slack enforcement on τ_3 which leads to a deadline-miss (marked in red).

4.1.2 Counterexample

Consider the following implicit-deadline sporadic task set $\mathbb{T} = \{\tau_1, \tau_2, \tau_3, \tau_4\}$:

- $\tau_1 \in \text{Spor}(7) \cap \text{SegSS}(1)$,
- $\tau_2 \in \text{Spor}(24) \cap \text{SegSS}(10)$,
- $\tau_3 \in \text{Spor}(36 + \delta) \cap \text{SegSS}(1, \delta, 1)$,
- $\tau_4 \in \text{Spor}(36 + 2\delta) \cap \text{SegSS}(2, 5, 2)$,

where $0 < \delta < 0.5$. That is, tasks τ_1 and τ_2 do not suspend, task τ_3 suspends for δ time units between two execution segments of length 1, and task τ_4 suspends for 5 time units between two execution segments of length 2. We apply preemptive Rate Monotonic (RM) scheduling, i.e., τ_1 has the highest priority, whereas τ_4 has the lowest priority.

For the counterexample, we need the WCRT of τ_3 and an upper bound on the WCRT of τ_4, which are $15 + \delta$ and $36 + \delta$, respectively, as we show in detail in Appendix B.1. The concrete example in Figure 4.3 demonstrates the behavior of the dynamic slack enforcement mechanism from [LR10]. According to the dynamic slack enforcement mechanism, the second execution segment of τ_3 is delayed to the latest time such that it still meets its WCRT of $15 + \delta$. That is, the second segment is delayed to release at time 23 such that it finishes no later than at $12 + 15 + \delta = 27 + \delta$. This leads to a deadline miss of τ_4. In particular, this disproves Property P1.

For Property P2, we consider the schedule depicted in Figure 4.4, which treats all higher-priority tasks as non-suspending tasks. Since the obtainable schedules without suspension of τ_3 are a subset of the obtainable schedules of $\mathbb{T}$ with suspension, the WCRT of τ_4 is again bounded by $36 + \delta$. However, we have already shown that dynamic slack enforcement leads to a deadline miss. This disproves Property P2.

We note that the stated properties for the dynamic slack enforcement mechanism are invalidated even if the mechanism is restricted to periodic or synchronous task sets due to the following consideration. Let $\delta = 0.2$ and consider $\mathbb{T}$ to be a synchronous periodic task set, i.e., job releases are aligned with the previous deadline and the first job release of each task is at time 0. In this case, the release pattern depicted in Figure 4.3 occurs at time

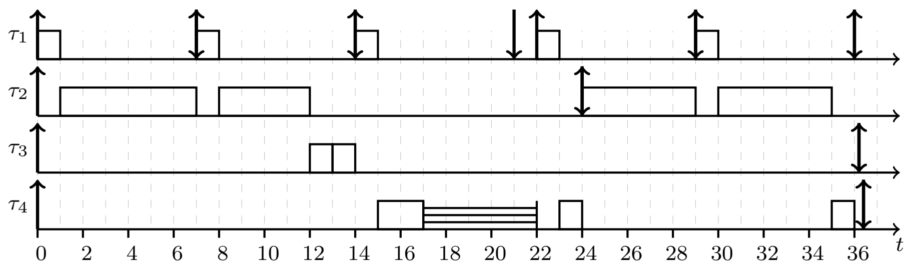


Figure 4.4.: T-FP schedule of $\mathbb{T}$ for achieving the WCRT of τ_4 when all higher-priority tasks have no suspension.

393 120, i.e., 393 120 is an integer multiple of 7, 24 and 36.4, and $10\,860 \cdot 36.2 - 393\,120 = 12$. Hence, the dynamic slack enforcement causes a deadline miss of τ_4 at time $393\,120 + 36.4$.

4.1.3 Source of Misconception

We believe that the main source of the misconception of the dynamic slack enforcement mechanism is inherited from the misconception of the CIT for self-suspending task systems, claimed in [LR10]. The authors argued that the dynamic slack enforcement mechanism delays the release of the second execution segment to the latest possible time, thereby not worsen the schedulability of lower-priority tasks. However, this argument is flawed, as disproved above. Our counterexample is based on the following conditions:

- If task τ_3 interferes with only one execution segment of a job of τ_4, the response time of the job of τ_4 is at most $36 + \delta$.
- If task τ_3 interferes with two execution segments of a job of τ_4, the response time of the job of τ_4 can be up to 37.

The dynamic slack enforcement mechanism delays the second computation segment of τ_3 in this counterexample and forces the latter case to take place, whilst the original T-FP scheduler has a safe WCRT of $36 + \delta$.

This is the counterpart of the misconception of the CIT claimed in [LR10]. Imagine that we split task τ_3 into two ordinary sporadic tasks τ_3^1 and τ_3^2 that do not suspend themselves, both with execution time 1 and minimum inter-arrival time 36. If we apply the (incorrect) CIT in [LR10], the WCRT of τ_4 follows from the schedule depicted in Figure 4.4. However, the actual worst case for this pattern is to release τ_3^1 and τ_3^2 so that each of them interferes with one execution segment of τ_4, i.e., as in Figure 4.3.

The proof of Lemma 4 in [LR10] is incorrect because the proof did not inspect the impact of the two execution segments of τ_3 on the two execution segments of τ_4. It solely argues that $I_j^{ns}(R) = I_j^1(R) + I_j^2(R)$ (here, the notation is directly from [LR10]), i.e., for an interval length R, the interference $I_j^{ns}(R) = \left\lceil \frac{R}{T_j} \right\rceil (C_j^1 + C_j^2)$ is always equal to $I_j^1(R) + I_j^2(R) = \left\lceil \frac{R}{T_j} \right\rceil C_j^1 + \left\lceil \frac{R}{T_j} \right\rceil C_j^2$. However, this argument is irrelevant to a formal

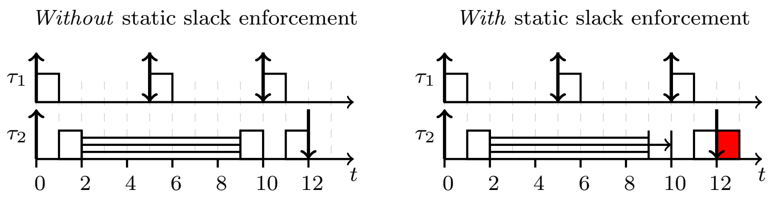

Without static slack enforcement
With static slack enforcement
τ_1
τ_2
0
2
4
6
8
10
12
t

Figure 4.5.: Deadline miss with static slack enforcement.

proof of the WCRT. Instead, the proof in [LR10] should in analyze the WCRT of a task for both cases, e.g., using the iterative approach like Time-Demand Analysis (TDA), and demonstrate their equivalence.

We note that our counterexample does not follow the call for rigorous proof of Lemma 4 in [LR10] by Chen et al. [Che+19b]. The main argument in [Che+19b] was due to the incomplete proof of the level-i busy period, which is irrelevant in our counterexample.

4.1.4 Static Slack Enforcement

Lakshmanan and Rajkumar [LR10] also present a static slack enforcement mechanism, that approximates the dynamic slack enforcement mechanism and reduces the runtime complexity. However, the authors state that static slack enforcement is not optimal in the sense that it might have negative impact on the WCRT. In the following, we show that this mechanism may even provoke deadline misses.

The static slack enforcement mechanism from [LR10, Section V] delays the second execution segment of each self-suspending job generated by a self-suspending task τ, such that the processor indeed idles the maximal suspension time S_τ^1 between both segments. Its formulation relies on the definition of *level-τ slack*[2]:

Definition 4.2 (Level-τ slack in [LR10, Section IV])**.** *The level-τ slack over any time interval $[t_1, t_2]$ (with $t_2 \geq t_1$) is defined as the total time within $[t_1, t_2]$ during which no tasks with priority greater than or equal to τ are executing.*

Definition 4.3 (Static slack enforcement in [LR10, Section V])**.** Static slack enforcement *is defined as an execution control policy that delays the release of the second segment of a self-suspending task τ such that the level-τ slack between the two segments of τ is at least S_τ^1.*

The work of Lakshmanan and Rajkumar [LR10] does not explain how self-suspending tasks may meet their deadlines utilizing this mechanism. In fact, the static slack enforcement is a source of deadline miss for self-suspending tasks, since the response time is increased if the slack is less than the suspension time. Figure 4.5 shows a schedule

[2]In their work, Lakshmanan and Rajkumar assume that every task is equipped with an index representing the task priority. Therefore, they use level-i slack to denote the slack on the level of task τ_i. We reformulate their definitions to be congruent with the notation of this dissertation.

where the static slack enforcement leads to a deadline miss: Consider an implicit-deadline task set $\mathbb{T} = \{\tau_1, \tau_2\}$ with two tasks

- $\tau_1 \in \mathrm{Spor}(5) \cap \mathrm{SegSS}(1)$
- $\tau_2 \in \mathrm{Spor}(12) \cap \mathrm{SegSS}(1, 7, 2)$

ordered by priority, i.e., τ_1 has higher priority than τ_2. At most one job of τ_1 interferes with each execution segment of τ_2. Hence, the WCRT of τ_2 is 12, as depicted on the left-hand side of Figure 4.5. The level-τ_2 slack in $[2, 9]$ is 6 since τ_1 utilizes the processor for 1 time unit. To obtain level-τ_2 slack of 7, the second segment of the job of τ_2 is delayed. This leads to a deadline miss as depicted on the right-hand side of Figure 4.5. Moreover, since the schedule on the left-hand side does not consider any suspension from the higher-priority task τ_1, this also shows that static slack enforcement does not guarantee the same WCRT as without enforcement when all higher priority self-suspending tasks behave like ordinary periodic tasks.

4.2 Analyzing Earliest-Deadline-First for Self-Suspending Tasks

In 2003, Devi [Dev03] presented a schedulability test for periodic task sets scheduled by Earliest-Deadline-First (EDF). In their Section 4, they provide extensions to account for practical overheads, like interference by interrupt handlers, self-suspension, and priority-space limitations. However, they do not provide a formal proof of their extension to self-suspending tasks. Hence, in the review paper by Chen et al., they request a "rigorous proof, since self-suspension behavior has induced several non-trivial phenomena" [Che+19b]. In the following we provide a counterexample of Theorem 8 in [Dev03, Section 4.5] and disprove the schedulability test. The results presented in this section appeared in the Real Time Systems Journal [GC20].

4.2.1 Devi's Analysis

Devi's analysis [Dev03] for self-suspending tasks in 2003 is designed for periodic, dynamic self-suspending tasks scheduled under preemptive EDF. That is, each task $\tau \in \mathbb{T}$ can be abstracted as

$$\tau \in \mathrm{Per}(T_\tau) \cap \mathrm{DynSS}(C_\tau, S_\tau). \tag{4.2}$$

The analysis is rephrased as follows:

Theorem 4.4 (Devi [Dev03][3])**.** *Let* $\mathbb{T} = \{\tau_1, \tau_2, \dots, \tau_n\}$ *such that* $T_{\tau_i} \le T_{\tau_j}$ *for all* $i \le j$.

[3]The definition is reformulated to match the notation of this dissertation. Moreover, we restrict to the implicit-deadline case since that formulation is sufficient for the counterexample.

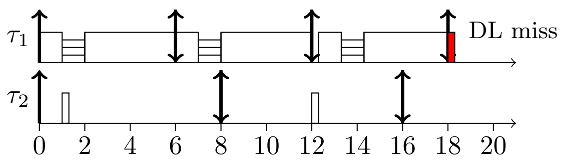


Figure 4.6.: A concrete EDF schedule with a deadline miss.

The task set $\mathbb{T}$ *is schedulable using preemptive EDF if for all* k *with* $1 \leq k \leq n$ *inequality*

$$\frac{B_k + B'_k}{T_{\tau_k}} + \sum_{i=1}^{k} \frac{C_{\tau_i}}{T_{\tau_i}} \leq 1 \tag{4.3}$$

holds, where $B_k = \sum_{i=1}^{k} \min\{S_{\tau_i}, C_{\tau_i}\}$ *and* $B'_k = \max_{1 \leq i \leq k} (\max\{0, S_{\tau_i} - C_{\tau_i}\})$.

4.2.2 Counterexample

In the following we provide a counterexample for Devi's analysis. While Devi considered arbitrary-deadline task systems with asynchronous arrival times, our counterexample is valid by considering two implicit-deadline periodic tasks released at the same time and therefore disproves also the general case. More specifically, we consider the task set $\mathbb{T} = \{\tau_1, \tau_2\}$ with:

- $\tau_1 \in \text{PerOff}(6, 0) \cap \text{DynSS}(5, 1)$
- $\tau_2 \in \text{PerOff}(8, 0) \cap \text{DynSS}(\varepsilon, 0)$ for any $\varepsilon \in (0, \frac{1}{3}]$

The test of Theorem 4.4 is as follows:

- When $k = 1$, we have $B_1 = 1$ and $B'_1 = 0$. Therefore, when $k = 1$, we obtain $\frac{B_k + B'_k}{T_{\tau_k}} + \sum_{i=1}^{k} \frac{C_{\tau_i}}{T_{\tau_i}} = 1$.
- When $k = 2$, we have $B_2 = 1$ and $B'_2 = 0$. Therefore, when $k = 2$, we obtain $\frac{B_k + B'_k}{T_{\tau_k}} + \sum_{i=1}^{k} \frac{C_{\tau_i}}{T_{\tau_i}} = \frac{1}{8} + \frac{\varepsilon}{8} + \frac{5}{6} = \frac{23+3\varepsilon}{24} \leq 1$, since $\varepsilon \leq 1/3$.

Therefore, Devi's schedulability test concludes that the task set is schedulable under preemptive EDF. But, a concrete schedule as demonstrated in Figure 4.6 shows that one of the jobs of task τ_1 misses its deadline even when both tasks release their first jobs at the same time. More specifically, the example in Figure 4.6 shows that a job of task τ_1 may be blocked by a job of task τ_2, which results in a deadline miss of the job of task τ_1. However, in Devi's schedulability analysis, such blocking is never considered since B_1 and B'_1 do not have any term related to τ_2.

The counterexample only requires task τ_1 to suspend once. It shows that applying Devi's analysis in [Dev03] is unsafe even for the segmented self-suspension model under EDF scheduling. We note that the above counterexample is only for Theorem 8 in [Dev03]. We do not examine any other schedulability tests in [Dev03].

4.3 Critical Instant Theorem for Probabilistic Scenarios

A classical, hard real-time analysis aims to determine whether tasks fulfill their timing constraints under all circumstances. The seminal work by Liu and Layland [LL73] provides fundamental knowledge to ensure worst-case timeliness when scheduling periodic real-time tasks on a uniprocessor system. More specifically, in their work they formulate the Critical Instant Theorem (CIT), as recapped in Section 2.5.

However, when occasional deadline misses can be tolerated, probabilistic guarantees are an important measure. To that end, Maxim and Cucu-Grosjean [MC13] presented a probabilistic response-time analysis, extending the CIT to probabilistic scenarios. Similarly, Chen and Chen analyze the probability for deadline misses [CC17] based on the CIT. In the following, we revisit the critical instant for the probabilistic timing analysis of sporadic real-time task sets, formally defined as the Worst-Case Response Time Exceedance Probability (WCRTEP) in Definition 4.6 and the Worst-Case Deadline Failure Probability (WCDFP) in Definition 4.8.

In this section, we show that their calculation of the WCDFP and WCRTEP is flawed, by providing counterexamples. Moreover, we propose two methods to derive the WCRTEP and WCDFP for sporadic real-time task systems with Probabilistic Execution Times (pETs). One method is to account for one additional *carry-in* job of a higher-priority task, whilst another is to *inflate* the execution of certain higher-priority jobs with a sampling process. More specifically, this section is structured as follows:

- In Section 4.3.1, we recap the critical instant and its analytical implications as stated by Maxim and Cucu-Grosjean [MC13].
- In Section 4.3.2, we present a counterexample to demonstrate that calculating the WCRTEP and WCDFP based on the CIT for T-FP scheduling can be *too optimistic* even without backlog.
- In Section 4.3.3, we propose two methods to derive safe WCRTEP and WCDFP.
- In Section 4.3.4, we discuss the impact of the unsound CIT for the probabilistic setting in the literature.

The results presented in this section appeared in RTSS 2022 [Che+22a].

4.3.1 Critical Instant in the Probabilistic Setup

We consider a set of sporadic real-time tasks $\mathbb{T}$ with independent and identically distributed (iid) Probabilistic Worst-Case Execution Time (pWCET) scheduled by a preemptive T-FP scheduling algorithm $\mathcal{A}$ on a uniprocessor system. Each task $\tau \in \mathbb{T}$ can be abstracted as

$$\tau \in \mathrm{Spor}(T_\tau) \cap \mathrm{pWCET}^{\mathrm{iid}}(\mathcal{D}_\tau), \tag{4.4}$$

where $\mathcal{D}_\tau$ describes the distribution of upper bounds $\bar{c}^{\omega}_{\tau(j)}$ on the execution demand $c^{\omega}_{\tau(j)}$ of jobs $\tau(j)$ under system evolutions $\omega \in \Omega$. We assume that the jobs' execution demands are independent not only on the intra-task level (i.e., jobs of the same task) but also on the inter-task level (i.e., jobs of different tasks); an assumption common in the literature, e.g., [Brü+18; CC17; Che+19c; DC19; MC13]. Moreover, we assume that the execution demand is independent of the job releases. We assume that once a job misses its deadline, it is immediately aborted.

In the probabilistic setup, the upper bound on the execution demand of a job $\tau(j)$ can be considered as a random variable $c^{\bullet}_{\tau(j)} \colon \Omega \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ on the sample space Ω. Furthermore, the response time of the j-th job of task τ is a random variable $R^{\mathcal{A}(\bullet),\bullet}_{\tau(j)} \colon \Omega \to \mathbb{R}_{\geq 0} \cup \{\infty\}$. If a deadline is missed in a specific system evolution $\omega \in \Omega$, then under that sample the response time is $R^{\mathcal{A}(\omega),\omega}_{\tau(j)} = \infty$ as the job never finishes. We are interested in the worst-case behavior of the probabilistic response time and the Deadline Failure Probability (DFP). As it may seem strange at the first glance to combine the worst-case phrase with the probabilistic argument, the following definitions provide concrete statements.

Definition 4.5 (Job-Level Response Time Exceedance Probability (RTEP))**.** *The RTEP of job $\tau(j)$ is the probability that the response time of the j-th job of task τ is greater than a target value $R \in \mathbb{R}_{\geq 0}$, i.e.,*

$$\mathbb{P}\left(R^{\mathcal{A}(\bullet),\bullet}_{\tau(j)} > R\right). \tag{4.5}$$

Definition 4.6 (Worst-Case Response Time Exceedance Probability (WCRTEP))**.** *The WCRTEP of task τ is an upper bound on the probability that the response time of a job of τ is greater than a target value $R \in \mathbb{R}_{\geq 0}$, i.e.,*

$$\sup_{j \in \mathbb{N}} \left\{ \mathbb{P}\left(R^{\mathcal{A}(\bullet),\bullet}_{\tau(j)} > R\right) \right\}. \tag{4.6}$$

We are often not interested in the complete response-time distribution but in the probability that the response time exceeds the relative deadline. That is, we are interested in the probability that a job misses its deadline.

Definition 4.7 (Deadline Failure Probability (DFP))**.** *The DFP of job $\tau(j)$ is the probability that the j-th job of task τ misses its relative deadline D_τ, i.e.,*

$$\mathbb{P}\left(R^{\mathcal{A}(\bullet),\bullet}_{\tau(j)} > D_\tau\right). \tag{4.7}$$

Definition 4.8 (Worst-Case Deadline Failure Probability (WCDFP))**.** *The WCDFP of task τ is an upper bound on the probability that a job of τ misses its relative deadline D_τ, i.e.,*

$$\sup_{j \in \mathbb{N}} \left\{ \mathbb{P}\left(R^{\mathcal{A}(\bullet),\bullet}_{\tau(j)} > D_\tau\right) \right\}. \tag{4.8}$$

We note that this WCDFP definition is identical to the one by Davis and Cucu-Grosjean in their survey paper [DC19]. However, they use the term probabilistic Worst-Case Response Time (pWCRT) to denote the WCRTEP defined here.

By following the CIT, Theorem 1 from [MC13] claims that the worst-case distribution of the response time occurs *for the first job* if the task set follows synchronous release. In the following, we provide a concrete counterexample to invalidate this statement. Note that implicit deadlines are assumed in [MC13]. For completeness, we first show their original theorem, slightly reformulated to match the notation of this dissertation.

Theorem 4.9 (From [MC13, Theorem 1]). *We consider a task system $\mathbb{T} = \{\tau_1, \dots, \tau_n\}$ of n tasks with τ_i described by probabilistic C_{τ_i} and T_{τ_i}, $\forall i \in \{1, 2, \dots, n\}$. The set is ordered according to the priorities of the tasks and the system is scheduled preemptively on a single processor. The response-time distribution $R^{\mathcal{A}(\bullet),\bullet}_{\tau_i(1)}$ of the first job of task τ_i is greater than the response-time distribution $R^{\mathcal{A}(\bullet),\bullet}_{\tau_i(j)}$ of any j-th job of task τ_i, $\forall i \in \{1, 2, \dots, n\}$.*

It is assumed in [MC13] that all tasks are synchronous, i.e., every task releases its first job at time 0, following the CIT. That is, it is claimed that the WCRTEP is observed for $j = 1$ since $\mathbb{P}\left(R^{\mathcal{A}(\bullet),\bullet}_{\tau(1)} > t\right) \geq \mathbb{P}\left(R^{\mathcal{A}(\bullet),\bullet}_{\tau(j)} > t\right)$ for all $t \in \mathbb{R}$. Similarly, Theorem 1 from [CC17] states that the WCDFP of a task τ can be derived under the critical instant.

Theorem 4.10 (Reworded from [CC17, Theorem 1]). *Let S_t be the sum of the execution times of one job of τ and $\left\lceil \frac{t}{T_{\tau'}} \right\rceil$ jobs of each higher-priority task $\tau' \in hp(\tau)$, i.e.,*

$$S_t := \bar{c}^{\bullet}_{\tau(1)} + \sum_{\tau' \in hp(\tau)} \sum_{q=1}^{\left\lceil \frac{t}{T_{\tau'}} \right\rceil} \bar{c}^{\bullet}_{\tau'(q)} \,. \tag{4.9}$$

If the TDA succeeds using the Worst-Case Execution Time (WCET) of each task, then the probability of deadline misses of task τ is 0. Otherwise, the probability of deadline misses of task τ is upper bounded by

$$\inf_{0<t\leq D_\tau} \mathbb{P}(S_t \geq t). \tag{4.10}$$

4.3.2 Counterexample

The statements in Theorem 4.9 and Theorem 4.10 are seemingly correct by simply applying the sketched proof of Theorem 2.21 considering pWCET. However, the analysis in the critical time zone does not consider the execution time distribution because only the WCET is needed for analyzing the WCRT.

The key issue is how the interval extension from $r^{\omega}_{\tau(\ell)}$ to t_0 in the proof of Theorem 2.21 changes the response-time distribution of $\tau(\ell)$. Considering the execution time distribution of the higher-priority jobs, the interval extension from $r^{\omega}_{\tau(\ell)}$ to t_0 is not deterministic and

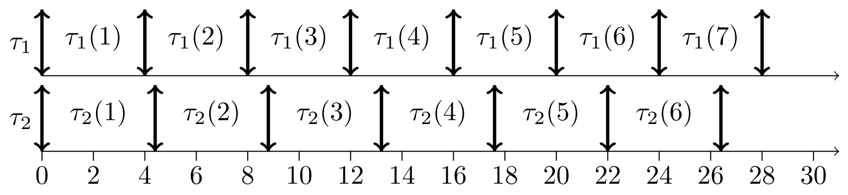


Figure 4.7.: Release pattern of the counterexample task set, where both tasks release synchronously at time 0.

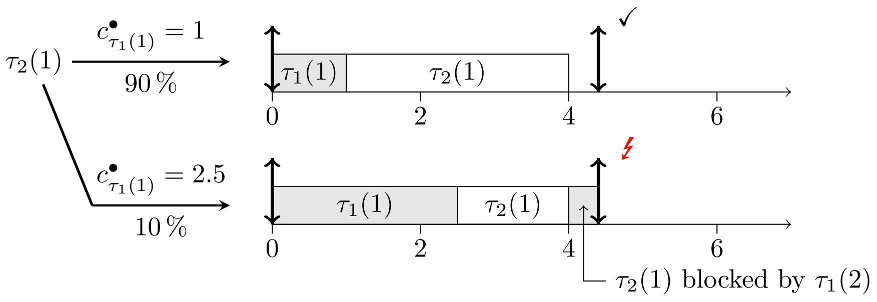

$\tau_2(1)$
$c^\bullet_{\tau_1(1)} = 1$
90 %
$c^\bullet_{\tau_1(1)} = 2.5$
10 %
$\tau_1(1)$
$\tau_2(1)$
0 2 4 6
✓
$\tau_2(1)$ blocked by $\tau_1(2)$

Figure 4.8.: State space for job $\tau_2(1)$. Note that, since job $\tau_{2,1}$ misses its deadline at 4.4, it is aborted in the schedule below.

is related to the probability of the execution times of the higher-priority tasks. Therefore, ignoring the probability of the feasibility of the interval extension may result in incorrect quantification of response-time distribution. This observation leads to the following counterexample.

Consider the periodic synchronous task set $\mathbb{T} = \{\tau_1, \tau_2\}$ with

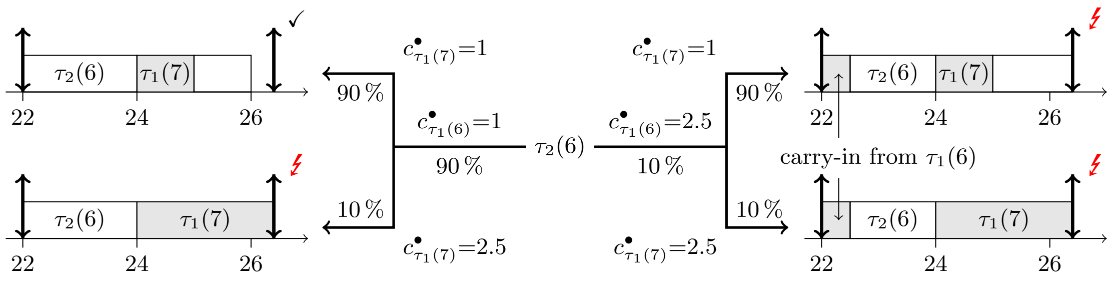
- $\tau_1 \in \text{PerOff}(T_{\tau_1}{=}4, \phi_{\tau_1}{=}0) \cap \text{pET}^{\text{iid}}(\mathcal{D}_{\tau_1}) \subseteq \text{PerOff}(4,0) \cap \text{pWCET}^{\text{iid}}(\mathcal{D}_{\tau_1})$
- $\tau_2 \in \text{PerOff}(T_{\tau_2}{=}4.4, \phi_{\tau_2}{=}0) \cap \text{pET}^{\text{iid}}(\mathcal{D}_{\tau_2}) \subseteq \text{PerOff}(4.4,0) \cap \text{pWCET}^{\text{iid}}(\mathcal{D}_{\tau_2})$

where $\mathcal{D}_{\tau_1}(t) = \mathbb{1}_{t\geq 1} \cdot 0.9 + \mathbb{1}_{t\geq 2.5} \cdot 0.1$ and $\mathcal{D}_{\tau_2}(t) = \mathbb{1}_{t\geq 3} \cdot 1.0$ are step functions. More specifically, the execution demand of every job $\tau_1(j)$ of τ_1 is a discrete random variable $c^\bullet_{\tau_1(j)}$ with $\mathbb{P}(c^\bullet_{\tau_1(j)} = 1) = 0.9$ and $\mathbb{P}(c^\bullet_{\tau_1(j)} = 2.5) = 0.1$, and $c^\bullet_{\tau_2(j)} = 3$ is deterministic for all jobs $\tau_2(j)$. The release pattern of that task set is depicted in Figure 4.7. In the following, we show that the probability $\mathbb{P}(R^{\mathcal{A}(\bullet),\bullet}_{\tau_2(6)} > t)$ is higher than the probability $\mathbb{P}(R^{\mathcal{A}(\bullet),\bullet}_{\tau_2(1)} > t)$, for some t with $0 < t \leq T_{\tau_2}$. That is, the RTEP of the first job of task τ_2 does not bound the RTEP of the other jobs of τ_2. To that end, we consider $t = 4.4$.

For job $\tau_2(1)$ (as shown in Figure 4.8): If $\tau_1(1)$ has an execution time of 1, then $\tau_2(1)$ finishes at time $1+3=4$. If $\tau_1(1)$ has an execution time of 2.5, then $\tau_2(1)$ does not finish its execution at time 4.4. Therefore, we obtain

$$\mathbb{P}(R^{\mathcal{A}(\bullet),\bullet}_{\tau_2(1)} > 4.4) = \mathbb{P}(c^\bullet_{\tau_1(1)} = 2.5) = 0.1 \tag{4.11}$$

For job $\tau_2(6)$ (as shown in Figure 4.9): If $\tau_1(6)$ has an execution time of 1, it contributes



$\tau_2(6)$
$\tau_1(7)$
22
24
26
$c^\bullet_{\tau_1(7)}$=1
90 %
10 %
$c^\bullet_{\tau_1(7)}$=2.5
$c^\bullet_{\tau_1(6)}$=1
$c^\bullet_{\tau_1(6)}$=2.5
carry-in from $\tau_1(6)$

Figure 4.9.: State space for job $\tau_2(6)$: For 3 out of 4 possible schedules in the time interval $[22, 26.4]$ job $\tau_2(6)$ misses its deadline.

no interference to $\tau_2(6)$. In this case, the response time of $\tau_2(6)$ is only larger than 4.4 if $\tau_1(7)$ executes for 2.5 time units. If $\tau_1(6)$ has an execution time of 2.5, then it contributes 0.5 time units as additional interference for $\tau_2(6)$. In this case, the response time for $\tau_2(6)$ is larger than 4.4 if $\tau_1(7)$ executes for 1 time unit or for 2.5 time units. Hence, $R^{\mathcal{A}(\bullet),\bullet}_{\tau_2(6)} > 4.4$ if either $\tau_1(6)$ or $\tau_1(7)$ or both have an execution time of 2.5. Therefore, the probability that the response time of $\tau_2(6)$ is larger than 4.4 is:

$$\mathbb{P}(R^{\mathcal{A}(\bullet),\bullet}_{\tau_2(6)} > 4.4) = \mathbb{P}(c^\bullet_{\tau_1(6)} = 2.5) + \mathbb{P}(c^\bullet_{\tau_1(6)} = 1) \cdot \mathbb{P}(c^\bullet_{\tau_1(7)} = 2.5) \tag{4.12}$$

$$= 0.1 + 0.9 \cdot 0.1 = 0.19 \tag{4.13}$$

We obtain $\mathbb{P}(R^{\mathcal{A}(\bullet),\bullet}_{\tau_2(1)} > 4.4) = 0.1 < 0.19 = \mathbb{P}(R^{\mathcal{A}(\bullet),\bullet}_{\tau_2(6)} > 4.4)$, which contradicts Theorem 4.9. The counterexample invalidates Theorem 4.10 as well if we assume that τ_2 has an implicit deadline (i.e., $D_{\tau_2} = T_{\tau_2} = 4.4$), since according to the counterexample $\tau_2(6)$ has a larger DFP than $\tau_2(1)$ which contradicts that the WCDFP can be observed for the first job of τ_2.

Consequence: With pWCETs, the synchronous release of all tasks does not necessarily generate the maximum interference and is thus not always a critical instant. Therefore, Theorem 4.9 (i.e., Theorem 1 from [MC13]) and Theorem 4.10 (i.e., Theorem 1 from [CC17]) may result in an unsound WCDFP, as well as unsound WCRTEP.

Detailing the Misconception: The proof in [CC17] follows a similar structure as the proof of the worst-case analysis of Theorem 2.21, whereas the proof in [MC13] directly refers to the classical CIT. Essentially, to prove that the worst case is to synchronously release one job from every higher-priority task together with the job of task τ under analysis, *interval extension* is needed. However, as demonstrated in Figure 4.9, the extension from $r_{\tau(\ell)}$ (namely, 22) to t_0 (namely, 20) is also probabilistic, depending on the execution time of the higher-priority jobs (namely, $\tau_1(6)$). It is therefore unsound to simply extend the interval of interest without considering the change of the probabilistic distribution in this regard.

4.3.3 Safe Bounds for WCDFP and WCRTEP

In the following, we provide two methods to derive a sound Worst-Case Deadline Failure Probability (WCDFP) and Worst-Case Response Time Exceedance Probability (WCRTEP). The first method is based on the inclusion of one additional job of a higher-priority task in the analysis window, typically called *carry-in*. This method has been widely used when the CIT does not work, e.g., for multiprocessor global scheduling [Bak03] and self-suspending task systems [CNH16].

Theorem 4.11 (WCRTEP with Carry-In)**.** *Consider a system of constrained-deadline sporadic real-time tasks $\mathbb{T}$ where a job is directly aborted when missing its deadline. The WCRTEP of task $\tau \in \mathbb{T}$ for a target value $R \in (0, T_\tau]$ under uniprocessor preemptive T-FP scheduling is upper bounded by*

$$\inf_{0<t\leq R} \mathbb{P}(S_t > t), \tag{4.14}$$

where S_t is a random variable which provides the sum of the execution times of one job of τ and $\left\lceil \frac{t+D_{\tau'}}{T_{\tau'}} \right\rceil$ jobs of each higher-priority task $\tau' \in hp(\tau)$, i.e.,

$$S_t := \bar{c}^{\bullet}_{\tau(1)} + \sum_{\tau' \in hp(\tau)} \sum_{q=1}^{\left\lceil \frac{t+D_{\tau'}}{T_{\tau'}} \right\rceil} \bar{c}^{\bullet}_{\tau'(q)}. \tag{4.15}$$

We note that the S_t from Equation (4.15) differs from S_t in Equation (4.9) by adding the execution demand of carry-in jobs. For the sake of readability, the proof for this theorem is included in Appendix B.2.

Corollary 4.12 (WCDFP with Carry-In)**.** *Under the same setup as in Theorem 4.11, the WCDFP of task τ under uniprocessor preemptive T-FP scheduling is at most*

$$\inf_{0<t\leq D_\tau} \mathbb{P}(S_t > t). \tag{4.16}$$

The second method quantifies the interference of the higher-priority tasks by the sum of $\left\lceil \frac{t}{T_{\tau'}} \right\rceil$ execution demands of each higher-priority task τ'. However, the previous counterexample shows that including only the execution demand of the first $\left\lceil \frac{t}{T_{\tau'}} \right\rceil$ jobs of τ' is unsound and that some additional jobs have to be considered. This method conquers this issue by considering the *inflation* of the execution time of some jobs in the random variables to ensure that the WCRTEP or the WCDFP is correctly bounded.

The idea behind inflation is as follows: Suppose that we have to consider $\lambda^t_{\tau'}$ jobs of τ' in the analysis window (to be defined later). We randomly sample the execution demand of $\lambda^t_{\tau'}$ jobs of τ' but only the *largest* $\left\lceil \frac{t}{T_i} \right\rceil$ of them are considered in the response-time analysis. This procedure is formalized as follows.

Definition 4.13. *For any positive integers a and b with $a \leq b$, a-*job execution time inflation of $\tau' \in \mathbb{T}$ out of b jobs *is a sampling process which first* samples *b random variables of the execution time of task τ' and then* selects *the a* largest *values among the samples as the inflated execution time. We denote by* SAI(τ', a, b) *(named after* sample and inflate*) the random variable which provides the sum of the a selected samples. The random variable can be formulated as:*

$$\mathrm{SAI}(\tau', a, b) = \max \left\{ \sum_{c \in S} c \;\middle|\; S \subseteq \left\{ \bar{c}^{\bullet}_{\tau'(1)}, \dots, \bar{c}^{\bullet}_{\tau'(b)} \right\}, |S| = a \right\} \tag{4.17}$$

The following theorem shows that it is sound to sample $\left\lceil \frac{t + \sum_{\tau'' \in hp(\tau) \setminus hp(\tau')} D_{\tau''}}{T_{\tau'}} \right\rceil$ jobs and inflate $\left\lceil \frac{t}{T_{\tau'}} \right\rceil$ jobs of each $\tau' \in hp(\tau)$ when analyzing the WCRTEP of τ.

Theorem 4.14 (WCRTEP with Inflation)**.** *Consider a system of constrained-deadline sporadic real-time tasks where a job is directly aborted when missing its deadline. The WCRTEP of task τ for a target value $R \in (0, T_\tau]$ under uniprocessor preemptive T-FP scheduling is at most*

$$\inf_{0 < t \leq R} \mathbb{P}\left(\bar{c}^{\bullet}_{\tau(1)} + \sum_{\tau' \in hp(\tau)} SAI\left(\tau', \left\lceil \frac{t}{T_{\tau'}} \right\rceil, \lambda^t_{\tau'}\right) > t \right), \tag{4.18}$$

where $\lambda^t_{\tau'}$ is $\left\lceil \frac{t + \sum_{\tau'' \in hp(\tau) \setminus hp(\tau')} D_{\tau''}}{T_{\tau'}} \right\rceil$.

Again, for the sake of readability, the proof for this theorem is in Appendix B.2.

Corollary 4.15 (WCDFP with Inflation)**.** *Under the same setup of Theorem 4.14, the WCDFP of task τ under uniprocessor preemptive T-FP scheduling is at most*

$$\inf_{0 < t \leq D_\tau} \mathbb{P}\left(\bar{c}^{\bullet}_{\tau(1)} + \sum_{\tau' \in hp(\tau)} SAI\left(\tau', \left\lceil \frac{t}{T_{\tau'}} \right\rceil, \lambda^t_{\tau'}\right) > t \right). \tag{4.19}$$

We demonstrate that these two methods do not dominate each other by providing two concrete task sets, one where carry-in provides a better bound on the WCDFP and one where inflation provides a better bound.

Example 4.16 (Inflation Outperforms Carry-In)**.** *We re-examine the scenario from the previously provided counterexample (here abstracted to the sporadic task model and using pWCET). In particular, we consider the implicit-deadline task set $\mathbb{T} = \{\tau_1, \tau_2\}$ with two tasks described by:*

- $\tau_1 \in \mathrm{Spor}(4) \cap \mathrm{pWCET}^{\mathrm{iid}}(\mathcal{D}_{\tau_1})$
- $\tau_2 \in \mathrm{Spor}(4.4) \cap \mathrm{pWCET}^{\mathrm{iid}}(\mathcal{D}_{\tau_2})$

where $\mathcal{D}_{\tau_1}(t) = 1_{t\geq 1} \cdot 0.9 + 1_{t\geq 2.5} \cdot 0.1$ and $\mathcal{D}_{\tau_2}(t) = 1_{t\geq 3} \cdot 1.0$. Task τ_2 is under analysis. It is shown in Section 4.3.2 that the WCDFP for τ_2 is at least 0.19.

First, we compute an upper bound on the WCDFP by applying the carry-in *method. By Corollary 4.12, the WCDFP of τ_2 is upper bounded by*

$$\inf_{0<t\leq D_{\tau_2}} \mathbb{P}(S_t > t), \tag{4.20}$$

where S_t is the sum of the execution time of one job of τ_2 and $\left\lceil \frac{t+D_{\tau_1}}{T_{\tau_1}} \right\rceil$ jobs of the higher-priority task τ_1. For all $t \in (0, D_{\tau_2}] = (0, 4.4]$, we have $\left\lceil \frac{t+D_{\tau_1}}{T_{\tau_1}} \right\rceil = \left\lceil \frac{t+4}{4} \right\rceil \geq 2$. Since the execution time of every job of τ_2 is 3 and the execution time of jobs of τ_1 is at least 1, we obtain $S_t \geq 5 > t$ for all $t \in (0, D_{\tau_2}]$. Hence, $\inf_{0<t\leq D_{\tau_2}} \mathbb{P}(S_t > t) = 1.0$ and the carry-in method states that the WCDFP of τ_2 is at most 1.0.

In this case, the inflation *method provides a tighter result. By Corollary 4.15, the WCDFP of τ_2 is upper bounded by*

$$\inf_{0<t\leq D_{\tau_2}} \mathbb{P}\left(\bar{c}^{\bullet}_{\tau_2(1)} + SAI(\tau_1, \left\lceil \frac{t}{T_{\tau_1}} \right\rceil, \lambda^t_{\tau_1}) > t\right), \tag{4.21}$$

where $\lambda^t_{\tau_1} = \left\lceil \frac{t+D_{\tau_1}}{T_{\tau_1}} \right\rceil$. The values of $\left\lceil \frac{t}{T_{\tau_1}} \right\rceil$ and $\lambda^t_{\tau_1}$ are dependent on t as follows:

$$\left\lceil \frac{t}{T_{\tau_1}} \right\rceil = \begin{cases} 1 & t \in (0,4] \\ 2 & t \in (4, 4.4] \end{cases} \qquad \lambda^t_{\tau_1} = \begin{cases} 2 & t \in (0,4] \\ 3 & t \in (4, 4.4] \end{cases} \tag{4.22}$$

i.e., checking Equation (4.21) for $t = 4$ and $t = 4.4$ is sufficient.

For $t = 4$, the random variable $SAI\left(\tau_1, \left\lceil \frac{t}{T_{\tau_1}} \right\rceil, \lambda^t_{\tau_1}\right)$ is $SAI(\tau_1, 1, 2)$ which returns 1 if both sampled jobs have execution of 1, i.e., with probability $0.9 \cdot 0.9 = 0.81$, and 2.5 with probability $1 - 0.81 = 0.19$. If $SAI(\tau_1, 1, 2)$ returns 1, then $\bar{c}^{\bullet}_{\tau_2(1)} + SAI(\tau_1, 1, 2)$ returns 4 which is $\leq t = 4$. If $SAI(\tau_1, 1, 2)$ returns 2.5, then $\bar{c}^{\bullet}_{\tau_2(1)} + SAI(\tau_1, 1, 2)$ returns 5.5 which is $> t = 4$. Hence, $\mathbb{P}(\bar{c}^{\bullet}_{\tau_2(1)} + SAI(\tau_1, \left\lceil \frac{t}{T_{\tau_1}} \right\rceil, \lambda^t_{\tau_1}) > t) = 0.19$ for $t = 4$.

For $t = 4.4$, $SAI(\tau_1, \left\lceil \frac{t}{T_{\tau_1}} \right\rceil, \lambda^t_{\tau_1})$ is $SAI(\tau_1, 2, 3)$ which returns at least 2. Therefore, $\bar{c}^{\bullet}_{\tau_2(1)} + SAI(\tau_1, 1, 2)$ returns at least 5 which is $> t = 4.4$ in all cases. Thus, the probability $\mathbb{P}(\bar{c}^{\bullet}_{\tau_2(1)} + SAI(\tau_1, \left\lceil \frac{t}{T_{\tau_1}} \right\rceil, \lambda^t_{\tau_1}) > t)$ for $t = 4.4$ is 1.0.

The WCDFP from the inflation method is at most 0.19, which is the lower bound from the counterexample, i.e., the inflation method provides the exact WCDFP for this example.

In fact, the inflation method (Theorem 4.14) is always better than the carry-in method (Theorem 4.11) when considering only two tasks.

Theorem 4.17. *If there are only two tasks in $\mathbb{T} = \{\tau_1, \tau_2\}$, with τ_1 having higher priority than τ_2, then*

$$\mathbb{P}\left(\bar{c}^{\bullet}_{\tau_2(1)} + SAI\left(\tau_1, \left\lceil \frac{t}{T_{\tau_1}} \right\rceil, \left\lceil \frac{t + D_{\tau_1}}{T_{\tau_1}} \right\rceil\right) > t\right) \leq \mathbb{P}\left(S_t > t\right) \tag{4.23}$$

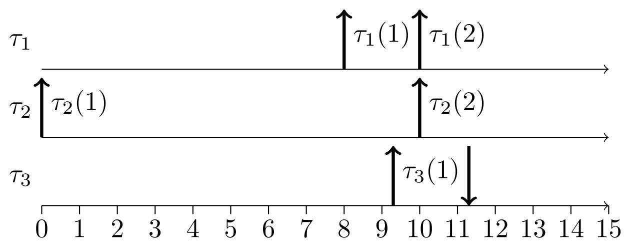


Figure 4.10.: Exemplary release pattern of the task set where the carry-in method outperforms the inflation method. Release times of the jobs are as follows: $r^\omega_{\tau_1(1)} = 8$, $r^\omega_{\tau_1(2)} = 10$, $r^\omega_{\tau_2(1)} = 0$, $r^\omega_{\tau_2(2)} = 10$, and $r^\omega_{\tau_3(1)} = 9.3$.

for all t with $0 < t \leq D_{\tau_2}$. That is, Theorem 4.14 dominates Theorem 4.11 when there are only two tasks in $\mathbb{T}$.

Proof. By definition, since $\lambda^t_{\tau_1} = \left\lceil \frac{t+D_{\tau_1}}{T_{\tau_1}} \right\rceil \geq \left\lceil \frac{t}{T_{\tau_1}} \right\rceil$, we have

$$\mathbb{P}\left(\bar{c}^\bullet_{\tau_2(1)} + SAI\left(\tau_1, \left\lceil \frac{t}{T_{\tau_1}} \right\rceil, \left\lceil \frac{t+D_{\tau_1}}{T_{\tau_1}} \right\rceil\right) > t\right) \tag{4.24}$$

$$\leq \mathbb{P}\left(\bar{c}^\bullet_{\tau_2(1)} + SAI\left(\tau_1, \left\lceil \frac{t+D_{\tau_1}}{T_{\tau_1}} \right\rceil, \left\lceil \frac{t+D_{\tau_1}}{T_{\tau_1}} \right\rceil\right) > t\right) \tag{4.25}$$

$$= \mathbb{P}\Big(\bar{c}^\bullet_{\tau_2(1)} + \sum_{q=1}^{\left\lceil \frac{t+D_{\tau_1}}{T_{\tau_1}} \right\rceil} \bar{c}^\bullet_{\tau_1(q)} > t\Big) = \mathbb{P}(S_t > t), \tag{4.26}$$

which leads to the condition in Equation (4.23). □

On the other hand, considering more than 2 tasks, there are cases where the carry-in method outperforms the inflation method.

Example 4.18 (Carry-In Outperforms Inflation)**.** *We consider the implicit-deadline task set $\mathbb{T} = \{\tau_1, \tau_2, \tau_3\}$ described by:*

- $\tau_1 \in \mathrm{Spor}(2) \cap \mathrm{pWCET}^{\mathrm{iid}}(D_{\tau_1})$ *with* $D_{\tau_1}(t) = 1_{t\geq 0.2} \cdot 0.9 + 1_{t\geq 2} \cdot 0.1$
- $\tau_2 \in \mathrm{Spor}(10) \cap \mathrm{pWCET}^{\mathrm{iid}}(D_{\tau_2})$ *with* $D_{\tau_2}(t) = 1_{t\geq 0.2} \cdot 0.9 + 1_{t\geq 10} \cdot 0.1$
- $\tau_3 \in \mathrm{Spor}(2) \cap \mathrm{pWCET}^{\mathrm{iid}}(D_{\tau_3})$ *with* $D_{\tau_3}(t) = 1_{t\geq 1} \cdot 1.0$

We assume that the task set is ordered by priority and that task τ_3 is under analysis. For a WCDFP lower bound we consider the release pattern shown in Figure 4.10. Under this release pattern, job $\tau_3(1)$ has a deadline miss if and only if at least one of the jobs $\tau_1(1)$, $\tau_1(2)$, $\tau_2(1)$, and $\tau_2(2)$ executes with the WCET. The WCDFP of τ_3 is therefore lower bounded by $1 - 0.9 \cdot 0.9 \cdot 0.9 \cdot 0.9 = 0.3439$.

First, we compute an upper bound on the WCDFP by applying the carry-in *method. By Corollary 4.12, the WCDFP of τ_3 is upper bounded by $\inf_{0<t\leq D_{\tau_3}} \mathbb{P}(S_t > t)$, where*

S_t is the sum of the execution time of one job of τ_3, $\left\lceil \frac{t+D_{\tau_2}}{T_{\tau_2}} \right\rceil$ jobs of the higher-priority task τ_2, and $\left\lceil \frac{t+D_{\tau_1}}{T_{\tau_1}} \right\rceil$ jobs of the higher-priority task τ_1. For all $t \in (0, D_{\tau_3}] = (0,2]$, $\left\lceil \frac{t+D_{\tau_2}}{T_{\tau_2}} \right\rceil = \left\lceil \frac{t+10}{10} \right\rceil = 2$ and $\left\lceil \frac{t+D_{\tau_1}}{T_{\tau_1}} \right\rceil = \left\lceil \frac{t+2}{2} \right\rceil = 2$. Therefore, S_t returns the sum of the execution demand of one job of τ_3, two jobs of τ_2, and two jobs of τ_1. We only get $S_t \leq 2$ if both jobs of τ_2 and both jobs of τ_1 execute the smaller execution time. This case occurs with probability 0.9^4. Hence, $\inf_{0<t\leq D_{\tau_3}} \mathbb{P}(S_t > t) = 1 - 0.9^4$, i.e., the upper bound on the WCDFP of τ_3 obtained by the carry-in method is exact.

For the inflation *method, by Corollary 4.15 the WCDFP of τ_3 is upper bounded by*

$$\inf_{0<t\leq D_{\tau_3}} \mathbb{P}\left(\bar{c}^{\bullet}_{\tau_3(1)} + SAI\left(\tau_2, \left\lceil \frac{t}{T_{\tau_2}} \right\rceil, \lambda^t_{\tau_2}\right) + SAI\left(\tau_1, \left\lceil \frac{t}{T_{\tau_1}} \right\rceil, \lambda^t_{\tau_1}\right) > t\right), \tag{4.27}$$

where

$$\lambda^t_{\tau_i} = \left\lceil \frac{t + \sum_{\tau'' \in hp(\tau_3)\setminus hp(\tau_i)} D_{\tau''}}{T_{\tau_i}} \right\rceil \text{ for } i = 1, 2.$$

For all $t \in (0, D_{\tau_3}] = (0,2]$, we have $\left\lceil \frac{t}{T_{\tau_2}} \right\rceil = \left\lceil \frac{t}{10} \right\rceil = 1$, $\left\lceil \frac{t}{T_{\tau_1}} \right\rceil = \left\lceil \frac{t}{2} \right\rceil = 1$, and

$$\lambda^t_{\tau_2} = \left\lceil \frac{t + D_{\tau_2}}{T_{\tau_2}} \right\rceil = \left\lceil \frac{t+10}{10} \right\rceil = 2,$$
$$\lambda^t_{\tau_1} = \left\lceil \frac{t + D_{\tau_2} + D_{\tau_1}}{T_{\tau_1}} \right\rceil = \left\lceil \frac{t+10+2}{2} \right\rceil = 7.$$

Therefore, the probability from Equation (4.27) can be formulated as

$$\mathbb{P}(\bar{c}^{\bullet}_{\tau_3(1)} + SAI(\tau_2, 1, 2) + SAI(\tau_1, 1, 7) > 2). \tag{4.28}$$

The random variable $SAI(\tau_2, 1, 2)$ returns the maximum execution demand of 2 jobs of τ_2, and the random variable $SAI(\tau_1, 1, 7)$ returns the maximum execution demand of 7 jobs of τ_1. Therefore, $SAI(\tau_2, 1, 2)$ is 0.2 if both jobs of τ_2 have an execution demand of 0.2, i.e., with probability 0.9^2, and 2 otherwise. Moreover, $SAI(\tau_1, 1, 7)$ is 0.2 if all 7 jobs of τ_1 have an execution demand of 0.2, i.e., with probability 0.9^7, and 10 otherwise. The random variable $\bar{c}^{\bullet}_{\tau_3(1)} + SAI(\tau_2, 1, 2) + SAI(\tau_1, 1, 7)$ is ≤ 2 if and only if $SAI(\tau_2, 1, 2) = 0.2$ and $SAI(\tau_1, 1, 7) = 0.2$, i.e., with probability 0.9^9. Hence, the WCDFP of τ_3 is upper bounded by $\mathbb{P}(\bar{c}^{\bullet}_{\tau_3(1)} + SAI(\tau_2, 1, 2) + SAI(\tau_1, 1, 7) > 2) = 1 - 0.9^9$ which is around 0.61. For this case, the upper bound on the WCDFP obtained by the inflation method is almost twice as large as the WCDFP obtained by the carry-in method.

4.3.4 Impact on Literature Results

Probabilistic response-time analysis suffers from high computational complexity, as it comprises two inherently difficult problems:

(i) How to efficiently bound the RTEP or the DFP of a specific job for a given release pattern (since a direct calculation via job-level convolution is intractable as time and space complexity are exponential with respect to the number of jobs in the pattern).

(ii) How to safely reduce the number of jobs/release patterns that must be considered in the analysis, as otherwise at least all jobs in the hyperperiod must be considered (see [Brü+21] for more detailed discussions).

Therefore, for an efficient analysis of the RTEP or DFP, solutions for both problems must be provided.

The results by Maxim and Cucu-Grosjean [MC13] and Chen and Chen [CC17] both provide two main contributions, one with respect to each of the problems: (i) a technique to efficiently bound the RTEP (DFP, respectively) for a job under analysis considering a specific release pattern, and (ii) a specific release pattern based on the critical instant that is claimed to provide the worst case among all jobs of a specific task under static-priority scheduling (i.e., to calculate the WCRTEP or the WCDFP). Although their concluded critical instant is unsound, the efficient calculations for a specific release pattern remains correct in both cases. Hence, results that directly build on the proposed unsound critical instant are affected, while their contributions to solutions for improving tractability remain unaffected. In the remainder of this section, how relevant publications in the literature are affected by our counterexample.

Direct Adoption of Unsound Critical Instant While the number of considered release patterns must be reduced to allow efficient computation, the derived analysis might become unsound, since the response-time distribution under the unsound critical instant is not necessarily the maximum.

For sensor networks, several results by Ren et al. analyze the response-time distribution, and Theorem 3.1 from [Ren+15] as well as Theorem 1 from [Ren+19] are affected. Their analysis assumes that the unsound critical instant from [MC13] is the worst-case scenario for bounding the response-time distribution. They derived a simulation-based analysis and resolved the intractability issue by abstracting it as an additional probabilistic function. However, the abstraction (e.g., Def 4.3 and 4.4 in [Ren+15]) still relies on the assumption of the unsound critical instant to limit the state space.

For mixed criticality systems [BBD11], Maxim et al. [Max+17] adapted the probabilistic response-time analysis derived in [MC13] as a backbone (i.e., Equations (4)–(6) in [MC13]) to build up several analyses for tasks with different criticality in different types of scheduling schemes. The influence of the unsound critical instant propagates by the direct adaption without further modifications. Therefore, the corresponding probabilistic response-time analysis and the schedulability test in [Max+17] are unsound.

The unsound critical instant was adopted by Chen et al. [CBC18] to efficiently calculate the deadline miss rate, assuming that a job is never aborted after any deadline miss. Their approach partitions the schedule into busy intervals and analyzes the probabilities of individual cases. They utilized the unsound critical instant as their analysis backbone, and, therefore, concluded an unsound worst-case miss rate analysis. The methods presented in Section 4.3.3 can unfortunately not be adopted to fix their approach, as we assume that jobs are aborted right after missing their deadlines, whilst *their problem definition disallows such a treatment.*

Techniques for Efficient Calculation When considering a specific job under a given release pattern, a direct computation through naïve convolution is intractable. The reason is that the time and space complexity is exponential in the number of jobs released in the considered interval, which can easily be hundreds or thousands of jobs for practical systems. Hence, several distinct approaches have been proposed to mitigate or even avoid these issues (by trading calculation speed for pessimism), e.g., down-sampling approaches [MPN21; MC13; Max+12; RH10], concentration inequalities [Brü+18; CC17; Che+19c], task-level convolution [Brü+18], and the Monte Carlo response-time analysis [BBB21].

When calculating the probabilistic response time of a specific job using convolution with down-sampling [MC13; Max+12; RH10] or the Monte Carlo response-time analysis [BBB21], the unsound critical instant is merely adopted as a specific evaluation scenario. Although the considered scenario is not the worst-case scenario, the technical contribution hence remains unaffected.

A similar misconception can be found in Theorem 4.10 (i.e., Theorem 1 from [CC17]), where the accumulated workload of higher-priority tasks also overlooks potential carry-in jobs. The analytical upper bounds by Chen and Chen [CC17], von der Brüggen et al. [Brü+18], and Chen et al. [Che+19c] as well as the task-level convolution by von der Brüggen et al. [Brü+18] utilize the same concept of accumulated workload.

The efficient calculation techniques mentioned above can still directly be applied to upper-bound the probabilistic response-time distribution (for [BBB21; MC13; RH10]) or the WCRTEP (for [Brü+18; CC17; Che+19c]) by replacing the adoption of the unsound CIT in these results with the sound analyses in Theorem 4.11 or Theorem 4.14.

4.4 Summary and Open Problems

In this chapter, we provided three counterexamples for well-established literature results. Specifically, in Section 4.1 we showed that slack enforcement mechanisms can provoke deadline misses for self-suspending tasks, in Section 4.2 we disproved a blocking-based schedulability test for self-suspending tasks, and in Section 4.3 we showed that the classical extension of the CIT to probabilistic scenarios is flawed. Hence, this chapter emphasizes the need for careful analysis and rigorous proofs.

The first two counterexamples close the unresolved issues of the review paper on self-suspending tasks by Chen et al. [Che+19b]. Hence, this concludes their investigation on the literature of self-suspending tasks before 2019. However, the underlying questions of the two papers under examination remain unclear: Can we shape self-suspending tasks such that they behave like classical non-self-suspending tasks? Can we analyze self-suspension as blocking under EDF scheduling?

Regarding the CIT, we also provided two safe approaches, namely carry-in and inflation, that can be used to extend the CIT to probabilistic scenarios. The safe approaches were picked up as a backbone to develop new research results regarding the analysis of probabilistic scenarios [MNP22; Mar+23]. Still, the proposed approaches are over-approximations, and it is unclear how close these over-approximations are to exact results or how to design a tractable exact analysis.

While the definite source of misconception for the three literature results investigated in this chapter are unknown, we believe that a lack of careful property-based analysis might have encouraged unsafe examination. Specifically, the common approach is to define the system model under analysis and only concretize system behavior where necessary (intuitively a top-down approach). However, this approach conceals the actual system behavior, favoring missing assumptions or limitation to unnecessary properties. Therefore, in this dissertation, we follow a more error-proof bottom-up strategy by deriving analytical properties from possible system evolutions.

5 Self-Suspension

The underlying assumption in typical response-time analysis and schedulability analysis is that no job can yield its ready state between its release and completion. In many real-world applications however, this model is insufficient to describe application demands. For instance, offloading parts of the computation to hardware accelerators [Don+18; Liu+14] or the partitioned scheduling of parallel Directed Acyclic Graph (DAG) tasks with subjob partitioning [Fon+16] are cases in which a job yields its ready state while waiting for offloaded computations or the completion of preceding subjobs before resuming to the ready state again. In the literature, such behavior is referred to as *self-suspension* as a job may *suspend* itself from the ready state and thus be exempted from the scheduling for the suspension duration. This behavior makes it non-trivial to resort to established concepts such as the busy-interval analysis for self-suspending task sets, which is required to analyze the Worst-Case Response Time (WCRT) of tasks, especially in the presence of backlog, e.g., for arbitrary-deadline task sets. The two predominantly studied task models for self-suspending tasks are the segmented and the dynamic self-suspending task model, as introduced in Section 3.1.3.

In this chapter, we discuss our analytical results for self-suspending tasks. To that end, in Section 5.1 we present analyses of the schedulability of self-suspending task sets. Section 5.2 discusses mechanisms to avoid timing anomalies for self-suspending tasks. The results of this chapter focus on preemptive scheduling on a uniprocessor system.

5.1 Schedulability Analysis

As supported by the existence of several flawed literature results, the analysis of WCRT or schedulability of self-suspending tasks is a challenging endeavor. The reason is that most classical analysis techniques, like the Critical Instant Theorem (CIT) [LL73], Time-Demand Analysis (TDA) [JP86; LSD89], or the demand bound function [BMR90], are based on the assumption that a job, after it is released, is either executed or waiting to be executed in the ready queue until it finishes. Contrarily, a job of a *self-suspending* task may release the processor before being completed, for instance when waiting to get access to a shared resource or offloading computation to an external device, and continue its execution later on. The irregular interference behavior of self-suspending task systems makes it mandatory to construct WCRT analyses from fundamental principles.

In this section, we provide analyses of the schedulability of self-suspending tasks under different scheduling algorithms, focusing on preemptive uniprocessor scheduling of dynamic self-suspending tasks. We cover results under Task-level Fixed-Priority (T-FP) scheduling in Section 5.1.1, under Earliest-Deadline-First (EDF) scheduling in Section 5.1.2 and under EDF-Like (EL) scheduling in Section 5.1.3. Moreover, an evaluation framework of aggregated bounds from the literature is presented in Section 5.1.4.

5.1.1 Under Task-level Fixed-Priority Scheduling

For dynamic self-suspending tasks under Task-level Fixed-Priority (T-FP) scheduling current analyses solely focus on periodic or sporadic task models [AB04; CNH16; Hua+15; Kim+95]. However, the results in [AB04; Kim+95] have been disproved [Che+19b] due to the fact that the classical CIT does not hold for self-suspending tasks. The current state-of-the-art analysis for dynamic self-suspending tasks under the sporadic task model is given by Chen et al. [CNH16]. In particular, they developed a unifying response-time analysis, that allows to either include the suspension times of higher-priority tasks explicitly in the analysis or include more workload modeled by a jitter term induced by self-suspension. However, their analysis is limited to constrained-deadline tasks, i.e., the deadline must be no more than the minimum inter-arrival time. To the best of our knowledge it is an open problem if a busy-interval equivalent concept can be established for self-suspending arbitrary-deadline task systems. Moreover, it is unclear how many jobs must be considered in the analysis even if a busy-interval concept would exist.

In this section, we present a suspension-aware busy-interval analysis for dynamic self-suspension tasks where the inter-arrival time of subsequent jobs can be bounded by an *arrival curve* (see Section 2.4.2). Based on the general analysis, we provide WCRT analyses and hence sufficient schedulability tests for sporadic self-suspension task systems with arbitrary deadlines. To the best of our knowledge, the analysis presented in this work is the first suspension-aware analysis for arbitrary-deadline sporadic real-time task systems and tasks abstracted with arrival curves under T-FP scheduling. The contributions of this section are as follows:

- We provide the first WCRT analysis for self-suspending real-time tasks described by arrival curves in Section 5.1.1.1 by extending the concept of busy-interval by Lehoczky [Leh90] to the *suspension-aware busy-interval.* Specifically, the analysis extends the window of interest in a way such that at most *one self-suspending job* of each higher-priority task has to be considered in the analysis. The interference from higher-priority self-suspending tasks can be arbitrarily modelled with one of two types of carry-in terms, in which one has self-suspending behavior and one does not.

- We derive a schedulability test for the special case where the worst-case (upper) arrival curve of a task is periodic, i.e., the classical sporadic real-time tasks, in Section 5.1.1.2.

- In Section 5.1.1.3 we discuss the trade-off of the two carry-in terms utilized in the analysis.
- We compare our approach with the state of the art for arbitrary-deadline tasks in Section 5.1.1.4, modelling the arrival curve for tasks with release jitter. Moreover, we demonstrate that our analysis exploits the optimism that is obtained when increasing the tasks' relative deadlines.

The results presented in this section appeared in RTSS 2021 [GUC21].

5.1.1.1 Analysis with Arrival Curves

We consider a task set $\mathbb{T} = \{\tau_1, \dots, \tau_n\}$ consisting of n tasks abstracted with dynamic self-suspension and arrival curves. That is, each task $\tau_i \in \mathbb{T}$ can be abstracted as

$$\tau_i \in \mathrm{Arr}(\alpha_{\tau_i}) \cap \mathrm{DynSS}(C_{\tau_i}, S_{\tau_i}). \tag{5.1}$$

We assume that the task set is ordered according to the priority level, i.e., a task τ_i has higher priority than a task τ_j if and only if $i < j$. In the following, we denote by $\mathcal{A}$ the preemptive T-FP scheduling algorithm. Hence, for a system evolution $\omega \in \Omega$, the corresponding schedule is $[\mathcal{S}] = \mathcal{A}(\omega)$.

On the basis of the WCRT $R^{\mathcal{A}}_{\tau_i}$ of a given task τ_i, the maximal amount of concurrently pending workload that is generated by τ_i is given by

$$C_i^* := \min(\alpha_{\tau_i}(R^{\mathcal{A}}_{\tau_i}) \cdot C_{\tau_i}, R^{\mathcal{A}}_{\tau_i}). \tag{5.2}$$

In particular, this means that at each time there is no more than $C_i^* \in [C_{\tau_i}, R^{\mathcal{A}}_{\tau_i}]$ amount of pending workload of jobs of τ_i in a ready state (ready queue). The reason for this is that the maximal number of concurrent jobs of τ_i which are released but not yet finished is upper bounded by $\alpha_{\tau_i}(R^{\mathcal{A}}_{\tau_i})$. Each of them can have pending workload of up to C_{τ_i} time units. This shows $C_i^* \leq \alpha_{\tau_i}(R^{\mathcal{A}}_{\tau_i}) \cdot C_{\tau_i}$. Moreover, if the concurrently pending workload at some time instant t would be higher than $R^{\mathcal{A}}_{\tau_i}$, then the most recently released job would finish no earlier than at $t + R^{\mathcal{A}}_{\tau_i}$, i.e., $R^{\mathcal{A}}_{\tau_i}$ would not be an upper bound of the WCRT of τ_i, which is a contradiction. Therefore, $C_i^* \leq R^{\mathcal{A}}_{\tau_i}$ as well.

In the remainder of this section, we derive a sufficient schedulability test. The proof structure is inspired by the constrained-deadline analysis in [CNH16]. However, the characterizations of arbitrary-deadline tasks are completely different from constrained-deadline tasks, and none of their lemmas can be directly applied without proper modifications. We start by considering a system evolution $\omega \in \Omega$ and the corresponding schedule $[\mathcal{S}] := \mathcal{A}(\omega)$ of the task set $\mathbb{T}$. Iteratively, we consider the task $\tau_k,\ k = 1, \dots, n$ and provide an upper bound on the WCRT for τ_k under the assumption that the upper bound for $\tau_1, \dots, \tau_{k-1}$ has been derived beforehand.

Let τ_k be the task under analysis. In the following, we consider some job $\tau_k(\ell)$ and bound its response time. We partition the higher-priority tasks $\tau_1, \dots, \tau_{k-1}$ into two sets denoted by $\mathbb{T}_0$ and $\mathbb{T}_1$. Our analysis assumes that this partition is given and provides a

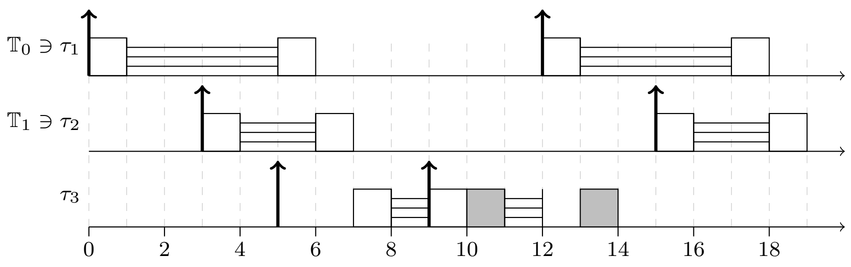


Figure 5.1.: Schedule $[\mathcal{S}]$ of 3 tasks from Example 5.2. The execution segments of job $\tau_3(2)$ are marked gray.

valid sufficient schedulability test. Depending on the partition, the analysis approach is different: For each task, we either cut or extend the analysis window. To find a suitable partition, one can examine all of them. We explain how to find suitable partitions in Section 5.1.1.3.

The proof is divided into four steps:

Step 1: Reducing the schedule $[\mathcal{S}]$ to $[\mathcal{S}_1]$ by removing jobs that do not contribute to the response time of $\tau_k(\ell)$.

Step 2: Analyzing the reduced schedule $[\mathcal{S}_1]$ and proving useful properties for Step 3 and Step 4.

Step 3: Providing a response-time upper bound for $\tau_k(\ell)$.

Step 4: Using the response-time bound from Step 3 to derive a schedulability test.

Our analysis is based on the *suspension-aware busy-interval* of τ_k defined as follows:

Definition 5.1 (Suspension-aware busy-interval)**.** *The half-opened interval $[v, w)$ is a* suspension-aware busy-interval *of τ_k if there is pending workload of task τ_k at all times during the interval $[v, w)$. That is, $[v, w)$ is a suspension-aware busy-interval if and only if $[v, w) \subseteq \bigcup_{j\in\mathbb{N}}[r^{\omega}_{\tau_k(j)}, f^{\mathcal{A}(\omega),\omega}_{\tau_k(j)})$.*

*Let $a \in \mathbb{N}$. The job $\tau_k(\ell)$ is the a-*th job in a suspension-aware busy-interval *of τ_k if $[r^{\omega}_{\tau_k(\ell-(a-1))}, f^{\mathcal{A}(\omega),\omega}_{\tau_k(\ell)})$ is a suspension-aware busy-interval of τ_k and all jobs of τ_k released before $r^{\omega}_{\tau_k(\ell-(a-1))}$ finish at latest at $r^{\omega}_{\tau_k(\ell-(a-1))}$.*

Example 5.2. *In Figure 5.1 we present an example schedule for $\mathbb{T} = \{\tau_1, \tau_2, \tau_3\}$ with $\mathbb{T}_0 = \{\tau_1\}$ and $\mathbb{T}_1 = \{\tau_2\}$ to give the reader guidance through the proof. The job $\tau_3(2)$ is under analysis. It is the second job in a suspension-aware busy-interval of τ_3.*

STEP 1: REDUCING THE SCHEDULE $[\mathcal{S}]$. First, we remove all tasks with lower priority than τ_k from the system. This does not affect the schedule of $\tau_1, \ldots, \tau_k$ as the lower-priority tasks are anyway preempted when there is pending workload of $\tau_1, \ldots, \tau_k$.

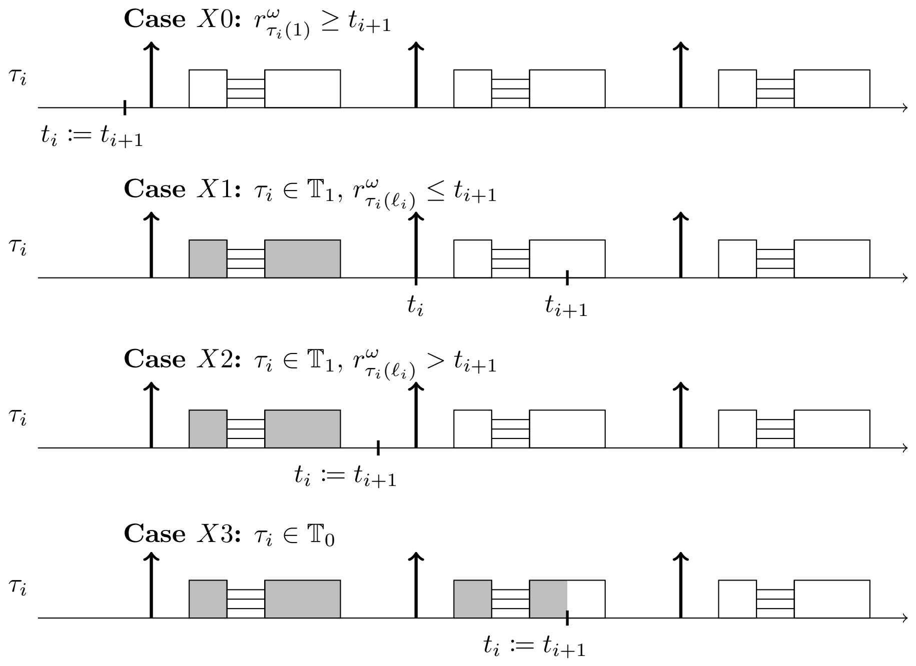

Case $X0$**:** $r^{\omega}_{\tau_i(1)} \geq t_{i+1}$
τ_i
$t_i := t_{i+1}$
Case $X1$**:** $\tau_i \in \mathbb{T}_1$, $r^{\omega}_{\tau_i(\ell_i)} \leq t_{i+1}$
τ_i
t_i
t_{i+1}
Case $X2$**:** $\tau_i \in \mathbb{T}_1$, $r^{\omega}_{\tau_i(\ell_i)} > t_{i+1}$
τ_i
$t_i := t_{i+1}$
Case $X3$**:** $\tau_i \in \mathbb{T}_0$
τ_i
$t_i := t_{i+1}$

Figure 5.2.: Procedure from $[\mathcal{S}_{i+1}]$ to $[\mathcal{S}_i]$ as in Step 1. Gray areas are removed from the schedule.

Afterwards, we remove further jobs from the schedule. In this step, the removal of jobs in the schedule has to be guaranteed not to affect the response time of $\tau_k(\ell)$ in the schedule.

Let $a \in \mathbb{N}$ such that $\tau_k(\ell)$ is the a-th job in a *suspension-aware busy-interval* of τ_k, as defined in Definition 5.1. We set $t_k := r^{\omega}_{\tau_k(\ell-(a-1))}$ as the beginning of the suspension-aware busy-interval and remove the jobs $\tau_k(1), \ldots, \tau_k(\ell - a)$ from the schedule. The removal of these jobs does not affect the schedule of the higher-priority tasks, as the higher-priority tasks would preempt job execution of τ_k anyway. Moreover, since τ_k has the lowest priority in the schedule $[\mathcal{S}]$ (after the treatment in the previous paragraph), there are no lower-priority tasks to be affected. The jobs released at or after t_k (including $\tau_k(\ell)$) are not affected as well, since all removed jobs are finished until time t_k.

We define ω_k and $[\mathcal{S}_k]$ to be the resulting reduced system evolution and reduced schedule, respectively. In the following, we describe how to derive $[\mathcal{S}_i]$, ω_i and t_i from $[\mathcal{S}_{i+1}]$, ω_{i+1} and t_{i+1} iteratively, for $i = k-1, k-2, \ldots, 1$. The main procedure is to *extend the analysis window* if $\tau_i \in \mathbb{T}_1$ and to *cut the overlapping job* of τ_i if $\tau_i \in \mathbb{T}_0$. More specifically, we distinguish four different cases as depicted in Figure 5.2. Please note that the four cases cover all possible scenarios.

Procedure from $[\mathcal{S}_{i+1}]$ to $[\mathcal{S}_i]$:

Case X0: All jobs of τ_i are released at or after t_{i+1}. In this case, we define $[\mathcal{S}_i] := [\mathcal{S}_{i+1}]$, $\omega_i := \omega_{i+1}$ and $t_i := t_{i+1}$.

The other three cases (**X1**, **X2**, and **X3**) involve scenarios in which there are jobs of

τ_i released before t_{i+1}. We denote by c_i^* the *residual workload* at time t_{i+1} of τ_i, i.e., the amount of remaining time that the processor needs to work on pending jobs of τ_i at time t_{i+1}. Moreover, let $\tau_i(\ell_i)$ be the first job of τ_i which finishes after t_{i+1}.

Case X1: $\tau_i \in \mathbb{T}_1$ and $r_{\tau_i(\ell_i)}^{\omega_{i+1}} \leq t_{i+1}$. In this case, we set $t_i := \max(f_{\tau_i(\ell_i-1)}^{[\mathcal{S}_{i+1}],\omega_{i+1}}, r_{\tau_i(\ell_i)}^{\omega_{i+1}})$ to be the maximum of the release of the first job that finishes after t_{i+1} and the finish of the previous job. We remove all jobs of τ_i before $\tau_{\tau_i(\ell_i)}$.

Case X2: $\tau_i \in \mathbb{T}_1, r_{\tau_i(\ell_i)}^{\omega_{i+1}} > t_{i+1}$. In this case, we set $t_i := t_{i+1}$ and remove all jobs of τ_i released before $\tau_i(\ell_i)$. Note that afterwards there is no job execution of or job release of τ_i before t_{i+1}.

Case X3: $\tau_i \in \mathbb{T}_0$. In this case, we define $t_i := t_{i+1}$. All jobs released after t_i remain unmodified in the schedule. All jobs released before t_i are replaced by *one* artificial job with execution time c_i^* and release t_i with the same priority and the same execution and suspension behavior that the other jobs had after t_{i+1}. In particular, the execution and suspension pattern of τ_i after t_i remains unchanged and there is no job release before t_i.

Please note that the transformation from $[\mathcal{S}_{i+1}]$ to $[\mathcal{S}_i]$ does not affect the response time of $\tau_k(\ell)$ due to the following reasoning: In the procedure from $[\mathcal{S}_{i+1}]$ to $[\mathcal{S}_i]$, only the schedule of τ_i before t_i is modified. Before t_i, there are no jobs of lower-priority tasks released (all of them are already removed in $[\mathcal{S}_{i+1}]$). Therefore, modifying the schedule of τ_i has no impact on the lower-priority tasks. As a result, the job $\tau_k(\ell)$ is not affected by the transformation from $[\mathcal{S}_{i+1}]$ to $[\mathcal{S}_i]$. We conclude the following.

Lemma 5.3. *The response time of $\tau_k(\ell)$ in $[\mathcal{S}]$ coincides with the response time of $\tau_k(\ell)$ in $[\mathcal{S}_1]$.*

Proof. In this subsection, we discussed that neither the transformation from $[\mathcal{S}]$ to $[\mathcal{S}_k]$ nor the transformation from $[\mathcal{S}_{i+1}]$ to $[\mathcal{S}_i]$ for any $i \in \{1, \ldots, k-1\}$ affects the response time of $\tau_k(\ell)$. Since the transformation from $[\mathcal{S}]$ to $[\mathcal{S}_1]$ is just a composition of the above transformations, this does not affect the response time of $\tau_k(\ell)$ as well. □

The procedure of this subsection is illustrated by the following example.

Example 5.4. *In Figure 5.3, we present the schedule $[\mathcal{S}_1]$ for the original schedule from Figure 5.1. We set t_3 to the release of the first job in the suspension-aware busy-interval of the job under analysis $\tau_3(2)$ at time 5. When going from $[\mathcal{S}_3]$ to $[\mathcal{S}_2]$, we set t_2 to time 3, according to Case X1. Finally, we set $t_1 := t_2$ and cut the job, as presented for Case X3. We obtain $[\mathcal{S}_1]$.*

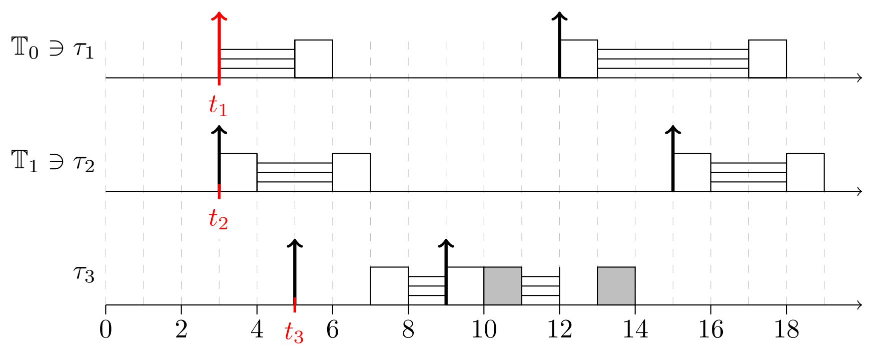


Figure 5.3.: Schedule $[\mathcal{S}_1]$ from Example 5.2 after Step 1.

STEP 2: ANALYZING $[\mathcal{S}_1]$. To deduce a WCRT of $\tau_k(\ell)$, in this step, we analyze the amount of time that the processor executes and idles in $[\mathcal{S}_1]$. The backbone for this step of the analysis is the fact that for any interval $[t_1, t)$ with $t_k \leq t \leq f^{[\mathcal{S}_1],\omega_1}_{\tau_k(\ell)}$, we have

$$\mathrm{idle}^{[\mathcal{S}_1]}(t_1, t) + \mathrm{exec}^{[\mathcal{S}_1]}(t_1, t) = t - t_1. \tag{5.3}$$

In the following, we derive upper bounds for $\mathrm{idle}^{[\mathcal{S}_1]}(t_1, t)$ and $\mathrm{exec}^{[\mathcal{S}_1]}(t_1, t)$, and conclude the response-time upper bound. We utilize

$$\mathrm{idle}^{[\mathcal{S}_1]}(t_1, t) = \sum_{i=1}^{k-1} \mathrm{idle}^{[\mathcal{S}_1]}(t_i, t_{i+1}) + \mathrm{idle}^{[\mathcal{S}_1]}(t_k, t) \tag{5.4}$$

$$\mathrm{exec}^{[\mathcal{S}_1]}(t_1, t) = \sum_{i=1}^{k} \mathrm{exec}^{[\mathcal{S}_1]}_{\tau_i}(t_1, t) \tag{5.5}$$

Moreover, we know that $\mathrm{exec}^{[\mathcal{S}_1]}_{\tau_i}(t_1, t) = \mathrm{exec}^{[\mathcal{S}_1]}_{\tau_i}(t_i, t)$ for all i since no job of τ_i is executed before t_i in $[\mathcal{S}_1]$.

Lemma 5.5 shows that for each segment $[t_i, t_{i+1})$ the idle time is upper bounded by the maximum suspension time S_{τ_i} in $[\mathcal{S}_1]$.

Lemma 5.5. *Consider the schedule $[\mathcal{S}_1]$.*

(i) Let $i \in \{1, \dots, k-1\}$. At any time instant[1] during $[t_i, t_{i+1})$ a job of τ_i is executed or suspends itself, or a job from a higher-priority task is executed.

(ii) At any time instant during $[t_k, f^{[\mathcal{S}_1],\omega_1}_{\tau_k(\ell)})$ a job of τ_k is executed or suspends itself, or a job from a higher-priority tasks is executed.

[1]When we say "*at any time instant ... holds*", we formally mean: *For all t there exists an $\varepsilon > 0$ such that during $(t, t+\varepsilon)$ ... holds.* We avoid this formalism in the proof of Lemma 5.5 for the sake of readability.

Proof. By the construction of t_i in Step 1, at any time instant during $[t_i, t_{i+1})$ there is pending work of τ_i. More specifically, either $t_i = t_{i+1}$ or there is a job of τ_i which is released no later than t_i and finishes after t_{i+1}. Therefore, at all time instants during $[t_i, t_{i+1})$ in the schedule $[\mathcal{S}]$ a job of τ_i is executed or suspends itself, or a job of a higher-priority task is executed. When we construct $[\mathcal{S}_1]$ from $[\mathcal{S}]$, during $[t_i, t_{i+1})$ we only remove execution and suspension from jobs of lower-priority tasks. Hence, (i) holds.

During $[t_k, f^{[\mathcal{S}_1],\omega_1}_{\tau_k(\ell)})$ in $[\mathcal{S}]$, there is a suspension-aware busy-interval of task τ_k, i.e., there is pending work of τ_k at all time instants during $[t_k, f^{[\mathcal{S}_1],\omega_1}_{\tau_k(\ell)})$. Similar as above, at all time instants during $[t_k, f^{[\mathcal{S}_1],\omega_1}_{\tau_k(\ell)})$ in $[\mathcal{S}]$ a job of τ_k is executed or suspends itself, or a job of a higher-priority task is executed. However, when $[\mathcal{S}_1]$ is constructed from $[\mathcal{S}]$, during $[t_k, f^{[\mathcal{S}_1],\omega_1}_{\tau_k(\ell)})$ only execution and suspension from jobs of lower-priority tasks is removed. Hence, (ii) holds. $\square$

This lemma implies that each time the processor idles during $[t_i, t_{i+1})$, a job of task τ_i suspends itself, and each time the processor idles during $[t_k, f^{[\mathcal{S}_1],\omega_1}_{\tau_k(\ell)})$, a job of task τ_k suspends itself. Let x_i be 1 if $\tau_i \in \mathbb{T}_1$ and 0 if $\tau_i \in \mathbb{T}_0$. We define by $\text{susp}_i(t_i, t_{i+1})$ the amount of time that jobs of τ_i suspend during $[t_i, t_{i+1})$ in $[\mathcal{S}_1]$.

Lemma 5.6. *For any* $i = 1, \dots, k-1$ *we have* $\text{idle}^{[\mathcal{S}_1]}(t_i, t_{i+1}) \leq x_i \cdot \text{susp}_i(t_i, t_{i+1}) \leq x_i \cdot S_{\tau_i}$.

Proof. If t_i and t_{i+1} coincide, then by definition the equations $\text{idle}^{[\mathcal{S}_1]}(t_i, t_{i+1}) = 0$ and $\text{susp}_i(t_i, t_{i+1}) = 0$ hold. Moreover, $x_i \cdot S_{\tau_i} \geq 0$, which concludes this case.

If t_i and t_{i+1} do not coincide, then $t_i < t_{i+1}$. This can only be achieved if Case X1 is applied when going from $[\mathcal{S}_{i+1}]$ to $[\mathcal{S}_i]$, i.e., $x_i = 1$. It remains to show that $\text{idle}^{[\mathcal{S}_1]}(t_i, t_{i+1}) \leq \text{susp}_i(t_i, t_{i+1}) \leq S_{\tau_i}$. Since $t_i = \max(f^{[\mathcal{S}_1],\omega_1}_{\tau_i(\ell_i-1)}, r^{\omega_1}_{\tau_i(\ell_i)})$, all jobs of τ_i prior to $\tau_i(\ell_i)$ finish at or before t_i. Moreover, $\tau_i(\ell_i)$ finishes after t_{i+1}. Hence, $\tau_i(\ell_i)$ is the only job of τ_i that suspends itself during $[t_i, t_{i+1})$, i.e., $\text{susp}_i(t_i, t_{i+1}) \leq S_{\tau_i}$. Due to Lemma 5.5, during $[t_i, t_{i+1})$ a job of τ_i suspends itself whenever the processor idles. Hence, $\text{idle}^{[\mathcal{S}_1]}(t_i, t_{i+1}) \leq \text{susp}_i(t_i, t_{i+1})$. $\square$

Utilizing the second part of Lemma 5.5, we provide a similar statement about the idle time during $[t_k, t)$ for any $t \in [t_k, f^{[\mathcal{S}_1],\omega_1}_{\tau_k(\ell)}]$. However, during this interval there are a many jobs of τ_k that may suspend themselves.

Lemma 5.7. *For any* $t \in [t_k, f^{[\mathcal{S}_1],\omega_1}_{\tau_k(\ell)}]$ *we have* $\text{idle}^{[\mathcal{S}_1]}(t_k, t) \leq a \cdot S_{\tau_k}$.

Proof. Due to Lemma 5.5, whenever the processor idles during the interval $[t_k, t)$, there is some job of τ_k that suspends itself. By the definition of t_k, the interval $[t_k, f^{[\mathcal{S}_1],\omega_1}_{\tau_k(\ell)})$ is a suspension-aware busy-interval of τ_k with a jobs. Hence, there can be at most a jobs of τ_k that suspend themselves during $[t_k, t) \subset [t_k, f^{[\mathcal{S}_1],\omega_1}_{\tau_k(\ell)})$ and $\text{idle}^{[\mathcal{S}_1]}(t_k, t) \leq a \cdot S_{\tau_k}$. $\square$

After an estimation of the idle time of the processor during $[t_1, t)$, we focus on the execution time. For each task $\tau_i \neq \tau_k$, we do this by setting Δ to $t - t_i$ and providing an upper bound for $\mathrm{exec}_{\tau_i}^{[\mathcal{S}_1]}(t_i, t_i + \Delta)$. In Lemma 5.8, we present a general bound which is applicable to all tasks. Afterwards, we consider the case $\tau_i \in \mathbb{T}_1$ in Lemma 5.11, we consider the case $\tau_i \in \mathbb{T}_0$ in Lemma 5.12, and we consider the case $\tau_i = \tau_k$ in Lemma 5.13.

Lemma 5.8. *Let τ_i in $\mathbb{T}$. For any $\Delta \geq 0$, we have*

$$\mathrm{exec}_{\tau_i}^{[\mathcal{S}_1]}(t_i, t_i + \Delta) \leq \alpha_{\tau_i}(\Delta + R_{\tau_i}^{\mathcal{A}}) \cdot C_{\tau_i} \tag{5.6}$$

where $R_{\tau_i}^{\mathcal{A}}$ is an upper bound on the WCRT of the task τ_i.

Proof. To be executed during the interval $[t_i, t_i + \Delta)$ a job of τ_i must be released before $t_i + \Delta$. Moreover, it must not be finished until t_i, which is only possible if it is released after $t_i - R_{\tau_i}^{\mathcal{A}}$. We conclude that only jobs that are released during the interval $(t_i - R_{\tau_i}^{\mathcal{A}}, t_i + \Delta)$ may be executed during $[t_i, t_i + \Delta)$. The number of those jobs is upper bounded by $\alpha_{\tau_i}(\Delta + R_{\tau_i}^{\mathcal{A}})$. Each of them can be executed for at most C_{τ_i} time units. □

For a precise proof of Lemma 5.11, 5.12 and 5.13, we first introduce the notation of interference I_i and derive useful properties in Lemma 5.10:

Definition 5.9 (Interference)**.** *For all $t \in \mathbb{R}$ and $\Delta \geq 0$, we define by*

$$I_i(t, t + \Delta) := \sum_{j \in \mathbb{N}, r_{\tau_i(j)}^{\omega_1} \in [t, t+\Delta)} \mathrm{exec}_{\tau_i(j)}^{[\mathcal{S}_1], \omega_1}(t, t + \Delta) \tag{5.7}$$

the interference *during the interval $[t, t+\Delta)$ from task τ_i, which is the amount of execution time during $[t, t + \Delta)$ from jobs of τ_i which are released during $[t, t + \Delta)$.*

Moreover, we define $I_i(\Delta)$ to be the maximum interference *from task τ_i during an interval of length Δ, i.e.,*

$$I_i(\Delta) = \begin{cases} \sup_t I_i(t, t + \Delta) & \Delta \geq 0 \\ 0 & \Delta < 0 \end{cases}. \tag{5.8}$$

Lemma 5.10. *For the maximum interference function I_i the following properties hold for all $\Delta \geq 0$:*

(i) $I_i(\Delta) \leq I_i(\Delta - \delta) + \delta$ *for all* $\delta \geq 0$

(ii) $I_i(\Delta) \leq \alpha_{\tau_i}(\Delta) \cdot C_{\tau_i}$

Proof. Let $\Delta, \delta \geq 0$ be fixed. During an interval of length δ, there can be at most δ amount of workload being executed. Therefore, for all t we have $I_i(t, t + \Delta) - I_i(t, t + \Delta - \delta) \leq \delta$. By using the supremum, we obtain the first part of the lemma.

The maximum number of job releases during an interval of length Δ is upper bounded by $\alpha_{\tau_i}(\Delta)$. Each of these jobs can be executed for at most C_{τ_i} time units. □

In the following three lemmas, we provide the upper bound for $\mathrm{exec}_{\tau_i}^{[\mathcal{S}_1]}(t_i, t_i + \Delta)$ if $\tau_i \in \mathbb{T}_1$, $\tau_i \in \mathbb{T}_0$ or $\tau_i = \tau_k$.

Lemma 5.11. *Let τ_i in $\mathbb{T}_1$. For any $\Delta \geq 0$, we have*

$$\text{exec}^{[\mathcal{S}_1]}_{\tau_i}(t_i, t_i + \Delta) \leq \alpha_{\tau_i}(\Delta + \max(R^{\mathcal{A}}_{\tau_i} - T_i, 0)) \cdot C_{\tau_i} \tag{5.9}$$

where $R^{\mathcal{A}}_{\tau_i}$ is the WCRT or an upper bound on the WCRT of the task τ_i, and $0 \leq T_i \leq \inf\{t \in \mathbb{R}_{\geq 0} \mid \alpha_{\tau_i}(t) \geq 2\}$ is a lower bound on the minimal time between two consecutive job releases.

Proof. Since $\tau_i \in \mathbb{T}_1$, t_i can be derived by **Case X0**, **Case X1** or **Case X2** during the procedure from $[\mathcal{S}_{i+1}]$ to $[\mathcal{S}_i]$.

If t_i is derived by **Case X0**, then all jobs of τ_i are released at or after t_i. Therefore, $\text{exec}^{[\mathcal{S}_1]}_{\tau_i}(t_i, t_i + \Delta) = I_i(t_i, t_i + \Delta) \leq I_i(\Delta) \leq \alpha_{\tau_i}(\Delta) \cdot C_{\tau_i}$. Due to the monotonicity of the arrival curve, we obtain the result from Equation (5.9).

If t_i is derived by **Case X1**, then all jobs of τ_i prior to $\tau_i(\ell_i)$ are finished at time t_i and are therefore not executed after t_i. If $t_i = r^{\omega_1}_{\tau_i(\ell_i)}$, then all jobs that execute after t_i are released at or after t_i, similar to **Case X0**. More specifically, we have:

$$\text{exec}^{[\mathcal{S}_1]}_{\tau_i}(t_i, t_i + \Delta) = I_i(t_i, t_i + \Delta) \leq I_i(\Delta) \leq \alpha_{\tau_i}(\Delta) \cdot C_{\tau_i} \tag{5.10}$$

$$\leq \alpha_{\tau_i}(\Delta + \max(R^{\mathcal{A}}_{\tau_i} - T_i, 0)) \cdot C_{\tau_i} \tag{5.11}$$

If $t_i = f^{[\mathcal{S}_1],\omega_1}_{\tau_i(\ell_i - 1)}$, then $\tau_i(\ell_i)$ is released no earlier than $t_i - R^{\mathcal{A}}_{\tau_i} + T_i$. We obtain:

$$\text{exec}^{[\mathcal{S}_1]}_{\tau_i}(t_i, t_i + \Delta) \leq I_i(t_i - R^{\mathcal{A}}_{\tau_i} + T_i, t_i + \Delta) \leq I_i(\Delta + R^{\mathcal{A}}_{\tau_i} - T_i) \tag{5.12}$$

$$\leq \alpha_{\tau_i}(\Delta + R^{\mathcal{A}}_{\tau_i} - T_i) \cdot C_{\tau_i} \tag{5.13}$$

Since the arrival curve α_{τ_i} is monotonically increasing, replacing $R^{\mathcal{A}}_{\tau_i} - T_i$ by $\max(R^{\mathcal{A}}_{\tau_i} - T_i, 0)$ yields the result.

If t_i is derived by **Case X2**, then all jobs of τ_i which are executed after t_i are released at or after t_i. Analogously to **Case X0**, we obtain $\text{exec}^{[\mathcal{S}_1]}_{\tau_i}(t_i, t_i + \Delta) = I_i(t_i, t_i + \Delta) \leq I_i(\Delta) \leq \alpha_{\tau_i}(\Delta) \cdot C_{\tau_i} \leq \alpha_{\tau_i}(\Delta + \max(R^{\mathcal{A}}_{\tau_i} - T_i, 0)) \cdot C_{\tau_i}$ as in Equation (5.9). □

We note that the bound from Lemma 5.11 is tighter than the bound from Lemma 5.8, since $\alpha_{\tau_i}(\Delta + \max(R^{\mathcal{A}}_{\tau_i} - T_i, 0)) \leq \alpha_{\tau_i}(\Delta + \max(R^{\mathcal{A}}_{\tau_i}, 0)) \leq \alpha_{\tau_i}(\Delta + R^{\mathcal{A}}_{\tau_i})$.

Lemma 5.12. *Let τ_i in $\mathbb{T}_0$. For any $\Delta \geq 0$, we have*

$$\text{exec}^{[\mathcal{S}_1]}_{\tau_i}(t_i, t_i + \Delta) \leq \alpha_{\tau_i}(\Delta - T_i + R^{\mathcal{A}}_{\tau_i} - C^*_i) \cdot C_{\tau_i} + C^*_i \tag{5.14}$$

where $R^{\mathcal{A}}_{\tau_i}$ is an upper bound on the WCRT of τ_i and $C^*_i := \min(\alpha_{\tau_i}(R^{\mathcal{A}}_{\tau_i}) \cdot C_{\tau_i}, R^{\mathcal{A}}_{\tau_i})$ is an upper bound on the maximum current workload.

Proof. Since $\tau_i \in \mathbb{T}_0$, t_i can be derived by **Case X0** or **Case X3** during the procedure from $[\mathcal{S}_{i+1}]$ to $[\mathcal{S}_i]$.

If t_i is derived by **Case X0**, then analogously to the proof of Lemma 5.11 we obtain $\text{exec}^{[\mathcal{S}_1]}_{\tau_i}(t_i, t_i + \Delta) \leq \alpha_{\tau_i}(\Delta) \cdot C_{\tau_i}$. For the arrival curve we have $\alpha_{\tau_i}(\Delta) \leq \alpha_{\tau_i}(\Delta - T_i) + 1$.

Due to the monotonicity of α_{τ_i} and since $R^{\mathcal{A}}_{\tau_i} - C^*_i \geq 0$, $\alpha_{\tau_i}(\Delta - T_i)$ is less than or equal to $\alpha_{\tau_i}(\Delta - T_i + R^{\mathcal{A}}_{\tau_i} - C^*_i)$. by using $C_{\tau_i} \leq C^*_i$ we obtain the result from Equation (5.14).

If t_i is derived by **Case X3**, then $c^*_i \leq C^*_i$ denotes the residual workload at time t_i. Let $\tau_i(\ell_i), \ldots, \tau_i(\ell_i+p)$ be the jobs that contribute to c^*_i, then $\tau_i(\ell_i+p)$ finishes no earlier than $f^{[\mathcal{S}_1],\omega_1}_{\tau_i(\ell_i+p)} \geq t_i + c^*_i$. Since $f^{[\mathcal{S}_1],\omega_1}_{\tau_i(\ell_i+p)} \leq r^{\omega_1}_{\tau_i(\ell_i+p)} + R^{\mathcal{A}}_{\tau_i} \leq r^{\omega_1}_{\tau_i(\ell_i+p+1)} - T_i + R^{\mathcal{A}}_{\tau_i}$, we obtain that $\tau_{\tau_i(\ell_i+p+1)}$ and all following jobs are released no earlier than $r^{\omega_1}_{\tau_i(\ell_i+p+1)} \geq t_i + c^*_i + T_i - R^{\mathcal{A}}_{\tau_i}$. This yields $\mathrm{exec}^{[\mathcal{S}_1]}_{\tau_i}(t_i, t_i + \Delta) \leq c^*_i + I_i(\Delta - c^*_i - T_i + R^{\mathcal{A}}_{\tau_i})$. Lemma 5.10 with δ set to $C^*_i - c^*_i$ yields the inequality $\mathrm{exec}^{[\mathcal{S}_1]}_{\tau_i}(t_i, t_i + \Delta) \leq C^*_i + I_i(\Delta - C^*_i - T_i + R^{\mathcal{A}}_{\tau_i})$ which is at most $C^*_i + \alpha_{\tau_i}(\Delta - C^*_i - T_i + R^{\mathcal{A}}_{\tau_i}) \cdot C_{\tau_i}$. □

In general, the bounds from Lemma 5.8 and Lemma 5.12 do not dominate each other. Hence, when $\tau_i \in \mathbb{T}_0$ both bounds have to be considered.

Lemma 5.13. *For $t \in [t_k, f^{[\mathcal{S}_1],\omega_1}_{\tau_k(\ell)}]$ we have $\mathrm{exec}^{[\mathcal{S}_1]}_{\tau_k}(t_k, t) \leq a \cdot C_{\tau_k}$.*

Proof. The interval $[t_k, f^{[\mathcal{S}_1],\omega_1}_{\tau_k(\ell)})$ is a suspension aware busy-interval of τ_k with a jobs. Hence, there are only a jobs of τ_k that can be executed by the processor during $[t_k, f^{[\mathcal{S}_1],\omega_1}_{\tau_k(\ell)})$. As a result, $\mathrm{exec}^{[\mathcal{S}_1]}_{\tau_k}(t_k, t) \leq \mathrm{exec}^{[\mathcal{S}_1]}_{\tau_k}(t_k, f^{[\mathcal{S}_1],\omega_1}_{\tau_k(\ell)}) \leq a \cdot C_{\tau_k}$. □

For τ_k we have an upper bound for the idle time and execution time formulated in Lemma 5.7 and Lemma 5.13 when $t \in [t_k, f^{[\mathcal{S}_1],\omega_1}_{\tau_k(\ell)}]$. If $t \neq f^{[\mathcal{S}_1],\omega_1}_{\tau_k(\ell)}$, i.e., $\tau_k(\ell)$ is not already finished at time t, we know that there is remaining execution or suspension time from $\tau_k(\ell)$. In this case we provide the following lemma.

Lemma 5.14. *For all $t \in [t_k, f^{[\mathcal{S}_1],\omega_1}_{\tau_k(\ell)})$ we have*

$$\mathrm{exec}^{[\mathcal{S}_1]}_{\tau_k}(t_k, t) + \mathrm{idle}^{[\mathcal{S}_1]}(t_k, t) < a \cdot (C_{\tau_k} + S_{\tau_k}). \tag{5.15}$$

Proof. We proof this lemma by contradiction and assume that there exists one $t \in [t_k, f^{[\mathcal{S}_1],\omega_1}_{\tau_k(\ell)})$ such that $\mathrm{exec}^{[\mathcal{S}_1]}_{\tau_k}(t_k, t) + \mathrm{idle}^{[\mathcal{S}_1]}(t_k, t) \geq a \cdot (C_{\tau_k} + S_{\tau_k})$. Since $\mathrm{exec}^{[\mathcal{S}_1]}_{\tau_k}(t_k, t) \leq a \cdot C_{\tau_k}$ and $\mathrm{idle}^{[\mathcal{S}_1]}(t_k, t) \leq a \cdot S_{\tau_k}$ by Lemmas 5.13 and 5.7, both take their highest value, i.e., $\mathrm{exec}^{[\mathcal{S}_1]}_{\tau_k}(t_k, t) = a \cdot C_{\tau_k}$ and $\mathrm{idle}^{[\mathcal{S}_1]}(t_k, t) = a \cdot S_{\tau_k}$. We conclude that all a jobs of τ_k in the suspension-aware busy-interval finished their complete execution time. Moreover, $\mathrm{idle}^{[\mathcal{S}_1]}(t_k, t) \leq \mathrm{susp}_k(t_k, t)$ by Lemma 5.5. In particular, all a jobs suspend themselves for S_{τ_k} time units each. As a result, all a jobs of τ_k are finished and $t \geq f^{[\mathcal{S}_1],\omega_1}_{\tau_k(\ell)}$. This contradicts the assumption. □

Step 3: Provide response-time upper bound. In this step we provide a response-time upper bound for the job $\tau_k(\ell)$. For this purpose, we safely enlarge the intervals for which we estimate the execution time from $[t_i, t)$ to $[t^*_i, t)$. We do this to ensure the property $t^*_{i+1} - t^*_i = \mathrm{idle}^{[\mathcal{S}_1]}(t_i, t_{i+1}) \leq S_{\tau_i}$, which is then utilized to bound the left boundary of the analysis intervals $[t^*_i, t)$. Subsequently, we compose the upper bounds from Step 2 to obtain a response-time bound.

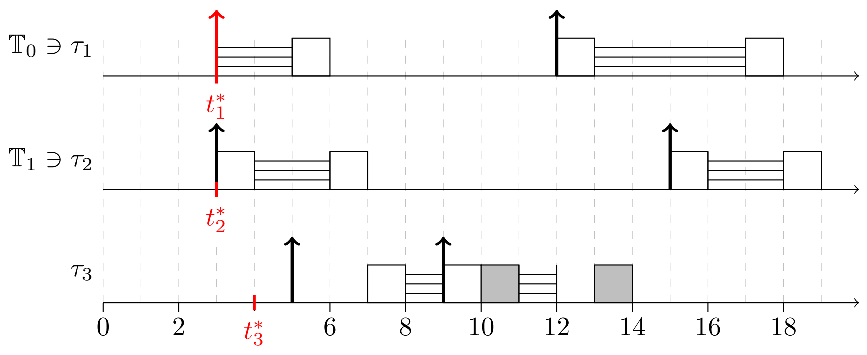


Figure 5.4.: Schedule from Example 5.2 after Step 3.

Definition 5.15. *Iteratively, we define*

$$t_1^* := t_1 \tag{5.16}$$

$$t_i^* := t_{i-1}^* + x_{i-1} \cdot \text{idle}^{[\mathcal{S}_1]}(t_{i-1}, t_i) \tag{5.17}$$

for all $i = 2, \ldots, k$, where $\text{idle}^{[\mathcal{S}_1]}(t_{i-1}, t_i)$ *is the amount of idle time during $[t_{i-1}, t_i)$ in the schedule $[\mathcal{S}_1]$.*

Example 5.16. *In Figure 5.4, we present the choice of t_i^* that is obtained from the schedule $[\mathcal{S}_1]$ in Figure 5.3. We start by setting $t_1^* := t_1 = 3$. Since there is no idle time between t_1 and t_2 in $[\mathcal{S}_1]$, we define $t_1^* := t_2^*$. Between t_2 and t_3 the processor idles for one time unit, namely during the interval $[4, 5)$. We set $t_3^* := t_2^* + 1 = 4$.*

We start the analysis by stating a simple property which is later used to describe the boundaries of the analysis intervals.

Lemma 5.17. *For all $i = 1, \ldots, k$ we have $t_i^* \leq t_i$.*

Proof. This follows from Definition 5.15 using that $\text{idle}^{[\mathcal{S}_1]}(t_{i-1}, t_i) \leq (t_i - t_{i-1})$. $\square$

With the following definition, the execution time bounds from Step 2 can be summarized by $\text{exec}_{\tau_i}^{[\mathcal{S}_1]}(t_i, t_i + \Delta) \leq A_i^1(\Delta)$ whenever $\tau_i \in \mathbb{T}_1$ and $\leq A_i^0(\Delta)$ whenever $\tau_i \in \mathbb{T}_0$.

Definition 5.18 (A_i^1 and A_i^0)**.** *We define the A_i^1 and A_i^0 by*

$$A_i^1(\Delta) := \alpha_{\tau_i}(\Delta + \max(R_{\tau_i}^{\mathcal{A}} - T_i, 0)) \cdot C_{\tau_i} \tag{5.18}$$

$$A_i^0(\Delta) := \min\begin{pmatrix} \alpha_{\tau_i}(\Delta + R_{\tau_i}^{\mathcal{A}}) \cdot C_{\tau_i}, \\ \alpha_{\tau_i}(\Delta - T_i + R_{\tau_i}^{\mathcal{A}} - C_i^*) \cdot C_{\tau_i} + C_i^* \end{pmatrix} \tag{5.19}$$

for all $\Delta \in \mathbb{R}$, where $C_i^* = \min(\alpha_{\tau_i}(R_{\tau_i}^{\mathcal{A}}) \cdot C_{\tau_i}, R_{\tau_i}^{\mathcal{A}})$ as defined in Equation (5.2).

For the response-time upper bound we formulate a property that only holds when $t < f_{\tau_k(\ell)}^{[\mathcal{S}_1],\omega_1}$, in the following lemma. Whenever the property does not hold, we assure that the finishing time must be exceeded. In particular, this allows to indicate response-time upper bounds.

Lemma 5.19. *For all $t \in [t_k^*, f_{\tau_k(\ell)}^{[\mathcal{S}_1],\omega_1})$ the inequality*

$$a \cdot (C_{\tau_k} + S_{\tau_k}) + \sum_{i=1}^{k-1} \left(x_i \cdot A_i^1(t - t_i^*) + (1 - x_i) A_i^0(t - t_i^*)\right) > t - t_k^* \tag{5.20}$$

holds.

Proof. Equation (5.3) states that $\mathrm{idle}^{[\mathcal{S}_1]}(t_1, t) + \mathrm{exec}^{[\mathcal{S}_1]}(t_1, t) = t - t_1$. We know that $\mathrm{idle}^{[\mathcal{S}_1]}(t_1, t) = \mathrm{idle}^{[\mathcal{S}_1]}(t_1, t_k) + \mathrm{idle}^{[\mathcal{S}_1]}(t_k, t)$ and $t_1 + \mathrm{idle}^{[\mathcal{S}_1]}(t_1, t_k) = t_k^*$. Hence, subtracting $\mathrm{idle}^{[\mathcal{S}_1]}(t_1, t_k)$ in Equation (5.3) yields

$$\mathrm{idle}^{[\mathcal{S}_1]}(t_k, t) + \mathrm{exec}^{[\mathcal{S}_1]}(t_1, t) = t - t_k^*. \tag{5.21}$$

For the execution part, we know that $\mathrm{exec}^{[\mathcal{S}_1]}(t_1, t) = \sum_{i=1}^{k-1} \mathrm{exec}_{\tau_i}^{[\mathcal{S}_1]}(t_i, t) + \mathrm{exec}^{[\mathcal{S}_1]}(t_k, t)$ holds. By Lemma 5.11 and Lemma 5.12,

$$\mathrm{exec}_{\tau_i}^{[\mathcal{S}_1]}(t_i, t) \leq x_i \cdot A_i^1(t - t_i) + (1 - x_i) \cdot A_i^0(t - t_i) \tag{5.22}$$

for all $i < k$, since $x_i = 1$ iff $\tau_i \in \mathbb{T}_1$ and $x_i = 0$ iff $\tau_i \in \mathbb{T}_0$.

Since the arrival curve is monotonically increasing, A_i^1 and A_i^0 are monotonically increasing as well. Hence, (5.22) is upper bounded by $x_i \cdot A_i^1(t - t_i^*) + (1 - x_i) \cdot A_i^0(t - t_i^*)$ since $t_i^* \leq t_i$ by Lemma 5.17. Using this together with the bound from Lemma 5.14 yields the result from Equation (5.20). □

In the following lemma, we make the property in Equation (5.20) independent of t_i^* by introducing $Q_i^{\vec{x}}$.

Lemma 5.20. *For $i = 1, \ldots, k-1$ we define $Q_i^{\vec{x}} := \sum_{j=i}^{k-1} x_j S_j$. The inequality*

$$a \cdot (C_{\tau_k} + S_{\tau_k}) + \sum_{i=1}^{k-1} \left(x_i \cdot A_i^1(\theta + Q_i^{\vec{x}}) + (1 - x_i) A_i^0(\theta + Q_i^{\vec{x}})\right) > \theta \tag{5.23}$$

holds for all $\theta \in [0, f_{\tau_k(\ell)}^{[\mathcal{S}_1],\omega_1} - t_k^*)$.

Proof. We obtain Equation (5.23) be replacing the variable t in Lemma 5.19 by $\theta + t_k^*$, i.e., $\theta = t - t_k^*$. Moreover, we have $(t_k^* - t_i^*) \leq Q_i^{\vec{x}}$ since $(t_k^* - t_i^*) = \sum_{j=i}^{k-1} x_j \cdot \mathrm{idle}^{[\mathcal{S}_1]}(t_j, t_{j+1}) \leq \sum_{j=i}^{k-1} x_j \cdot S_j$ because of Lemma 5.6. □

From Lemma 5.20 we derive that any $\theta \geq 0$, such that Equation (5.23) does not hold, is an upper bound on the response time of $\tau_k(\ell)$.

Theorem 5.21. *Let $\vec{x} = (x_1, \ldots, x_{k-1}) \in \{0, 1\}^{k-1}$ and $a \in \mathbb{N}$. If there exist some $\theta \geq 0$ such that*

$$a \cdot (C_{\tau_k} + S_{\tau_k}) + \sum_{i=1}^{k-1} \left(x_i \cdot A_i^1(\theta + Q_i^{\vec{x}}) + (1 - x_i) A_i^0(\theta + Q_i^{\vec{x}})\right) \leq \theta \tag{5.24}$$

with $Q_i^{\vec{x}}$ defined as in Lemma 5.20, then θ is an upper bound on $f_{\tau_k(\ell)}^{[\mathcal{S}_1],\omega_1} - r_{\tau_k(\ell-a+1)}^{\omega_1}$ for all a-th jobs $\tau_k(\ell)$ in a suspension-aware busy-interval of τ_k. In particular,

$$R_k^a := \theta - \inf\{\Delta \geq 0 \,|\, \alpha_k(\Delta) \geq a\} \tag{5.25}$$

is an upper bound on the response time of any a-th job in a suspension-aware busy-interval of τ_k.

Proof. Let $\tau_k(\ell)$ be any a-th job in a suspension-aware busy-interval of τ_k. We prove this theorem by contraposition.

Assume that $\theta \geq 0$ is not an upper bound on $f_{\tau_k(\ell)}^{[\mathcal{S}_1],\omega_1} - r_{\tau_k(\ell-a+1)}^{\omega_1}$ but Equation (5.24) holds. In this case, we know that $f_{\tau_k(\ell)}^{[\mathcal{S}_1],\omega_1} > \theta + r_{\tau_k(\ell-a+1)}^{\omega_1} = \theta + t_k \geq \theta + t_k^*$. In particular, $\theta \in [0, f_{\tau_k(\ell)}^{[\mathcal{S}_1],\omega_1} - t_k^*)$. Applying Lemma 5.20 yields that (5.24) does not hold. This contradicts the assumption.

Since we have shown that $\theta \geq f_{\tau_k(\ell)}^{[\mathcal{S}_1],\omega_1} - r_{\tau_k(\ell-a+1)}^{\omega_1}$, we conclude

$$\begin{aligned} R_k^a &= \theta - \inf\{\Delta \geq 0 \,|\, \alpha_k(\Delta) \geq a\} && (5.26)\\ &\geq f_{\tau_k(\ell)}^{[\mathcal{S}_1],\omega_1} - r_{\tau_k(\ell-a+1)}^{\omega_1} - \inf\{\Delta \geq 0 \,|\, \alpha_k(\Delta) \geq a\} && (5.27)\\ &\geq f_{\tau_k(\ell)}^{[\mathcal{S}_1],\omega_1} - r_{\tau_k(\ell-a+1)}^{\omega_1} - (r_{\tau_k(\ell)}^{\omega_1} - r_{\tau_k(\ell-a+1)}^{\omega_1}) && (5.28)\\ &= f_{\tau_k(\ell)}^{[\mathcal{S}_1],\omega_1} - r_{\tau_k(\ell)}^{\omega_1}, && (5.29) \end{aligned}$$

i.e., R_k^a is a response-time upper bound. Since $\tau_k(\ell)$ was chosen to be any a-th job in a suspension-aware busy-interval of τ_k, the response-time upper bound is valid for all of them. □

Step 4: The Schedulability Test. To provide a response-time upper bound for all jobs of τ_k using Theorem 5.21, we need to figure out which job in a suspension-aware busy-interval has the highest response time. In this regard, we first need to figure out the maximal number $\tilde{a}$ of jobs that belong to one suspension-aware busy-interval. A sufficient condition for the maximal number is $R_k^{\tilde{a}} \leq \inf\{\Delta \geq 0 \,|\, \alpha_k(\Delta) \geq \tilde{a}+1\} - \inf\{\Delta \geq 0 \,|\, \alpha_k(\Delta) \geq \tilde{a}\}$, i.e., the $\tilde{a}$-th job is finished before the next job is released. If the maximal number can be detected with this condition, then a WCRT upper bound is provided by the following corollary.

Corollary 5.22. *Let $\tilde{a} \in \mathbb{N}$ be the lowest natural number such that the response-time upper bound $R_k^{\tilde{a}}$ derived by Theorem 5.21 is at most*

$$\inf\{\Delta \geq 0 \,|\, \alpha_k(\Delta) \geq \tilde{a}+1\} - \inf\{\Delta \geq 0 \,|\, \alpha_k(\Delta) \geq \tilde{a}\} \tag{5.30}$$

Then $R_k := \max_{a=1,\dots,\tilde{a}}(R_k^a)$ is an upper bound on the WCRT of τ_k.

Proof. To prove this corollary, we need to show that each job of τ_k is an a-th job in a suspension-aware busy-interval of τ_k for some $a \in \{1, \dots, \tilde{a}\}$. We do this by contraposition.

Let $\tau_k(\ell)$ be the first job of τ_k which is not an a-th job in a suspension-aware busy-interval of τ_k with $a \leq \tilde{a}$ according to Definition 5.1.

Let a be the smallest positive integer, such that $\tau_k(\ell)$ is the a-th job in a suspension-aware busy-interval of τ_k. Due to our assumption, $a > \tilde{a}$. The job $\tau_k(\ell - a + \tilde{a})$ is the $\tilde{a}$-th job in a suspension-aware busy-interval of τ_k. By Theorem 5.21, $\tau_k(\ell - a + \tilde{a})$ finishes no later than at time

$$r^{\omega_1}_{\tau_k(\ell-a+\tilde{a}-\tilde{a}+1)} + R^{\tilde{a}}_k + \inf\{\Delta \geq 0 \,|\, \alpha_k(\Delta) \geq \tilde{a}\} \tag{5.31}$$

$$\leq r^{\omega_1}_{\tau_k(\ell-a+1)} + \inf\{\Delta \geq 0 \,|\, \alpha_k(\Delta) \geq \tilde{a}+1\} \tag{5.32}$$

$$\leq r^{\omega_1}_{\tau_k(\ell-a+\tilde{a}+1)}. \tag{5.33}$$

In particular, this means that $\tau_k(\ell - a + \tilde{a})$ and all previous jobs are finished at time $r^{\omega_1}_{\tau_k(\ell-a+\tilde{a}+1)}$. As a result, $\tau_k(\ell)$ is also an $(a-\tilde{a})$-th job in a suspension-aware busy-interval of τ_k which contradicts the minimality of a.

We have proven that each job of τ_k is an a-th job in a suspension-aware busy-interval of τ_k for some $a \in \{1, \dots, \tilde{a}\}$, and we use Theorem 5.21 to provide response-time upper bounds R^a_k for all $\tilde{a}$ cases. □

We know that any sub-additive upper arrival curve α_{τ_i} is monotonically increasing for all tasks τ_i. Hence, the left-hand side of Equation (5.24) is monotonically increasing with respect to θ. We use this monotonicity to apply a similar search strategy as classical TDA. We start by setting $\theta \coloneqq 0$ and compute the left-hand side of Equation (5.24). Whenever θ is less than the left-hand side, we set θ to the value of the left-hand side and compute left-hand side again. When θ is bigger than or equal to left-hand side, then θ can be used to compute the response-time upper bound as in Theorem 5.21. The procedure is presented in Algorithm 1. Please note that we artificially set an upper bound a_{max} to exit the algorithm when the number of jobs in a suspension-aware busy-interval is unbounded.

Upper arrival curves are typically modeled as step functions. Under the assumption that the upper arrival curve α_{τ_i} is a step function with minimal step size > 0 for all i, the left-hand side increases by a certain minimal step size in each iteration as well, until either left-hand side $\leq \theta$ or left-hand side $> D_{\tau_k}$. As a result Algorithm 1 is deterministic in such a scenario.

In the following, we show that our method dominates the Compositional Performance Analysis (CPA) [Hen+05; HAE17].

Corollary 5.23. *The WCRT analysis by using CPA for task τ_k with suspension as computation, i.e., execution time $\alpha_k(\Delta) \cdot (C_{\tau_k} + S_{\tau_k})$ for any interval length of $\Delta \geq 0$, and jitter-based higher-priority preemptive interference of task τ_i with execution time $\alpha_{\tau_i}(\Delta + R^{\mathcal{A}}_{\tau_i}) \cdot (C_{\tau_i})$, is dominated by the analysis in Corollary 5.22 when all higher-priority tasks are in $\mathbb{T}_0$.*

Proof. When all tasks are in the set $\mathbb{T}_0$, then Equation (5.24) simplifies to $a \cdot (C_{\tau_k} + S_{\tau_k}) + \sum_{i=1}^{k-1} A^0_i(\theta) \leq \theta$. However, $A^0_i(\theta)$ is upper bounded by $\alpha_{\tau_i}(\Delta + R^{\mathcal{A}}_{\tau_i}) \cdot C_{\tau_i}$. This

Algorithm 1 Sufficient schedulability test.

```
for $k = 1, \dots, n$ do                                        ▷ Loop tasks.
    for $a = 1, 2, 3, \dots$ do
        if $a > a_{max}$ then
            return False
        $comp := \inf\{\Delta \geq 0 \,|\, \alpha_k(\Delta) \geq a + 1\}$
                $- \inf\{\Delta \geq 0 \,|\, \alpha_k(\Delta) \geq a\}$
        $\theta := 0$
        while True do
            Compute left-hand side of Equation (5.24).
            if left-hand side $\leq \theta$ then                 ▷ Result found.
                break
            else if left-hand side $> D_{\tau_k}$ then            ▷ Too high.
                return False
            else                                                ▷ Continue search.
                $\theta :=$ left-hand side
        $R_k^a := \theta - \inf\{\Delta \geq 0 \,|\, \alpha_k(\Delta) \geq a\}$
        if $R_k^a \leq comp$ then
            $\tilde{a} := a$
            break
    $R_k := \max_{a=1,\dots,\tilde{a}}(R_k^a)$                  ▷ WCRT upper bound.
return True
```

is the formula that is used for the a-th job in the busy-interval. In our schedulability test, we increase a, until the subsequent job release of τ_k is outside the busy-interval, i.e., until $R_k^a \leq \inf\{\Delta \geq 0 \,|\, \alpha_k(\Delta) \geq \tilde{a} + 1\} - \inf\{\Delta \geq 0 \,|\, \alpha_k(\Delta) \geq \tilde{a}\}$. This coincides with the test used in CPA analysis. □

5.1.1.2 Analysis for Sporadic Tasks

In this Section, we derive a schedulability test for sporadic tasks with arbitrary deadlines. To that end, we consider a task set $\mathbb{T} = \{\tau_1, \dots, \tau_n\}$ where each task is abstracted as

$$\tau_i \in \text{DynSS}(C_{\tau_i}, S_{\tau_i}) \cap \text{Spor}(T_{\tau_i}) \tag{5.34}$$

with $T_{\tau_k} > 0$. Again, we assume that the task set is ordered according to the priority level, and we denote by $\mathcal{A}$ the preemptive T-FP scheduling algorithm.

We do this by constructing an arrival curve and using the result from Section 5.1.1.1. In the end, we show that by restricting to the constrained-deadline case, our schedulability test coincides with the test in [CNH16], which is limited to constrained-deadline task sets. Hence, in this work we provide a natural extension of their analysis to the generalized case with arbitrary deadlines and arrival curves.

For any **arbitrary-deadline** task τ_i, an upper arrival curve is given by

$$\alpha_{\tau_i} = \left(\Delta \mapsto \left\lceil \frac{\max(\Delta, 0)}{T_{\tau_i}} \right\rceil = \begin{cases} \left\lceil \frac{\Delta}{T_{\tau_i}} \right\rceil & \Delta \geq 0 \\ 0 & \Delta < 0 \end{cases} \right). \tag{5.35}$$

Applying this to A_i^1 and A_i^0 with $T_i \coloneqq T_{\tau_i}$ yields

$$A_i^1(\Delta) = \left\lceil \frac{\Delta + \max(R_{\tau_i}^{\mathcal{A}} - T_{\tau_i}, 0)}{T_{\tau_i}} \right\rceil \cdot C_{\tau_i} \tag{5.36}$$

$$A_i^0(\Delta) = \min \left(\begin{array}{l} \left\lceil \frac{\Delta + R_{\tau_i}^{\mathcal{A}}}{T_{\tau_i}} \right\rceil, \\ \left\lceil \frac{\max(\Delta - T_{\tau_i} + R_{\tau_i}^{\mathcal{A}} - C_i^*, 0)}{T_{\tau_i}} \right\rceil \cdot C_{\tau_i} + C_i^* \end{array} \right) \tag{5.37}$$

for all $\Delta \geq 0$. Please note that $\Delta + \max(R_{\tau_i}^{\mathcal{A}} - T_{\tau_i}, 0) \geq 0$ which is why $\max(\bullet, 0)$ is omitted in the numerator of (5.36). Moreover, we have $C_i^* = \min(\left\lceil \frac{R_{\tau_i}^{\mathcal{A}}}{T_{\tau_i}} \right\rceil \cdot C_{\tau_i}, R_{\tau_i}^{\mathcal{A}})$ and $\inf \{\Delta \geq 0 \,|\, \alpha_{\tau_i}(\Delta) \geq a\} = (a-1) \cdot T_{\tau_i}$. This leads to the following bound on the WCRT.

Theorem 5.24. *Let $\vec{x} = (x_1, \ldots, x_{k-1}) \in \{0,1\}^{k-1}$ and $a \in \mathbb{N}$. If there exist some $\theta \geq 0$ such that*

$$a \cdot (C_{\tau_k} + S_{\tau_k}) + \sum_{i=1}^{k-1} \left(\begin{array}{l} x_i \cdot \left\lceil \frac{\theta + Q_i^{\vec{x}} + \max(R_{\tau_i}^{\mathcal{A}} - T_{\tau_i}, 0)}{T_{\tau_i}} \right\rceil \cdot C_{\tau_i} + (1 - x_i) \cdot \\ \min \left(\begin{array}{l} \left\lceil \frac{\theta + Q_i^{\vec{x}} + R_{\tau_i}^{\mathcal{A}}}{T_{\tau_i}} \right\rceil, \\ \left\lceil \frac{\max(\theta + Q_i^{\vec{x}} - T_{\tau_i} + R_{\tau_i}^{\mathcal{A}} - C_i^*, 0)}{T_{\tau_i}} \right\rceil \cdot C_{\tau_i} + C_i^* \end{array} \right) \end{array} \right) \leq \theta \tag{5.38}$$

with $Q_i^{\vec{x}}$ defined as in Lemma 5.20, then θ is an upper bound on $f_{\tau_k(\ell)}^{[\mathcal{S}_1], \omega_1} - r_{\tau_k(\ell - a + 1)}^{\omega_1}$ for all a-th jobs $\tau_k(\ell)$ in a suspension-aware busy-interval of τ_k. In particular,

$$R_k^a \coloneqq \theta - (a-1) T_{\tau_i} \tag{5.39}$$

is an upper bound on the response time of any a-th job in a suspension-aware busy-interval of τ_k.

Proof. This follows from Theorem 5.21 using the arrival curve from Equation (5.35), and A_i^1 from Equation (5.36) and A_i^0 from Equation (5.37). □

Similar to Corollary 5.22, we formulate the following.

Corollary 5.25. *Let $\tilde{a} \in \mathbb{N}$ be the lowest natural number such that the response-time upper bound derived by Theorem 5.24 is at most T_{τ_k}. Then $R_k \coloneqq \max_{a=1,\ldots,\tilde{a}}(R_k^a)$ is an upper bound on the WCRT of τ_k.*

Proof. The corollary follows from the correctness of Corollary 5.22 by using that

$$\inf\{\Delta \geq 0 \,|\, \alpha_k(\Delta) \geq \tilde{a}+1\} - \inf\{\Delta \geq 0 \,|\, \alpha_k(\Delta) \geq \tilde{a}\} = T_{\tau_k} \tag{5.40}$$

□

For any **constrained-deadline** task τ_i with $T_{\tau_i} > 0$, if its schedulability is already ensured, which is the case for all $i < k$, then $R^{\mathcal{A}}_{\tau_i} \leq D_{\tau_i} \leq T_{\tau_i}$. In particular, $C^*_i = C_{\tau_i}$ and the formulas for A^1_i and A^0_i from Equation (5.36) and (5.37) further simplify to

$$A^1_i(\Delta) = \left\lceil \frac{\Delta}{T_{\tau_i}} \right\rceil \cdot C_{\tau_i} \tag{5.41}$$

$$A^0_i(\Delta) = \left\lceil \frac{\Delta + R^{\mathcal{A}}_{\tau_i} - C_{\tau_i}}{T_{\tau_i}} \right\rceil \cdot C_{\tau_i} \tag{5.42}$$

for all $\Delta > 0$. Please note that Equation (5.42) is obtained from (5.37) in the following way: For all $\Delta > 0$ we have $\Delta - T_{\tau_i} + R^{\mathcal{A}}_{\tau_i} - C_{\tau_i} \geq \Delta - T_{\tau_i} > -T_{\tau_i}$. Therefore, we obtain

$$\left\lceil \frac{\max(\Delta - T_{\tau_i} + R^{\mathcal{A}}_{\tau_i} - C_{\tau_i}, 0)}{T_{\tau_i}} \right\rceil = \max(\left\lceil \frac{\Delta - T_{\tau_i} + R^{\mathcal{A}}_{\tau_i} - C_{\tau_i}}{T_{\tau_i}} \right\rceil, 0) \tag{5.43}$$

$$= \left\lceil \frac{\Delta - T_{\tau_i} + R^{\mathcal{A}}_{\tau_i} - C_{\tau_i}}{T_{\tau_i}} \right\rceil \tag{5.44}$$

for all $\Delta > 0$. Moreover,

$$A^0_i(\Delta) = \left\lceil \frac{\Delta - T_{\tau_i} + R^{\mathcal{A}}_{\tau_i} - C_{\tau_i}}{T_{\tau_i}} \right\rceil \cdot C_{\tau_i} + C_{\tau_i} \tag{5.45}$$

$$= \left(\left\lceil \frac{\Delta - T_{\tau_i} + R^{\mathcal{A}}_{\tau_i} - C_{\tau_i}}{T_{\tau_i}} \right\rceil + 1 \right) \cdot C_{\tau_i} \tag{5.46}$$

$$= \left\lceil \frac{\Delta + R^{\mathcal{A}}_{\tau_i} - C_{\tau_i}}{T_{\tau_i}} \right\rceil \cdot C_{\tau_i}. \tag{5.47}$$

For the constrained-deadline case, we only need to compute the WCRT upper bound for $a = 1$, since if $R^1_i > T_{\tau_k}$ then $R^1_i > D_{\tau_k}$ as well and the schedulability test fails. The schedulability test formulated with A^1_i and A^0_i from Equation (5.41) and (5.42) with only $a = 1$ coincides with the schedulability test in [CNH16, Corollary 1]. The high performance of the abovementioned schedulability test demonstrated in [CNH16] indicates high performance of the schedulability test derived in this work.

5.1.1.3 Choice of $\mathbb{T}_0$ and $\mathbb{T}_1$

For the schedulability test presented in Section 5.1.1.1, any partition of the task set $\mathbb{T}$ into $\mathbb{T}_0$ and $\mathbb{T}_1$ can be used. To fully utilize the power of the proposed test, all possible combinations of $\mathbb{T}_0$ and $\mathbb{T}_1$ should be explored. However, this results in an exhaustive search of 2^{k-1} different partitions of the $k-1$ higher-priority tasks, for which every schedulability test itself also takes high time complexity.

This limitation was also presented by Chen et al. [CNH16] in their analysis for constrained-deadline task systems. They presented some heuristics, which can also be applied here for arbitrary-deadline task systems and tasks with arrival curves. Here, we shortly present the heuristics that can be used to reduce the complexity significantly. They are compared in the subsequent section. However, those heuristics are not the focus of this paper and should be further examined in the future.

The most simple heuristic is to include all tasks into $\mathbb{T}_0$, i.e., $\mathbb{T}_0 = \mathbb{T}$ and $\mathbb{T}_1 = \emptyset$, or to include all tasks into $\mathbb{T}_1$, i.e., $\mathbb{T}_0 = \emptyset$ and $\mathbb{T}_1 = \mathbb{T}$. In Section 5.1.1.4, we observe that such a simple heuristic is already sufficient in many cases.

As argued in Section 5.1.1.1, our analysis is a natural extension to arbitrary-deadline task systems from the analysis for constrained-deadline task systems in [CNH16]. In their work they provide a *linear approximation* that is stated as follows: τ_i is in $\mathbb{T}_1$ if $\frac{C_{\tau_i}}{T_{\tau_i}} \cdot (R^{\mathcal{A}}_{\tau_i} - C_{\tau_i}) > S_{\tau_i} \cdot (\sum_{j=1}^{i} \frac{C_{\tau_j}}{T_{\tau_j}})$; otherwise $\tau_i \in \mathbb{T}_0$. Their heuristic is supported by a mathematical reasoning which goes beyond the scope of this section.

Our analysis for arbitrary-deadline task systems is much more involving, and therefore it is more difficult to judge whether it is better to place τ_i in $\mathbb{T}_0$ or in $\mathbb{T}_1$. Although we do not have concrete mathematical approximations to derive any classification strategies to partition $\mathbb{T}$, the above linear approximation can still be applied. Unfortunately, when considering arrival curves, it is possible that T_{τ_i} is 0 and $\frac{C_{\tau_i}}{T_{\tau_i}} \to \infty$ for $T_{\tau_i} \to 0$. In such a case, the linear approximation becomes invalid. One possible patch is to apply the linear approximation of the arrival curve by defining $slope_i$ and $const_i$ for every task τ_i such that $\alpha_{\tau_i}(\Delta) \leq const_i + slope_i \cdot \Delta$ holds for all $\Delta \geq 0$. Then, we can heuristically put task τ_i in $\mathbb{T}_1$ if $slope_i \cdot (R^{\mathcal{A}}_{\tau_i} - C_{\tau_i}) > S_{\tau_i} \cdot (\sum_{j=1}^{i} slope_j)$; otherwise $\tau_i \in \mathbb{T}_0$.

5.1.1.4 Evaluation

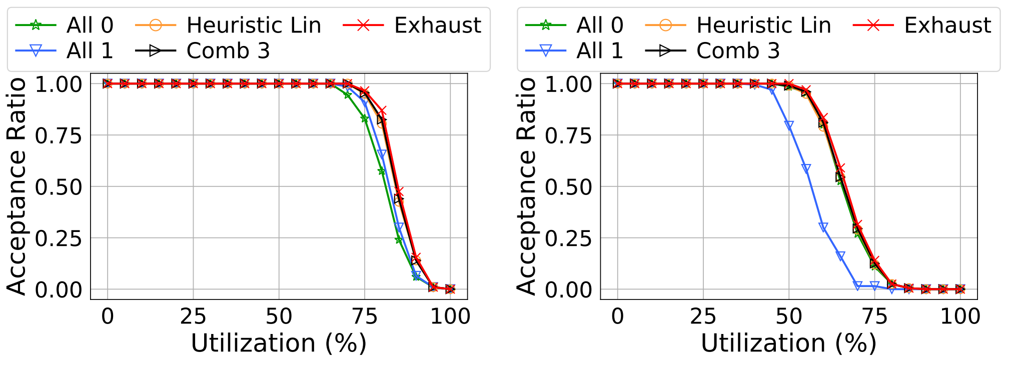
In order to evaluate the performance of our proposed schedulability analysis presented in Section 5.1.1.1, we conduct three different experiments using synthetically generated tasks sets as follows:

(i) We demonstrate how the heuristics from Section 5.1.1.3 perform for arbitrary-deadline task sets (**Experiment—Suspension Time**).

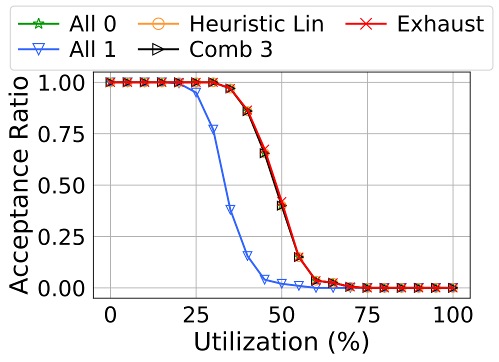
(ii) We show that the increase of deadlines can be exploited in our schedulability test, i.e., the schedulability is increased with increased deadlines (**Experiment—Varying Deadline**).

(iii) We examine the performance of our schedulability test for arrival curves obtained from tasks with release jitter (**Experiment—Release Jitter**).

The source code that is used to conduct the experiments is released on Github [TU 21b]. In all our experiments, we use Algorithm 1 configured with $a_{max} = 10$. The values



(a) Low suspension in $[0, 0.1] \cdot (T_{\tau_i} - C_{\tau_i})$. (b) Medium suspension in $[0.1, 0.3] \cdot (T_{\tau_i} - C_{\tau_i})$.



(c) High suspension in $[0.3, 0.5] \cdot (T_{\tau_i} - C_{\tau_i})$.

Figure 5.5.: Acceptance ratio of our schedulability test with different heuristics as described in Section 5.1.1.3.

of $\inf \{\Delta \geq 0 \,|\, \alpha_{\tau_i}(\Delta) \geq a\}$ for different a are stored in a list to improve the execution efficiency of the schedulability test. For all three experiments, we present the *acceptance ratio* of the test under analysis, i.e., the number of task sets that are deemed schedulable by the test divided by the total number of task sets. We consider Deadline Monotonic (DM) scheduling, i.e., the task with a lower relative deadline has a higher priority, and ties are broken arbitrarily. We emphasize that we do not consider a constrained-deadline scenario, since in this case our method is identical to the current state-of-the-art analysis in [CNH16], which dominates all other valid analyses for the studied problem.

Experimental Setup and Generation For each total utilization between 0 % and 100 % in steps of 5 % we randomly generate 200 task sets according to the description below. In a first step, given the required length of task sets, we use the UUniFast [BB05] algorithm to synthesize task utilizations that add up to a specified cumulative (total) utilization. In a second step, the minimum inter-arrival time for each sporadic task is drawn log-uniformly from the interval $[1, 100]$[ms] as suggested in [ESD10]. Based on the utilization U_{τ_i} and minimum inter-arrival time T_{τ_i}, the Worst-Case Execution Time (WCET) is calculated by $C_{\tau_i} = U_{\tau_i} \cdot T_{\tau_i}$ [ms]. The arrival curve of the sporadic tasks is given by $\alpha_{\tau_i}(\Delta) = \left\lceil \frac{\Delta}{T_{\tau_i}} \right\rceil$.

In our experiments, we evaluate Algorithm 1 for different partitioning strategies of tasks in $\mathbb{T}$ into $\mathbb{T}_0$ and $\mathbb{T}_1$ as described in Section 5.1.1.3. Namely, these are:

All 0**:** All tasks $\tau_i \in \mathbb{T}$ are put to $\mathbb{T}_0$.

All 1**:** All tasks $\tau_i \in \mathbb{T}$ are put in $\mathbb{T}_1$.

Heuristic Lin: The linear heuristic from [CNH16] is adopted.

Comb 3**:** We choose the best partition of the above three heuristics All 0, All1, and Heuristic Lin (in each step).

Exhaust: In the exhaustive approach we apply our schedulability test for all 2^{k-1} partitions.

EXPERIMENTS In this section, we present the details of the experiments and describe the results.
Experiment—Suspension Time: In this experiment, we restrict to 10 tasks per task set due to the time complexity of (Exhaust). Moreover, we draw the relative deadline D_{τ_i} uniformly from the interval $[0.8T_{\tau_i}, 1.2T_{\tau_i}]$ and consider three different configurations of self-suspension time:

- *low*: S_{τ_i} is drawn uniformly from $[0, 0.1](T_{\tau_i} - C_{\tau_i})$,
- *medium*: S_{τ_i} is drawn uniformly from $[0.1, 0.3](T_{\tau_i} - C_{\tau_i})$,
- *high*: S_{τ_i} is drawn uniformly from $[0.3, 0.5](T_{\tau_i} - C_{\tau_i})$.

Figure 5.5 shows the evaluation results, in which (All 0) and (All 1) do not dominate each other. When the suspension time is short, (All 1) is better than (All 0), in Figure 5.5a. When the suspension time is long, (All 0) is better than (All 1), in Figure 5.5c. Moreover, (Heuristic Lin) outperforms both of them for all scenarios. The benefit of exhaust is marginal compared to the linear heuristic as can be seen in all Figures 5.5a, 5.5b, and 5.5c.
Experiment—Varying Deadlines: In this experiment, we consider 30 tasks per task set and draw the suspension time S_{τ_i} uniformly at random from the interval $[0, 0.5(T_{\tau_i} - C_{\tau_i})]$ for each task. In addition, the deadline of all tasks τ_i is set to $D_{\tau_i} = X \cdot T_{\tau_i}$ **(DX)** for $X = 1.0, 1.1, \ldots, 1.5$. We evaluate Algorithm 1 with the partitioning strategy (Comb 3) and show the results in Figure 5.6. The results show that the acceptance ratio of our algorithm improves with increased deadlines.
Experiment—Release Jitter: In this experiment, we consider 10 tasks per task set and draw the suspension time S_{τ_i} uniformly from the interval $[0, 0.1](T_{\tau_i} - C_{\tau_i})$ for each task. The relative deadline of task τ_i is drawn uniformly from the interval $[0.8, 1.2]T_{\tau_i}$. Moreover, we consider task sets with release jitter of 10 % or 20 %. That is, after the generation of

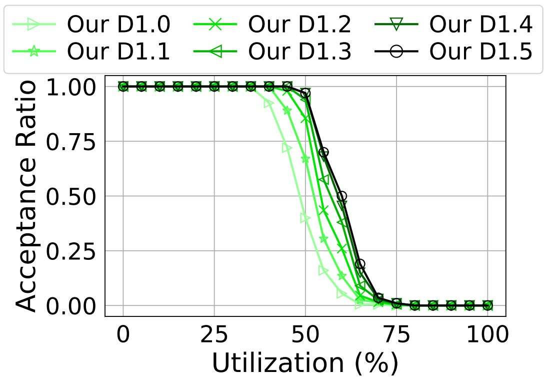
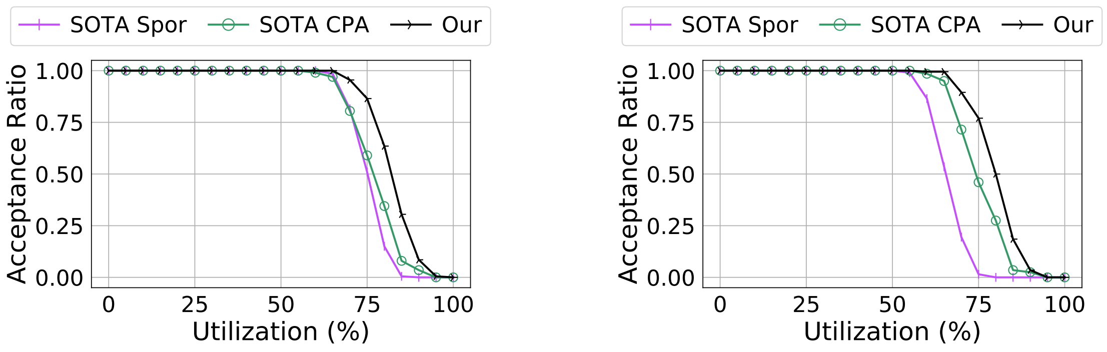


Figure 5.6.: Acceptance ratio of our schedulability test with extended deadlines.



(a) Release jitter: 10 % of T_{τ_i}.

(b) Release jitter: 20 % of T_{τ_i}.

Figure 5.7.: Acceptance ratio of task sets with release jitter. Our schedulability test and two state of the art are presented.

$C_{\tau_i}, T_{\tau_i}, D_{\tau_i}, S_{\tau_i}$ for τ_i, we add $jitter_i$ as $0.1T_{\tau_i}$ or $0.2T_{\tau_i}$. The corresponding arrival curve of τ_i is then $\alpha_{\tau_i}(\Delta) = \left\lceil \frac{\Delta + jitter_i}{T_{\tau_i}} \right\rceil$. There are two different state-of-the-art methods that we compare with. For the CPA state of the art **(SOTA CPA)**, we adhere to the state of the art presented in Corollary 5.23. For the sporadic state of the art **(SOTA Spor)**, we transform the task set into a constrained-deadline task set by reducing the deadline of each τ_i to $\min(D_{\tau_i}, T_{\tau_i})$ and keeping its original priority. Then we adopt [CNH16] using the heuristic with three partitions, similar to (Comb 3). The schedulability of the transformed task set indicates the schedulability of the original task set. The arrival curve suggests that the minimum inter-arrival time of two consecutive jobs is $T_{\tau_i} - jitter_i$, and we can shorten the relative deadline to $T_{\tau_i} - jitter_i$ if $D_{\tau_i} > T_{\tau_i} - jitter_i$.

The results for the release jitter of 10 % or 20 % are shown in Figure 5.7a and Figure 5.7b respectively. Our schedulability test **(Our)** performs much better than the state of the art (SOTA Spor) and (SOTA CPA) for the scenario with release jitter. We note that we also conducted experiments with higher suspension and larger number of tasks per task set as well. In such a case, the suspension time per task becomes longer and the scenario of having tasks in $\mathbb{T}_1$ is less beneficial. When the analysis with all tasks in $\mathbb{T}_0$ is superior, our analysis becomes equivalent to jitter-based analysis, which is still superior to (SOTA CPA) but the gain in the acceptance ratio becomes smaller.

5.1.2 Under Earliest-Deadline-First Scheduling

Regarding suspension-aware schedulability tests for Task-level Dynamic-Priority (T-DP) scheduling algorithms, results have been limited to the Earliest-Deadline-First (EDF) algorithm, in which the priority of each job is given by its absolute deadline; and although EDF is not optimal for scheduling self-suspending task systems [RRC04], it remains one of the most adopted scheduling strategies. Besides the trivial suspension-oblivious schedulability analysis, the following results have been provided in the literature:

- Analysis by Devi [Dev03, Theorem 8], dating back to 2003, dedicated to uniprocessor EDF. The review paper in [Che+19b] notes that Devi does not give a proof of her analysis. This analysis is proven to be wrong by a counterexample in Section 4.2.

- Analysis by Liu and Anderson [LA13], for global multiprocessor EDF. The analysis is valid for uniprocessor EDF by setting the number of processors to 1.

- Analysis by Dong and Liu [DL16], for global multiprocessor EDF. The analysis is valid for uniprocessor EDF by setting the number of processors to 1.

Despite the possibility to apply the analyses of Liu and Anderson [LA13] and Dong and Liu [DL16], fundamental research questions regarding schedulability tests for uniprocessor EDF considering dynamic self-suspension remain open. For example, *there is no schedulability analysis that is superior to the trivial suspension-oblivious analysis for uniprocessor EDF*. We believe that fundamental knowledge of uniprocessor EDF is a cornerstone to achieve tight schedulability tests for more advanced multiprocessor scenarios.

In this section, we focus on schedulability analysis for the dynamic self-suspension model under uniprocessor preemptive EDF and provide two sufficient schedulability tests for this setting. We limit our attention to *implicit-deadline* real-time task systems, i.e., the relative deadline of a task is the same as its period (for periodic releases) or minimum inter-arrival time (for sporadic releases). Our contributions are as follows:

- We recap existing schedulability analyses, and prove that the utilization-based schedulability test by Dong and Liu [DL16] does not improve the suspension-oblivious approach if it is utilized for uniprocessor systems, in Section 5.1.2.1.

- We provide a response-time analysis for EDF in Section 5.1.2.2. If all tasks have suspension, then this test dominates the test developed by Liu and Anderson [LA13] applied to implicit-deadline *sporadic* real-time tasks in uniprocessor systems. We also discuss how this analysis can be extended to multiprocessor global EDF.

- We further provide a utilization-based schedulability test for *periodic* task systems in Section 5.1.2.3, where we estimate the amount of time that the processor is used by other jobs during self-suspension. To the best of our knowledge, this is the only work explicitly considering periodic tasks with dynamic self-suspension, showing that dedicated analysis of the periodic job release pattern of a task can be beneficial. Furthermore, this is the *first* schedulability test which dominates and improves the trivial suspension-oblivious approach for uniprocessor systems.
- In addition to the theoretical improvement and dominance of existing methods under different conditions, the evaluation results in Section 5.1.2.4 show that the response-time-aware schedulability test performs significantly better than the state of the art.

The results presented in this section appeared in EMSOFT 2020 [GBC20].

5.1.2.1 Existing Methods

In this section, we recap existing schedulability tests for dynamic self-suspending tasks under preemptive EDF scheduling [LL73] that are applicable to implicit-deadline uniprocessor systems. The method by Devi [Dev03, Theorem 8] is left out, since a counterexample for the analysis is provided in Section 4.2.

Suspension-Oblivious Suspension-oblivious analysis interprets suspension time as additional computation time. This approach can be very pessimistic, especially if some tasks have a large maximum suspension time. Since implicit-deadline task sets without suspension are schedulable under EDF if and only if their total utilization is less than or equal to 1, the schedulability test (detailed in [Che+19b, Section 4]) is defined as follows.

Theorem 5.26 (Suspension-Oblivious)**.** *Let $\mathbb{T} = \{\tau_1, \tau_2, \dots, \tau_n\}$ be a system of n implicit-deadline sporadic tasks with dynamic self-suspension, i.e.,*

$$\tau_i \in \mathrm{Spor}(T_{\tau_i}) \cap \mathrm{DynSS}(C_{\tau_i}, S_{\tau_i}) \tag{5.48}$$

for all $\tau_i \in \mathbb{T}$. Then $\mathbb{T}$ is schedulable using preemptive EDF if $\sum_{i=1}^{n} \frac{C_{\tau_i}+S_{\tau_i}}{T_{\tau_i}} \leq 1$.

Workload-Based Schedulability Test The schedulability test developed by Liu and Anderson [LA13] is actually formulated for multiprocessor systems, considering arbitrary-deadline sporadic tasks. Furthermore, they added a *tardiness threshold*, which is an upper bound on the amount of time by which individual jobs may miss their deadline, to make their analysis also available for soft real-time systems (where a deadline overrun is considered a service degradation but not system failure). Setting the number of processors to 1 and the tardiness threshold to 0, their method can be applied to implicit-deadline sporadic tasks in uniprocessor systems. For each task $\tau_\ell \in \mathbb{T}$ they estimate the workload W_c and W_{nc} with and without carry-in jobs[2] inside an interval of

[2]Carry-in jobs are those which are released before the interval under analysis and still interfere during the interval.

length ξ_ℓ and check whether the processor still has enough capacity to work on and be suspended for a job of τ_ℓ.[3]

Theorem 5.27 (Liu and Anderson [LA13])**.** *Let $\mathbb{T} = \{\tau_1, \tau_2, \dots, \tau_n\}$ be a system of n implicit-deadline sporadic tasks with dynamic self-suspension, i.e.,*

$$\tau_i \in \mathrm{Spor}(T_{\tau_i}) \cap \mathrm{DynSS}(C_{\tau_i}, S_{\tau_i}) \tag{5.49}$$

for all $\tau_i \in \mathbb{T}$. The task set is schedulable using preemptive EDF if for all $\tau_\ell \in \mathbb{T}$, for all $s_{\ell,j} \in \{0, 1, \dots, S_{\tau_\ell}\}$, and for all $\xi_\ell \in \left[T_{\tau_\ell}, \frac{C_{\tau_\ell} + s_{\ell,j} + \sum_{i=1}^n C_{\tau_i}}{\left(1 - \sum_{i=1}^n \frac{C_{\tau_i}}{T_{\tau_i}}\right)} \right) \cap \mathbb{Z}$ the property

$$\sum_{\tau_i \in \tau^s} \max\{W_{nc}(\tau_i), W_c(\tau_i)\} + \sum_{\tau_j \in \tau^e} W_{nc}(\tau_j) \leq \xi_\ell - C_{\tau_\ell} - s_{\ell,j} \tag{5.50}$$

holds, where τ^s and τ^e are subsets of $\mathbb{T}$ consisting of those tasks τ_i which do have suspension time ($S_{\tau_i} \neq 0$) and those which do not have suspension ($S_{\tau_i} = 0$), i.e., which are execution-only*, respectively. Moreover, the values $W_c(\tau_i)$ and $W_{nc}(\tau_i)$ are estimations for relevant work of τ_i with and without carry-in jobs, defined by*

$$W_c(\tau_i) = \begin{cases} \min\{\Delta(\tau_i, \xi_\ell), \xi_\ell - C_{\tau_\ell} - s_{\ell,j} + 1\} & , i \neq l \\ \min\{\Delta(\tau_\ell, \xi_\ell) - C_{\tau_\ell}, \xi_\ell - T_{\tau_\ell}\} & , i = l \end{cases} \tag{5.51}$$

with $\Delta(\tau_i, t) = \left(\left\lceil \frac{t}{T_{\tau_i}} \right\rceil - 1 \right) C_{\tau_i} + \min\left\{ C_{\tau_i}, t - \left\lceil \frac{t}{T_{\tau_i}} \right\rceil T_{\tau_i} + T_{\tau_i} \right\}$, and

$$W_{nc}(\tau_i) = \begin{cases} \min\left\{ \left\lfloor \frac{\xi_\ell}{T_{\tau_i}} \right\rfloor C_{\tau_i}, \xi_\ell - C_{\tau_\ell} - s_{\ell,j} + 1 \right\} & , i \neq l \\ \min\left\{ \left\lfloor \frac{\xi_\ell}{T_{\tau_\ell}} \right\rfloor C_{\tau_\ell} - C_{\tau_\ell}, \xi_\ell - T_{\tau_\ell} \right\} & , i = l \end{cases}. \tag{5.52}$$

We note that Liu and Anderson [LA13] restrict to the discrete time domain for their analysis, i.e., the periods, WCETs, and maximum suspension times of all tasks are in $\mathbb{Z}_{\geq 0}$. This is not necessary for our methods in Sections 5.1.2.2 and 5.1.2.3.

Utilization-Based Schedulability Test The method by Dong and Liu [DL16, Theorem 2] is also formulated for multiprocessor systems. By setting the number of processors to 1, their method can be utilized for uniprocessor systems.

Theorem 5.28 (Dong and Liu [DL16, Theorem 2])**.** *Let $\mathbb{T} = \{\tau_1, \dots, \tau_n\}$ be a system of n implicit-deadline sporadic tasks with dynamic self-suspension, i.e.,*

$$\tau_i \in \mathrm{Spor}(T_{\tau_i}) \cap \mathrm{DynSS}(C_{\tau_i}, S_{\tau_i}) \tag{5.53}$$

[3]We note that there is a typing error in the original paper, corrected here.

for all $\tau_i \in \mathbb{T}$. The task set is schedulable using preemptive EDF scheduling if

$$\sum_{i=1}^{n} \frac{C_{\tau_i} + S_{\tau_i}}{T_{\tau_i}} \leq 1 \tag{5.54}$$

holds, or if there exists some $\mathbb{T}_0 \subseteq \mathbb{T}$ and some $k \in \{2, \dots, |\mathbb{T}_0|\}$ with

$$\sum_{i=1}^{\lfloor \frac{k}{2} \rfloor} E^i(\mathbb{T}_0) \geq T_{max}(\mathbb{T}_0), \tag{5.55}$$

where $E^i(\mathbb{T}_0)$ is the i-th minimum $(C_{\tau_j} + S_{\tau_j})$ among tasks $\tau_j \in \mathbb{T}_0$ and $T_{max}(\mathbb{T}_0)$ is the maximum task period of tasks in $\mathbb{T}_0$, such that

$$\sum_{\tau_i \in \mathbb{T}} \frac{C_{\tau_i}}{T_{\tau_i}} + \sum_{\tau_i \in \mathbb{T} - \mathbb{T}_0} \frac{S_{\tau_i}}{T_{\tau_i}} + \sum_{i=1}^{k} v^i(\mathbb{T}_0) \leq 1, \tag{5.56}$$

where $v^i(\mathbb{T}_0)$ is the i-th maximum suspension ratio $\left(\frac{S_{\tau_j}}{T_{\tau_j}}\right)$ among tasks $\tau_j \in \mathbb{T}_0$.

In fact, this method is no improvement compared to the suspension-oblivious schedulability test if it is formulated for only one processor.

Proposition 5.29. *The schedulability test by Dong and Liu as formulated in Theorem 5.28 for uniprocessor systems is identical to the suspension-oblivious test as formulated in Theorem 5.26.*

Proof. Assume that the schedulability test from Theorem 5.28 is superior to the suspension-oblivious approach for the task set $\mathbb{T} = \{\tau_1, \dots, \tau_n\}$. Then we find some $\mathbb{T}_0 \subseteq \mathbb{T}$ and some $k \in \{2, \dots, |\mathbb{T}_0|\}$ with $\sum_{i=1}^{\lfloor \frac{k}{2} \rfloor} E^i(\mathbb{T}_0) \geq T_{max}(\mathbb{T}_0)$. By reordering we assume that $\mathbb{T}_0 = \{\tau_1, \dots, \tau_{n_0}\}$ and $E^i(\mathbb{T}_0) = C_{\tau_i} + S_{\tau_i}$. Then we have

$$1 \leq \sum_{i=1}^{\lfloor \frac{k}{2} \rfloor} \frac{E^i(\mathbb{T}_0)}{T_{max}(\mathbb{T}_0)} < \sum_{i=1}^{k} \frac{E^i(\mathbb{T}_0)}{T_{max}(\mathbb{T}_0)} \leq \sum_{i=1}^{k} \frac{C_{\tau_i} + S_{\tau_i}}{T_{\tau_i}} \tag{5.57}$$

$$\leq \sum_{i=1}^{n} \frac{C_{\tau_i}}{T_{\tau_i}} + \sum_{i=1}^{k} v^i(\mathbb{T}_0) + \sum_{i=n_0+1}^{n} \frac{S_{\tau_i}}{T_{\tau_i}}. \tag{5.58}$$

In other words, $\sum_{i=1}^{\lfloor \frac{k}{2} \rfloor} E^i(\mathbb{T}_0) \geq T_{max}(\mathbb{T}_0)$ implies that Equation (5.56) does not hold. Therefore, the schedulability test in Theorem 5.28 coincides with the suspension-oblivious schedulability test. We note that in the proof we utilized that $E^{\lfloor \frac{k}{2} \rfloor + 1}(\mathbb{T}_0) > 0$, but this holds since

$$0 < T_{max}(\mathbb{T}_0) \leq \sum_{i=1}^{\lfloor \frac{k}{2} \rfloor} E^i(\mathbb{T}_0) \leq \left\lfloor \frac{k}{2} \right\rfloor \cdot E^{\lfloor \frac{k}{2} \rfloor}(\mathbb{T}_0) \leq \left\lfloor \frac{k}{2} \right\rfloor \cdot E^{\lfloor \frac{k}{2} \rfloor + 1}(\mathbb{T}_0). \tag{5.59}$$

□

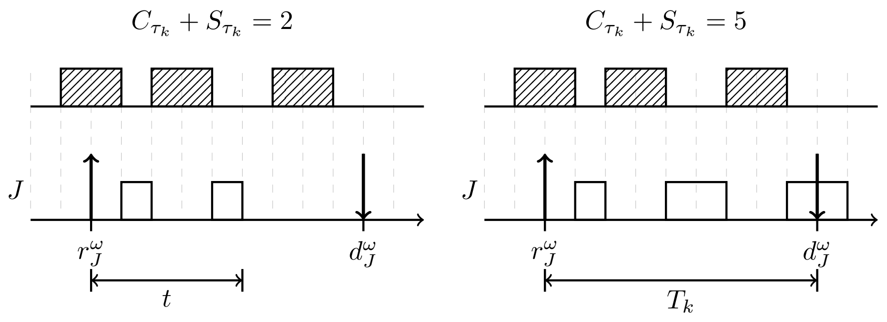


Figure 5.8.: Illustration of Observations 5.30 (left) and 5.31 (right). Patterned areas are interference by higher-priority jobs.

In the remaining part of Section 5.1.2, we omit this schedulability test in the discussions since it is the same as the suspension-oblivious approach for uniprocessor systems.

5.1.2.2 Method 1: Response Time Analysis

In this section, we introduce a new method to test the schedulability of an implicit-deadline sporadic task set under preemptive EDF scheduling. For this purpose, we estimate the interference from higher-priority jobs to calculate an upper bound on the WCRT for each task. We consider a task set $\mathbb{T} = \{\tau_1, \dots, \tau_n\}$ consisting of n implicit-deadline sporadic tasks abstracted with dynamic self-suspension. That is, each task $\tau_i \in \mathbb{T}$ can be abstracted as

$$\tau_i \in \text{Spor}(T_{\tau_i}) \cap \text{DynSS}(C_{\tau_i}, S_{\tau_i}). \tag{5.60}$$

In the following, we denote by $\mathcal{A}$ the preemptive EDF scheduling algorithm. Hence, for a system evolution $\omega \in \Omega$, the corresponding schedule is $[\mathcal{S}] = \mathcal{A}(\omega)$.

Let $\tau_k \in \mathbb{T} = \{\tau_1, \dots, \tau_n\}$ be any task and $J \in \mathbb{J}$ some job of τ_k. We define by

$$W_J^i(t) := \sum_{j \in \mathbb{N}, \tau_i(j) \geq_{prio} J} \text{exec}_{\tau_i(j)}^{[\mathcal{S}]}(r_J^\omega, r_J^\omega + t) \tag{5.61}$$

the interference by higher-priority jobs of τ_i during the interval $[r_J^\omega, r_J^\omega + t] \subseteq [r_J^\omega, d_J^\omega]$. In particular, the processor can work on and be suspended for J during an interval $[r_J^\omega, r_J^\omega + t]$ for $t - \sum_{i=1}^n W_J^i(t)$ time units. Hence, it finishes J during $[r_J^\omega, r_J^\omega + t]$ if $C_{\tau_k} + S_{\tau_k} \leq t - \sum_{i=1}^n W_J^i(t)$. Equivalently, if J cannot be finished during $[r_J^\omega, r_J^\omega + t]$, then $C_{\tau_k} + S_{\tau_k} > t - \sum_{i=1}^n W_J^i(t)$.

We derive the following two observations as depicted in Figure 5.8, which provide a general upper bound on the response time and a sufficient schedulability condition. These are (mainly implicitly) used similarly in classical response-time analysis and the underlying concepts are already applied for self-suspending tasks as in [LA13].

Observation 5.30. *If J meets its deadline, then we calculate an upper bound on the response time of J by choosing the minimal t with $0 \leq t \leq T_{\tau_k}$, such that*

$$C_{\tau_k} + S_{\tau_k} + \sum_{i=1}^{n} W_J^i(t) \leq t. \tag{5.62}$$

If no such t exists, then we cannot give a guaranteed WCRT, but the job may still meet its deadline, e.g., if the job completes earlier than in its worst case.

Observation 5.31. *If J does not meet its deadline, then it cannot be finished during the interval $[r_k, r_k + T_{\tau_k}]$, i.e.,*

$$C_{\tau_k} + S_{\tau_k} + \sum_{i=1}^{n} W_J^i(T_{\tau_k}) > T_{\tau_k}, \tag{5.63}$$

as illustrated on the right-hand side of Figure 5.9.

We start by determining the higher-priority jobs which may contribute to the response time of J. By the definition of EDF scheduling, jobs with a larger absolute deadline than J have a lower priority. We remove these jobs from the schedule for ease of notation, since they do not affect the response time of J. Furthermore, when determining which higher-priority jobs contribute to the response time of J, we assume that all higher-priority jobs meet their deadline. For simplicity, we assume that J is released at time 0. Other job releases and task phases are shifted accordingly (to negative time instants if necessary). This results in a modified system evolution ω' and a modified schedule $[\mathcal{S}'] = \mathcal{A}(\omega')$. In particular, $r_J^{\omega'} = 0$ and J is the job with maximal deadline $d_J^{\omega'}$.

We know that $W_J^k(t) = 0$ since all jobs of τ_k with higher priority than $J \in \tau_k$ meet their deadline which is at most $r_J^{\omega'} = 0$. For a task τ_i with $i \neq k$ there are at most $\left\lfloor \frac{T_{\tau_k}}{T_{\tau_i}} \right\rfloor$ of the remaining jobs released in $[0, t] \subseteq [0, T_{\tau_k}]$. Furthermore, there may be one additional overlapping job which is released before $r_J^{\omega'} = 0$ but finishes after 0.

Lemma 5.32. *For a task $\tau_i \neq \tau_k$ only the last $\left\lfloor \frac{T_{\tau_k}}{T_{\tau_i}} \right\rfloor + 1$ jobs may be executed in $[0, t]$.*[4]

Proof. The other jobs of τ_i have a deadline of at most

$$T_{\tau_k} - \left(\left\lfloor \frac{T_{\tau_k}}{T_{\tau_i}} \right\rfloor + 1 \right) \cdot T_{\tau_i} \leq T_{\tau_k} - \left(\frac{T_{\tau_k}}{T_{\tau_i}} \right) \cdot T_{\tau_i} = 0. \tag{5.64}$$

Since they meet their deadline, they are finished before the release of J. □

Remark 5.33. *In fact only the last $\left\lceil \frac{T_{\tau_k}}{T_{\tau_i}} \right\rceil$ jobs may be executed in $[0, t]$, which can be shown by a similar proof. Only for convenience, we use the bound given in the lemma and set the interference by the additional job to 0 if necessary.*

[4]We highlight that we consider a limited schedule here, which was obtained by the original schedule. The *last x jobs* are the first x jobs counted from the end of the schedule.

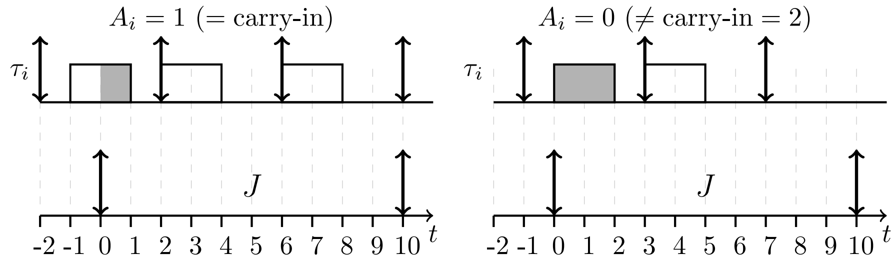


Figure 5.9.: Interference by higher-priority jobs of task $\tau_i = (2, 0, 4, 4)$ during the interval $[r_J^{\omega'}, f_J] = [0, 10]$. Left: interference = 5. Right: interference = 4.

For the last $\left\lfloor \frac{T_{\tau_k}}{T_{\tau_i}} \right\rfloor$ jobs of τ_i we estimate the time they may be executed in $[0, t]$ by C_{τ_i} each. Although this seems to be very pessimistic if t is very small, this estimation is sufficient for our analysis. However, we have to be more careful with the $\left(\left\lfloor \frac{T_{\tau_k}}{T_{\tau_i}} \right\rfloor + 1\right)$-th last job of τ_i which is released before 0 but may still finish after 0. Note that by our assumption, 0 is not a lower bound on the phase but the release of J.

Definition 5.34. *We define $J^i \in \mathbb{J}$ to be the $\left(\left\lfloor \frac{T_{\tau_k}}{T_{\tau_i}} \right\rfloor + 1\right)$-th last job of τ_i. The time J^i is interfering in $[0, t]$ is denoted by $A_i := \text{exec}_{J^i}^{[\mathcal{S}']}(0, t)$.*

To derive a good response time bound, in general, estimation of the carry-in, i.e., interference by higher-priority jobs which are released before the job under analysis, is essential as outlined in [BC07]. Since A_i is the interference by a certain job released before J, it serves the role of the carry-in. Nevertheless, there is a slight difference as depicted in Figure 5.9. Although in the scenario on the left-hand side the carry-in and A_i coincide, the value of A_i is 0 in the right scenario since the 3rd last job of τ_i finishes before J is released. We observe that A_i has its highest value when the interference is maximized, i.e., on the left-hand side. Hence, independently examining A_i is relatively safe for estimating interference. This is not the case for the carry-in, for which in general more caution is required.

With this definition of A_i, the time a task $\tau_i \neq \tau_k$ is interfering during $[0, t]$ is

$$W_J^i(t) \leq \left\lfloor \frac{T_{\tau_k}}{T_{\tau_i}} \right\rfloor \cdot C_{\tau_i} + A_i, \tag{5.65}$$

and we will examine how to estimate the interference A_i by J^i during $[0, t]$ later on. First, we show how the estimation for $W_J^i(t)$ can be utilized, assuming that A_i is given.

Lemma 5.35. *We consider a job J of τ_k. If all higher-priority jobs meet their deadline and*

$$S_{\tau_k} + C_{\tau_k} + \sum_{i \neq k} \left(\left\lfloor \frac{T_{\tau_k}}{T_{\tau_i}} \right\rfloor \cdot C_{\tau_i} + A_i \right) \leq T_{\tau_k} \tag{5.66}$$

holds, then also J meets its deadline.

Proof. If J does not meet its deadline, then we obtain by Observation 5.31 that

$$T_{\tau_k} < S_{\tau_k} + C_{\tau_k} + \sum_{i \neq k} W_J^i(T_{\tau_k}) \leq S_{\tau_k} + C_{\tau_k} + \sum_{i \neq k} \left(\left\lfloor \frac{T_{\tau_k}}{T_{\tau_i}} \right\rfloor \cdot C_{\tau_i} + A_i \right), \tag{5.67}$$

which contradicts (5.66). □

Under the assumption, that A_i is given for all i, our schedulability test can be formulated as follows:

Proposition 5.36. *Let $\mathbb{T} = \{\tau_1, \ldots, \tau_n\}$ be an implicit-deadline task set. If the equation*

$$\widetilde{r}_k := S_{\tau_k} + C_{\tau_k} + \sum_{i \neq k} \left(\left\lfloor \frac{T_{\tau_k}}{T_{\tau_i}} \right\rfloor \cdot C_{\tau_i} + A_i \right) \leq T_{\tau_k} \tag{5.68}$$

holds for all indices $k \in \{1, \ldots, n\}$, then $\mathbb{T}$ is schedulable by preemptive EDF and the WCRT $R^{\mathcal{A}}_{\tau_k}$ of τ_k is upper bounded by $\widetilde{r}_k$ for each $k \in \{1, \ldots, n\}$.

Proof. We use an indirect proof to show that $\mathbb{T}$ is schedulable by preemptive EDF. In the following, we assume that $\mathbb{T}$ is not schedulable. Let J be the job with the highest priority whose deadline is not met and let τ_k be the task corresponding to J. In this case, Lemma 5.35 shows that J is schedulable, which contradicts our assumption.

We have proven that the task set $\mathbb{T}$ is schedulable, and we are able to use Equation (5.65) together with (5.62) to upper bound the response time of each job of τ_k by $t = \widetilde{r}_k$ for $k = 1, \ldots, n$. □

The aforementioned proposition analyzes the scenario with worst-case release pattern, i.e., all jobs are released as late as possible, and the interference is maximized as on the left-hand side of Figure 5.9. In such a scenario, A_i and the carry-in coincide. Hence, in the following estimation of A_i we observe mainly typical carry-in bounds as provided in the literature [BC07; GH05; LA13]. Hereinafter, we provide rigorous proofs where suspension is integrated seamlessly into carry-in analysis.

For the estimation of A_i we introduce two approaches, where the first one examines the deadline of the overlapping job J^i and the second one utilizes its response time.

Lemma 5.37. *We can estimate A_i by*

$$A_i \leq \min \left\{ C_{\tau_i}, T_{\tau_k} - \left\lfloor \frac{T_{\tau_k}}{T_{\tau_i}} \right\rfloor \cdot T_{\tau_i} \right\} =: \widetilde{A}_i^1. \tag{5.69}$$

Proof. Since there are $\left\lfloor \frac{T_{\tau_k}}{T_{\tau_i}} \right\rfloor$ other jobs of τ_i between the deadline of J^i and T_{τ_k}, the deadline of J^i is at most $d_{Ji}^{\omega'} \leq T_{\tau_k} - \left\lfloor \frac{T_{\tau_k}}{T_{\tau_i}} \right\rfloor \cdot T_{\tau_i}$. Due to the assumption that all jobs with higher-priority than J meet their deadline, J^i meets its deadline as well and any possible interference by J^i must happen in the interval from $[0, d_{Ji}^{\omega'}]$.

On the other hand, we also know that the interference by J^i is bounded by the WCET C_{τ_i}. Since both values bound A_i, we take the minimum of them. □

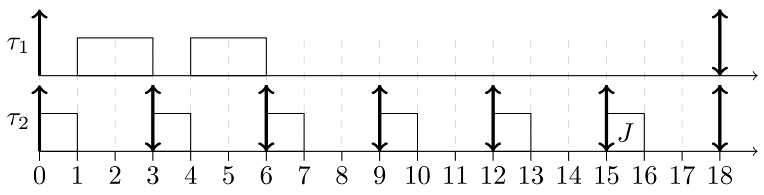


Figure 5.10.: For these tasks, Lemma 5.37 gives a bad estimation for A_1.

This estimation may be pessimistic, e.g., when $T_{\tau_1} \gg C_{\tau_1} > T_{\tau_2} > 0$, as shown in Figure 5.10 for the implicit-deadline task set $\mathbb{T} = \{\tau_1, \tau_2\}$ with:

- $\tau_1 \in \text{Spor}(T_{\tau_1} = 18) \cap \text{DynSS}(C_{\tau_1} = 4, S_{\tau_1} = 0)$
- $\tau_2 \in \text{Spor}(T_{\tau_2} = 3) \cap \text{DynSS}(C_{\tau_2} = 1, S_{\tau_2} = 0)$

If we estimate A_1 by $\widetilde{A}_1^1 = 3$, then we can bound $\widetilde{r}_2$ only by 4 which exceeds the relative deadline $T_{\tau_2} = 3$. Actually, $A_1 = 0$ and $\widetilde{r}_2 = 1$ is smaller than T_{τ_2}. We note that $\widetilde{r}_2 = 1$ is even a precise bound on the response time of τ_2.

If an upper bound R_i on the response time of J^i is known, A_i can be estimated with the following approach, that performs much better for the task set in Figure 5.10.

Lemma 5.38. *We can estimate A_i by*

$$A_i \leq \max\left\{T_{\tau_k} + R_i - \left(\left\lfloor \frac{T_{\tau_k}}{T_{\tau_i}} \right\rfloor + 1\right) \cdot T_{\tau_i}, 0\right\} =: \widetilde{A}_i^2, \tag{5.70}$$

where R_i is either the WCRT $R_{\tau_i}^{\mathcal{A}}$ of τ_i or any other upper bound on the response time of job J^i.

Proof. Due to its definition, we know that the job J^i is released no later than $T_{\tau_k} - \left(\left\lfloor \frac{T_{\tau_k}}{T_{\tau_i}} \right\rfloor + 1\right) \cdot T_{\tau_i}$ and therefore must be finished no later than $T_{\tau_k} - \left(\left\lfloor \frac{T_{\tau_k}}{T_{\tau_i}} \right\rfloor + 1\right) \cdot T_{\tau_i} + R_i$, which may be less than 0 if J^i is actually finished before time 0. As a result, the maximum amount of time that J^i may interfere with the interval $[0, t] \subseteq [0, T_{\tau_k}]$ is bounded by the maximum of 0 or $T_{\tau_k} + R_i - \left(\left\lfloor \frac{T_{\tau_k}}{T_{\tau_i}} \right\rfloor + 1\right) \cdot T_{\tau_i}$, since the interference cannot be negative. $\square$

The examination in this section leads to the following response-time bound $\widetilde{R}_k$ which is not as tight as $\widetilde{r}_k$ but directly computable since it uses an estimation $\widetilde{A}_i$ of A_i.

Lemma 5.39. *Let J be a job of τ_k and let all jobs with higher priority than J meet their deadline. Further, we define*

$$\widetilde{R}_k := C_{\tau_k} + S_{\tau_k} + \sum_{i \neq k} \left(\left\lfloor \frac{T_{\tau_k}}{T_{\tau_i}} \right\rfloor \cdot C_{\tau_i} + \widetilde{A}_i\right) \tag{5.71}$$

where each $\widetilde{A}_i$ is defined as the minimum of $\widetilde{A}_i^1$ and $\widetilde{A}_i^2$ from Equation (5.69) and (5.70), i.e.,

$$\widetilde{A}_i := \min\left\{C_{\tau_i}, \max\left\{T_{\tau_k} + R_i - \left(\left\lfloor \frac{T_{\tau_k}}{T_{\tau_i}} \right\rfloor + 1\right) \cdot T_{\tau_i}, 0\right\}\right\} \tag{5.72}$$

where R_i is an upper bound on the WCRT $R_{\tau_i}^{\mathcal{A}}$ of τ_i or $R_i := T_{\tau_i}$. If $\widetilde{R}_k \leq T_{\tau_k}$, then J meets its deadline and $\widetilde{R}_k$ is an upper bound on the response time of J.

Proof. Lemma 5.37 and Lemma 5.38 yield $\widetilde{r}_k \leq \widetilde{R}_k$. If $\widetilde{R}_k \leq T_{\tau_k}$, then we also have $\widetilde{r}_k \leq \widetilde{R}_k \leq T_{\tau_k}$ and by Proposition 5.36 the job J meets its deadline. The response time of J is upper bounded by $\widetilde{r}_k$ and hence, also by $\widetilde{R}_k$. □

Using this lemma, we obtain a response-time bound valid for all jobs J of τ_k. Hence, $\widetilde{R}_k$ serves as a bound on the WCRT of τ_k and can be used for the estimation of $\widetilde{A}_k$ for other tasks in Lemma 5.39.

Theorem 5.40. *Let $\mathbb{T} = \{\tau_1, \dots, \tau_n\}$ be a sporadic implicit-deadline task set. If for all $\tau_k \in \mathbb{T}$ the value $\widetilde{R}_k$ from (5.71) is at most T_{τ_k}, then the task set is schedulable by preemptive EDF.*

Proof. We know that $\widetilde{R}_k \geq \widetilde{r}_k$ for all $k \in \{1, \dots, n\}$. If $T_{\tau_k} \geq \widetilde{R}_k$ for all k, then also $T_{\tau_k} \geq \widetilde{r}_k$. Hence, by Proposition 5.36, the task set $\mathbb{T}$ is schedulable by preemptive EDF and the WCRT of each τ_k is upper bounded by $\widetilde{r}_k \leq \widetilde{R}_k$. □

We can further improve the test by the following observations. If we do not estimate A_i by C_{τ_i}, then we assume that J^i interferes for the whole time interval until the deadline (for $\widetilde{A}_i^1$) or until the finishing time (for $\widetilde{A}_i^2$). Although those intervals intersect for different jobs, we add the interference several times, which is overly pessimistic.

Proposition 5.41. *Let $I \subseteq \{1, \dots, n\} - \{k\}$ and let R_i be a bound on the response time of J^i for all $i \in I$. If no such bound is given, set $R_i = T_{\tau_i}$. We can estimate $\sum_{i \in I} A_i$ as*

$$\sum_{i \in I} A_i \leq \max\left\{\max_{i \in I}\left\{T_{\tau_k} + R_i - \left(\left\lfloor \frac{T_{\tau_k}}{T_{\tau_i}} \right\rfloor + 1\right) T_{\tau_i}\right\}, 0\right\}. \tag{5.73}$$

Proof. Recall, that $f_{J^i}^{[\mathcal{S}'],\omega'}$ and $d_{J^i}^{\omega'}$ are the finishing time and the absolute deadline of the job J^i from Definition 5.34, respectively. Moreover, we still assume that $r_J^{\omega'} = 0$. The job $J^i, i \in I$ may interfere after the release of J only during

$$\left[0, \max\left\{\max_{i \in I}\left\{f_{J^i}^{[\mathcal{S}'],\omega'}\right\}, 0\right\}\right]. \tag{5.74}$$

Furthermore, the processor can only be occupied by one job at a time. Hence, the sum of all interference of $J^i, i \in I$ is bounded by $\sum_{i \in I} A_i \leq \max\left\{\max_{i \in I}\left\{f_{J^i}^{[\mathcal{S}'],\omega'}\right\}, 0\right\}$.

We know that $d_{J^i}^{\omega'} \leq T_{\tau_k} - \left(\left\lfloor \frac{T_{\tau_k}}{T_{\tau_i}} \right\rfloor\right) T_{\tau_i}$ is a bound on the absolute deadline of J^i and $f_{J^i}^{[\mathcal{S}'],\omega'} \leq T_{\tau_k} + R_i - \left(\left\lfloor \frac{T_{\tau_k}}{T_{\tau_i}} \right\rfloor + 1\right) T_{\tau_i}$ is an upper bound on the finishing time of J^i if a bound R_i on the response time of J^i is given. Since J^i finishes on time, we have $f_{J^i}^{[\mathcal{S}'],\omega'} \leq d_{J^i}^{\omega'}$ and the bound for the absolute deadline holds for the finishing time. Using both bounds in $\sum_{i \in I} A_i \leq \max\left\{\max_{i\in I}\left\{f_{J^i}^{[\mathcal{S}'],\omega'}\right\},0\right\}$ concludes the proof. □

For a specific job J^i, the interval starting from 0 up to the threshold given by the right-hand side of Equation (5.73) could also contain the interference by other jobs which are not overlapping. We use this observation to improve the test.

Proposition 5.42. *Let $i \in \{1,\dots,n\} \setminus \{k\}$. The interference by the jobs of τ_i after a threshold $m \in [0, T_{\tau_k}]$ is bounded by*

$$\left\lceil \frac{T_{\tau_k} - m}{T_{\tau_i}} \right\rceil \cdot C_{\tau_i}. \tag{5.75}$$

Proof. The interference can only occur for tasks with deadline after m. For task τ_i, this is only possible for the last $\left\lceil \frac{T_{\tau_k}-m}{T_{\tau_i}} \right\rceil$ jobs. They contribute an interference of at most C_{τ_i} each. □

Both ideas can be used to define a better estimation $\widetilde{R}_k$ than in Equation (5.71). With this new definition of $\widetilde{R}_k$, Lemma 5.39 and Theorem 5.40 still hold. In the following we formulate this improved schedulability test.

Response-Time-Based Schedulability Test Let $\mathbb{T} = \{\tau_1, \dots, \tau_n\}$ be a sporadic implicit-deadline task set. Algorithm 2 is a sufficient test for the schedulability of $\mathbb{T}$ by preemptive EDF scheduling. If Algorithm 2 returns *schedulable*, then all $\widetilde{R}_k \leq T_{\tau_k}$, i.e., the task set $\mathbb{T}$ is schedulable according to Theorem 5.40 and its improvements in Proposition 5.41 and Proposition 5.42.

The algorithm estimates the response time of task τ_k for $k = n, \dots, 1$. At first, in line 4-5, it calculates $\widetilde{A}_i^k := T_{\tau_k} + R_i - \left(\left\lfloor \frac{T_{\tau_k}}{T_{\tau_i}} \right\rfloor + 1\right) T_{\tau_i}$ from Lemma 5.38 where R_i is an already computed response-time bound (for $i > k$) or T_{τ_i}. Then the algorithm tries out different thresholds m_j^k. For those tasks where A_i can be neglected (i.e., tasks with index in I_j^k) as in Proposition 5.41, it just adds the interference of the other $\left\lfloor \frac{T_{\tau_k}}{T_{\tau_i}} \right\rfloor$ jobs after the threshold, i.e., $\min\left\{\left\lfloor \frac{T_{\tau_k}}{T_{\tau_i}} \right\rfloor, \left\lceil \frac{T_{\tau_k}-m_j^k}{T_{\tau_i}} \right\rceil\right\} C_{\tau_i}$ due to Proposition 5.42. For the other tasks (i.e., tasks with index in $I^k \setminus I_j^k$) we estimate A_i by C_{τ_i} with the bound from Lemma 5.37. Together with Proposition 5.42 we obtain $\min\left\{\left\lfloor \frac{T_{\tau_k}}{T_{\tau_i}} \right\rfloor + 1, \left\lceil \frac{T_{\tau_k}-m_j^k}{T_{\tau_i}} \right\rceil\right\} C_{\tau_i}$. The algorithm then calculates the response-time bound $\widetilde{R}_k(j)$ by adding the threshold m_j^k, the interference after the threshold, and $C_{\tau_k} + S_{\tau_k}$. Finally, the algorithm computes a response-time bound $\widetilde{R}_k(0)$ for the threshold $m = 0$, in line 10. It derives the best bound by taking the minimum, in line 11.

Algorithm 2 Response-Time-Based Schedulability Test.

```
1: Sort $\tau_1, \dots, \tau_n$, such that $T_{\tau_1} \leq \dots \leq T_{\tau_n}$.
2: for $k = n, \dots, 1$ do
3:     $I^k := \{1, \dots, n\} - \{k\}$
4:     for $i \in I^k$ do                          ▷ Carry-in estimation from Lemma 5.38.
5:         $\widetilde{A}_i^k := \left\{ \begin{array}{ll} T_{\tau_k} - \left\lfloor \frac{T_{\tau_k}}{T_{\tau_i}} \right\rfloor \cdot T_{\tau_i}, & i < k \\ T_{\tau_k} + \widetilde{R}_i - \left( \left\lfloor \frac{T_{\tau_k}}{T_{\tau_i}} \right\rfloor + 1 \right) \cdot T_{\tau_i}, & i > k \end{array} \right\}$
6:     for $j \in I^k$ do                          ▷ Response-time bounds for different thresholds.
7:         $m_j^k := \max \left\{ \widetilde{A}_j^k, 0 \right\}$
8:         $I_j^k := \left\{ i \in I^k \;\middle|\; \widetilde{A}_i^k \leq \widetilde{A}_j^k \right\}$
9:         $\widetilde{R}_k(j) := \sum_{i \in I^k \setminus I_j^k} \min \left\{ \left\lfloor \frac{T_{\tau_k}}{T_{\tau_i}} \right\rfloor + 1, \left\lceil \frac{T_{\tau_k} - m_j^k}{T_{\tau_i}} \right\rceil \right\} C_{\tau_i}$
                   $+ \sum_{i \in I_j^k} \min \left\{ \left\lfloor \frac{T_{\tau_k}}{T_{\tau_i}} \right\rfloor, \left\lceil \frac{T_{\tau_k} - m_j^k}{T_{\tau_i}} \right\rceil \right\} C_{\tau_i}$
                   $+ C_{\tau_k} + S_{\tau_k} + m_j^k$
                                                   ▷ Response-time bound without threshold.
10:    $\widetilde{R}_k(0) := C_{\tau_k} + S_{\tau_k} + \sum_{i \in I^k} \left( \left\lfloor \frac{T_{\tau_k}}{T_{\tau_i}} \right\rfloor + 1 \right) C_{\tau_i}$
11:    $\widetilde{R}_k := \min \left\{ \widetilde{R}_k(j) \;\middle|\; j \in I^k \cup \{0\} \right\}$
12:    if $\widetilde{R}_k > T_{\tau_k}$ then
13:        return not schedulable
14: return schedulable
```

Theoretical Evaluation We compare the response-time analysis presented in this section with the existing tests summarized in Section 5.1.2.1.

Comparison with Suspension-Oblivious (Theorem 5.26): Although we show in Section 5.1.2.4 that our method has a better performance, both schedulability tests do not dominate each other. There are task sets which pass our schedulability test, although the suspension-oblivious approach cannot give any schedulability guarantees.

Example 5.43. *Consider the implicit-deadline task set* $\mathbb{T} = \{\tau_1, \tau_2\}$ *with:*

- $\tau_1 \in \text{Spor}(T_{\tau_1} = 5) \cap \text{DynSS}(C_{\tau_1} = 1, S_{\tau_1} = 2)$
- $\tau_2 \in \text{Spor}(T_{\tau_2} = 7) \cap \text{DynSS}(C_{\tau_2} = 1, S_{\tau_2} = 3)$

Then by our test, the WCRTs for the tasks are bounded by:

- $\widetilde{R}_2 = 1 + 3 + (\lfloor \frac{7}{5} \rfloor + 1) \cdot 1 = 6 \leq 7 = T_{\tau_2}$
- $\widetilde{R}_1 = 1 + 2 + (\lfloor \frac{5}{7} \rfloor + 1) \cdot 1 = 4 \leq 5 = T_{\tau_1}$

Hence, $\mathbb{T}$ is schedulable according to our test. However, for the condition in Theorem 5.26 we obtain $\frac{1+2}{5} + \frac{1+3}{7} > 1$, i.e., the task set $\mathbb{T}$ cannot be deemed schedulable by the suspension-oblivious schedulability test.

On the other hand, there are some task sets that are schedulable by Theorem 5.26 but not according to our test:

Example 5.44. *Consider the implicit-deadline task set $\mathbb{T} = \{\tau_1, \tau_2\}$ with:*

- $\tau_1 \in \mathrm{Spor}(T_{\tau_1} = 6) \cap \mathrm{DynSS}(C_{\tau_1} = 3, S_{\tau_1} = 0)$
- $\tau_2 \in \mathrm{Spor}(T_{\tau_2} = 20) \cap \mathrm{DynSS}(C_{\tau_2} = 10, S_{\tau_2} = 0)$

Then the task set is schedulable by Theorem 5.26. However, our method computes the value $\widetilde{R}_2 = 10 + \left\lfloor \frac{20-2}{6} \right\rfloor \cdot 3 + 2 = 21$ which is higher than the relative deadline of τ_2.

Comparison with the Workload-Based Schedulability Test (Theorem 5.27) by Liu and Anderson: Comparing their bounds with the ones described in this section yields the following proposition.

Proposition 5.45. *If all tasks are implicit-deadline self-suspending tasks, our method dominates the analysis by Liu and Anderson [LA13] applied to uniprocessor systems.*

Proof. For contradiction, we assume that our schedulability test fails but the one described in Theorem 5.27 succeeds. First, we want to find a possible value for ξ_ℓ. If Algorithm 2 stops without determining the schedulability of the task set, then there is some $\ell \in \{0, 1, \ldots, n\}$ with $\widetilde{R}_\ell > T_{\tau_\ell}$. We deduce that $T_{\tau_\ell} < \widetilde{R}_\ell \leq \widetilde{R}_\ell(0) = C_{\tau_\ell} + S_{\tau_\ell} + \sum_{i \neq \ell} \left(\left\lfloor \frac{T_{\tau_\ell}}{T_{\tau_i}} \right\rfloor + 1 \right) \cdot C_{\tau_i} \leq C_{\tau_\ell} + S_{\tau_\ell} + T_{\tau_\ell} \sum_i \frac{C_{\tau_i}}{T_{\tau_i}} + \sum_i C_{\tau_i}$. This implies $C_{\tau_\ell} + S_{\tau_\ell} + \sum_i C_{\tau_i} > T_{\tau_\ell} \cdot \left(1 - \sum_i \frac{C_{\tau_i}}{T_{\tau_i}}\right)$ and hence $\frac{C_{\tau_\ell} + S_{\tau_\ell} + \sum_i C_{\tau_i}}{1 - \sum_i \frac{C_{\tau_i}}{T_{\tau_i}}} > T_{\tau_\ell}$. We conclude that the interval $\left[T_{\tau_\ell}, \frac{C_{\tau_\ell} + S_{\tau_\ell} + \sum_i C_{\tau_i}}{1 - \sum_i \frac{C_{\tau_i}}{T_{\tau_i}}} \right)$ is not empty and $\xi_\ell = T_{\tau_\ell}$ has to be considered in the schedulability test according to Theorem 5.27.

By setting $\xi_\ell = T_{\tau_\ell}$ and $s_{\ell,j} = S_{\tau_\ell}$, we obtain

$$\sum_{\tau_i \in \tau^s} \max \{W_{nc}(\tau_i), W_c(\tau_i)\} + \sum_{\tau_j \in \tau^e} W_{nc}(\tau_j) \leq T_{\tau_\ell} - C_{\tau_\ell} - S_{\tau_\ell}. \tag{5.76}$$

Furthermore, we know that the set τ^e is empty and $\tau^s = \mathbb{T}$ by assumption. Hence, the left-hand side (LHS) of Inequality (5.76) is $\sum_{\tau_i \in \mathbb{T}} \max \{W_{nc}(\tau_i), W_c(\tau_i)\}$, and we can bound it from below by $W_{nc}(\tau_\ell) + \sum_{\tau_i \neq \tau_\ell} W_c(\tau_i) \leq LHS$.

The computation of $W_{nc}(\tau_\ell)$ yields that $W_{nc}(\tau_\ell) = \min \left\{ \left\lfloor \frac{T_{\tau_\ell}}{T_{\tau_\ell}} \right\rfloor C_{\tau_\ell} - C_{\tau_\ell}, T_{\tau_\ell} - T_{\tau_\ell} \right\} = 0$, and $W_c(\tau_i), i \neq \ell$ is defined by $W_c(\tau_i) = \min \{\Delta(\tau_i, T_{\tau_\ell}), T_{\tau_\ell} - C_{\tau_\ell} - S_{\tau_\ell} + 1\}$ with $\Delta(\tau_i, T_{\tau_\ell}) = \left(\left\lceil \frac{T_{\tau_\ell}}{T_{\tau_i}} \right\rceil - 1 \right) C_{\tau_i} + \min \left\{ C_{\tau_i}, T_{\tau_\ell} - \left\lceil \frac{T_{\tau_\ell}}{T_{\tau_i}} \right\rceil T_{\tau_i} + T_{\tau_i} \right\}$.

By using the four inequalities[5] $T_{\tau_\ell} - C_{\tau_\ell} - S_{\tau_\ell} + 1 \geq 0$, $\left(\left\lceil \frac{T_{\tau_\ell}}{T_{\tau_i}} \right\rceil - 1\right) \geq 0$, $C_{\tau_i} \geq 0$ and $T_{\tau_\ell} - \left\lceil \frac{T_{\tau_\ell}}{T_{\tau_i}} \right\rceil T_{\tau_i} + T_{\tau_i} \geq T_{\tau_\ell} - \left(\frac{T_{\tau_\ell}}{T_{\tau_i}} + 1\right) T_{\tau_i} + T_{\tau_i} = 0$, we obtain that $W_c(\tau_i)$ is not negative for all $i \neq \ell$. We conclude that $W_c(\tau_i) = \Delta(\tau_i, T_{\tau_i})$ since otherwise the left-hand side from Equation (5.76) would be at least $T_{\tau_\ell} - C_{\tau_\ell} - S_{\tau_\ell} + 1$ and hence $T_{\tau_\ell} - C_{\tau_\ell} - S_{\tau_\ell} + 1 \leq LHS \leq T_{\tau_\ell} - C_{\tau_\ell} - S_{\tau_\ell}$, which is a contradiction.

In the following, we show that even

$$\Delta(\tau_i, T_{\tau_\ell}) = \left\lfloor \frac{T_{\tau_\ell}}{T_{\tau_i}} \right\rfloor C_{\tau_i} + \min\left\{C_{\tau_i}, T_{\tau_\ell} - \left\lfloor \frac{T_{\tau_\ell}}{T_{\tau_i}} \right\rfloor T_{\tau_i}\right\} \tag{5.77}$$

holds. For all T_{τ_ℓ} which are not in $\mathbb{Z}T_{\tau_i}$, we know that $\left\lceil \frac{T_{\tau_\ell}}{T_{\tau_i}} \right\rceil - 1 = \left\lfloor \frac{T_{\tau_\ell}}{T_{\tau_i}} \right\rfloor$. By using this in the definition of $\Delta(\tau_i, T_{\tau_\ell})$, we obtain the result from Equation (5.77). In the other case $T_{\tau_\ell} \in \mathbb{Z}T_{\tau_i}$, we have $\left\lceil \frac{T_{\tau_\ell}}{T_{\tau_i}} \right\rceil = \frac{T_{\tau_\ell}}{T_{\tau_i}} = \left\lfloor \frac{T_{\tau_\ell}}{T_{\tau_i}} \right\rfloor$ which yields $\Delta(\tau_i, T_{\tau_\ell}) = \left(\frac{T_{\tau_\ell}}{T_{\tau_i}} - 1\right) C_{\tau_i} + \min\left\{C_{\tau_i}, T_{\tau_\ell} - \frac{T_{\tau_\ell}}{T_{\tau_i}} \cdot T_{\tau_i} + T_{\tau_i}\right\} = \frac{T_{\tau_\ell}}{T_{\tau_i}} C_{\tau_i} = \left\lfloor \frac{T_{\tau_\ell}}{T_{\tau_i}} \right\rfloor C_{\tau_i} + \min\left\{C_{\tau_i}, T_{\tau_\ell} - \left\lfloor \frac{T_{\tau_\ell}}{T_{\tau_i}} \right\rfloor T_{\tau_i}\right\}$.

We conclude that if the test by Liu and Anderson as in Theorem 5.27 indicates schedulability of the task system, then the inequality $C_{\tau_\ell} + S_{\tau_\ell} + \sum_{\tau_i \neq \tau_\ell} \left\lfloor \frac{T_{\tau_\ell}}{T_{\tau_i}} \right\rfloor C_{\tau_i} + \min\{C_{\tau_i}, L(i)\} \leq T_{\tau_\ell}$ with $L(i) = T_{\tau_\ell} - \left\lfloor \frac{T_{\tau_\ell}}{T_{\tau_i}} \right\rfloor T_{\tau_i}$ holds. It is left to show that from this inequality also $\widetilde{R}_\ell \leq T_{\tau_\ell}$ follows.

Let $I := \{i \in \{1, 2, \ldots, n\} \mid i \neq \ell, L(i) \leq C_{\tau_i}\}$. If the set I is empty, then $\widetilde{R}_\ell(0) = C_{\tau_\ell} + S_{\tau_\ell} + \sum_{\tau_i \neq \tau_\ell} \left\lfloor \frac{T_{\tau_\ell}}{T_{\tau_i}} \right\rfloor C_{\tau_i} + \min\{C_{\tau_i}, L(i)\}$ and we get $\widetilde{R}_\ell \leq \widetilde{R}_\ell(0) \leq T_{\tau_\ell}$, which contradicts the assumption that $\widetilde{R}_\ell > \widetilde{T}_{\tau_\ell}$. Otherwise, if I is not empty, we define $\tilde{j} := \arg\max_{i \in I}\{\widetilde{A}_i^\ell\}$. If there is not a unique integer for the maximum, we choose one among them. In the following we show that the right-hand side of

$$\widetilde{R}_\ell(\tilde{j}) \leq C_{\tau_\ell} + S_{\tau_\ell} + \sum_{\tau_i \neq \tau_\ell} \left\lfloor \frac{T_{\tau_\ell}}{T_{\tau_i}} \right\rfloor C_{\tau_i} + \sum_{i \in I^\ell - I^\ell_{\tilde{j}}} C_{\tau_i} + \max\left\{\widetilde{A}^\ell_{\tilde{j}}, 0\right\} \tag{5.78}$$

is upper bounded by $C_{\tau_\ell} + S_{\tau_\ell} + \sum_{\tau_i \neq \tau_\ell} \left(\left\lfloor \frac{T_{\tau_\ell}}{T_{\tau_i}} \right\rfloor C_{\tau_i} + \min\{C_{\tau_i}, L(i)\}\right)$. Since the term $C_{\tau_\ell} + S_{\tau_\ell} + \sum_{\tau_i \neq \tau_\ell} \left\lfloor \frac{T_{\tau_\ell}}{T_{\tau_i}} \right\rfloor C_{\tau_i}$ exists in both expressions, we consider only the remaining part $\sum_{i \in I^\ell - I^\ell_{\tilde{j}}} C_{\tau_i} + \max\left\{\widetilde{A}^\ell_{\tilde{j}}, 0\right\}$.

Since $\widetilde{A}^\ell_{\tilde{j}} \leq L(\tilde{j}) \leq C_{\tilde{j}}$ and $0 \leq L(\tilde{j})$, we have $\max\left\{\widetilde{A}^\ell_{\tilde{j}}, 0\right\} \leq L(\tilde{j}) = \min\left\{C_{\tilde{j}}, L(\tilde{j})\right\}$ for the second summand. Furthermore, if $i \in I^\ell - I^\ell_{\tilde{j}}$, then by definition $\widetilde{A}^\ell_i > \widetilde{A}^\ell_{\tilde{j}}$ and $i \notin I$. Hence, we obtain $L(i) > C_{\tau_i}$ and $\sum_{i \in I^\ell - I^\ell_{\tilde{j}}} C_{\tau_i} = \sum_{i \in I^\ell - I^\ell_{\tilde{j}}} \min\{C_{\tau_i}, L(i)\} \leq \sum_{\substack{i \neq \ell \\ i \neq \tilde{j}}} \min\{C_{\tau_i}, L(i)\}$ for the first summand. We conclude that in the case $I \neq \emptyset$, the

[5] We know that $T_{\tau_\ell} \geq C_{\tau_\ell} + S_{\tau_\ell}$ since otherwise τ_ℓ can miss its deadline, $\mathbb{T}$ is not schedulable, and both schedulability tests would fail.

inequality $\widetilde{R}_\ell(\tilde{j}) \leq C_{\tau_\ell} + S_{\tau_\ell} + \sum_{i \neq \ell} \left\lfloor \frac{T_{\tau_\ell}}{T_{\tau_i}} \right\rfloor C_{\tau_i} + \sum_{i \neq \ell} \min\{C_{\tau_i}, L(i)\} \leq T_{\tau_\ell}$ also holds, which contradicts our assumption. $\square$

We note that we did not examine, whether our schedulability test still dominates the method by Liu and Anderson [LA13] if some tasks are execution-only (i.e., $\tau^e \neq \emptyset$).

Remarks on Extensions For a more rigorous proof and dominance discussion we focused our attention on the fundamental case with implicit-deadline uniprocessor systems. Nevertheless, with minor adjustment, the response-time analysis can also be applied to constrained-deadline sporadic real-time task sets, in which $D_{\tau_i} \leq T_{\tau_i}$ for every τ_i. Furthermore, this analysis, which is based on Equation (5.68) in Proposition 5.36, can be modified to handle global EDF on M processors as follows:

$$\widetilde{r}_k := S_{\tau_k} + C_{\tau_k} + \frac{\sum_{i \neq k} \left(\left\lfloor \frac{T_{\tau_k}}{T_{\tau_i}} \right\rfloor \cdot C_{\tau_i} + A_i \right)}{M} \leq T_{\tau_k}. \tag{5.79}$$

That is, (i) when task τ_k suspends, no job is executed on any of the M processors, and (ii) when task τ_k executes on one processor, all $M-1$ processors idle.

5.1.2.3 Method 2: Redundant Self-Suspension Analysis

Our second approach is an improvement of the suspension-oblivious schedulability test, which is stated in Theorem 5.26. The underlying utilization-based test [LL73] is exact for periodic tasks without suspension, i.e., the test succeeds if and only if the task set is schedulable. The test would become incorrect if the implicit-deadline constraint would be dropped. The overall aim of this section is to provide a suspension-aware version of that utilization-based test.

The central observation is that the processor can be suspended for several jobs and additionally work on a lower-priority job at the same time. The suspension-oblivious approach is not aware of this fact and adds the time that the processor is suspended for a job as additional execution time to the workload. In the following, we estimate the number of jobs by which the processor is only suspended while it is also suspended for or working on lower-priority jobs, and exclude their suspension time from the workload. To make the estimations possible, we have to restrict to periodic task sets. We note that by omitting the implicit-deadline restriction our analysis would become incorrect, too.

We consider a task set $\mathbb{T} = \{\tau_1, \dots, \tau_n\}$ consisting of n periodic implicit-deadline tasks abstracted with dynamic self-suspension. That is, each task $\tau_i \in \mathbb{T}$ can be abstracted as

$$\tau_i \in \mathrm{PerOff}(T_{\tau_i}, \phi_{\tau_i}) \cap \mathrm{DynSS}(C_{\tau_i}, S_{\tau_i}). \tag{5.80}$$

We denote by $\mathcal{A}$ the preemptive EDF scheduling algorithm. In the following we assume that $\mathbb{T}$ is *not* schedulable, i.e., there exists a system evolution $\omega \in \Omega$ such that the resulting schedule $[\mathcal{S}] := \mathcal{A}(\omega)$ is *not* feasible.

Preparation Since we consider a periodic task system, the release pattern of the jobs is fixed, depending on the given phases. For a given schedule, we reorganize the task system and the schedule to fulfill the following conditions:

Remark 5.46 (Revision of the EDF schedule)**.**

(i) *The property $C_{\tau_1} + S_{\tau_1} \leq \cdots \leq C_{\tau_n} + S_{\tau_n}$ holds.* This is accomplished simply by re-indexing.

(ii) *For each task, no additional job at the beginning can be added without preserving the minimal phase, i.e., for $\phi := \min\{\phi_{\tau_i}\}$ the property $\phi_{\tau_i} - \phi < T_{\tau_i}$ holds for all i. More specifically, $\phi_{\tau_i} \in [\phi, \phi + T_{\tau_i}]$ for all i.* This is achieved by adding empty jobs without any execution or suspension time of τ_i periodically between ϕ_{τ_i} and ϕ until $\phi_{\tau_i} - \phi < T_{\tau_i}$ holds.

(iii) *All jobs have* full *execution and suspension time, i.e., the processor is working on and suspended for each job the amount of time given by the WCET and maximum suspension time.* This is achieved by the undermentioned procedure. Lemma 5.47 proves that this step does not transform any EDF schedule with deadline misses into an EDF schedule without deadline misses.

We note that the addition of empty jobs in step (ii) has no impact on the response times of the other jobs. After ensuring Property (ii), we denote the job set by $\mathbb{J}' \supseteq \mathbb{J}$. Furthermore, we denote the jobs of $\mathbb{J}'$ by $J_1, J_2, \ldots$, sorted by their priorities. Inductively, we modify the schedule again by using the following procedure for the m-th job J_m, $m = 1, 2, \ldots$, to obtain Property (iii):

- If the processor is now working on any of the jobs $J_1, \ldots, J_{m-1}$ at a time interval where it was before suspended for or working on J_m, we move this execution or suspension time of J_m to the end of that job.
- We add additional (pseudo) execution and suspension time at the end of J_m such that it equals the worst case.

Lemma 5.47. *By using the procedure described above, the response time of each job does not decrease.*

Proof. We use induction and prove that after the m-th step, $J_1, \ldots, J_m$ interfere at least the same time intervals as before and the response time of J_m is not decreased.

For the first job, we can just add additional suspension and execution time at the end. This cannot decrease the response time of that job and J_1 interferes at least the same time intervals as before, possibly even more.

A modification of the m-th job does not affect the response times of the higher-priority jobs $J_1, \ldots, J_{m-1}$. By induction, those higher-priority jobs interfere for at least the same time intervals as before. Therefore and because we only add execution and suspension time, the response time of J_m does not decrease. If the processor was before suspended for or working on J_m, then it is now either still in the same state or it is occupied by any of the jobs $J_1, \dots, J_{m-1}$. Hence, the processor is occupied by $J_1, \dots, J_m$ at least at the same time intervals as before, i.e., the interference by $J_1, \dots, J_m$ may only be increased. □

The resulting schedule and system evolution after the modifications in this section are denoted by $[\mathcal{S}']$ and ω', respectively. By Lemma 5.47, if the original EDF schedule $[\mathcal{S}]$ has a deadline miss for ω, the new EDF schedule $[\mathcal{S}']$ with its jobs in the setting described in Remark 5.46 also has a deadline miss for ω'.

Proof of the Schedulability Test We assume that the requirements from Remark 5.46 are met by the task set. Let $J^\dagger$ be the job with the highest priority which does not meet its deadline. Let this deadline be $d^{\omega'}_{J^\dagger}$. Furthermore, let t_{-1} be the last time before $d^{\omega'}_{J^\dagger}$ where the processor is not suspended, or working on $J^\dagger$ or jobs with higher priority. We delete all jobs which are finished before t_{-1} or have lower priority than $J^\dagger$. This does not affect the response times of the remaining jobs.

The remaining job set, system evolution, and schedule from time t_{-1} to $d^{\omega'}_{J^\dagger}$ are denoted as $\mathbb{J}''$, ω'', and $[\mathcal{S}'']$, respectively, for the remainder of this proof. We note that $J^\dagger \in \mathbb{J}''$ and $d^{\omega'}_{J^\dagger} = d^{\omega''}_{J^\dagger}$. According to the above definition, all jobs that are executed in the time interval $[t_{-1}, d^{\omega''}_{J^\dagger}]$ have release time $\geq t_{-1}$ and deadline $\leq d^{\omega''}_{J^\dagger}$. Hence, there are at most $\left\lfloor \frac{d^{\omega''}_{J^\dagger} - t_{-1}}{T_{\tau_i}} \right\rfloor$ jobs of task τ_i executed in the schedule $[\mathcal{S}'']$.

During the whole interval $[t_{-1}, d^{\omega''}_{J^\dagger}]$ the processor is either executing a job or idling, i.e., $d^{\omega''}_{J^\dagger} - t_{-1} = \text{exec}^{[\mathcal{S}'']}(t_{-1}, d^{\omega''}_{J^\dagger}) + \text{idle}^{[\mathcal{S}'']}(t_{-1}, d^{\omega''}_{J^\dagger})$. Since by definition, during the whole interval $[t_{-1}, d^{\omega''}_{J^\dagger}]$ jobs are either executed or suspended, each time the processor idles, a job suspends, i.e., $\text{idle}^{[\mathcal{S}'']}(t_{-1}, d^{\omega''}_{J^\dagger}) \leq \sum_{\tau_i \in \mathbb{T}} \left\lfloor \frac{d^{\omega''}_{J^\dagger} - t_{-1}}{T_{\tau_i}} \right\rfloor S_{\tau_i}$. Furthermore, the execution time can be bounded by $\text{exec}^{[\mathcal{S}'']}(t_{-1}, d^{\omega''}_{J^\dagger}) \leq \sum_{\tau_i \in \mathbb{T}} \left\lfloor \frac{d^{\omega''}_{J^\dagger} - t_{-1}}{T_{\tau_i}} \right\rfloor C_{\tau_i}$, and since a job misses its deadline, even $\text{exec}^{[\mathcal{S}'']}(t_{-1}, d^{\omega''}_{J^\dagger}) < \sum_{\tau_i \in \mathbb{T}} \left\lfloor \frac{d^{\omega''}_{J^\dagger} - t_{-1}}{T_{\tau_i}} \right\rfloor C_{\tau_i}$. This leads to:

$$\begin{aligned} d^{\omega''}_{J^\dagger} - t_{-1} &= \text{exec}^{[\mathcal{S}'']}(t_{-1}, d^{\omega''}_{J^\dagger}) + \text{idle}^{[\mathcal{S}'']}(t_{-1}, d^{\omega''}_{J^\dagger}) \\ &< \sum_{\tau_i \in \mathbb{T}} \left\lfloor \frac{d^{\omega''}_{J^\dagger} - t_{-1}}{T_{\tau_i}} \right\rfloor C_{\tau_i} + \sum_{\tau_i \in \mathbb{T}} \left\lfloor \frac{d^{\omega''}_{J^\dagger} - t_{-1}}{T_{\tau_i}} \right\rfloor S_{\tau_i} \end{aligned} \tag{5.81}$$

The condition $d^{\omega''}_{J^\dagger} - t_{-1} < \sum_{\tau_i \in \mathbb{T}} \left\lfloor \frac{d^{\omega''}_{J^\dagger} - t_{-1}}{T_{\tau_i}} \right\rfloor (C_{\tau_i} + S_{\tau_i})$ implies that $1 < \sum_{\tau_i \in \mathbb{T}} \frac{C_{\tau_i} + S_{\tau_i}}{T_{\tau_i}}$, which concludes the suspension-oblivious schedulability test in Theorem 5.26. However, instead of relying on $\text{idle}^{[\mathcal{S}'']}(t_{-1}, d^{\omega''}_{J^\dagger}) \leq \sum_{\tau_i \in \mathbb{T}} \left\lfloor \frac{d^{\omega''}_{J^\dagger} - t_{-1}}{T_{\tau_i}} \right\rfloor S_{\tau_i}$, our utilization-based analysis applies a tighter bound for $\text{idle}^{[\mathcal{S}'']}(t_{-1}, d^{\omega''}_{J^\dagger})$, based on the following observation:

Observation 5.48. *Suppose that there are two jobs J and $\tilde{J}$ in schedule $[\mathcal{S}'']$. If*

$$[r_J^{\omega''}, d_J^{\omega''}] \subseteq [r_{\tilde{J}}^{\omega''}, f_{\tilde{J}}^{[\mathcal{S}''],\omega''}], \tag{5.82}$$

then the suspension time of job J only contributes to the idle time in the schedule $[\mathcal{S}'']$ when $\tilde{J}$ suspends at the same time. In this case, the suspension of the processor due to J is already covered by the suspension of $\tilde{J}_j$ and does not have to be considered in the bound for $\text{idle}^{[\mathcal{S}'']}(t_{-1}, d_{J^\dagger}^{\omega''})$*.*

Let ℓ be the highest index among those tasks which do have remaining jobs executed in $[\mathcal{S}'']$. Let t_r^ℓ and t_d^ℓ be the first release and last deadline of the remaining jobs of τ_ℓ in $[\mathcal{S}'']$, respectively. If $\ell = 1$, then there is only one task τ_ℓ in the remaining schedule in the time interval $[t_{-1}, d_{J^\dagger}^{\omega''}]$ and the deadline miss of τ_ℓ implies that $C_{\tau_\ell} + S_{\tau_\ell} > D_{\tau_\ell} = T_{\tau_\ell}$, which violates our assumption that $C_{\tau_\ell} + S_{\tau_\ell} \leq D_{\tau_\ell}$. We therefore only have to consider the scenario where $\ell \geq 2$.

For each of the $\frac{t_d^\ell - t_r^\ell}{T_{\tau_\ell}}$ jobs, we utilize it as $\tilde{J}$ in Observation 5.48 and then quantify the jobs J that satisfy the condition $[r_J^{\omega''}, d_J^{\omega''}] \subseteq [r_{\tilde{J}}^{\omega''}, f_{\tilde{J}}^{[\mathcal{S}''],\omega''}]$.

Lemma 5.49. *Suppose that job $\tilde{J}$ is a job of τ_ℓ in the schedule $[\mathcal{S}'']$ and $C_{\tau_\ell} + S_{\tau_\ell} \geq T_{\tau_i}$ for a certain task τ_i. Then, at least $\left\lfloor \frac{C_{\tau_\ell} + S_{\tau_\ell}}{T_{\tau_i}} \right\rfloor - 1$ different jobs of task τ_i are jobs J that satisfy Equation 5.82 from Observation 5.48.*

Proof. Since the execution time plus the suspension time of job $\tilde{J}$ is $C_{\tau_\ell} + S_{\tau_\ell}$ in the schedule $[\mathcal{S}'']$ due to Property (iii) in Remark 5.46, the response time of job $J^\dagger$ is at least $C_{\tau_\ell} + S_{\tau_\ell}$. Moreover, since task τ_i is periodically released with Property (ii) in Remark 5.46, the first job release of task τ_i after $r_{J^\dagger}^{\omega''}$ must be earlier than $r_{J^\dagger}^{\omega''} + T_{\tau_i}$. Therefore, there are at least $\left\lfloor \frac{r_{J^\dagger}^{\omega''} + C_{\tau_\ell} + S_{\tau_\ell} - (r_{J^\dagger}^{\omega''} + T_{\tau_i})}{T_{\tau_i}} \right\rfloor = \left\lfloor \frac{C_{\tau_\ell} + S_{\tau_\ell}}{T_{\tau_i}} \right\rfloor - 1$ jobs of task τ_i with release time $\geq r_{J^\dagger}^{\omega''}$ and deadline $\leq r_{J^\dagger}^{\omega''} + C_{\tau_\ell} + S_{\tau_\ell}$. □

We already calculated the number of jobs of τ_ℓ in the schedule $[\mathcal{S}'']$ by $\frac{t_d^\ell - t_r^\ell}{T_{\tau_\ell}}$. However, to perform a similar proof as for the suspension-oblivious test in Equation (5.81), the term $(t_d^\ell - t_r^\ell)$ is impractical, and we need to relate that term to $(d_{J^\dagger}^{\omega''} - t_{-1})$ instead. Hence, we estimate $(t_d^\ell - t_r^\ell)$ with respect to $(d_{J^\dagger}^{\omega''} - t_{-1})$.

Lemma 5.50. *The ratio of $t_d^\ell - t_r^\ell$ to $d_{J^\dagger}^{\omega''} - t_{-1}$ is at least $\frac{1}{3}$.*

Proof. Since the other jobs are deleted, we have $[t_r^\ell, t_d^\ell] \subseteq [t_{-1}, d_{J^\dagger}^{\omega''}]$. If the remaining part on the right-hand side of the interval would be more than T_{τ_ℓ}, i.e., $d_{J^\dagger}^{\omega''} - t_d^\ell > T_{\tau_\ell}$, then there would be another job of τ_ℓ released at t_d^ℓ with higher priority than J, which does not exist by the definition of t_d^ℓ. Furthermore, if $t_r^\ell - t_{-1} > T_{\tau_\ell}$, then there would be enough space for another job with deadline at t_r^ℓ and release time after t_{-1}. By Property (ii) in Remark 5.46, the phase of τ_ℓ is smaller than t_r^ℓ and the additional job in fact exists. This contradicts the definition of t_r^ℓ. We conclude that $[t_r^\ell, t_d^\ell] \subseteq [t_{-1}, d_{J^\dagger}^{\omega''}] \subseteq [t_r^\ell - T_{\tau_\ell}, t_d^\ell + T_{\tau_\ell}]$.

Let h be the number of jobs of τ_ℓ in $[\mathcal{S}'']$ defined by the quotient $\frac{t_d^\ell - t_r^\ell}{T_{\tau_\ell}}$, then we obtain $\frac{t_d^\ell - t_r^\ell}{d_{J\dagger}^{\omega''} - t_{-1}} \geq \frac{h \cdot T_{\tau_\ell}}{h \cdot T_{\tau_\ell} + 2 \cdot T_{\tau_\ell}} = \frac{h}{h+2}$. Since $h \in \mathbb{Z}_{\geq 1}$ and $\frac{h}{h+2} \leq \frac{(h+1)}{(h+1)+2}$ for all $h \in \mathbb{Z}_{\geq 1}$, we get $\frac{h}{h+2} \geq \frac{1}{3}$ and $\frac{t_d^\ell - t_r^\ell}{d_{J\dagger}^{\omega''} - t_{-1}} \geq \frac{1}{3}$. □

Now, we can conclude the schedulability test.

Theorem 5.51. *Let $\mathbb{T} = \{\tau_1, \ldots, \tau_n\}$ be a set of implicit-deadline periodic tasks with dynamic self-suspension and $C_{\tau_1} + S_{\tau_1} \leq \cdots \leq C_{\tau_n} + S_{\tau_n}$. For notational brevity, let δ_i^ℓ denote the indication function which is 1 if $C_{\tau_\ell} + S_{\tau_\ell} \geq T_{\tau_i}$ and 0 otherwise. If for all $\ell \in \{1, \ldots, n\}$ the property*

$$\frac{C_{\tau_\ell} + S_{\tau_\ell}}{T_{\tau_\ell}} + \sum_{i=1}^{\ell-1} \frac{C_{\tau_i} + S_{\tau_i}\left(1 - \frac{1}{3}\frac{T_{\tau_i}}{T_{\tau_\ell}}\left(\left\lfloor \frac{C_{\tau_\ell}+S_{\tau_\ell}}{T_{\tau_i}} \right\rfloor - 1\right) \cdot \delta_i^\ell\right)}{T_{\tau_i}} \leq 1 \tag{5.83}$$

holds, then the task set is schedulable by preemptive EDF. We set $\sum_{i=1}^{\ell-1} X_i = 0$ when ℓ is 1 for notational brevity.

Proof. We prove this theorem by contraposition. Suppose that there is a job missing its deadline, and we construct the remaining schedule $[\mathcal{S}'']$ in time interval $[t_{-1}, d_{J\dagger}^{\omega''}]$ as defined above, in which a job misses its deadline in the schedule $[\mathcal{S}'']$. Let ℓ be the highest index among those tasks with at least one job in schedule $[\mathcal{S}'']$. We define $C'_i \coloneqq C_{\tau_i} + S_{\tau_i}$ for all i.

We first focus on the scenario when $\ell \geq 2$. Let h be the number of jobs of task τ_ℓ in the time interval $[t_{-1}, d_{J\dagger}^{\omega''}]$, i.e., h is $\frac{t_d^\ell - t_r^\ell}{T_{\tau_\ell}}$. By Lemma 5.50 and the definition of periodic tasks, we have $\frac{d_{J\dagger}^{\omega''} - t_{-1}}{3T_{\tau_\ell}} \leq h \leq \frac{d_{J\dagger}^{\omega''} - t_{-1}}{T_{\tau_\ell}}$.

We can use Lemma 5.49 and Observation 5.48 to reduce the contribution of the suspension time to $\text{idle}^{[\mathcal{S}'']}(t_{-1}, d_{J\dagger}^{\omega''})$ from a certain task τ_i for $i = 1, 2, \ldots, l-1$. Together with an argument similar to that resulting in the condition in Equation (5.81), we have

$$d_{J\dagger}^{\omega''} - t_{-1} < hC'_\ell + \sum_{i=1}^{\ell-1}\left(\left\lceil \frac{d_{J\dagger}^{\omega''} - t_{-1}}{T_{\tau_i}} \right\rceil C'_i - h\left(\left\lfloor \frac{C'_\ell}{T_{\tau_i}} \right\rfloor - 1\right)\delta_i^\ell S_{\tau_i}\right) \tag{5.84}$$

$$\leq \frac{d_{J\dagger}^{\omega''} - t_{-1}}{T_{\tau_\ell}} C'_\ell + \sum_{i=1}^{\ell-1}\left(\frac{d_{J\dagger}^{\omega''} - t_{-1}}{T_{\tau_i}} C'_i - \frac{d_{J\dagger}^{\omega''} - t_{-1}}{3T_{\tau_\ell}}\left(\left\lfloor \frac{C'_\ell}{T_{\tau_i}} \right\rfloor - 1\right)\delta_i^\ell S_{\tau_i}\right) =: A. \tag{5.85}$$

The second inequality is due to $\frac{d_{J\dagger}^{\omega''} - t_{-1}}{3T_{\tau_\ell}} \leq h \leq \frac{d_{J\dagger}^{\omega''} - t_{-1}}{T_{\tau_\ell}}$.

Moreover, when ℓ is 1, the schedule $[\mathcal{S}'']$ has only jobs from τ_1 in $[t_{-1}, d_{J\dagger}^{\omega''}]$. That means $d_{J\dagger}^{\omega''} - t_{-1} < h \cdot C'_\ell \leq \frac{d_{J\dagger}^{\omega''} - t_{-1}}{T_{\tau_\ell}} \cdot C'_\ell$, i.e., the condition $d_{J\dagger}^{\omega''} - t_{-1} < A$ holds also when $\ell = 1$.

Therefore, by dividing both sides of $d^{\omega''}_{J^\dagger} - t_{-1} < A$ with $d^{\omega''}_{J^\dagger} - t_{-1}$, which is > 0, the deadline miss of a job in the schedule $[\mathcal{S}'']$ implies the existence of $\ell = 1, 2, \ldots, n$ with

$$1 < \frac{C'_\ell}{T_{\tau_\ell}} + \sum_{i=1}^{\ell-1} \frac{C_{\tau_i}}{T_{\tau_i}} + \frac{S_{\tau_i}\left(1 - \frac{1}{3}\frac{T_{\tau_i}}{T_{\tau_\ell}}\left(\left\lfloor \frac{C'_\ell}{T_{\tau_i}} \right\rfloor - 1\right) \cdot \delta_i^\ell\right)}{T_{\tau_i}}. \tag{5.86}$$

As a result, the negation of the above condition, i.e., for all $\ell = 1, 2, \ldots, n$ with the condition in (5.83), implies the schedulability of the task set under preemptive EDF. □

THEORETICAL EVALUATION We provide some observations about the theoretical behavior of the schedulability test provided in this section in comparison to the prior methods detailed in Section 5.1.2.1 and our response-time analysis in Section 5.1.2.2.

Comparison with Suspension-Oblivious (Theorem 5.26): The redundant self-suspension schedulability test proposed in Theorem 5.51 dominates the existing utilization-based suspension-oblivious analysis applied to periodic task sets, since the left-hand side of Equation (5.83) is at most the total utilization if we consider suspension time as additional execution time. Our method performs better if the sum of execution and suspension time of one task is large compared to the period of the other tasks.

Example 5.52. *Consider the implicit-deadline task set* $\mathbb{T} = \{\tau_1, \tau_2\}$ *with:*

- $\tau_1 \in \mathrm{Per}(T_{\tau_1} = 1) \cap \mathrm{DynSS}(C_{\tau_1} = \frac{1}{17}, S_{\tau_1} = \frac{1}{3})$
- $\tau_2 \in \mathrm{Per}(T_{\tau_2} = 21) \cap \mathrm{DynSS}(C_{\tau_2} = 14, S_{\tau_2} = 0)$

Suspension-oblivious analysis concludes that the task set is not schedulable, since the total utilization is > 1. *Our test however indicates schedulability: At first we observe that for* $\ell = 1$, *we have* $\frac{C_{\tau_1}+S_{\tau_1}}{T_{\tau_1}} = \frac{\frac{1}{17}+\frac{1}{3}}{1} = \frac{20}{51} \leq 1$. *For* $\ell = 2$, *we compute* $\frac{1}{3}\frac{T_{\tau_1}}{T_{\tau_2}}\left(\left\lfloor \frac{C_{\tau_2}+S_{\tau_2}}{T_{\tau_1}} \right\rfloor - 1\right) = \frac{1}{3}\frac{1}{21} \cdot 13 = \frac{13}{63}$. *The test in Equation (5.83) yields* $\frac{14}{21} + \frac{\frac{1}{17}+\frac{1}{3}(1-\frac{13}{63})}{1} \leq 1$. *Hence, the task set is schedulable.*

Comparison with Response Time Analysis in Section 5.1.2.2: Both schedulability tests proposed in this paper do not dominate each other. To prove this, we again use Example 5.43 and Example 5.44 with periodic task sets. Since neither $C_{\tau_1} + S_{\tau_1} \geq T_{\tau_2}$ nor $C_{\tau_2} + S_{\tau_2} \geq T_{\tau_1}$, our utilization-based approach coincides with the suspension-oblivious schedulability test, which is superior in Example 5.44 and inferior in Example 5.43.

Comparison with the Workload-Based Schedulability Test (Theorem 5.27) by Liu and Anderson: To show that the schedulability test by Liu and Anderson [LA13] and the one described in this section also do not dominate each other, we use similar examples with periodic task sets.

The method by Liu and Anderson is superior for Example 5.43. For $\ell = 1$, only $s_{\ell,j} = 2$ and $\xi_\ell = 5$ is possible. By computation, we obtain $W_c(\tau_1) = 0, W_{nc}(\tau_1) = 0, W_c(\tau_2) = 1$ and $W_{nc}(\tau_2) = 0$. Since $1 \leq 5 - 1 - 2 = 2$, Equation (5.50) holds. For $\ell = 2$, no choice of $s_{\ell,j}$ and ξ_ℓ is possible. Hence, the test says that the task set is schedulable. Since

$C_{\tau_2} + S_{\tau_2} = 4 < 5 = T_{\tau_1}$, our schedulability test is the same as the suspension-oblivious approach. Therefore, our test cannot conclude schedulability.

For a task set that is similar to the one presented in Example 5.44, our method is superior to the one by Liu and Anderson. Specifically, consider the implicit-deadline task set $\mathbb{T} = \{\tau_1, \tau_2\}$ with

- $\tau_1 \in \mathrm{Per}(T_{\tau_1} = 6) \cap \mathrm{DynSS}(C_{\tau_1} = 3 - \varepsilon, S_{\tau_1} = \varepsilon)$
- $\tau_2 \in \mathrm{Per}(T_{\tau_2} = 20) \cap \mathrm{DynSS}(C_{\tau_2} = 10 - \varepsilon, S_{\tau_2} = \varepsilon)$

such that $\varepsilon \in (0, \frac{1}{3})$. The periodic task set is not deemed schedulable by our response-time-based analysis in Section 5.1.2.2. Since it dominates the schedulability test by Liu and Anderson [LA13] if all tasks have suspension, as discussed in the theoretical evaluation in Section 5.1.2.2, their test also fails for this example. We have $\widetilde{R}_2(0) = 10 + \left(\left\lfloor \frac{20}{6} \right\rfloor + 1\right) \cdot (3 - \varepsilon)$ and $\widetilde{R}_2(1) = 10 + \left\lfloor \frac{20-2}{6} \right\rfloor \cdot (3 - \varepsilon) + 2$. Hence, the value for $\widetilde{R}_2$ is $10 + 3 \cdot (3 - \varepsilon) + 2 = 21 - 3\varepsilon$ which is bigger than 20 for $\varepsilon < \frac{1}{3}$. Our response-time-based analysis in Section 5.1.2.2 as well as the approach by Liu and Anderson [LA13] both fail.

On the other hand, we know that the example is schedulable by the suspension-oblivious approach since the value for $\sum_i \frac{C_{\tau_i} + S_{\tau_i}}{T_{\tau_i}}$ does not change compared to Example 5.44. Since our redundant self-suspension schedulability test dominates the suspension-oblivious test, it also implies schedulability.

5.1.2.4 Experimental Evaluation

We evaluate synthesized task sets to compare the proposed methods with the approaches from the literature. The metric to compare is the *acceptance ratio* with respect to the task set utilization, i.e., the percentage of task sets that has been accepted for the specific utilization value. We generate 1 000 task sets for each of the analyzed utilization levels in the range of 0 % to 100 % with steps of 1 %.

For each task set, we first generate a set of implicit-deadline tasks. We adopt the UUniFast method [BB05] to generate a set of utilization values U_{τ_i} with the given goal U_{sum}, and apply the suggestion by Emberson et al. [ESD10] to generate the periods of the tasks according to a log-uniform distribution with two orders of magnitude. To be precise, $\log_{10} T_{\tau_i}$ is uniformly distributed, and the resulting T_{τ_i} values are in $[1\text{ms}, 100\text{ms}]$. Accordingly, the execution time is set to $C_{\tau_i} = T_{\tau_i} \cdot U_{\tau_i}$ and the relative deadline is set to the task periods, i.e., $D_{\tau_i} = T_{\tau_i}$. We convert them to dynamic self-suspending tasks by generating the suspension lengths of each task according to a uniform distribution in either of three ranges, i.e., the targeted self-suspension length:

- Short suspension: $[0.0(T_{\tau_i} - C_{\tau_i}), 0.1(T_{\tau_i} - C_{\tau_i})]$
- Moderate suspension: $[0.1(T_{\tau_i} - C_{\tau_i}), 0.3(T_{\tau_i} - C_{\tau_i})]$
- Long suspension: $[0.3(T_{\tau_i} - C_{\tau_i}), 0.6(T_{\tau_i} - C_{\tau_i})]$

Furthermore, we consider self-suspension that is log-uniformly distributed in the interval $[0.0001(T_{\tau_i} - C_{\tau_i}), 0.1(T_{\tau_i} - C_{\tau_i})]$. We compare the following methods:

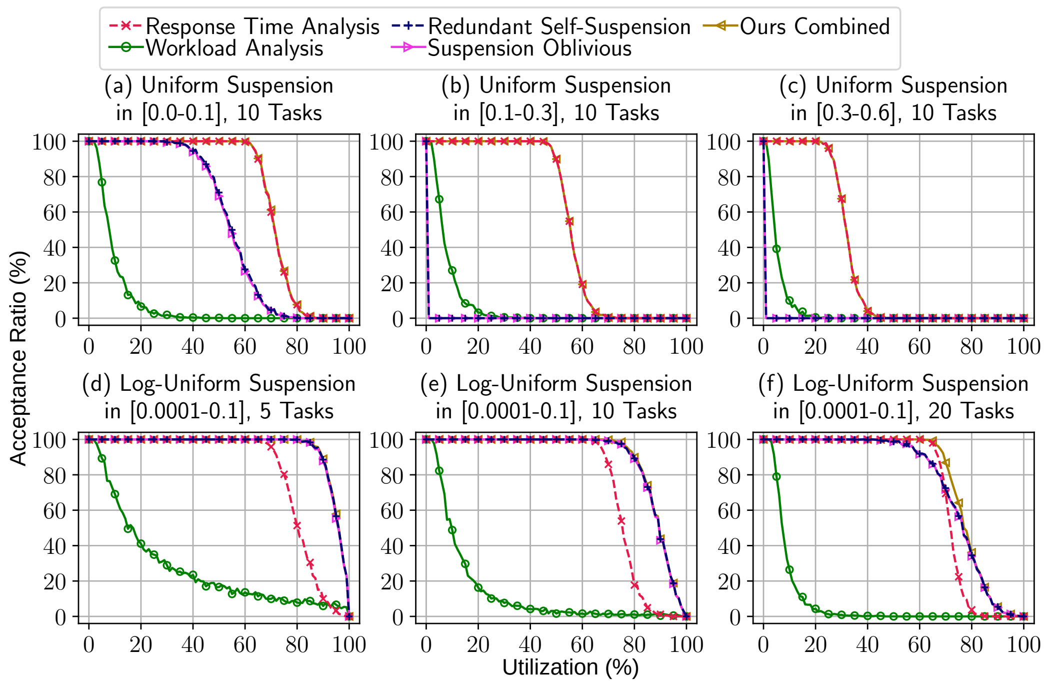


Figure 5.11.: Comparison of our schedulability tests with the state-of-the-art.

- Our **Response Time Analysis** detailed in Section 5.1.2.2.
- Our utilization-based analysis that examines **Redundant Self-Suspension** (Section 5.1.2.3).
- Combination of the above two methods, denoted by **Ours Combined**, i.e., if one of them returns *schedulable*, this test succeeds.
- The **Suspension Oblivious** approach, see Section 5.1.2.1.
- The **Workload Analysis** by Liu and Anderson [LA13], see Section 5.1.2.1.
- The **Utilization-Based Method** by Dong and Liu [DL16], see Section 5.1.2.1.

The implementation of our analyses is integrated in an evaluation framework on Github [TU 21a, Methods RTEDF and RSS].

The results with uniformly distributed self-suspension with 10 tasks per task set are shown in Figure 5.11 (a)–(c), and the results with log-uniformly distributed self-suspension are shown in Figure 5.11 (d)–(f) with 5, 10 and 20 tasks per task set. The **Utilization-Based Method** is omitted, since it provided the same results as the **Suspension Oblivious** approach. While for comparison with the **Redundant Self-Suspension** method we assume that all tasks are periodic, comparison between the other methods is based even on the sporadic counterparts.

Utilization range [%]	(d)	(e)	(f)	(d)’	(e)’	(f)’
1 – 10	0.00	0.00	0.00	0.00	0.00	0.00
11 – 20	0.00	0.00	0.00	0.00	0.00	0.00
21 – 30	0.00	0.00	0.01	0.00	0.00	0.00
31 – 40	0.00	0.00	0.02	0.00	0.00	0.02
41 – 50	0.00	0.00	0.02	0.00	0.00	0.11
51 – 60	0.00	0.00	0.16	0.00	0.01	0.53
61 – 70	0.00	0.02	0.39	0.00	0.08	1.26
71 – 80	0.00	0.19	0.52	0.01	0.74	1.37
81 – 90	0.39	0.79	0.25	0.69	1.89	0.69
91 – 100	0.90	0.31	0.02	1.44	0.79	0.03

Table 5.1.: Average acceptance ratio gain in percent of **Redundant Self-Suspension** compared to **Suspension Oblivious**. (d)–(f) are experiments from Figure 5.11 and (d)’–(f)’ are the same experiments with periods pulled from $[1, 10\,000]$.

The **Response Time Analysis** clearly outperforms the state-of-the-art for all cases with uniform distributed self-suspension length as depicted in Figure 5.11 (a)–(c). As expected, the acceptance ratio is reduced for all schedulability test if the suspension interval gets longer. The **Workload Analysis** by Liu and Anderson [LA13] always drops very quickly, but, contrary to the utilization-based analyses, still accepts some task sets for longer suspension intervals. The **Redundant Self-Suspension** and the **Suspension Oblivious** approach only perform reasonably good for short suspension as in Figure 5.11 (a).

Redundant Self-Suspension and **Suspension Oblivious** show enhanced performance for short log-uniformly distributed self-suspension as in Figure 5.11 (d)–(f). Interestingly, we can see that those results depend on the number of tasks. The fewer tasks are in the task set, the better performs **Redundant Self-Suspension** compared to **Response Time Analysis**, i.e., in Figure 5.11 (d) and Figure 5.11 (e) the former provides better results while in Figure 5.11 (f) it depends on the utilization.

The result of **Ours-Combined** shows another aspect of the relation of our proposed methods. For Figure 5.11 (a)–(e), its curve lies almost directly on the individually better performing curve, i.e., the largest gap is 1.3 %. On the other hand, for Figure 5.11 (f), we observe an additional benefit for combining both tests by an increase in acceptance ratio of up to 14.6 %.

As proven in Section 5.1.2.3, the **Redundant Self-Suspension** method dominates the **Suspension Oblivious** approach. Table 5.1 summarizes the acceptance ratio gain, where the gain of acceptance ratio of A compared to B is computed by subtracting the acceptance ratio of B from the acceptance ratio of A in different utilization ranges of experiments (d)–(f). We further examine how this improvement changes if the period range is extended from $[1, 100]$ to $[1, 10\,000]$, indicated by (d)’–(f)’. In almost all cases, the average acceptance ratio gain increases. We note that the extended periods do not significantly impact the performance of the schedulability test compared to each other which is why the plots for (d)'–(f)' are omitted. As depicted in the table, an average acceptance ratio gain of up to 1.89 percent is observed.

5.1.3 Under EDF-Like Scheduling

Current suspension-aware analyses are limited to T-FP scheduling or EDF scheduling. In this section, we consider the *window-constrained* scheduler EDF-Like (EL). The category of *window-constrained* schedulers, where at each time job priorities are assigned according to a *priority point*, has originally been proposed by Leontyev and Anderson [LA07] to provide general tardiness bounds for multiprocessor scheduling. Popular T-DP algorithms, such as EDF, First In – First Out (FIFO), and Earliest-Quasi-Deadline-First (EQDF) [BCS12] fall into this category.

In the following, we provide a unifying schedulability analysis for uniprocessor EL schedulers of arbitrary-deadline task sets. Our analysis is the first suspension-aware schedulability analysis for arbitrary-deadline sporadic real-time task systems under T-DP scheduling, such as EDF, and it is the first unifying suspension-aware schedulability analysis framework that covers a wide range of scheduling algorithms. Our contributions are as follows:

- In Section 5.1.3.1, we discuss the capabilities and limitations of EL scheduling algorithms and demonstrate how they can be configured to behave as EDF, FIFO, EQDF, Suspension-Aware EDF (SAEDF), T-FP algorithms, and hierarchical scheduling.
- In Section 5.1.3.2, we introduce a unifying schedulability test for uniprocessor EL scheduling algorithms, that is applicable to self-suspending arbitrary-deadline task systems. To the best of our knowledge, this is the first result that can handle arbitrary-deadline task sets under T-DP scheduling and cover a wide range of scheduling algorithms in one analysis framework for self-suspending task systems.
- Our numerical evaluation in Section 5.1.3.3 shows that our schedulability test outperforms the state of the art for EDF and performs slightly worse than the test by Chen et al. [CNH16] for DM scheduling under constrained-deadlines. We also examine the performance of different configurations for EQDF and SAEDF.

The results presented in this section appeared in RTSS 2022 [Gün+22].

5.1.3.1 Capabilities and Limitations of EDF-Like Scheduling

In EDF-Like (EL) scheduling, as introduced in Section 2.4.3, the priority of each job $\tau(j)$ is based on a job-specific *priority point* $\pi^\omega_{\tau(j)} \in \mathbb{R}$. More specifically, a job $\tau(j)$ has higher priority than $\tau'(j')$ in system evolution ω if $\pi^\omega_{\tau(j)} < \pi^\omega_{\tau'(j')}$. The priority point is induced by the release of the job and a task-specific parameter Π_τ denoted as *relative priority point*, i.e., $\pi^\omega_{\tau(j)} = r^\omega_{\tau(j)} + \Pi_\tau$. As a result, smaller Π_τ in comparison to the other relative

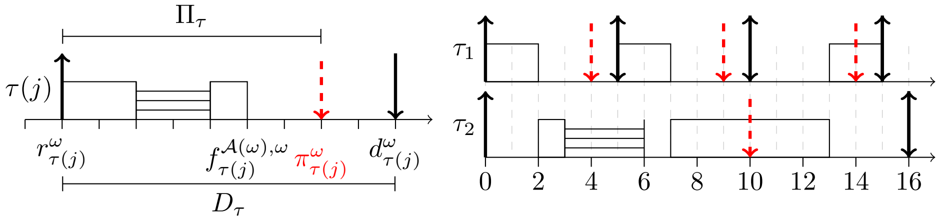


Figure 5.12.: Left: Presentation of our Notation. Right: Jobs of two tasks scheduled by EL scheduling. The schedule is feasible, and the job priority is given by $\pi^{\omega}_{\tau_1(1)} < \pi^{\omega}_{\tau_1(2)} < \pi^{\omega}_{\tau_2(1)} < \pi^{\omega}_{\tau_1(3)}$.

$$\text{T-FP} \quad \subseteq \quad \textbf{EL} \quad \subseteq \quad \text{J-FP}$$

Figure 5.13.: Expressiveness of the scheduling policies Task-level Fixed-Priority (T-FP), EDF-Like (EL), and Job-level Fixed-Priority (J-FP).

priority points favor the jobs of τ to be scheduled first. The left-hand side of Figure 5.12 depicts the notation for EL scheduling.

Under an assignment of relative priority points $(\Pi_\tau)_{\tau \in \mathbb{T}}$, whenever a job is added to the ready queue, the new highest-priority job is determined and executed. Whenever a job finishes or suspends itself, it is removed from the ready queue and a new highest-priority job is determined and executed.

Example 5.53. *The right-hand side of Figure 5.12 shows a schedule for two implicit-deadline tasks*

- $\tau_1 \in \text{PerOff}(5, 0) \cap \text{DynSS}(2, 0)$
- $\tau_2 \in \text{PerOff}(16, 0) \cap \text{DynSS}(7, 3)$

under EL scheduling with relative priority points $\Pi_{\tau_1} = 4$ *and* $\Pi_{\tau_2} = 10$.

Since the class of EL scheduling algorithms is considered sparsely in the literature, we first discuss how they relate to T-FP and Job-level Fixed-Priority (J-FP). This relation is depicted in Figure 5.13. Since all jobs are assigned priorities based on a fixed priority point, EL scheduling algorithms are by design a subclass of J-FP algorithms, i.e., if one job has higher priority than another job at some point in time, then it has higher priority at all times.[6] While the EL scheduling algorithms are a subset of J-FP, it contains many frequently used J-FP algorithms. Specifically, for a task set $\mathbb{T}$ with tasks $\tau \in \text{DynSS}(C_\tau, S_\tau)$ for all $\tau \in \mathbb{T}$, we model the following scheduling algorithms:

- Earliest-Deadline-First (EDF) [LL73] ($\Pi_\tau = D_\tau$)

[6]Please note that Leontyev and Anderson [LA07] define priority points as well. However, they obtain priority points by prioritization functions, thus their model can express all J-FP and Job-level Dynamic-Priority (J-DP) scheduling algorithms.

- Earliest-Quasi-Deadline-First (EQDF) [BCS12] ($\Pi_\tau = D_\tau + \lambda C_\tau$ for some predefined $\lambda \in \mathbb{R}$)
- Suspension-Aware EDF (SAEDF)[7] ($\Pi_\tau = D_\tau + \lambda S_\tau$ for some predefined $\lambda \in \mathbb{R}$)
- First In – First Out (FIFO) ($\Pi_\tau = 0$)

As a result, the schedulability test that we present in Section 5.1.3.2 is applicable to all these scheduling algorithms by configuring the relative priority points Π_{τ_i} accordingly.

Furthermore, T-FP algorithms can be treated as EL scheduling algorithms in the analysis as well. More specifically, each T-FP algorithm can be transferred into a related EL scheduling algorithm with the same behavior. Hence, if a task set is determined to be schedulable under this EL scheduling algorithm, it is schedulable under the T-FP algorithm as well. For the construction, we consider a task set $\mathbb{T} = \{\tau_1, \dots, \tau_n\}$, where the task indices indicate their priority, i.e., τ_k has higher priority than $\tau_{k'}$ if and only if $k < k'$ (with τ_1 having the highest priority). We assume that a WCRT upper bound K_j for each task either for the schedule under EL scheduling or under T-FP scheduling is given. If we set the relative priority point of each task τ_i to $\Pi_{\tau_i} = \sum_{j=1}^{i} K_j$, then EL and T-FP coincide. An EL schedulability test can also be utilized when the relative deadline D_{τ_i} is taken as an upper bound on the WCRT. In both cases, if a task set is schedulable according to an EL schedulability test, it is also schedulable under the given T-FP assignment.

Proposition 5.54 (T-FP as EL.). *Consider $\mathbb{T} = \{\tau_1, \dots, \tau_n\}$ and the T-FP scheduling algorithm with priorities indicated by task indexes. Let $K_1, \dots, K_n \in \mathbb{R}_{\geq 0}$ and*

$$\Pi_{\tau_i} := \sum_{j=1}^{i} K_j \tag{5.87}$$

for $i = 1, \dots, n$. If for all $j = 1, \dots, n$ the value K_j is an upper bound on the WCRT of τ_j in EL or in T-FP, then the schedule of $\mathbb{T}$ under EL and the schedule of $\mathbb{T}$ under T-FP coincide.

Proof. For an indirect proof we assume that there exists a system evolution ω of the job set $\mathbb{J}$ obtained by $\mathbb{T}$, such that the corresponding schedules $[\mathcal{S}_{EL}]$ under EL and $[\mathcal{S}_{FP}]$ under T-FP do not coincide. Let $\tau_k(\ell)$ be the job with the highest priority in the EL schedule, such that the schedules of $\tau_k(\ell)$ do not coincide, i.e., $\mathcal{S}_{EL}^{-1}(\{\tau_k(\ell)\})$ and $\mathcal{S}_{FP}^{-1}(\{\tau_k(\ell)\})$ do not coincide almost everywhere. We define the interval

$$I := [r^\omega_{\tau_k(\ell)}, r^\omega_{\tau_k(\ell)} + K_k). \tag{5.88}$$

Let $\mathbb{J}_{FP} \subseteq \mathbb{J}$ and $\mathbb{J}_{EL} \subseteq \mathbb{J}$ be the set of jobs with higher priority than $\tau_k(\ell)$ under T-FP and under EL, respectively, that are executed during I. We will reach a contradiction

[7]We note that Chen [Che16] also defined a suspension-aware version of EDF in 2016 by giving the job with the lowest absolute deadline minus *remaining* suspension the highest priority. However, that version is not compatible with EL scheduling since it is not a J-FP algorithm.

by showing that $\mathbb{J}_{FP} = \mathbb{J}_{EL}$ and that further the schedule of the jobs in $\mathbb{J}_{FP} = \mathbb{J}_{FP}$ coincides under T-FP and under EL scheduling. For that purpose we partition the job sets into three subsets $\mathbb{J}_{FP} = \coprod_{i\in\{+,-,0\}} \mathbb{J}^i_{FP}$ and $\mathbb{J}_{EL} = \coprod_{i\in\{+,-,0\}} \mathbb{J}^i_{EL}$, where each of them denotes the subset of jobs released by tasks of $\mathbb{T}_+ = \{\tau_{k+1}, \dots, \tau_n\}$, $\mathbb{T}_- = \{\tau_1, \dots, \tau_{k-1}\}$ or $\mathbb{T}_0 = \{\tau_k\}$. In the following we show that $\mathbb{J}^i_{FP} = \mathbb{J}^i_{EL}$ for all $i \in \{+,-,0\}$.

Proof of $\mathbb{J}^+_{FP} = \mathbb{J}^+_{EL}$: Because of the task-level priority order under T-FP, all jobs of the tasks of $\mathbb{T}_+$ have a lower priority than $\tau_k(\ell)$, i.e., $\mathbb{J}^+_{FP} = \emptyset$. Under EL, we choose any job $\tau_i(j)$, with $\tau_i \in \mathbb{T}_+$, that has higher priority than $\tau_k(\ell)$, i.e., $\pi^\omega_{\tau_i(j)} = r^\omega_{\tau_i(j)} + \Pi_{\tau_i} \leq \pi^\omega_{\tau_k(\ell)} = r^\omega_{\tau_k(\ell)} + \Pi_{\tau_k}$ holds. By subtracting Π_{τ_k} we obtain

$$r^\omega_{\tau_i(j)} + K_i \leq r^\omega_{\tau_i(j)} + (\Pi_{\tau_i} - \Pi_{\tau_k}) \leq r^\omega_{\tau_k(\ell)}. \tag{5.89}$$

Since $\tau_i(j)$ has higher priority than $\tau_k(\ell)$, by assumption, the schedule of $\tau_i(j)$ coincides under EL and T-FP, and we have $f^{[\mathcal{S}_{FP}],\omega}_{\tau_i(j)} = f^{[\mathcal{S}_{EL}],\omega}_{\tau_i(j)} \leq r^\omega_{\tau_i(j)} + K_i$. With Equation (5.89), we conclude $f^{[\mathcal{S}_{FP}],\omega}_{\tau_i(j)} = f^{[\mathcal{S}_{EL}],\omega}_{\tau_i(j)} \leq r^\omega_{\tau_k(\ell)}$. In particular, the job $\tau_i(j)$ is not executed during I. Since $\tau_i(j)$ was chosen arbitrarily, this means that $\mathbb{J}^+_{EL} = \emptyset$ as well.

Proof of $\mathbb{J}^-_{FP} = \mathbb{J}^-_{EL}$: Under T-FP and under EL, jobs of the tasks in $\mathbb{T}_-$ can only be executed during I if they are released before $r^\omega_{\tau_k(\ell)} + K_k$, i.e., let $\mathbb{J}'^-$ be the set of jobs released before $r^\omega_{\tau_k(\ell)} + K_k$ by tasks of $\mathbb{T}_-$, then $\mathbb{J}^-_{FP}, \mathbb{J}^-_{EL} \subseteq \mathbb{J}'^-$. Under T-FP, all jobs in $\mathbb{J}'^-$ have higher priority than $\tau_k(\ell)$ since they are released by the tasks of $\mathbb{T}_-$. Under EL, we show the same: Let $\tau_i(j) \in \mathbb{J}'^-$, i.e., $r^\omega_{\tau_i(j)} < r^\omega_{\tau_k(\ell)} + K_k$. It directly follows that

$$\pi^\omega_{\tau_i(j)} = r^\omega_{\tau_i(j)} + \Pi_{\tau_i} < r^\omega_{\tau_k(\ell)} + K_k + \Pi_{\tau_i} \leq r^\omega_{\tau_k(\ell)} + \Pi_{\tau_k} = \pi^\omega_{\tau_k(\ell)}. \tag{5.90}$$

In particular, $\tau_i(j)$ has higher priority than $\tau_k(\ell)$. We have shown that all jobs in $\mathbb{J}'^-$ have higher priority than $\tau_k(\ell)$ under EL and under T-FP scheduling. By assumption, the schedule of the jobs in $\mathbb{J}'^-$ coincides under T-FP and EL. Therefore, the same jobs of $\mathbb{J}'^-$ are executed during I under T-FP and EL, i.e., $\mathbb{J}^-_{FP} = \mathbb{J}^-_{EL}$.

Proof of $\mathbb{J}^0_{FP} = \mathbb{J}^0_{EL}$: Under T-FP and EL, $\mathbb{J}'^0 := \{\tau_k(1), \dots, \tau_k(\ell-1)\}$ are the jobs of τ_k that have higher priority than $\tau_k(\ell)$, i.e., $\mathbb{J}^0_{FP}, \mathbb{J}^0_{EL} \subseteq \mathbb{J}'^0$. By assumption, the schedule of the jobs in $\mathbb{J}'^0$ coincides. Therefore, the same jobs of $\mathbb{J}'^0$ are executed during I, i.e., $\mathbb{J}^0_{FP} = \mathbb{J}^0_{EL}$.

We have shown that $\mathbb{J}_{FP} = \mathbb{J}_{EL}$ by the above discussion. Since the schedule of the jobs $\mathbb{J}_{FP} = \mathbb{J}_{EL}$ coincides during I, $\tau_k(\ell)$ is preempted/blocked during the same time intervals under T-FP and EL scheduling during I. Hence, the schedule of $\tau_k(\ell)$ during I coincides. Since by assumption K_k is an upper bound on the response time under EL or T-FP scheduling, the job $\tau_k(\ell)$ finishes during I. This proves that the whole schedule of $\tau_k(\ell)$ coincides. □

Even without knowledge about the WCRTs, we can use a schedulability test based on EL scheduling for T-FP scheduling by setting the relative priority points to $\Pi_{\tau_i} = \sum_{j=1}^{i} D_{\tau_j}$. If the schedulability test assures that all jobs meet their deadline, then D_{τ_j} is an upper bound on the WCRT. In this case, the schedule obtained by EL scheduling coincides with the T-FP schedule and is feasible.

Corollary 5.55. *If the task set $\mathbb{T} = \{\tau_1, \dots, \tau_n\}$ is schedulable under EL with $\Pi_{\tau_i} := \sum_{j=1}^{i} D_{\tau_j}$ for all $i = 1, \dots, n$, then $\mathbb{T}$ is schedulable under T-FP, with priorities indicated by task indices, as well.*

Proof. If $\mathbb{T}$ is schedulable under EL with the given relative priority points Π_{τ_i}, then D_{τ_j} is an upper bound on the WCRT of τ_j for all $\tau_j \in \mathbb{T}$ under EL scheduling. In this case, by Proposition 5.54 the schedule under T-FP and EL coincide. Therefore, D_{τ_j} is also an upper bound on the WCRT of τ_j for all $\tau_j \in \mathbb{T}$ under T-FP scheduling. Hence, $\mathbb{T}$ is schedulable under T-FP as well. □

We note that the response time of tasks may be unbounded and that for such cases a T-FP algorithm cannot be treated as EL scheduling algorithm. However, these cases do not apply in practical scenarios if it is required that the jobs of all tasks finish at or before their absolute deadline.

The assignment of relative priority points also allows mixing different scheduling algorithms or hierarchical scheduling algorithms as shown in the following example.

Example 5.56. *We consider a task set $\mathbb{T} = \{\tau_1, \tau_2, \tau_3, \tau_4\}$ of 4 tasks. In the following we demonstrate how to assign priorities such that τ_1 and τ_2 are on one priority-level, and τ_3 and τ_4 are on another priority-level, and on each priority-level EDF is utilized. We assign the relative priority points $\Pi_{\tau_1} = D_{\tau_1}$, $\Pi_{\tau_2} = D_{\tau_2}$, $\Pi_{\tau_3} = D_{\tau_1} + D_{\tau_2} + D_{\tau_3}$ and $\Pi_{\tau_4} = D_{\tau_1} + D_{\tau_2} + D_{\tau_4}$. If $\mathbb{T}$ is schedulable under EL scheduling with the given relative priority points, then EL produces the same schedule as the desired scheduling policy: Since $\Pi_{\tau_i} - \Pi_{\tau_j} \geq D_{\tau_i}$ for all $i = 3, 4$ and $j = 1, 2$, a job J of τ_1 or τ_2 can only have higher priority than a job J' of τ_3 or τ_4 if J' is already finished when J is released. τ_1 and τ_2 are scheduled according to EDF, since their relative priority points are set to the deadline. The tasks τ_3 and τ_4 are also scheduled according to EDF, since for all $\omega \in \Omega$ the difference between the global priorities $\pi^{\omega}_{\tau_3(j)}$ and $\pi^{\omega}_{\tau_4(j')}$ of each two jobs $\tau_3(j)$ and $\tau_4(j')$ is the same as the difference between the absolute deadlines $r^{\omega}_{\tau_3(j)} + D_{\tau_3}$ and $r^{\omega}_{\tau_4(j')} + D_{\tau_4}$.*

To conclude, the expressiveness of EL scheduling algorithms is between T-FP and J-FP scheduling, but includes many important J-FP algorithms like EDF, EQDF, FIFO, and SAEDF. Furthermore, EL scheduling allows mixing different scheduling strategies and hierarchical scheduling.

5.1.3.2 Schedulability Test for EDF-Like Scheduling Algorithms

In this section, we derive a sufficient schedulability test for EDF-Like (EL) scheduling. That is, for an arbitrary-deadline task set $\mathbb{T} = \{\tau_1, \dots, \tau_n\}$ and an assignment of relative priority points $(\Pi_{\tau_1}, \dots, \Pi_{\tau_n})$ the test returns *True* if $\mathbb{T}$ is schedulable by the corresponding EL scheduling algorithm. We assume that each task $\tau_i \in \mathbb{T}$ can be abstracted as

$$\tau_i \in \mathrm{Spor}(T_{\tau_i}) \cap \mathrm{DynSS}(C_{\tau_i}, S_{\tau_i}). \tag{5.91}$$

In the following, we denote by $\mathcal{A}$ the preemptive EL scheduling algorithm with given priority points. For a system evolution $\omega \in \Omega$, the corresponding schedule is denoted

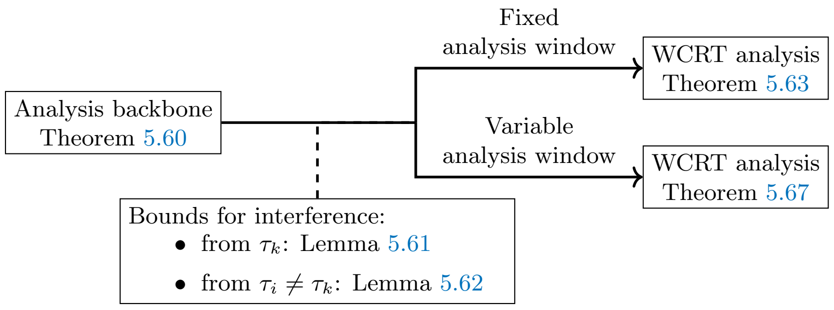


Figure 5.14.: Roadmap of the analysis in Section 5.1.3.2.

by $[\mathcal{S}] = \mathcal{A}(\omega)$. We assume that $\mathcal{A}$ schedules jobs of the *same* task $\tau_i \in \mathbb{T}$ in a FIFO manner. That is, if a job $\tau_i(j)$ of task τ_i is only eligible for execution (i.e., ready to be executed) after all jobs of τ_i released prior to $\tau_i(j)$ finish execution. Specifically, $\tau_i(j)$ cannot be executed while $\tau_i(j-1)$ is suspended.

Our high-level idea is to bound the WCRT of a task $\tau_k \in \mathbb{T}$ by considering an arbitrary job $\tau_k(\ell)$, bounding the time the job needs to run, the time the job can be suspended, and the possible interference, both from higher-priority tasks and from earlier jobs of the same task τ_k. We summarize the individual steps in the following roadmap, which we depict in Figure 5.14:

- First, we formulate the analysis backbone in Theorem 5.60 by taking into account the execution of $\tau_k(\ell)$ itself, self-interference from preceding jobs of τ_k, and interference from other tasks.
- Second, we provide upper bounds for the number of self-interfering jobs, denoted by $a_{k,\ell}$ in Lemma 5.61, and for the interference from other tasks, denoted by $B^i_{k,\ell}$ in Lemma 5.62, when EL scheduling is applied.
- Third, we present two approaches to conduct the WCRT analysis based on the analysis backbone and the bounds for the interference:
  - Analysis when considering a fixed analysis window, i.e., we only analyze active intervals starting when $\tau_k(\ell)$ is already released, in Theorem 5.63.
  - Analysis when gradually increasing the analysis window, in Theorem 5.67.

Although increasing the analyzed active interval may reduce the WCRT bound from the analysis, we discuss in Remark 5.64 that the analyses from Theorems 5.63 and 5.67 do not dominate each other.

Analysis Backbone First, we formalize the analysis backbone that we utilize for the schedulability analysis. The backbone is based on the observation that whenever a job of task $\tau_k \in \mathbb{T}$ is released but not finished, the processor is either (i) executing higher-priority jobs, or (ii) executing or suspended by a job of task τ_k. We define the following terminology to formalize the analysis backbone.

Definition 5.57 (Active and Current Job). *For a schedule $\mathcal{A}(\omega)$ of system evolution ω, a job $\tau_k(\ell)$ of a task τ_k is* active *at time t, if it is released but not already finished by time t, i.e., $t \in [r^{\omega}_{\tau_k(\ell)}, f^{\mathcal{A}(\omega),\omega}_{\tau_k(\ell)})$. When there are active jobs of task τ_k at a time instant t, then we call the active job of τ_k with the earliest release time the* current *job of τ_k at time t. We say that task τ_k is active at t, if there exists an active job of τ_k at t.*

To describe the two states of the processor (i) and (ii), we use the following notation.

Definition 5.58. *Let $\tau_k(\ell)$ be a job of τ_k. Furthermore, let $[c, d)$ with $c < d$ be any half opened interval. We define by*

$$\mathrm{B}^i_{k,\ell}(c, d) := \sum_{j\in\mathbb{N},\ \tau_i(j)>_{prio}\tau_k(\ell)} \mathrm{exec}^{\mathcal{A}(\omega)}_{\tau_i(j)}(c, d) \tag{5.92}$$

the amount of time during $[c, d)$ that the processor is executing jobs of τ_i with higher priority than $\tau_k(\ell)$. If $c \geq d$, we set both terms to 0 for simplicity. Furthermore, we define by

$$a_{k,\ell}(c) := \left| \left\{ \tau_k(j) \,\middle|\, j < \ell,\ f^{\mathcal{A}(\omega),\omega}_{\tau_k(j)} > c \right\} \right| \tag{5.93}$$

the number of jobs of τ_k before $\tau_k(\ell)$ that can potentially execute or suspend during $[c, d)$.

We utilize this notation to formulate the following lemma. Specifically, we state that if a job $\tau_k(\ell)$ does not finish until d, then the amount of execution of higher-priority jobs plus execution and suspension of jobs of τ_k exceeds d.

Lemma 5.59. *Consider some interval $[c, d)$ with $c \leq d$. If $\tau_k(\ell)$ is not finished by time d and the task τ_k is active during the whole interval $[c, d)$, then*

$$(C_{\tau_k} + S_{\tau_k}) + \sum_{i\neq k} \mathrm{B}^i_{k,\ell}(c, d) + a_{k,\ell}(c) \cdot (C_{\tau_k} + S_{\tau_k}) > (d - c). \tag{5.94}$$

Proof. Since the processor executes a job or idles during the whole interval $[c, d)$, we have

$$d - c = \sum_{J\in\mathbb{J}} \mathrm{exec}^{\mathcal{A}(\omega)}_J(c, d) + \mathrm{idle}^{\mathcal{A}(\omega)}(c, d). \tag{5.95}$$

By definition, the amount of time that jobs with higher priority than $\tau_k(\ell)$ are executed can be formulated as

$$\sum_{\substack{j\in\mathbb{N}\\ \tau_i(j)>_{prio}\tau_k(\ell)}} \mathrm{exec}^{\mathcal{A}(\omega)}_{\tau_i(j)}(c, d) = B^i_{k,\ell}(c, d) \tag{5.96}$$

for all $\tau_i \neq \tau_k$. Furthermore, since τ_k is active during the whole interval $[c, d)$, whenever a job of $\tau_i \neq \tau_k$ with lower priority than $\tau_k(\ell)$ is executed and whenever the processor idles, a job of τ_k suspends itself. Since $\tau_k(\ell)$ is not finished at time d, the amount of jobs of τ_k that can suspend themselves is upper bounded by $a_{k,\ell}(c)+1$. Hence, we obtain

$$\text{idle}^{\mathcal{A}(\omega)}(c,d) + \sum_{i\neq k} \sum_{\substack{j\in\mathbb{N}\\ \tau_i(j) <_{prio} \tau_k(\ell)}} \text{exec}^{\mathcal{A}(\omega)}_{\tau_i(j)}(c,d) \leq (a_{k,\ell}(c)+1)\cdot S_{\tau_k}. \tag{5.97}$$

Similarly, only $a_{k,\ell}(c)+1$ jobs of τ_k can be executed during $[c,d)$, and the execution of one job ($\tau_k(\ell)$) cannot be finished until d. Hence,

$$\sum_{j\in\mathbb{N}} \text{exec}^{\mathcal{A}(\omega)}_{\tau_k(j)}(c,d) < (a_{k,\ell}(c)+1)\cdot C_{\tau_k}. \tag{5.98}$$

Applying Equations (5.96), (5.97) and (5.98) to Equation (5.95) proves the lemma. □

By contraposition, if Equation (5.94) does not hold, then d is an upper bound on the finishing time of $\tau_k(\ell)$. We utilize Lemma 5.59 to provide a response-time bound $\tilde{R}_{k,\ell} \leq D_{\tau_k}$ for $\tau_k(\ell)$ by setting $d = r^\omega_{\tau_k(\ell)} + \tilde{R}_{k,\ell}$. Enlarging the window of interest from $[c,d)$ to $[c, d^\omega_{\tau_k(\ell)})$ enables the following response-time bound which serves as the analysis backbone for the remainder of this section.

Theorem 5.60 (Analysis Backbone). *Let $c \leq d^\omega_{\tau_k(\ell)} \in \mathbb{R}$ such that either (i) $\tau_k(\ell)$ is released before c, i.e., $c \geq r^\omega_{\tau_k(\ell)}$, or (ii) $c < r^\omega_{\tau_k(\ell)}$ and τ_k is active during $[c, r^\omega_{\tau_k(\ell)})$. If*

$$\tilde{R}_{k,\ell} := (C_{\tau_k}+S_{\tau_k}) + \sum_{i\neq k} \mathrm{B}^i_{k,\ell}(c, d^\omega_{\tau_k(\ell)}) + a_{k,\ell}(c)\cdot(C_{\tau_k}+S_{\tau_k}) + c - r^\omega_{\tau_k(\ell)}, \tag{5.99}$$

is at most D_{τ_k}, then $\tilde{R}_{k,\ell}$ is an upper bound on the response time of $\tau_k(\ell)$.

Proof. We prove the theorem by contradiction and assume that $\tilde{R}_{k,\ell}$ is not an upper bound on the response time of $\tau_k(\ell)$, i.e., the job $\tau_k(\ell)$ is not finished at time $r^\omega_{\tau_k(\ell)} + \tilde{R}_{k,\ell}$. We set $d := r^\omega_{\tau_k(\ell)} + \tilde{R}_{k,\ell}$. Lemma 5.59 can be applied for the following reasons:

- $d = r^\omega_{\tau_k(\ell)} + \tilde{R}_{k,\ell} = (C_{\tau_k}+S_{\tau_k}) + \sum_{i\neq k} \mathrm{B}^i_{k,\ell}(c, d^\omega_{\tau_k(\ell)}) + a_{k,\ell}(c)\cdot(C_{\tau_k}+S_{\tau_k}) + c \geq c$ by the definition of $\tilde{R}_{k,\ell}$ in Equation (5.99).
- $\tau_k(\ell)$ is not finished at time d by assumption.
- Since (i) or (ii) from the description of the theorem holds, this means that τ_k is active during the interval $[c,d)$.

Since Lemma 5.59 can be applied this means that Equation (5.94) holds. We have $\mathrm{B}^i_{k,\ell}(c,d) \leq \mathrm{B}^i_{k,\ell}(c, d^\omega_{\tau_k(\ell)})$ for all $i \neq k$ since $d \leq d^\omega_{\tau_k(\ell)}$. Hence, the left-hand side of Equation (5.94) is upper bounded by $\tilde{R}_{k,\ell} - c + r^\omega_{\tau_k(\ell)}$. We conclude that $\tilde{R}_{k,\ell} - c + r^\omega_{\tau_k(\ell)} > d - c = \tilde{R}_{k,\ell} + r^\omega_{\tau_k(\ell)} - c$ which is a contradiction. □

So far, Theorem 5.60 examines a given interval that starts at time c and ends at time $d^{\omega}_{\tau_k(\ell)}$, i.e., the absolute deadline of the analyzed job, under the assumption that the inference in the interval is known. However, how to choose the starting value c and how the interference in $[c, d^{\omega}_{\tau_k(\ell)})$ can actually be calculated has not yet been discussed. Hence, to apply the upper bound in Theorem 5.60, the following questions have to be answered:

- **Question 1:** What are the values of $\mathrm{B}^{i}_{k,\ell}(c, d^{\omega}_{\tau_k(\ell)})$ and $a_{k,\ell}(c)$? Since computing the values directly has high complexity, we use over-approximation. In Lemmas 5.61 and 5.62 we derive upper bounds for $\mathrm{B}^{i}_{k,\ell}(c, d^{\omega}_{\tau_k(\ell)})$ and $a_{k,\ell}(c)$ with $i \neq k$.

- **Question 2:** Which are good values for c? Trying out all possible c for the estimation would result in very high complexity. Therefore, we discuss two strategies to choose c in Theorems 5.67 and 5.63. More precisely, with the first procedure we restrict c to be in the interval $[r^{\omega}_{\tau_k(\ell)}, d^{\omega}_{\tau_k(\ell)})$, which has benefits on the runtime of our analysis due to the *fixed analysis windows*. For the second strategy, we examine *active intervals* for τ_k, and gradually increase the analysis window. Afterwards, we discuss that these two methods do not dominate each other.

UPPER BOUNDS ON HIGHER-PRIORITY INTERFERENCE In this subsection, we bound the interference of higher-priority jobs during the interval $[c, d^{\omega}_{\tau_k(\ell)})$, providing upper bounds for $a_{k,\ell}(c)$ in Lemma 5.61 and for $\mathrm{B}^{i}_{k,\ell}(c, d^{\omega}_{\tau_k(\ell)})$ in Lemma 5.62. We do this under the assumption that all jobs with higher priority than $\tau_k(\ell)$ meet their deadline since this is our induction hypothesis later used in Theorem 5.63 and Lemma 5.66. As mentioned earlier, how to choose c is discussed in Theorems 5.63 and 5.67.

For arbitrary deadlines, there might be several active jobs of one task at the same time, which makes the analysis in general more complicated. Note that, according to the FIFO mechanism of jobs of the same task, the jobs of τ_k must be executed one after another, i.e., even if the processor idles, a job of τ_k cannot start its execution unless all jobs of τ_k released prior to it are finished.

We achieve a bound for $a_{k,\ell}(c)$ by counting the number of jobs of τ_k with deadline in $[c, d^{\omega}_{\tau_k(\ell)})$.

Lemma 5.61 (Bound for interference from τ_k)**.** *The number of jobs of task τ_k with higher priority than $\tau_k(\ell)$ that are executed or suspended in the interval $[c, d^{\omega}_{\tau_k(\ell)})$ is upper bounded by*

$$a_{k,\ell}(c) \leq \left\lceil \frac{d^{\omega}_{\tau_k(\ell)} - c}{T_{\tau_k}} \right\rceil - 1. \tag{5.100}$$

Proof. All higher-priority jobs of τ_k finish until their deadline. Therefore, the processor can only work on or be suspended by a higher-priority job of τ_k during $[c, d^{\omega}_{\tau_k(\ell)})$ if the deadline of the job is after c. Moreover, the job of τ_k can only have higher priority than $\tau_k(\ell)$ if its deadline is no later than $d^{\omega}_{\tau_k(\ell)} - T_{\tau_k}$. At most $\left\lceil \frac{d^{\omega}_{\tau_k(\ell)} - c}{T_{\tau_k}} \right\rceil - 1$ jobs of τ_k have their deadline during $(c, d^{\omega}_{\tau_k(\ell)} - T_{\tau_k}]$. □

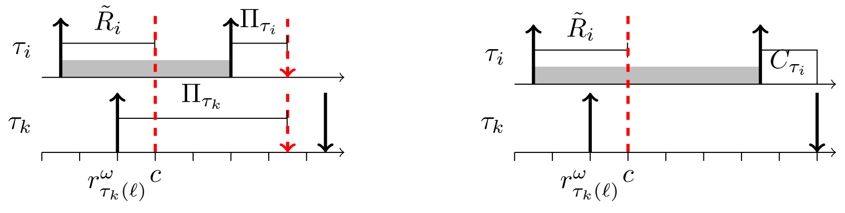


Figure 5.15.: Intuition for Lemma 5.62. The left schedule describes the approach for Equation (5.102) and the right schedule describes the approach for Equation (5.103). Only jobs of τ_i that are released during the gray boxes can have higher priority than $\tau_k(\ell)$ and be executed during the analysis interval $[c, d^{\omega}_{\tau_k(\ell)})$.

For the interference from $\tau_i \neq \tau_k$, we estimate the number of job releases during a certain interval under analysis, as depicted by the gray boxes in Figure 5.15, and account for C_{τ_i} time units for each of those jobs. For each task $\tau_i \neq \tau_k$, let $\tilde{R}_i$ be an upper bound on the WCRT of jobs of τ_i with higher priority than $\tau_k(\ell)$. We set $\tilde{R}_i := D_{\tau_i}$ if no upper bound is known, as all jobs with higher priority meet their deadline.

Lemma 5.62 (Bound for interference from $\tau_i \neq \tau_k$)**.** *For $i \neq k$, the amount of time during $[c, d^{\omega}_{\tau_k(\ell)})$ that jobs of τ_i with higher priority than $\tau_k(\ell)$ are executed is*

$$\mathrm{B}^i_{k,\ell}(c, d^{\omega}_{\tau_k(\ell)}) \leq \max\left(\left\lceil \frac{G^i_k + \tilde{R}_i + r^{\omega}_{\tau_k(\ell)} - c}{T_{\tau_i}} \right\rceil, 0\right) C_{\tau_i} \tag{5.101}$$

where $G^i_k := \min(D_{\tau_k} - C_{\tau_i}, \Pi_{\tau_k} - \Pi_{\tau_i})$*.*

Proof. The bound from Equation (5.101) is achieved by proving the two upper bounds for $\mathrm{B}^i_{k,\ell}$ we get for the two cases of G^i_k. That is, we prove the following two bounds individually:

$$\mathrm{B}^i_{k,\ell}(c, d^{\omega}_{\tau_k(\ell)}) \leq \max\left(\left\lceil \frac{\Pi_{\tau_k} - \Pi_{\tau_i} + \tilde{R}_i + r^{\omega}_{\tau_k(\ell)} - c}{T_{\tau_i}} \right\rceil, 0\right) C_{\tau_i} \tag{5.102}$$

$$\mathrm{B}^i_{k,\ell}(c, d^{\omega}_{\tau_k(\ell)}) \leq \max\left(\left\lceil \frac{D_{\tau_k} - C_{\tau_i} + \tilde{R}_i + r^{\omega}_{\tau_k(\ell)} - c}{T_{\tau_i}} \right\rceil, 0\right) C_{\tau_i} \tag{5.103}$$

Bound in Equation (5.102): The bound is based on the following two observations:

- Jobs of τ_i have higher priority than $\tau_k(\ell)$ only if they are released no later than $r^{\omega}_{\tau_k(\ell)} + \Pi_{\tau_k} - \Pi_{\tau_i}$.
- Jobs of τ_i can be executed after c only if they are released after $c - \tilde{R}_i$.

Due to these two observations, the number of jobs that contribute to $\mathrm{B}^i_{k,\ell}(c, d^\omega_{\tau_k(\ell)})$ is upper bounded by the number of releases in $(c - \tilde{R}_i, r^\omega_{\tau_k(\ell)} + \Pi_{\tau_k} - \Pi_{\tau_i}]$, which is at most $\max\left(\left\lceil \frac{\Pi_{\tau_k} - \Pi_{\tau_i} + \tilde{R}_i + r^\omega_{\tau_k(\ell)} - c}{T_{\tau_i}} \right\rceil, 0\right)$. The processor can work on each of them for at most C_{τ_i} time units.

Bound in Equation (5.103)**:** Before formally proving Equation (5.103), we first examine the worst-case scenario depicted in Figure 5.15. Intuitively, the maximal interference from task τ_i is obtained when the *last* interfering job of τ_i is released at $d^\omega_{\tau_k(\ell)} - C_{\tau_i}$ and executed for C_{τ_i} time units during $[d^\omega_{\tau_k(\ell)} - C_{\tau_i}, d^\omega_{\tau_k(\ell)})$ as depicted in Figure 5.15. The maximum number of jobs that contribute to $\mathrm{B}^i_{k,\ell}(c, d^\omega_{\tau_k(\ell)})$ is therefore upper bounded by the number of releases in $(c - \tilde{R}_i, d^\omega_{\tau_k(\ell)} - C_{\tau_i}]$, which is at most $\max\left(\left\lceil \frac{D_{\tau_k} - C_{\tau_i} + \tilde{R}_i + r^\omega_{\tau_k(\ell)} - c}{T_{\tau_i}} \right\rceil, 0\right)$. The processor can work on each of them for at most C_{τ_i} time units.

In the following we present a formal proof for the bound from Equation (5.103). If the processor is not working on any job of τ_i during $[c, d^\omega_{\tau_k(\ell)})$, then $\mathrm{B}^i_{k,\ell}(c, d^\omega_{\tau_k(\ell)}) = 0$ and the lemma is proven. Otherwise, let $\tau_i(j')$ be the last job of τ_i that the processor is working on during $[c, d^\omega_{\tau_k(\ell)})$. We isolate the job $\tau_i(j')$ in the following way. Let $r^\omega_{\tau_i(j')}$ be the release time of $\tau_i(j')$, let $s^{\mathcal{A}(\omega),\omega}_{\tau_i(j')}$ be the start time of $\tau_i(j')$, i.e., the first time that the processor is working on $\tau_i(j')$, and let C^* be the amount of time that the processor is working on $\tau_i(j')$ during the interval $[c, d^\omega_{\tau_k(\ell)})$. Therefore, by definition, we have

$$r^\omega_{\tau_i(j')} + C^* \leq d^\omega_{\tau_k(\ell)} = r^\omega_{\tau_k(\ell)} + D_{\tau_k} \tag{5.104}$$

and

$$\mathrm{B}^i_{k,\ell}(c, d^\omega_{\tau_k(\ell)}) \leq \mathrm{B}^i_{k,\ell}(c, s^{\mathcal{A}(\omega),\omega}_{\tau_i(j')}) + C^*. \tag{5.105}$$

We distinguish two cases:

Case 1: $s^{\mathcal{A}(\omega),\omega}_{\tau_i(j')} - c < (C_{\tau_i} - C^*)$: In this case, the left-hand side of Equation (5.103) is at most $\mathrm{B}^i_{k,\ell}(c, s^{\mathcal{A}(\omega),\omega}_{\tau_i(j')}) + C^* \leq (s^{\mathcal{A}(\omega),\omega}_{\tau_i(j')} - c) + C^* \leq C_{\tau_i}$. Moreover, the right-hand side of Equation (5.103) is at least $\left\lceil \frac{D_{\tau_k} - C_{\tau_i} + r^\omega_{\tau_k(\ell)} - c + \tilde{R}_i}{T_{\tau_i}} \right\rceil C_{\tau_i} \geq \left\lceil \frac{D_{\tau_k} + r^\omega_{\tau_k(\ell)} - c}{T_{\tau_i}} \right\rceil C_{\tau_i} \geq C_{\tau_i}$ since:

- $\tilde{R}_i \geq C_{\tau_i}$ by definition of $\tilde{R}_i$.
- $\tau_i(j')$ is executed during $[c, d^\omega_{\tau_k(\ell)})$ and therefore $c < D_{\tau_k} + r^\omega_{\tau_k(\ell)}$ must hold.

In this case, Equation (5.103) is proven.

Case 2: $s^{\mathcal{A}(\omega),\omega}_{\tau_i(j')} - c \geq (C_{\tau_i} - C^*)$: Since during the interval $[c, c + (C_{\tau_i} - C^*))$ the processor can work on jobs of τ_i for at most $(C_{\tau_i} - C^*)$ time units, we have $\mathrm{B}^i_{k,\ell}(c, s^{\mathcal{A}(\omega),\omega}_{\tau_i(j')}) \leq$

$(C_{\tau_i} - C^*) + \mathrm{B}^i_{k,\ell}(c + (C_{\tau_i} - C^*), s^{\mathcal{A}(\omega),\omega}_{\tau_i(j')})$. Hence, we obtain:

$$\mathrm{B}^i_{k,\ell}(c, d^\omega_{\tau_k(\ell)}) \overset{\text{by (5.105)}}{\leq} \mathrm{B}^i_{k,\ell}(c, s^{\mathcal{A}(\omega),\omega}_{\tau_i(j')}) + C^* \tag{5.106}$$

$$\leq (C_{\tau_i} - C^*_{\tau_i}) + \mathrm{B}^i_{k,\ell}(c + (C_{\tau_i} - C^*), s^{\mathcal{A}(\omega),\omega}_{\tau_i(j')}) + C^*_{\tau_i} \tag{5.107}$$

$$= \mathrm{B}^i_{k,\ell}(c + (C_{\tau_i} - C^*), s^{\mathcal{A}(\omega),\omega}_{\tau_i(j')}) + C_{\tau_i} \tag{5.108}$$

The number of jobs of τ_i that contribute to $\mathrm{B}^i_{k,\ell}(c + (C_{\tau_i} - C^*), s^{\mathcal{A}(\omega),\omega}_{\tau_i(j')})$ is at most the number of releases of τ_i during the interval $(c + C_{\tau_i} - C^* - \tilde{R}_i, r^\omega_{\tau_i(j')} - T_{\tau_i}]$, which is

$$\begin{aligned}\left\lceil \frac{r^\omega_{\tau_i(j')} - T_{\tau_i} - c - C_{\tau_i} + C^* + \tilde{R}_i}{T_{\tau_i}} \right\rceil &\overset{\text{by (5.104)}}{\leq} \left\lceil \frac{r^\omega_{\tau_k(\ell)} - T_{\tau_i} - c - C_{\tau_i} + D_{\tau_k} + \tilde{R}_i}{T_{\tau_i}} \right\rceil \\ &= \left\lceil \frac{r^\omega_{\tau_k(\ell)} - c - C_{\tau_i} + D_{\tau_k} + \tilde{R}_i}{T_{\tau_i}} \right\rceil - 1.\end{aligned}$$

Therefore, for this case, we conclude that

$$\begin{aligned}\mathrm{B}^i_{k,\ell}(c, d^\omega_{\tau_k(\ell)}) &\leq \mathrm{B}^i_{k,\ell}(c + (C_{\tau_i} - C^*), s^{\mathcal{A}(\omega),\omega}_{\tau_i(j')}) + C_{\tau_i} \\ &\leq \left(\left\lceil \frac{r^\omega_{\tau_k(\ell)} - c - C_{\tau_i} + D_{\tau_k} + \tilde{R}_i}{T_{\tau_i}} \right\rceil - 1 \right) C_{\tau_i} + C_{\tau_i} \\ &= \left\lceil \frac{r^\omega_{\tau_k(\ell)} - c - C_{\tau_i} + D_{\tau_k} + \tilde{R}_i}{T_{\tau_i}} \right\rceil C_{\tau_i}.\end{aligned}$$

Hence, for this case, Equation (5.103) is proven as well. □

Fixed Analysis Window In the following, we fix the analysis window, i.e., the possible range of $[c, d^\omega_{\tau_k(\ell)})$, to the interval $[r^\omega_{\tau_k(\ell)}, d^\omega_{\tau_k(\ell)})$. We utilize the upper bounds on $\mathrm{B}^i_{k,\ell}(c, d^\omega_{\tau_k(\ell)})$ and $a_{k,\ell}(c)$ provided previously to obtain the following schedulability test.

Theorem 5.63 (Sufficient Schedulability Test)**.** *Let $\mathbb{T} = \{\tau_1, \dots, \tau_n\}$ be an arbitrary-deadline task set with relative priority points $(\Pi_{\tau_1}, \dots, \Pi_{\tau_n})$. If for all $k = 1, \dots, n$ there exists some $b_k \in [0, D_{\tau_k})$ such that*

$$\tilde{R}_k(b_k) \leq D_{\tau_k}, \tag{5.109}$$

where $\tilde{R}_k(b_k) := \sum_{i \neq k} \max\left(\left\lceil \frac{G^i_k + \tilde{R}_i - b_k}{T_{\tau_i}} \right\rceil, 0 \right) C_{\tau_i} + \left\lceil \frac{D_{\tau_k} - b_k}{T_{\tau_k}} \right\rceil (C_{\tau_k} + S_{\tau_k}) + b_k$ *and* $G^i_k = \min(D_{\tau_k} - C_{\tau_i}, \Pi_{\tau_k} - \Pi_{\tau_i})$*, then the task set is schedulable by EL scheduling with the given relative priority points and the WCRT of τ_k is upper bounded by $\tilde{R}_k := \tilde{R}_k(b_k)$.*

Algorithm 3 Schedulability test with fixed analysis window.

```
Input: $\mathbb{T} = \{\tau_1, \dots, \tau_n\}$, $(\Pi_{\tau_1}, \dots, \Pi_{\tau_n})$, $\eta$, $depth$
Output: True: schedulable, False: no decision

1: Order $\tau_1, \dots, \tau_n$, s.th. $D_{\tau_1} \geq \dots \geq D_{\tau_n}$.
2: Set $\tilde{R}_i := D_{\tau_i}$ for all $i$.
3: for $i = 1, 2, \dots, depth$ do
4:     $solved := True$
5:     for $k = 1, 2, \dots, n$ do
6:         $cand := [\,]$; $step := \eta \cdot D_{\tau_k}$                ▷ Preparation.
7:         for $b = 0, step, 2 \cdot step, \dots < D_{\tau_k}$ do          ▷ Compute.
8:             $cand.append(\tilde{R}_k(b))$ using Equation (5.109).
9:         $\tilde{R}_k := \min(cand)$                         ▷ Compare candidates.
10:        if $\tilde{R}_k > D_{\tau_k}$ then                      ▷ Check condition.
11:            $solved := False$; $\tilde{R}_k := D_{\tau_k}$; break
12: return $solved$
```

Proof. Assume we have found b_k, $k = 1, \dots, n$ such that Equation (5.109) holds. We consider some schedule obtained by the task set $\mathbb{T}$ and denote by Seq the sequence of all jobs in the schedule ordered by their priority. Via induction, we prove that the first ξ jobs in Seq have the required response-time upper bound, for all $\xi \in \mathbb{N}_0$. Consequently, $\tilde{R}_k$ is an upper bound on the WCRT of τ_k for all k and the task set is schedulable.

Initial case: $\xi = 0$. In this case, the set of the first ξ jobs in Seq is the empty set. Trivially, all of them have the required response-time upper bound.

Induction step: $\xi \mapsto \xi + 1$. By assumption, the first ξ jobs in Seq have the required response-time upper bound. We denote the $(\xi+1)$-th job in Seq by $\tau_k(\ell)$. We aim to use the analysis backbone from Theorem 5.60 to prove that the response time of $\tau_k(\ell)$ is upper bounded by $\tilde{R}_k$. By definition, we have $\tilde{R}_{k,\ell} = (C_{\tau_k} + S_{\tau_k}) + \sum_{i \neq k} \mathrm{B}^i_{k,\ell}(c, d^\omega_{\tau_k(\ell)}) + a_{k,\ell}(c) \cdot (C_{\tau_k} + S_{\tau_k}) + c - r^\omega_{\tau_k(\ell)}$. Since all higher-priority jobs have the required response-time upper bound, we can use the estimation for $a_{k,\ell}(c)$ and $\mathrm{B}^i_{k,\ell}(c, d^\omega_{\tau_k(\ell)})$ presented in Lemmas 5.61 and 5.62, respectively. In particular, $\tilde{R}_{k,\ell}$ is upper bounded by

$$
\begin{aligned}
&(C_{\tau_k} + S_{\tau_k}) + \sum_{i \neq k} \max\left(\left\lceil \frac{G^i_k + \tilde{R}_i + r^\omega_{\tau_k(\ell)} - c}{T_{\tau_i}} \right\rceil, 0\right) C_{\tau_i} \\
&+ \left(\left\lceil \frac{d^\omega_{\tau_k(\ell)} - c}{T_{\tau_k}} \right\rceil - 1\right) \cdot (C_{\tau_k} + S_{\tau_k}) + c - r^\omega_{\tau_k(\ell)}.
\end{aligned}
\tag{5.110}
$$

When choosing $c := b_k + r^\omega_{\tau_k(\ell)}$ we obtain that $\tilde{R}_{k,\ell}$ is upper bounded by $\tilde{R}_k$. Due to Equation (5.109), we know $\tilde{R}_{k,\ell} \leq \tilde{R}_k \leq D_{\tau_k}$, i.e., the job meets its deadline. We use Theorem 5.60 to conclude that $\tilde{R}_{k,\ell}$ is an upper bound on the response time of $\tau_k(\ell)$ and therefore $\tilde{R}_k$ is an upper bound on the response time of $\tau_k(\ell)$ as well. Since all jobs meet their deadline, the task set is schedulable. □

Although Theorem 5.63 looks like a classical mechanism extended from TDA [JP86; LSD89], implementing an efficient schedulability test based on it requires some efforts since the values of $\tilde{R}_k$ for every task τ_k are dependent on each other. To apply this schedulability test, two critical points have to be addressed:

(i) Finding good values for b_k with low complexity. Without an efficient mechanism, there are D_{τ_k} options for b_k, provided that all input parameters are integers, and infinitely many options in general.

(ii) Computing the dependent values of $\tilde{R}_k$ for every task τ_k correctly and efficiently.

To determine the values of b_k, we take a user-specified parameter η and discretize the search space into $\frac{1}{\eta}$ values with a step size $\eta \cdot D_{\tau_k}$. To determine $\tilde{R}_k$, we go through the task set several times and compute upper bounds for the values of $\tilde{R}_k$ in each iteration. Improving this search algorithm is out of scope for this dissertation but may be discussed in future work.

The search algorithm is depicted in pseudocode in Algorithm 3. It takes as input the task set $\mathbb{T}$, the relative priority points $(\Pi_{\tau_1}, \dots, \Pi_{\tau_n})$, a step size parameter $\eta \in (0, 1]$ and *depth* to indicate the number improving runs of the search algorithm. It returns *True* if the task set is schedulable by EL scheduling with the given relative priority points. We start by setting $\tilde{R}_k = D_{\tau_k}$ for all $k = 1, \dots, n$, and go *depth*-times through the task set ordered by the relative deadline, as we obtained the best results with this ordering. With a step size of $\textit{step} = \eta \cdot D_{\tau_k}$, i.e., a certain share of D_{τ_k} like 1 percent, we compute $\tilde{R}_k(b_k)$ for $b_k = 0, 1 \cdot \textit{step}, 2 \cdot \textit{step}, \dots$ until $b_k \geq D_{\tau_k}$ is reached. We then take the minimal value of all these candidates and define it as the new $\tilde{R}_k$. The time complexity of Algorithm 3 is $\mathcal{O}\left(\frac{\textit{depth} \cdot n^2}{\eta}\right)$.

Please note that the computed values of $\tilde{R}_k$ are in fact only upper bounds of $\tilde{R}_k$ from Theorem 5.63. A reduction of $\tilde{R}_i$, $i \neq k$ in subsequent iterations reduces the actual value of $\tilde{R}_k$ as well, since $\tilde{R}_k$ is monotonically increasing with respect to $\tilde{R}_i$, for all $i \neq k$.

Variable Analysis Window In the following, we detail an approach based on *active intervals*. More specifically, if all jobs finish until the next job release is reached, i.e., $R^{\mathcal{A}}_{\tau_k} \leq T_{\tau_k}$, then no previous jobs contribute interference to the job under analysis, and they can be safely removed from the computation of the WCRT upper bound. However, if $R^{\mathcal{A}}_{\tau_k} > T_{\tau_k}$, then interference from previous jobs has to be considered. We utilize that a job $\tau_k(\ell - a)$ can only interfere with $\tau_k(\ell)$, if τ_k is active during $[r^{\omega}_{\tau_k(\ell-a)}, r^{\omega}_{\tau_k(\ell)})$. For the schedulability test with variable analysis window, we gradually increase the length of the active interval (i.e., $a = 0, 1, 2, \dots$) and analyze the window $[c, d^{\omega}_{\tau_k(\ell)})$ with $c \in [r^{\omega}_{\tau_k(\ell-a)}, d^{\omega}_{\tau_k(\ell)})$. With this approach, the pessimism of the interference estimation from higher-priority jobs of the same task is reduced in some cases.

Remark 5.64. *Please note that the approach with variable analysis window only differs from the approach with fixed analysis window when considering arbitrary-deadline tasks. For constrained-deadline task sets, the variable analysis window approach coincides with the fixed analysis window approach, as the algorithm stops at $a = 0$ without enlarging the analysis window. The reason is that for constrained-deadline task sets in the variable analysis window approach we get one of the following two cases:*

Case 1: If the response-time bound obtained for the first job in an active interval is larger than the task's deadline, then the schedulability cannot be guaranteed, and the algorithm stops at $a = 0$.

Case 2: If the response-time bound for the first job in an active interval is at most the task's deadline, then $R^{\mathcal{A}}_{\tau_k} \leq D_{\tau_k} \leq T_{\tau_k}$ is guaranteed for the first job in an active interval. Therefore, the subsequent job is a first job in some active interval as well. By induction, all jobs of τ_k are first job in some active interval. Therefore, the algorithm stops at $a = 0$.

We start by formally defining active intervals.

Definition 5.65. *Let $a \in \mathbb{N}_0$. A job $\tau_k(\ell)$ is the $(a+1)$-th job in an* active interval *of τ_k, if the following two conditions hold.*

- *τ_k is active during $[r^\omega_{\tau_k(\ell-a)}, f^{\mathcal{A}(\omega),\omega}_{\tau_k(\ell)})$.*
- *At time $r^\omega_{\tau_k(\ell-a)}$ there is no active job which is released* before *$r^\omega_{\tau_k(\ell-a)}$.*

If $\tau_k(\ell)$ is the $(a+1)$-th job in an active interval of τ_k, then only $\tau_k(\ell-a), \dots, \tau_k(\ell)$ are current jobs of τ_k during $[r^\omega_{\tau_k(\ell-a)}, f^{\mathcal{A}(\omega),\omega}_{\tau_k(\ell)})$. More specifically, in this case the value of $B_{k,j}(c, d^\omega_{\tau_k(\ell)})$ is 0 if $c \geq r^\omega_{\tau_k(\ell-a)}$ and $j < \ell - a$. We formalize this by the following lemma.

Lemma 5.66. *Let $\tau_k(\ell)$ be the $(a+1)$-th job in an active interval of τ_k and let all higher-priority jobs meet their deadline. Let $\tilde{R}_i$ be an upper bound on the response time of all higher-priority jobs of τ_i with $i \neq k$. If there exists some $c \in [r^\omega_{\tau_k(\ell-a)}, d^\omega_{\tau_k(\ell)})$ such that*

$$\begin{aligned} &\min\left(a+1, \left\lceil \frac{d^\omega_{\tau_k(\ell)} - c}{T_{\tau_k}} \right\rceil\right)(C_{\tau_k} + S_{\tau_k}) \\ &+ \sum_{i \neq k} \max\left(\left\lceil \frac{G^i_k + \tilde{R}_i + r^\omega_{\tau_k(\ell)} - c}{T_{\tau_i}} \right\rceil, 0\right) C_{\tau_i} + c - r^\omega_{\tau_k(\ell)} \end{aligned} \tag{5.111}$$

is at most D_{τ_k}, with $G^i_k := \min(D_{\tau_k} - C_{\tau_i}, \Pi_{\tau_k} - \Pi_{\tau_i})$, then (5.111) is an upper bound on the response time of $\tau_k(\ell)$.

Proof. For the proof, we apply the analysis backbone from Theorem 5.60. Since $\tau_k(\ell)$ is the $(a+1)$-th job in an active interval, τ_k is active during $[r^\omega_{\tau_k(\ell-a)}, r^\omega_{\tau_k(\ell)})$. Hence, the restriction on c in the formulation of Theorem 5.60 is fulfilled if c is chosen from the interval $[r^\omega_{\tau_k(\ell-a)}, d^\omega_{\tau_k(\ell)})$. Moreover, since $\tau_k(\ell)$ is the $(a+1)$-th job in an active interval, the jobs $\tau_{k,1}, \dots, \tau_{k,\ell-a-1}$ are finished by time $r^\omega_{\tau_k(\ell-a)}$. We obtain $a_{k,\ell}(c) \leq a$. We combine this upper bound on $a_{k,\ell}(c)$ with the upper bounds from Lemma 5.61 and

Lemma 5.62, and obtain that $\tilde{R}_{k,\ell}$ from the analysis backbone, i.e., Equation (5.99), is upper bounded by the value in Equation (5.111). If Equation (5.111) is at most D_{τ_k}, then $\tilde{R}_{k,\ell} \leq D_{\tau_k}$. The analysis backbone from Theorem 5.60 states that $\tilde{R}_{k,\ell}$ is an upper bound on the response time of $r^{\omega}_{\tau_k(\ell)}$ and therefore, also Equation (5.111) is an upper bound on the response time. □

Similar to the proof of the response-time upper bound for the fixed analysis window in Theorem 5.63, we derive a response-time bound for the variable analysis window. However, the different cases for $a = 0, 1, \dots$ have to be considered for each task τ_k and the value of c can be chosen from the interval $[r^{\omega}_{\tau_k(\ell)} - aT_{\tau_k}, r^{\omega}_{\tau_k(\ell)} + D_{\tau_k})$. To improve readability, in the following we replace $r^{\omega}_{\tau_k(\ell)} - c$ by $aT_{\tau_k} - x$, where x can be chosen from the interval $[0, aT_{\tau_k} + D_{\tau_k})$.

Theorem 5.67 (Sufficient Schedulability Test)**.** *Let $\mathbb{T} = \{\tau_1, \dots, \tau_n\}$ be an arbitrary-deadline task set with relative priority points $(\Pi_{\tau_1}, \dots, \Pi_{\tau_n})$. We define the function $\tilde{R}^a_k \colon [0, aT_{\tau_k} + D_{\tau_k}) \to \mathbb{R}_{\geq 0}$ by the assignment*

$$\begin{aligned} x \mapsto \min\left(a+1, \left\lceil \frac{D_{\tau_k} - x + aT_{\tau_k}}{T_{\tau_k}} \right\rceil\right) (C_{\tau_k} + S_{\tau_k}) \\ + \sum_{i \neq k} \max\left(\left\lceil \frac{G^k_i + \tilde{R}_i - x + aT_{\tau_k}}{T_{\tau_i}} \right\rceil, 0\right) C_{\tau_i} + x - aT_{\tau_k}. \end{aligned} \tag{5.112}$$

If for all $k = 1, \dots, n$ there exists $\tilde{a}_k \in \mathbb{N}_0$, such that for all $a = 0, \dots, \tilde{a}_k$ there exists $b^a_k \in [0, aT_{\tau_k} + D_{\tau_k})$, such that

$$\tilde{R}^a_k(b^a_k) \leq D_{\tau_k} \qquad \text{and} \qquad \tilde{R}^{\tilde{a}_k}_k(b^{\tilde{a}_k}_k) \leq T_{\tau_k}, \tag{5.113}$$

then the task set is schedulable by preemptive EL scheduling with the given relative priority points and $\tilde{R}_k := \max_{a=0,\dots,\tilde{a}} \tilde{R}^a_k(b^a_k)$ is an upper bound on the WCRT of τ_k for all k.

Proof. The proof is similar to the one of the sufficient schedulability test for the fixed analysis window presented in Theorem 5.63. Let *Seq* be the sequence of all jobs in the schedule ordered by their priority. By induction, we show that the following response-time upper bounds hold for the first $\xi \in \mathbb{N}_0$ jobs in *Seq*:

(i) $\tilde{R}_k$ is a response-time upper bound for all jobs of τ_k.

(ii) T_{τ_k} is a response-time upper bound for all $\tilde{a}_k$-th jobs in an active interval of τ_k.

Initial case: ξ_0. The initial case is again trivially fulfilled, since there has nothing to be checked when there are no jobs.

Induction step: $\xi \mapsto \xi + 1$. The first ξ jobs in *Seq* have the required response-time upper bounds (i) and (ii) by induction. We denote by $\tau_k(\ell)$ the $(\xi + 1)$-th job of *Seq*. Let a be the lowest value in $\mathbb{N}_0$, such that $\tau_k(\ell)$ is the $(a + 1)$-th job in an active interval of task τ_k.

Algorithm 4 Schedulability test with variable analysis window.

```
Input: $\mathbb{T} = \{\tau_1, \dots, \tau_n\}$, $(\Pi_{\tau_1}, \dots, \Pi_{\tau_n})$, $\eta$, $max\_a$, $depth$
Output: True: schedulable, False: no decision

1: Order $\tau_1, \dots, \tau_n$, s.th. $D_{\tau_1} \geq \dots \geq D_{\tau_n}$.
2: Set $\tilde{R}_i := D_{\tau_i}$ for all $i$.
3: for $i = 1, 2, \dots, depth$ do
4:     $solved := True$
5:     for $k = 1, 2, \dots, n$ do
6:         for $a = 0, 1, \dots, max\_a$ do                                  ▷ Different $a$.
7:             $cand := [\,]$; $step := \eta \cdot D_{\tau_k}$                      ▷ Preparation.
8:             for $b = 0, step, 2 \cdot step, \dots < aT_{\tau_k} + D_{\tau_k}$ do       ▷ Compute candidate.
9:                 $cand.append(\tilde{R}_k^a(b))$ from Equation (5.112)
10:            $\tilde{R}_k^a := \min(cand)$                                  ▷ Compare candidates.
11:            if $\tilde{R}_k^a \leq T_{\tau_k}$ then                              ▷ Check cond. 1.
12:                $\tilde{a} := a$; $\tilde{R}_k := \min_{a=0,\dots,\tilde{a}} \tilde{R}_k^a$; break       ▷ WCRT upper bound.
13:            if $\tilde{R}_k^a > D_{\tau_k}$ or $a = max\_a$ then                  ▷ Check cond. 2.
14:                $solved := False$; $\tilde{R}_k := D_{\tau_k}$; break
15: return $solved$
```

We first show that $a \leq \tilde{a}_k$ by contraposition. In this regard, we assume $a > \tilde{a}_k$ and consider the job $\tau_{k,\ell-(a-\tilde{a}_k)}$. The job $\tau_{k,\ell-(a-\tilde{a}_k)}$ is the $(\tilde{a}_k+1)$-th job in an active interval of τ_k. Moreover, this job is one of the first ξ jobs in Seq and therefore has a response time of at most T_{τ_k} due to (ii). Hence, the job $\tau_{k,\ell-(a-\tilde{a}_k)}$ is finished by time $r_{k,\ell-(a-\tilde{a}_k)+1}$. We conclude that $\tau_k(\ell)$ is the $(a-\tilde{a}_k)$-th job in an active interval of τ_k, which contradicts the minimality of a.

We now choose $c := b_k^a - a \cdot T_{\tau_k} + r^{\omega}_{\tau_k(\ell)}$. Applying the response-time upper bound provided by Equation (5.111) in Lemma 5.66 with this c shows that $\tilde{R}_k^a(b_k^a)$ is a response-time upper bound of $\tau_k(\ell)$, which shows that (i) holds for $\tau_k(\ell)$. If $a = \tilde{a}_k$, then $\tilde{R}_k^{\tilde{a}_k}(b_k^{\tilde{a}_k})$ is an upper bound on the response time of $\tau_k(\ell)$. By Equation (5.113) the response time of $\tau_k(\ell)$ is upper bounded by T_{τ_k}, which shows that (ii) holds for $\tau_k(\ell)$ as well. □

We apply a similar search strategy as for the case with fixed analysis window. However, the values of $\tilde{R}_k$ are computed through an additional loop over the values of a until $\tilde{R}_k^a \leq T_{\tau_k}$. Algorithm 4 depicts an implementation of the schedulability test in pseudocode. As the value of $\tilde{R}_k^a$ can be between T_{τ_k} and D_{τ_k} for all iterations of a, the program may never return a result. To make the schedulability test deterministic, we introduce an additional parameter max_a which aborts the loop when no result is obtained for $a = 0, 1, ..., max\_a$.

Dominance of Fixed and Variable Analysis Window At first glance, the analysis with variable window size seems to improve the analysis with fixed window size in all cases: When setting $x = b_k + a \cdot T_{\tau_k}$ in Theorem 5.67, then the result is lower bounded by $\tilde{R}_k(b_k)$ from Theorem 5.63. However, both methods do not dominate each other, as demonstrated in the discussion of Figure 5.18 in Section 5.1.3.3, due to the following reasons. First, the analysis with variable analysis window can only analyze schedules where the length of active intervals is bounded. More specifically, if the response-time upper bound $\tilde{R}_k^a$ is in the interval (T_{τ_k}, D_{τ_k}) for all a, then the schedulability test with the variable analysis window never deems the task schedulable. Second, by setting max_a this effect is even intensified: The analysis with variable analysis window has to find $\tilde{R}_k^a \leq T_{\tau_k}$ for $a \leq max\_a$. Third, the discretization using η in Algorithm 3 and Algorithm 4 ensures the same number of points for each analysis interval. As a result, not all points $b + aT_{\tau_k}$ with b from Algorithm 3 are checked during Algorithm 4 as well.

5.1.3.3 Evaluation

In this section, we evaluate the performance of our schedulability tests **(EL)** presented in Algorithm 3 for the fixed analysis window and in Algorithm 4 for the variable analysis window. More precisely, we show the following:

(i) Our schedulability test performs similar to already existing schedulability tests for DM and improves the state of the art for EDF scheduling.

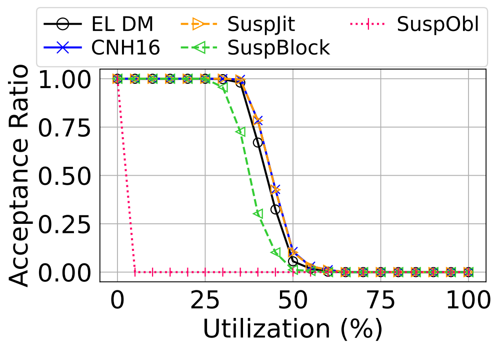
(ii) Our schedulability test can be used to compare different configurations of EQDF and SAEDF (see Section 5.1.3.1).

(iii) Our schedulability test exploits the optimism introduced when the deadline of tasks is extended over their minimum inter-arrival time.

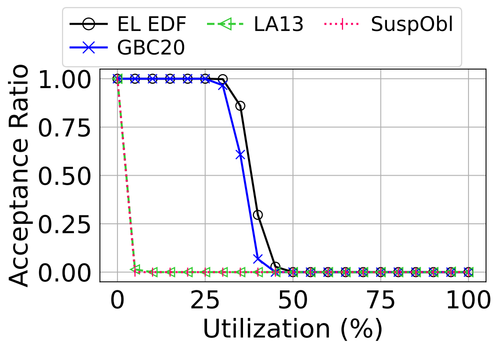
Please note that for constrained deadlines we do not distinguish between fixed and variable analysis window since both schedulability tests coincide, as explained in Remark 5.64. When applying our schedulability test, we choose the configuration $\eta = 0.01$, $depth = 5$, and $max\_a = 10$. In each figure, we present the *acceptance ratio*, which is the share of task sets that are deemed schedulable by the schedulability test under consideration. The evaluation is released on GitHub [TU 22a].

For the evaluation, we synthesize 500 task sets with 50 tasks each for each total system utilization from 0 % to 100 % in steps of 5 %. We first generate 50 utilization values U_{τ_i} using the UUniFast [BB05] method with the given total utilization goal, and adopt the suggestion by Emberson et al. [ESD10] to draw the minimum inter-arrival time T_{τ_i} according to a log-uniform distribution from the interval $[1, 100][ms]$. The WCET is computed as $C_{\tau_i} = T_{\tau_i} \cdot U_{\tau_i}$ and the deadline is set to the minimum inter-arrival time, i.e., $D_{\tau_i} = T_{\tau_i}$. For each task, we draw the maximum suspension time S_{τ_i} uniformly at random from $[0, 0.5(T_{\tau_i} - C_{\tau_i})]$. The tasks in each set are ordered by their deadline.

In Figure 5.16a, we show the results when applying our schedulability test for EL, with relative priority points $\Pi_{\tau_i} = \sum_{j=1}^{i} D_{\tau_j}$ to obtain a schedulability test for DM scheduling **(EL DM)**. We compare with the methods *Suspension as Jitter* **(SuspJit)** [Che+19b, Page 163] and *Suspension as Blocking* **(SuspBlock)** [Che+19b, Page 165]. Moreover, we



(a) Deadline Monotonic (DM).



(b) Earliest-Deadline-First (EDF).

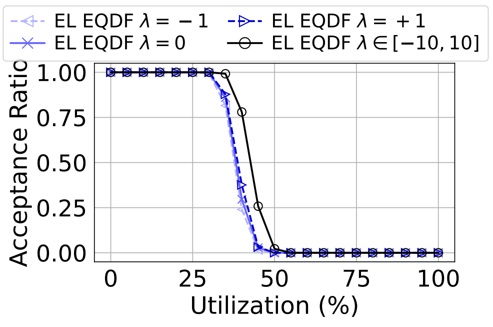
Figure 5.16.: Acceptance ratio of different schedulability tests. Our EDF-Like (EL) schedulability test (EL DM and EL EDF, black curve) performs similar to the state of the art.

compare with the Suspension-Oblivious Analysis **(SuspObl)** [Che+19b, Page 162] and the Unifying Analysis Framework from Chen, Nelissen, and Huang **(CNH16)** [CNH16] configured with three vectors according to Equation (27), Lemma 15, and Lemma 16 of their paper. Our schedulability test performs similar to the state of the art.

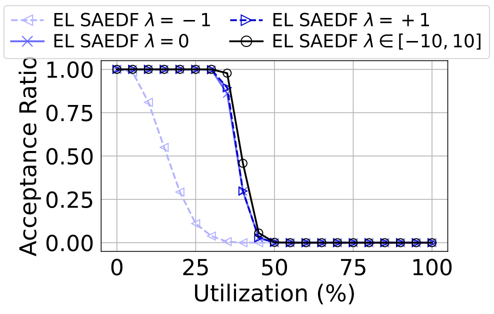
In Figure 5.16b, we compare our schedulability test **(EL EDF)** with state-of-the-art methods for EDF. We compare with the method by Liu and Anderson **(LA13)** [LA13]. Moreover, we show the schedulability test proposed in Section 5.1.2.2 **(GBC20)** and the Suspension-Oblivious Analysis from Theorem 5.26 in Section 5.1.2.1 **(SuspObl)**. The method from Dong and Liu [DL16] is not displayed since it is dominated by **SuspObl**, as shown in Section 5.1.2.1. We observe that **EL EDF** improves the state of the art.

In Figure 5.17, the performance of our schedulability test is shown for different configurations for Earliest-Quasi-Deadline-First (EQDF) ($\Pi_{\tau_i} = D_{\tau_i} + \lambda C_{\tau_i}$) and for Suspension-Aware EDF (SAEDF) ($\Pi_{\tau_i} = D_{\tau_i} + \lambda S_{\tau_i}$). Choosing λ to be the best integer in the interval $[-10, 10]$ improves the acceptance ratio compared to the standard EDF with $\lambda = 0$, especially for EQDF.

Figures 5.18 and 5.19 show the performance of our schedulability test for arbitrary deadlines. More specifically, we set the deadline to $x = 1.0, 1.1, 1.2, 1.5$ times the minimum inter-arrival time **(Dx)** and apply our schedulability test. We see that both the fixed and the variable analysis window lead to better acceptance ratios in certain scenarios, depending on the size of x and the scheduling algorithm under analysis. From the theoretical discussion in Remark 5.64, we know that **EL-fix DM 1.0** (**EL-fix EDF 1.0**, respectively) and **EL-var DM 1.0** (**EL-var EDF 1.0**, respectively) coincide. We observe that **EL-var** already benefits from small enlargements of the deadline, whereas **EL-fix** can reach better guarantees for larger deadline extensions in some scenarios. The non-dominance, discussed at the end of Section 5.1.3.2, can be observed in Figure 5.18 for D1.2 and D1.5.

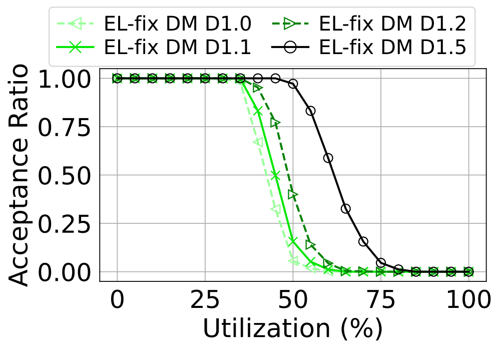


(a) EQDF ($\Pi_{\tau_i} = D_{\tau_i} + \lambda C_{\tau_i}$).

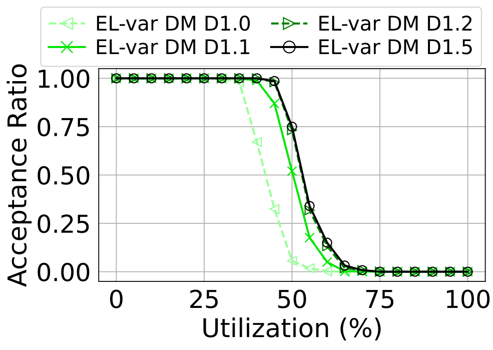


(b) SAEDF ($\Pi_{\tau_i} = D_{\tau_i} + \lambda S_{\tau_i}$).

Figure 5.17.: Acceptance ratio of variants of EDF using our EDF-Like (EL) schedulability test. Choosing the best $\lambda \in [-10, 10]$ for each task set (black line) improves standard EDF ($\lambda = 0$).

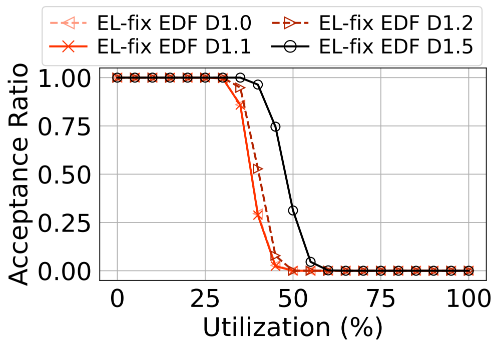


(a) Fixed analysis window.

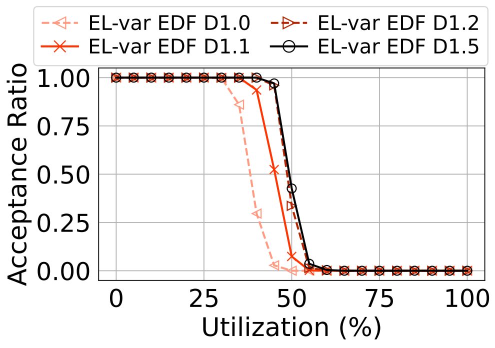


(b) Variable analysis window.

Figure 5.18.: Arbitrary-deadline evaluation for Deadline Monotonic (DM) scheduling.

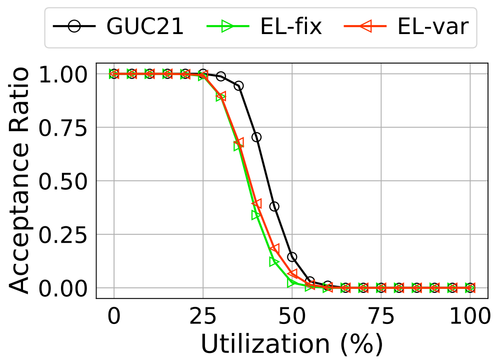


(a) Fixed analysis window.

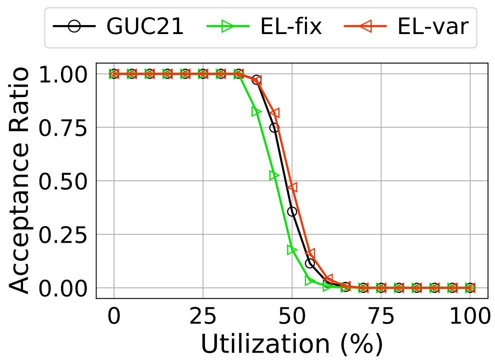


(b) Variable analysis window.

Figure 5.19.: Arbitrary-deadline evaluation for EDF scheduling.



(a) $D_{\tau_i} \in [0.8, 1.2]T_{\tau_i}$.



(b) $D_{\tau_i} \in [1.0, 1.2]T_{\tau_i}$.

Figure 5.20.: Comparison of arbitrary-deadline schedulability tests under Deadline Monotonic (DM) scheduling.

In Figure 5.20, we compare the performance of our schedulability test **(EL-fix, EL-var)** with the test proposed in Section 5.1.1.2 **(GUC21)**, applying them to arbitrary-deadline tasks under DM scheduling. We observe that in the more general case with $[0.8, 1.2]T_{\tau_i}$, **GUC21** outperforms **EL-fix** and **EL-var**. However, as shown in Figure 5.20b, there are some configurations where **EL-var** performs better than **GUC21**.

Moreover, we examine the **runtime** of our analysis. We create 100 task sets for each utilization in 0 % to 100 % in steps of 10 % and measure the runtime to receive a schedulability decision. To obtain the measurements, we run an implementation with Python3 on a machine with 2x AMD EPYC 7742 running Linux, i.e., in total 256 threads with 2.25 GHz and 256 GB RAM. Each of the measurements runs on one independent thread. For sets with 200 tasks, our schedulability test takes on average 12.87 seconds and at most 17.77 seconds to return the result. The runtime for other relative priority points is comparable.

5.1.4 Evaluation Framework

Numerical simulations often play an important role when evaluating and comparing the performance of schedulability tests, as they allow to empirically demonstrate their applicability using synthesized task sets under various configurations. In order to provide a fair comparison of various schedulability tests, von der Brüggen et al. [von+19] presented the first version of an evaluation framework for self-suspending task sets in WATERS 2019. It provides an easy-to-use framework for evaluating the performance of various schedulability tests, based on synthesized task sets. However, only a few schedulability tests were implemented. We enhance the framework by implementing the state of the arts we are aware of and by providing additional features.

The functionality of the framework is threefold: First, it can be used to generate self-suspending task sets, using existing synthesis approaches [ESD10]. Second, it provides schedulability tests that can be applied to the generated task sets or external task sets, which are loaded into the framework. Third, it illustrates the performance of schedulability tests by plotting acceptance ratio, i.e., the amount of task sets that are deemed schedulable by the test divided by the total amount of task sets, over the total utilization of task sets.

The schedulability tests are categorized using two criteria:

- Self-suspension behavior
  (segmented self-suspension, dynamic self-suspension, hybrid self-suspension)
- Deadline constraint
  (implicit deadline, constrained deadline, arbitrary deadline)

However, as the tests are not always compatible with constrained deadlines or arbitrary deadlines, the framework only considers the implicit-deadline model for generating automatic comparisons. Furthermore, since there is no standardized way of constructing constrained-deadline and arbitrary-deadline task sets yet in the literature, the users are recommended to adjust the program for evaluating constrained-deadline or arbitrary-deadline task systems if necessary.

We further enhance the framework by integrating the state of the arts we are aware of. By now baseline approaches and schedulability tests proposed by 18 papers are realized in the framework.[8] In addition, we provide more features to ease the use, e.g., Python 3 support, an improved Graphical User Interface (GUI), multiprocessing, and evaluation of external task sets. The enhanced framework also integrates the Gurobi optimization solver [Gur21] to supply tests which use linear programming. The framework is published on Github [TU 21a] and there is an ongoing effort to keep the framework up to date. The results presented in this section appeared in the Work-in-Progress track of RTSS 2021 [Gün+21b].

5.1.4.1 Framework Description

In this section, we introduce the features of the current framework in detail. In Figure 5.21, the GUI of our framework is presented. Besides the general setting of the framework (①), the framework consists of three components (②, ③ and ④):

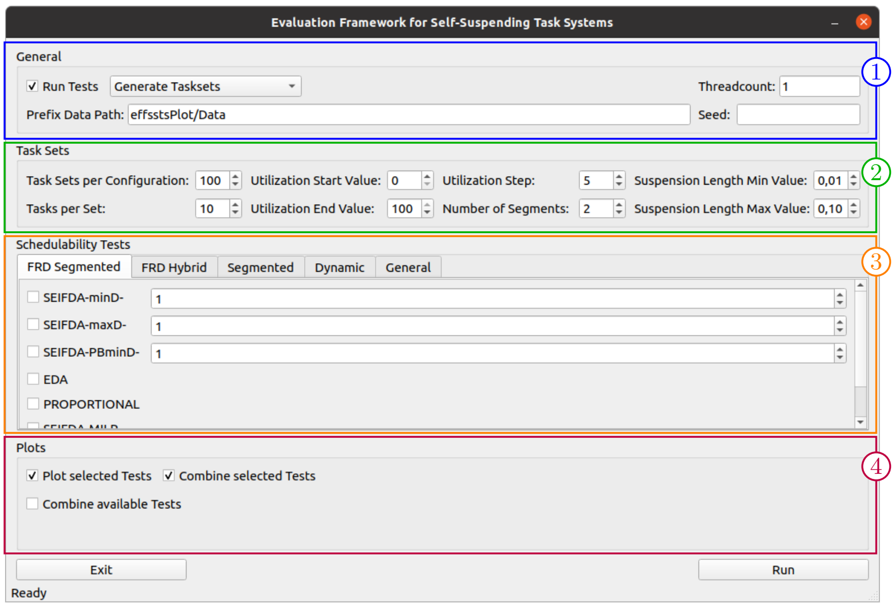
(i) A *task set generator* to provide periodic/sporadic task sets for evaluation.

(ii) The *schedulability test evaluator* applies a choice of schedulability tests to the generated task sets.

(iii) The *plotting tool configurator* illustrates the performance of the schedulability tests.

Please note that the following description is explained in a sequential manner to demonstrate the usage from beginning to end. However, these components can also be used independently for different scenarios (see Section 5.1.4.2).

[8]This includes the 17 papers that were available at the time of publication in 2021, plus one additional test from Casini et al. [Cas+18] that has been integrated after publication.


Evaluation Framework for Self-Suspending Task Systems

General
[✓] Run Tests Generate Tasksets Threadcount: 1
Prefix Data Path: effsstsPlot/Data Seed:

Task Sets
Task Sets per Configuration: 100 Utilization Start Value: 0 Utilization Step: 5 Suspension Length Min Value: 0,01
Tasks per Set: 10 Utilization End Value: 100 Number of Segments: 2 Suspension Length Max Value: 0,10

Schedulability Tests
FRD Segmented | FRD Hybrid | Segmented | Dynamic | General
[ ] SEIFDA-minD- 1
[ ] SEIFDA-maxD- 1
[ ] SEIFDA-PBminD- 1
[ ] EDA
[ ] PROPORTIONAL

Plots
[✓] Plot selected Tests [✓] Combine selected Tests
[ ] Combine available Tests

Exit Run
Ready

Figure 5.21.: Overview of the enhanced framework.

General Setting In part ①, the user selects the option to either generate task sets in the framework or to load an external file that contains the task set data. The task sets can be reused by saving them after evaluation and reloading them at a later point in time. Additionally, the number of threads used to evaluate the task sets in-parallel can be set. The generated or loaded task sets are evenly distributed to the selected number of threads and evaluated in-parallel. The evaluation of a large number of task sets can therefore be done in a fraction of the time compared to the case without multiprocessing. Furthermore, the seed for the random number generator can be set, so that the results of a previous run can be reproduced.

Task Set Configurations The parameters of synthesized task sets can be configured in part ②. The task sets are generated using the integrated UUnifast algorithm [BB05] and can be configured by using the parameters that are provided in the framework. In particular, these parameters are: the number of task sets, the number of tasks per set, the number of suspension segments, the utilization values to evaluate, and the suspension parameters for the tasks. At first, the utilization of each task is drawn randomly according to UUnifast [BB05]. Afterwards, the period and deadline are drawn from a log-uniform distribution, where the default is over two orders of magnitude, i.e., [1,100]. Depending on the considered suspension models, the suspension time of each task is drawn accordingly[9].

[9]Details of implementation can be found in the readme file on [TU 21a].

Year	Papers	Methods	Suspension	Deadline	New
	Baseline approaches	Suspension as computation, i.e., SCEDF, SCRM, SUSPOBL	—	(up to tests)	
2000	Liu [Liu00, pp. 164–165]	SUSPBLOCK (proof in [CNH16])	Dynamic	Implicit	✓
2014	Liu et al. [Liu+14]	PROPORTIONAL	Segmented	Implicit	
2014	Chen and Liu [CL14]	EDA	Segmented	Implicit	
2014	Liu and Chen [LC14]	Idv-Burst-RM	Dynamic	Implicit	✓
2015	Liu and Anderson et al. [LA15][1]	WLAEDF	Dynamic	Implicit	✓
2015	Huang et al. [Hua+15]	PASS-OPA	Dynamic	Constrained	
2015	Nelissen et al. [Nel+17; Nel+15], Biondi et al. [Bio+16][2]	MILP-ReleaseJitter	Segmented	Constrained	✓
2016	Dong and Liu [DL16][1]	UDLEDF	Dynamic	Implicit	✓
2016	Mohaqeqi et al. [MEY16]	SRSR	Segmented	Implicit	✓
2016	von der Brüggen et al. [Brü+16]	SEIFDA-minD, SEIFDA-maxD, SEIFDA-PBminD, SEIFDA-MILP, NC	Segmented	Implicit	
2016	Peng and Fisher [PF17]	GMFPA	Segmented	Arbitrary	✓
2016	Huang and Chen [HC16]	EDAGMF-OPA	Segmented	Constrained	✓
2016	Chen et al. [CNH16]	UNIFRAMEWORK, SUSPOBL, SUSPJIT, SUSPBLOCK	Dynamic	Constrained	✓
2017	von der Brüggen et al. [BHC17]	Oblivious-IUB, Clairvoyant-SSSD, Oblivious-MP, Clairvoyant-PDAB	Hybrid	Implicit	
2018	Casini et al. [Cas+18]	Casini18	Segmented	Constrained	✓[4]
2018	Schönberger et al. [Sch+18b][3]	SCAIR-RM, SCAIR-OPA	Dynamic	Constrained	
2019	Yalcinkaya et al. [YNB19]	UPPAAL	Dynamic	Implicit	✓
2020	Günzel et al. [GBC20]	RSS, RTEDF	Dynamic	Implicit	✓

[1] Uniprocessor version is presented in [GBC20].
[2] Originally proposed in Section VI in [Nel+15], revised in [Nel+17], implemented in [Bio+16] with open source.
[3] Originally proposed in [HC15] and revised in [Sch+18b].
[4] This work has been added after publication in the Work-in-Progress track of RTSS 2021 [Gün+21b].

Table 5.2.: Schedulability tests included in the framework.

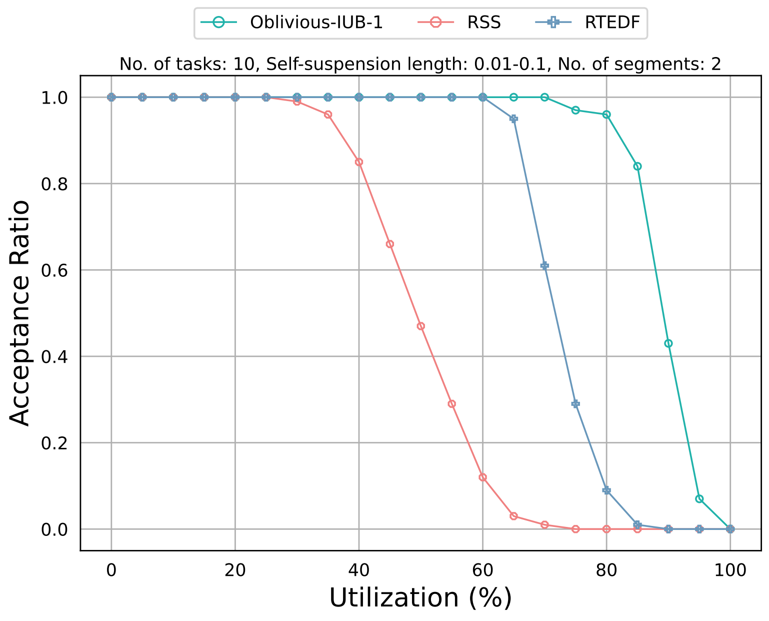


Figure 5.22.: Exemplary plot provided by the framework schedulability analysis with three tests and 100 tasks per set.

Test Selection After configuring the parameters, the user can select a set of schedulability tests in part ③ to evaluate the task sets. All currently available schedulability tests are presented in Table 5.2. Please note that some schedulability tests can set custom parameters, e.g., SEIFDA-based approaches [Brü+16]. After pressing the `Run` button at the bottom, each generated task set will be tested by the selected schedulability tests iteratively. If a tested task set is deemed schedulable, the considered test will return *True*. After checking all task sets, the framework counts the number of *True*s, and calculates the percentage of task sets that are deemed schedulable by each test. We note that the framework can also support tests which require linear programming solvers, e.g., MILP-ReleaseJitter, by using the Gurobi optimization solver [Gur21].

Plotting Format The results of the schedulability analysis can be plotted using an integrated tool of the framework. The user can determine the format of the plots in part ④, either plot each selected test by itself, or combine the plots of all selected tests, so that they can be directly compared. A resulting exemplary plot is shown in Figure 5.22. The framework relies on a plotting library, namely `matplotlib` in Python. The raw data of the evaluation can also be post-processed by other plotting schemes.

5.1.4.2 Research Applications

The framework can be applied in different research scenarios. In the following, we detail three scenarios, in which the framework can be used:

- **Scenario 1:** Evaluate and compare the performance of an own schedulability test.
- **Scenario 2:** Analyze own task sets with already known schedulability tests.
- **Scenario 3:** Utilize already known schedulability tests in an own python program.

Scenario 1 Additional schedulability tests can be integrated into the framework. This involves two steps: First, the analysis needs to be implemented in Python. Alternatively for C++-algorithms, a pre-built binary can also be executed from a Python script. Second, the additional test needs to be integrated into the framework infrastructure. The test has to be integrated into the testing component of the framework, and it has to be made available in part ③ of the GUI. In particular, several things should be added into `effsstsMain.py`, e.g., add an entry to the GUI and an additional case to the `switchTest` method. After integration, it is possible to evaluate the test using the provided task generator and plotting tool.

Scenario 2 Custom task sets can be loaded into the framework to evaluate them using the provided schedulability tests. For this scenario the *task set generator* component (②) is replaced, but the other two components, i.e., ③ and ④ are utilized. In this regard, we additionally provide a script `SaveTaskSet` to convert task sets into the required format. This script will guide the user to create a serialized file for custom task sets, so that they can be loaded into the framework properly. To this end, the general setup ① also has to be configured accordingly to load the own task sets, i.e., ticking *Load Taskset* and enter the name of the saved task set. In fact, this feature is particularly useful if many task sets need to be evaluated all at once.

Scenario 3 The realized schedulability tests in the framework can also be imported into other programs, which can be done by a standard import statement in Python. The names for the schedulability tests are identical to the ones in part ③ of the GUI. After the import, it is possible to evaluate custom external task sets, as long as the required arguments are given correctly. Please note that the return value of every schedulability test is always in a boolean expression.

5.2 Timing Anomalies

While for tasks without self-suspending behavior the WCRT is achieved when all jobs execute for their WCET, self-suspending behavior can lead to *timing anomalies*, i.e., the reduction of execution or suspension time of some jobs enlarges the response time of another job. The reason for timing anomalies is that by reducing the actual execution/suspension time, succeeding segments may become ready for execution earlier and interfere with segments from other tasks. Such a timing anomaly involving only two jobs, J_1 and J_2, with one self-suspension segment is depicted in Figure 3.1 of Section 3.1. In that example, the second segment of J_1 becomes ready earlier and interferes with J_2.

Timing anomalies can affect the feasibility of a task set. To be more specific, it is possible that a task set, determined to be schedulable based on the WCET and the maximum suspension time of the segments, can still have deadline violations during runtime due to timing anomalies (cf. [Sch+18b]). Therefore, existing schedulability analyses for self-suspending tasks account for timing anomalies by over-approximation. To avoid that analytical pessimism, a treatment for eliminating the timing anomalies is essential. To that end, different mechanisms to reduce the impact of timing anomalies have been developed in the literature:

- *Period enforcer* [Raj91] intends to apply a runtime rule so that *"it forces tasks to behave like ideal periodic tasks from the scheduling point of view with no associated scheduling penalties."*, as summarized in Section 4.3.1 in the survey paper [Che+19b]. However, Chen and Brandenburg [CB17] show that *"period enforcement [Raj91] is not strictly superior (compared to the base case without enforcement) as it can cause deadline misses in self-suspending task sets that are schedulable without enforcement."*

- *Release guard* [SL96] and *release enforcement* [HC16] enforce the computation segments to be released with an offset, which is either set statically [HC16] or dynamically increased [SL96]. The approach is summarized in Section 4.3.2 in the survey paper [Che+19b].

- *Slack enforcement* [LR10] creates execution enforcement by utilizing the available *slack*. In Section 4.1, we provide counterexamples, indicating that slack enforcement may provoke deadline misses and does not guarantee the same WCRT as without slack enforcement.

The period enforcer and slack enforcement pursue the ultimate goal to completely *ignore the self-suspension behavior* of higher-priority tasks, which would consequently avoid timing anomalies. However, none of them achieves this ultimate goal, as shown by Chen and Brandenburg [CB17] and in Section 4.1, and therefore they are not able to guarantee that timing anomalies are eliminated. The release guard and release enforcement do not intend to ignore the self-suspension behavior of higher-priority tasks or eliminate timing anomalies, but only aim for *easier schedulability analyses*. This is achieved by decoupling the segment release time from the finishing time of earlier segments. Although not explicitly stated, such treatments avoid timing anomalies as a side effect. However, the treatments do not sustain the WCRT of a task, meaning that the WCRT of a task with treatment may be much larger than the WCRT without treatment.

In the following, we propose two treatments, namely *segment release time enforcement* in Section 5.2.2 and *segment priority modification* in Section 5.2.3, and prove that both treatments eliminate timing anomalies without negative impact on the WCRT. To that end, we focus on preemptive uniprocessor scheduling of segmented self-suspending tasks with a fixed arrival pattern, i.e.,

$$\tau \in \text{FixRel}((r_{\tau(j)})_{j\in\mathbb{N}}) \cap \text{SegSS}(C_\tau^1, S_\tau^1, C_\tau^2, \dots, S_\tau^{m_\tau-1}, C_\tau^{m_\tau}) \tag{5.114}$$

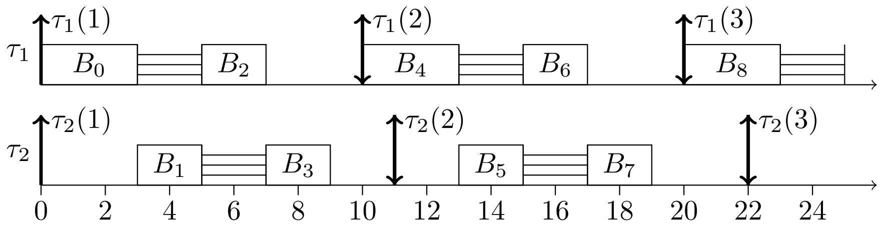


Figure 5.23.: Example of a nominal schedule of a segmented self-suspension task set.

for all $\tau \in \mathbb{T}$. We assume that any Segment-level Fixed-Priority (S-FP) scheduling algorithm $\mathcal{A}$ is applied. That is, there is a total priority ordering $<_{prio}$ of all computation segments $\mathbb{B} := \sqcup_{\tau(j)\in\mathbb{J}}\mathbb{B}_{\tau(j)}$ independent of the system evolution $\omega \in \Omega$.[10] We write $\mathrm{B}_1 <_{prio} B_2$ if computation segment B_2 has higher priority than B_1. Any T-FP and J-FP algorithm that has fixed priorities under a fixed arrival pattern falls under this category. This includes for example EDF, EL, DM and Rate Monotonic (RM).

Our solutions are based on the following two steps:

- **Step 1**: A *nominal schedule* $[\mathcal{S}_0]$ is constructed and recorded offline purely based on the WCET and the maximum suspension time. More specifically, $[\mathcal{S}_0] = \mathcal{A}(\omega_0)$ is the schedule generated by the system evolution

$$\omega_0 = \Big(\big(r_{\tau(j)}\big)_{\tau(j)\in\mathbb{J}}, \big((1 \to \cdots \to m_\tau), (S_\tau^i)_{(i,i+1)}, (C_\tau^i)_i\big)\Big). \tag{5.115}$$

  Figure 5.23 demonstrates the nominal schedule $[\mathcal{S}_0]$ of two tasks:
  - $\tau_1 \in \mathrm{PerOff}(10,0) \cap \mathrm{SegSS}(3,2,2)$
  - $\tau_2 \in \mathrm{PerOff}(11,0) \cap \mathrm{SegSS}(2,2,2)$

- **Step 2**: The *online schedule* $[\mathcal{S}_1]$ refers to the nominal schedule to make scheduling decisions for an arbitrary system evolution $\omega_1 \in \Omega$ *without any risk of timing anomalies*. As a result, the schedulability of the task set is guaranteed as long as feasibility of the nominal schedule is guaranteed.

From a schedulability standpoint, avoiding timing anomalies is desirable. That is, instead of considering all possible schedules, schedulability can be checked by simulating the nominal schedule. Specifically for synchronous, periodic tasks with constrained deadlines, this can be achieved by simulating the schedule over one hyperperiod.

Remark 5.68 (Enforcement for periodic tasks.)**.** *For periodic, synchronous tasks with constrained deadlines, the nominal schedule $[\mathcal{S}_0]$ repeats every hyperperiod if it no deadline miss occurs in the first hyperperiod. Therefore, it is sufficient to schedule only one hyperperiod and apply the mechanism for one hyperperiod at a time. Moreover, the simulation of one hyperperiod for Step 1 directly serves as an* exact *schedulability test for*

[10]Please note that the index ω is omitted in $\mathbb{B}_{\tau(j)}$, because by the definition of segmented self-suspending tasks, $\mathbb{B}^\omega_{\tau(j)} \simeq (1 \to \cdots \to m_\tau)$ for all ω.

scheduling under the treatments: There are no deadline misses in the schedule obtained under the treatment if and only if there are no deadline misses in the first hyperperiod of the nominal schedule $[\mathcal{S}_0]$.

In this section, we focus on eliminating timing anomalies without negative impact on the WCRT analysis when scheduling periodic tasks with segmented self-suspension behavior. To that end, we propose two treatments, *segment release time enforcement* and *segment priority modification*, and prove that both treatments eliminate timing anomalies. The contributions are as follows:

- In Section 5.2.1, we formulate the terminology and observations that serve as backbone of the two treatments.
- In Section 5.2.2, we propose the treatment *segment release time enforcement*. With *segment release time enforcement*, a task segment is enforced to start its execution **no earlier** than its release time in the nominal schedule.
- In Section 5.2.3, we propose the treatment *segment priority modification*. For *segment priority modification*, the priorities of the segments are adjusted based on their nominal finishing times. The rationale behind *segment priority modification* is that a segment with an earlier nominal finishing time should not be interfered by segments with later nominal finishing times. We implement the S-FP scheduling mechanism in RTEMS [RTE23], an open-source Real-Time Operating System (RTOS), so that the treatment *segment priority modification* can be performed in RTEMS.
- In Section 5.2.4, our empirical results demonstrate that the proposed treatments achieve higher acceptance ratios in terms of schedulability compared to state-of-the-art over-approximations by considering segmented self-suspension periodic tasks under different task set configurations.

The results presented in this section appeared in RTAS 2023 [Lin+23].

5.2.1 Terminology and Observations

An online S-FP scheduling algorithm $\mathcal{A}$ chooses the segment with the highest priority among all segments in the ready queue for execution. A segment is *ready* and inserted into the ready queue according to Definition 5.69. If the chosen segment has a higher priority than the one currently being executed, the current executing segment is preempted and moved to the ready queue. We make the following definitions for any arbitrary schedule $[\mathcal{S}]$ with system evolution ω.

Definition 5.69 (Ready). *A computation segment* $B \in \mathbb{B}$ *is* ready *at time* $t \in \mathbb{R}$ *in a schedule* $[\mathcal{S}]$ *if:*

(i) there is remaining workload of B *to be executed at time* t *in* $[\mathcal{S}]$*;*

(ii) B is or has been released *at time t in $[\mathcal{S}]$.*

Definition 5.70 (Release)**.** *Let $J \in \mathbb{J}$ be a job of task τ consisting of computation segments $(B_1 \to \cdots \to B_{m_\tau})$. The first segment B_1 is released when the job J is released, i.e., at time r_J. A subsequent segment B_i is released as soon as the previous segment B_{i-1} finishes and the delay runs out, i.e., at time $f^{[\mathcal{S}],\omega}_{B_{i-1}} + \delta^{\omega}_{(B_{i-1},B_i)}$. In general, we denote by $r^{[\mathcal{S}],\omega}_B$ the release time of a segment $B \in \mathbb{B}$ in the schedule $[\mathcal{S}]$.*

With this definition of a segment being *ready*, the standard S-FP preemptive scheduler is *work-conserving* on the segment-level, in the sense that whenever there are ready segments, the segment of the highest priority task is executed. This leads to the following observation for the nominal schedule $[\mathcal{S}_0]$.

Observation 5.71. *Let $B \in \mathbb{B}$ be a segment. In $[\mathcal{S}_0]$, the segment B finishes at the lowest $t \in \mathbb{R}$ such that*

$$t \geq r^{[\mathcal{S}_0],\omega_0}_B + W^{[\mathcal{S}_0]}_B(r^{[\mathcal{S}_0],\omega_0}_B, t) + c^{\omega_0}_B, \tag{5.116}$$

where $W^{[\mathcal{S}_0]}_B(r^{[\mathcal{S}_0],\omega_0}_B, t)$ is the total amount of time that higher-priority segments are executed during the interval $[r^{[\mathcal{S}_0],\omega_0}_B, t)$ in $[\mathcal{S}_0]$, i.e.,

$$W^{[\mathcal{S}_0]}_B(a,b) := \sum_{\mathrm{B}' >_{prio} B \in \mathbb{B}} \mathrm{exec}^{[\mathcal{S}_0]}_{B'}(a,b) \tag{5.117}$$

for $a \leq b \in \mathbb{R}$.

At all times during the interval $[r^{[\mathcal{S}_0],\omega_0}_B, s^{[\mathcal{S}_0],\omega_0}_B)$, segments with higher priorities than B are executed. Hence, $W^{[\mathcal{S}_0]}_B(r^{[\mathcal{S}_0],\omega_0}_B, t) = (s^{[\mathcal{S}_0],\omega_0}_B - r^{[\mathcal{S}_0],\omega_0}_B) + W^{[\mathcal{S}_0]}_B(s^{[\mathcal{S}_0],\omega_0}_B, t)$ holds, and the observation can be reformulated as follows.

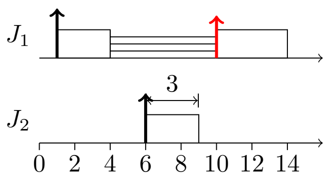
Observation 5.72. *Let $B \in \mathbb{B}$ be a segment. In $[\mathcal{S}_0]$, the segment B finishes at the lowest $t \in \mathbb{R}$ such that*

$$t \geq s^{[\mathcal{S}_0],\omega_0}_B + W^{[\mathcal{S}_0]}_B(s^{[\mathcal{S}_0],\omega_0}_B, t) + c^{\omega_0}_B. \tag{5.118}$$

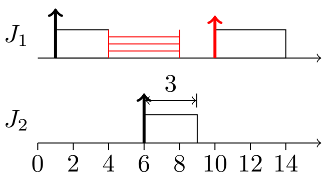
In the following two subsections, we utilize these observations to formulate treatments to prevent timing anomalies while preserving the WCRT of jobs in the nominal schedule.

5.2.2 Segment Release Time Enforcement

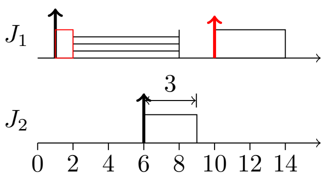
Timing anomalies occur if a computation segment experiences higher interference in the online schedule $[\mathcal{S}_1]$ than in the nominal schedule $[\mathcal{S}_0]$. This may happen if a higher-priority segment starts its execution before its release time in $[\mathcal{S}_0]$ due to the early completion and/or reduced suspension of a previous segment from the same job. This higher-priority segment blocks or preempts segments from lower-priority jobs, therefore



(a) Nominal schedule.



(b) Reduced suspension.



(c) Reduced execution.

Figure 5.24.: Example of the treatment *segment release time enforcement*. A computation segment cannot start before its nominal release time (red arrow) even if the processor idles.

increasing their response times. To eliminate timing anomalies, one method is to enforce the release time of the segments, such that all the segments start no earlier than their release time in the nominal schedule.

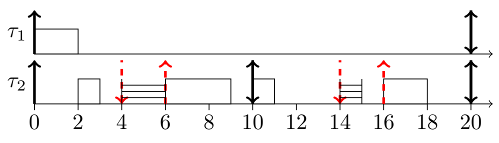
Our first treatment, *segment release time enforcement*, works as follows. If a segment B is ready at time t according to Definition 5.69 but the release time in the nominal schedule $r_B^{[\mathcal{S}_0],\omega_0}$ is not reached, then the segment execution is delayed further until $r_B^{[\mathcal{S}_0],\omega_0}$. Formally, we *redefine the term released* for the online schedule under segment release time enforcement as follows:

Definition 5.73. *Let $J \in \mathbb{J}$ be a job of task τ consisting of segments $(B_1 \to \dots \to B_{m_\tau})$. In the online schedule $[\mathcal{S}_1]$ under* segment release time enforcement*,*

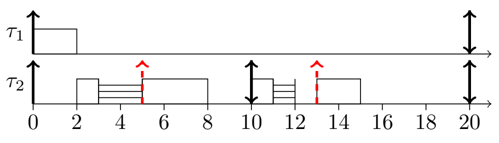
- *the first segment B_1 is* released *when the job J is released.*
- *a subsequent segment B_i is* released *as soon as the previous segment B_{i-1} finishes and the suspension time is consumed, and the release time in the nominal schedule $r_{B_i}^{[\mathcal{S}_0],\omega_0}$ is reached. That is, B_i is released at time $\max(f_{B_{i-1}}^{[\mathcal{S}_1],\omega_1} + \delta_{(B_{i-1},B_i)}^{\omega_1}, r_{B_i}^{[\mathcal{S}_0],\omega_0})$.*

We denote by $[\mathcal{S}_1^{rt}]$ the online schedule for system evolution ω_1 obtained by segment release enforcement, as described above. Hence, $r_B^{[\mathcal{S}_1^{rt}],\omega_1}$ denotes the release time of a segment $B \in \mathbb{B}$ in the online schedule under that treatment. Figure 5.24 demonstrates the new execution states with *segment release time enforcement*. In practice, we can enforce the segment release time by maintaining an additional queue. All the segments which are supposed to be inserted into the ready queue are inserted to the new queue instead. Only the segments that have a nominal starting time no lesser than the current time t can be moved to the ready queue.

We note that our proposed enforcement is different from the *release guard* [SL96] and *release enforcement* [HC16]. In release guard and release enforcement, the offset of segments is a task-specific value, whereas in our proposed method, the release time of each segment is determined for each job individually according to the nominal schedule. Figure 5.25 provides a visual comparison between the online schedules under two different enforcement methods: *release enforcement* as described in [HC16], and our proposed



(a) *Release enforcement* proposed in [HC16].



(b) *Segment release time enforcement.*

Figure 5.25.: The online schedules of task set $\mathbb{T} = \{\tau_1, \tau_2\}$ with under different release time enforcement policies. The red upward (downward) arrows indicate the enforced release time (deadline) of each computation segment. Note that the inter-arrival time of each segment is fixed in [HC16].

approach, *segment release time enforcement.* In the figure, we consider the implicit-deadline task set $\mathbb{T} = \{\tau_1, \tau_2\}$ with:

- $\tau_1 \in \mathrm{PerOff}(20, 0) \cap \mathrm{SegSS}(1)$
- $\tau_2 \in \mathrm{PerOff}(10, 0) \cap \mathrm{SegSS}(2, 2, 3)$

As in Definition 5.69, a segment is ready to be executed under segment release time enforcement if there is remaining workload to be executed and if the segment is released. With this definition of being ready, based on the definition of being released in Definition 5.73, the S-FP preemptive scheduler under segment release time enforcement executes segments that are ready in a work-conserving manner, leading to the following observation.

Observation 5.74. *Let $B \in \mathbb{B}$ be a segment. Then the segment B finishes in the online schedule with segment release time enforcement, i.e., in $[\mathcal{S}_1^{rt}]$ at the lowest $t \in \mathbb{R}$ with*

$$t \geq r_B^{[\mathcal{S}_1^{rt}],\omega_1} + W_B^{[\mathcal{S}_1^{rt}]}(r_B^{[\mathcal{S}_1^{rt}],\omega_1}, t) + c_B^{\omega_1}, \tag{5.119}$$

where $W_B^{[\mathcal{S}_1^{rt}]}(r_B^{[\mathcal{S}_1^{rt}],\omega_1}, t)$ is the amount of time that higher-priority segments are executed during the interval $[r_B^{[\mathcal{S}_1^{rt}],\omega_1}, t)$ in the schedule $[\mathcal{S}_1^{rt}]$, i.e.,

$$W_B^{[\mathcal{S}_1^{rt}]}(r_B^{[\mathcal{S}_1^{rt}],\omega_1}, t) := \sum_{B' >_{prio} B \in \mathbb{B}} \mathrm{exec}_B^{[\mathcal{S}_1^{rt}]}(r_B^{[\mathcal{S}_1^{rt}],\omega_1}, t). \tag{5.120}$$

We now prove that with the treatment *segment release time enforcement*, timing anomalies cannot occur in the online schedule $[\mathcal{S}_1^{rt}]$. Intuitively, if the release time of a segment $B \in \mathbb{B}$ is fixed, the segment finishing time can only become larger if $W_B^{[\mathcal{S}_1^{rt}]}$ is larger than $W_B^{[\mathcal{S}_0]}$. However, since no segment release can be moved forward under the treatment, and if all previous segments finish no later than their finishing time in the nominal schedule, $W_B^{[\mathcal{S}_1^{rt}]}$ cannot be larger than $W_B^{[\mathcal{S}_0]}$. To make this proof formal, we start by rewriting $W_B^{[\mathcal{S}_1^{rt}]}$ and $W_B^{[\mathcal{S}_0]}$ using the following lemma.

Lemma 5.75. *Let $B \in \mathbb{B}$ be a segment. With segment release time enforcement, equations*

$$W_B^{[\mathcal{S}_0]}(a,b) = \lambda\left([a,b) \cap \bigcup_{B' >_{prio} B \in \mathbb{B}} [r_{B'}^{[\mathcal{S}_0],\omega_0}, f_{B'}^{[\mathcal{S}_0],\omega_0})\right) \tag{5.121}$$

$$W_B^{[\mathcal{S}_1^{rt}]}(a,b) = \lambda\left([a,b) \cap \bigcup_{B' >_{prio} B \in \mathbb{B}} [r_{B'}^{[\mathcal{S}_1^{rt}],\omega_1}, f_{B'}^{[\mathcal{S}_1^{rt}],\omega_1})\right) \tag{5.122}$$

hold for all $a \leq b \in \mathbb{R}$.

Proof. In the following, we provide the proof for the nominal schedule $[\mathcal{S}_0]$ (Equation (5.121)). The proof for the online schedule $[\mathcal{S}_1^{rt}]$ (Equation (5.122)) is analogous.

By definition, we have

$$W_B^{[\mathcal{S}_0]}(a,b) = \sum_{B' >_{prio} B} \text{exec}_{B'}^{[\mathcal{S}_0]}(a,b) = \sum_{B' >_{prio} B} \lambda\left((\mathcal{S}_0)^{-1}(\{B'\}) \cap (a,b)\right). \tag{5.123}$$

Moreover, since there are only finitely many B' with $B' >_{prio} B$, and $(\mathcal{S}_0)^{-1}(\{B'\})$ is disjoint for different B', we can reformulate this equation as

$$W_B^{[\mathcal{S}_0]}(a,b) = \lambda\left(\bigcup_{B' >_{prio} B} (\mathcal{S}_0)^{-1}(\{B'\}) \cap (a,b)\right). \tag{5.124}$$

Hence, it is sufficient to show that $\bigcup_{B' >_{prio} B} (\mathcal{S}_0)^{-1}(\{B'\})$ and $\bigcup_{B' >_{prio} B} [r_{B'}^{[\mathcal{S}_0],\omega_0}, f_{B'}^{[\mathcal{S}_0],\omega_0})$ coincide almost everywhere, i.e., up to a set of measure 0. We do this in two steps.

Step 1: $\lambda\left(\bigcup_{B' >_{prio} B} (\mathcal{S}_0)^{-1}(\{B'\}) \setminus \bigcup_{B' >_{prio} B} [r_{B'}^{[\mathcal{S}_0],\omega_0}, f_{B'}^{[\mathcal{S}_0],\omega_0})\right) \overset{!}{=} 0$. To achieve this, first we over-approximate the term by

$$\leq \lambda\left(\bigcup_{B' >_{prio} B} \left((\mathcal{S}_0)^{-1}(\{B'\}) \setminus [r_{B'}^{[\mathcal{S}_0],\omega_0}, f_{B'}^{[\mathcal{S}_0],\omega_0})\right)\right) \tag{5.125}$$

$$\leq \sum_{B' >_{prio} B} \lambda\left((\mathcal{S}_0)^{-1}(\{B'\}) \setminus [r_{B'}^{[\mathcal{S}_0],\omega_0}, f_{B'}^{[\mathcal{S}_0],\omega_0})\right). \tag{5.126}$$

By definition, a block cannot be executed before it is released or after it finished. Therefore, $\lambda\left((\mathcal{S}_0)^{-1}(\{B'\}) \setminus [r_{B'}^{[\mathcal{S}_0],\omega_0}, f_{B'}^{[\mathcal{S}_0],\omega_0})\right) = 0$.

Step 2: $\lambda\left(\bigcup_{B' >_{prio} B} [r_{B'}^{[\mathcal{S}_0],\omega_0}, f_{B'}^{[\mathcal{S}_0],\omega_0}) \setminus \bigcup_{B' >_{prio} B} (\mathcal{S}_0)^{-1}(\{B'\})\right) \overset{!}{=} 0$. An upper bound for this term can be formulated as follows:

$$\leq \lambda\left(\bigcup_{B' >_{prio} B} \left([r_{B'}^{[\mathcal{S}_0],\omega_0}, f_{B'}^{[\mathcal{S}_0],\omega_0}) \setminus \bigcup_{B'' \geq_{prio} B'} (\mathcal{S}_0)^{-1}(\{B''\})\right)\right) \tag{5.127}$$

$$\leq \sum_{B' >_{prio} B} \lambda\left([r_{B'}^{[\mathcal{S}_0],\omega_0}, f_{B'}^{[\mathcal{S}_0],\omega_0}) \setminus \bigcup_{B'' \geq_{prio} B'} (\mathcal{S}_0)^{-1}(\{B''\})\right) \tag{5.128}$$

By the work-conserving property of the schedule, at any time in $[r_{B'}^{[\mathcal{S}_O],\omega_O}, f_{B'}^{[\mathcal{S}_O],\omega_O})$ the job with the highest priority is executed. Hence,

$$\lambda\left([r_{B'}^{[\mathcal{S}_O],\omega_O}, f_{B'}^{[\mathcal{S}_O],\omega_O}) \setminus \bigcup_{B'' \geq_{prio} B'} (\mathcal{S}_O)^{-1}(\{B''\})\right) = 0. \tag{5.129}$$

□

With the previous lemma, we reformulate the execution time of higher-priority segments $W_B^{[\mathcal{S}_O]}(r_B^{[\mathcal{S}_O],\omega_O}, t)$ from Equation (5.117) as

$$W_B^{[\mathcal{S}_O]}(r_B^{[\mathcal{S}_O],\omega_O}, t) = \lambda\left([r_B^{[\mathcal{S}_O],\omega_O}, t) \cap \bigcup_{B' >_{prio} B \in \mathbb{B}} [r_{B'}^{[\mathcal{S}_O],\omega_O}, f_{B'}^{[\mathcal{S}_O],\omega_O})\right), \tag{5.130}$$

and we reformulate $W_B^{[\mathcal{S}_1^{rt}]}(r_B^{[\mathcal{S}_1^{rt}],\omega_1}, t)$ from Equation (5.120) as

$$W_B^{[\mathcal{S}_1^{rt}]}(r_B^{[\mathcal{S}_1^{rt}],\omega_1}, t) = \lambda\left([r_B^{[\mathcal{S}_1^{rt}],\omega_1}, t) \cap \bigcup_{B' >_{prio} B \in \mathbb{B}} [r_{B'}^{[\mathcal{S}_1^{rt}],\omega_1}, f_{B'}^{[\mathcal{S}_1^{rt}],\omega_1})\right). \tag{5.131}$$

This allows us to prove the following theorem which states that timing anomalies cannot occur under segment release time enforcement.

Theorem 5.76. *The finishing time of each segment in the online schedule $[\mathcal{S}_1^{rt}]$ with* segment release time enforcement *is no larger than the finishing time in the nominal schedule $[\mathcal{S}_O]$, i.e., $f_B^{[\mathcal{S}_O],\omega_O} \geq f_B^{[\mathcal{S}_1^{rt}],\omega_1}$ for all $B \in \mathbb{B}$.*

Proof. Let *Seg* $= (B_0, B_1, \dots)$ denote the list of all segments $\mathbb{B}$ ordered by their finishing time in the nominal schedule, i.e., $f_{B_0}^{[\mathcal{S}_O],\omega_O} < f_{B_1}^{[\mathcal{S}_O],\omega_O} < \dots$ holds. We consider the online schedule $[\mathcal{S}_1^{rt}]$ obtained under segment release time enforcement. By induction over the segments in *Seg* we show that for each B_n, $n = 0, 1, \dots$:

(i) B_n is released at the same time in the online and in the nominal schedule, i.e., $r_{B_n}^{[\mathcal{S}_1^{rt}],\omega_1} = r_{B_n}^{[\mathcal{S}_O],\omega_O}$.

(ii) The finishing time of B_n in the online schedule is no later than in the nominal schedule, i.e., $f_{B_n}^{[\mathcal{S}_1^{rt}],\omega_1} \leq f_{B_n}^{[\mathcal{S}_O],\omega_O}$.

Base case (n=0): Since B_0 has the earliest finishing time in the nominal schedule, it must be the first segment of some job. Therefore, by definition $r_{B_0}^{[\mathcal{S}_1^{rt}],\omega_1} = r_{B_0}^{[\mathcal{S}_O],\omega_O}$, which implies that (i) holds.

We prove (ii) by contradiction. By Observation 5.74, we know that $f_{B_0}^{[\mathcal{S}_1^{rt}],\omega_1}$ is the smallest $t \in \mathbb{R}$ such that $t \geq r_{B_0}^{[\mathcal{S}_1^{rt}],\omega_1} + W_{B_0}^{[\mathcal{S}_1^{rt}]}(r_{B_0}^{[\mathcal{S}_1^{rt}],\omega_1}, t) + c_{B_0}^{\omega_1} \overset{(i)}{=} r_{B_0}^{[\mathcal{S}_O],\omega_O} + W_{B_0}^{[\mathcal{S}_1^{rt}]}(r_{B_0}^{[\mathcal{S}_O],\omega_O}, t) + c_{B_0}^{\omega_1}$. We assume that $f_{B_0}^{[\mathcal{S}_O],\omega_O} < f_{B_0}^{[\mathcal{S}_1^{rt}],\omega_1}$ for the contradiction. Then we have

$$f_{B_0}^{[\mathcal{S}_O],\omega_O} < r_{B_0}^{[\mathcal{S}_O],\omega_O} + W_{B_0}^{[\mathcal{S}_1^{rt}]}(r_{B_0}^{[\mathcal{S}_O],\omega_O}, f_{B_0}^{[\mathcal{S}_O],\omega_O}) + c_{B_0}^{\omega_1}. \tag{5.132}$$

For all segments $B \in \mathbb{B}$ we have $r_B^{[\mathcal{S}_1^{rt}],\omega_1} \geq r_B^{[\mathcal{S}_0],\omega_0}$ by the enforcement mechanism. Moreover, since B_0 is the first segment in *Seg*, it has the lowest finishing time in $[\mathcal{S}_0]$, i.e., $f_{B_0}^{[\mathcal{S}_0],\omega_0} \leq f_B^{[\mathcal{S}_0],\omega_0}$. Hence,

$$\begin{aligned}
[r_B^{[\mathcal{S}_1^{rt}],\omega_1}, f_B^{[\mathcal{S}_1^{rt}],\omega_1}) \cap [r_{B_0}^{[\mathcal{S}_0],\omega_0}, f_{B_0}^{[\mathcal{S}_0],\omega_0}) &\subseteq [r_B^{[\mathcal{S}_0],\omega_0}, f_B^{[\mathcal{S}_1^{rt}],\omega_1}) \cap [r_{B_0}^{[\mathcal{S}_0],\omega_0}, f_{B_0}^{[\mathcal{S}_0],\omega_0}) && (5.133)\\
&\subseteq [r_B^{[\mathcal{S}_0],\omega_0}, f_{B_0}^{[\mathcal{S}_0],\omega_0}) \cap [r_{B_0}^{[\mathcal{S}_0],\omega_0}, f_{B_0}^{[\mathcal{S}_0],\omega_0}) && (5.134)\\
&\subseteq [r_B^{[\mathcal{S}_0],\omega_0}, f_B^{[\mathcal{S}_0],\omega_0}) \cap [r_{B_0}^{[\mathcal{S}_0],\omega_0}, f_{B_0}^{[\mathcal{S}_0],\omega_0}). && (5.135)
\end{aligned}$$

We obtain that

$$\begin{aligned}
&\bigcup_{B >_{prio} B_0} [r_B^{[\mathcal{S}_1^{rt}],\omega_1}, f_B^{[\mathcal{S}_1^{rt}],\omega_1}) \cap [r_{B_0}^{[\mathcal{S}_0],\omega_0}, r_{B_0}^{[\mathcal{S}_0],\omega_0})\\
&\subseteq \bigcup_{B >_{prio} B_0} [r_B^{[\mathcal{S}_0],\omega_0}, f_B^{[\mathcal{S}_0],\omega_0}) \cap [r_{B_0}^{[\mathcal{S}_0],\omega_0}, r_{B_0}^{[\mathcal{S}_0],\omega_0}).
\end{aligned} \tag{5.136}$$

Hence, by applying this to Lemma 5.75, we achieve

$$W_{B_0}^{[\mathcal{S}_1^{rt}]}(r_{B_0}^{[\mathcal{S}_0],\omega_0}, f_{B_0}^{[\mathcal{S}_0],\omega_0}) \leq W_{B_0}^{[\mathcal{S}_0]}(r_{B_0}^{[\mathcal{S}_0],\omega_0}, f_{B_0}^{[\mathcal{S}_0],\omega_0}). \tag{5.137}$$

We use that to obtain

$$\begin{aligned}
f_{B_0}^{[\mathcal{S}_0],\omega_0} &< r_{B_0}^{[\mathcal{S}_0],\omega_0} + W_{B_0}^{[\mathcal{S}_1^{rt}]}(r_{B_0}^{[\mathcal{S}_0],\omega_0}, f_{B_0}^{[\mathcal{S}_0],\omega_0}) + c_{B_0}^{\omega_1} && (5.138)\\
&\leq r_{B_0}^{[\mathcal{S}_0],\omega_0} + W_{B_0}^{[\mathcal{S}_0]}(r_{B_0}^{[\mathcal{S}_0],\omega_0}, f_{B_0}^{[\mathcal{S}_0],\omega_0}) + c_{B_0}^{\omega_1} && (5.139)\\
&\leq r_{B_0}^{[\mathcal{S}_0],\omega_0} + W_{B_0}^{[\mathcal{S}_0]}(r_{B_0}^{[\mathcal{S}_0],\omega_0}, f_{B_0}^{[\mathcal{S}_0],\omega_0}) + c_{B_0}^{\omega_0}. && (5.140)
\end{aligned}$$

Since $r_{B_0}^{[\mathcal{S}_0],\omega_0} + W_{B_0}^{[\mathcal{S}_0]}(r_{B_0}^{[\mathcal{S}_0],\omega_0}, f_{B_0}^{[\mathcal{S}_0],\omega_0}) + c_{B_0}^{\omega_0} \leq f_{B_0}^{[\mathcal{S}_0],\omega_0}$ by Observation 5.71, we obtain a contradiction. This proves (ii).

Induction Step ($n-1 \mapsto n$): We assume that (i) and (ii) hold for all segments in $Seg_n := (B_0, B_1, \dots, B_{n-1})$. In the following, we show that (i) and (ii) hold for B_n. Let J be the job that segment B_n belongs to.

For (i), if B_n is the first segment of J, then B_n is released at time $r_{B_n}^{[\mathcal{S}_1^{rt}],\omega_1} = r_J = r_{B_n}^{[\mathcal{S}_0],\omega_0}$, similar to the base case. If B_n is not the first segment of J, then we denote by B the segment of J prior to B_n. By definition, $r_{B_n}^{[\mathcal{S}_1^{rt}],\omega_1} \geq r_{B_n}^{[\mathcal{S}_0],\omega_0}$. Furthermore, $r_{B_n}^{[\mathcal{S}_1^{rt}],\omega_1} > r_{B_n}^{[\mathcal{S}_0],\omega_0}$ is only possible if $f_B^{[\mathcal{S}_1^{rt}],\omega_1} > f_B^{[\mathcal{S}_0],\omega_0}$. However, $B \in Seg_n$ since B finishes before B_n in the nominal schedule. Therefore, by induction, $f_B^{[\mathcal{S}_1^{rt}],\omega_1} \leq f_B^{[\mathcal{S}_0],\omega_0}$. We conclude that $r_{B_n}^{[\mathcal{S}_1^{rt}],\omega_1} = r_{B_n}^{[\mathcal{S}_0],\omega_0}$. This proves (i).

As in the base case, we prove (ii) by contradiction and assume that $f_{B_n}^{[\mathcal{S}_0],\omega_0} < f_{B_n}^{[\mathcal{S}_1^{rt}],\omega_1}$. Similar to the base case, we obtain

$$f_{B_n}^{[\mathcal{S}_0],\omega_0} < r_{B_n}^{[\mathcal{S}_0],\omega_0} + W_{B_n}^{[\mathcal{S}_1^{rt}]}(r_{B_n}^{[\mathcal{S}_0],\omega_0}, f_{B_n}^{[\mathcal{S}_0],\omega_0}) + c_{B_n}^{\omega_1}. \tag{5.141}$$

By the enforcement mechanism, for all segments $B \in \mathbb{B}$, $r_B^{[\mathcal{S}_1^{rt}],\omega_1} \geq r_B^{[\mathcal{S}_0],\omega_0}$ holds. Hence,

$$[r_B^{[\mathcal{S}_1^{rt}],\omega_1}, f_B^{[\mathcal{S}_1^{rt}],\omega_1}) \cap [r_{B_n}^{[\mathcal{S}_0],\omega_0}, f_{B_n}^{[\mathcal{S}_0],\omega_0}) \subseteq [r_B^{[\mathcal{S}_0],\omega_0}, f_B^{[\mathcal{S}_1^{rt}],\omega_1}) \cap [r_{B_n}^{[\mathcal{S}_0],\omega_0}, f_{B_n}^{[\mathcal{S}_0],\omega_0}). \tag{5.142}$$

As in the base case, in the following we show that

$$[r_B^{[\mathcal{S}_0],\omega_0}, f_B^{[\mathcal{S}_1^{rt}],\omega_1}) \cap [r_{B_n}^{[\mathcal{S}_0],\omega_0}, f_{B_n}^{[\mathcal{S}_0],\omega_0}) \overset{!}{\subseteq} [r_B^{[\mathcal{S}_0],\omega_0}, f_B^{[\mathcal{S}_0],\omega_0}) \cap [r_{B_n}^{[\mathcal{S}_0],\omega_0}, f_{B_n}^{[\mathcal{S}_0],\omega_0}). \tag{5.143}$$

However, for the induction step we distinguish two cases:

Case 1: $f_B^{[\mathcal{S}_0],\omega_0} < f_{B_n}^{[\mathcal{S}_0],\omega_0}$. In that case, we know that B is in Seg_n, and by induction $f_B^{[\mathcal{S}_1^{rt}],\omega_1} \leq f_B^{[\mathcal{S}_0],\omega_0}$ holds. Therefore, Equation (5.143) holds.

Case 2: $f_B^{[\mathcal{S}_0],\omega_0} \geq f_{B_n}^{[\mathcal{S}_0],\omega_0}$. In that case, we obtain

$$\begin{aligned}
[r_B^{[\mathcal{S}_0],\omega_0}, f_B^{[\mathcal{S}_1^{rt}],\omega_1}) \cap [r_{B_n}^{[\mathcal{S}_0],\omega_0}, f_{B_n}^{[\mathcal{S}_0],\omega_0}) &\subseteq [r_B^{[\mathcal{S}_0],\omega_0}, f_{B_n}^{[\mathcal{S}_0],\omega_0}) \cap [r_{B_n}^{[\mathcal{S}_0],\omega_0}, f_{B_n}^{[\mathcal{S}_0],\omega_0}) &(5.144)\\
&\subseteq [r_B^{[\mathcal{S}_0],\omega_0}, f_B^{[\mathcal{S}_0],\omega_0}) \cap [r_{B_n}^{[\mathcal{S}_0],\omega_0}, f_{B_n}^{[\mathcal{S}_0],\omega_0}) &(5.145)
\end{aligned}$$

as well.

By applying Equations (5.142) and (5.143), we obtain that

$$\begin{aligned}
&\bigcup_{B >_{prio} B_n} [r_B^{[\mathcal{S}_1^{rt}],\omega_1}, f_B^{[\mathcal{S}_1^{rt}],\omega_1}) \cap [r_{B_n}^{[\mathcal{S}_0],\omega_0}, r_{B_n}^{[\mathcal{S}_0],\omega_0})\\
&\quad \subseteq \bigcup_{B >_{prio} B_n} [r_B^{[\mathcal{S}_0],\omega_0}, f_B^{[\mathcal{S}_0],\omega_0}) \cap [r_{B_n}^{[\mathcal{S}_0],\omega_0}, r_{B_n}^{[\mathcal{S}_0],\omega_0}).
\end{aligned} \tag{5.146}$$

Hence, by applying this to Lemma 5.75, we achieve

$$W_{B_n}^{[\mathcal{S}_1^{rt}]}(r_{B_n}^{[\mathcal{S}_0],\omega_0}, f_{B_n}^{[\mathcal{S}_0],\omega_0}) \leq W_{B_n}^{[\mathcal{S}_0]}(r_{B_n}^{[\mathcal{S}_0],\omega_0}, f_{B_n}^{[\mathcal{S}_0],\omega_0}). \tag{5.147}$$

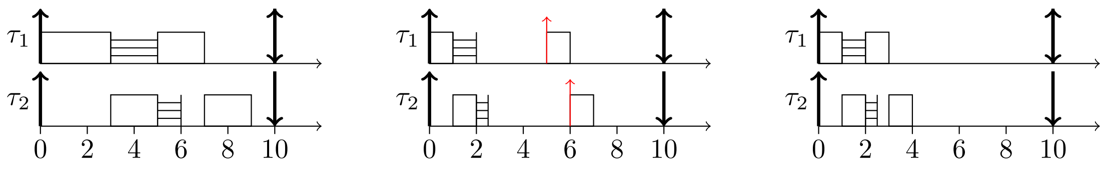
Similar to the base case, we use that to obtain

$$\begin{aligned}
f_{B_n}^{[\mathcal{S}_0],\omega_0} &< r_{B_n}^{[\mathcal{S}_0],\omega_0} + W_{B_n}^{[\mathcal{S}_1^{rt}]}(r_{B_n}^{[\mathcal{S}_0],\omega_0}, f_{B_n}^{[\mathcal{S}_0],\omega_0}) + c_{B_n}^{\omega_1} &(5.148)\\
&\leq r_{B_n}^{[\mathcal{S}_0],\omega_0} + W_{B_n}^{[\mathcal{S}_0]}(r_{B_n}^{[\mathcal{S}_0],\omega_0}, f_{B_n}^{[\mathcal{S}_0],\omega_0}) + c_{B_n}^{\omega_1} &(5.149)\\
&\leq r_{B_n}^{[\mathcal{S}_0],\omega_0} + W_{B_n}^{[\mathcal{S}_0]}(r_{B_n}^{[\mathcal{S}_0],\omega_0}, f_{B_n}^{[\mathcal{S}_0],\omega_0}) + c_{B_n}^{\omega_0}. &(5.150)
\end{aligned}$$

Since $r_{B_n}^{[\mathcal{S}_0],\omega_0} + W_{B_n}^{[\mathcal{S}_0]}(r_{B_n}^{[\mathcal{S}_0],\omega_0}, f_{B_n}^{[\mathcal{S}_0],\omega_0}) + c_{B_n}^{\omega_0} \leq f_{B_n}^{[\mathcal{S}_0],\omega_0}$ by Observation 5.71, we obtain a contradiction, and (ii) is proven. This concludes the induction step and therefore this also proves the theorem. □

5.2.3 Segment Priority Modification

In Section 5.2.2, we eliminate the timing anomalies and ensure that a feasible segmented self-suspending task set remains schedulable in online scheduling by enforcing the release



(a) T-FP schedule (nominal). (b) Segment release time enforcement (online). (c) No treatment (online).

Figure 5.26.: Schedules of the task set $\mathbb{T} = \{\tau_1, \tau_2\}$ under T-FP scheduling where τ_1 has higher priority than τ_2. Enforcing the segment release time leads to a longer per-job response time, compared to the schedule without treatment.

time of the segments. Although delaying the segment release has no negative impact on the worst-case behavior, this treatment may lead to poor average-case performance. To demonstrate this, consider the case shown in Figure 5.26, where the WCETs and the maximum suspension times of segments are significantly larger than the actual execution and suspension times. Figure 5.26a demonstrates the nominal schedule, Figure 5.26b shows the schedule with enforcement, and Figure 5.26c shows the schedules without enforcement. Compared to the schedule without enforcement, enforcing the segment release time leads to a longer per-job response time, i.e., the time elapsed between the release and the completion of a job. Therefore, a treatment to avoid timing anomalies without artificially delaying segments is desirable.

To that end, we propose a treatment that modifies the segment priorities to eliminate timing anomalies. In particular, we redefine the segment priorities according to their finishing time in the nominal schedule. The rationale is that a segment with later finishing time should not be able to interfere a segment with an earlier finishing time. To distinguish original segment priorities and modified segment priorities, we call the modified priorities *preference* instead. More specifically, we say that a segment $B \in \mathbb{B}$ has a higher preference than $B' \in \mathbb{B}$ if B finishes earlier than B' in the nominal schedule. We denote the preference as:

$$B >_{pref} B' \quad :\Leftrightarrow \quad f_B^{[\mathcal{S}_O],\omega_O} < f_{B'}^{[\mathcal{S}_O],\omega_O} \tag{5.151}$$

This leads to a total ordering of all segments in $\mathbb{B}$. In the online schedule, segments with higher preference are scheduled first. We denote by $[\mathcal{S}_1^{sp}]$ the online schedule with priority modification.

For example, consider the system of Figure 5.26. We denote by B_1^1 and B_2^1 the two segments of the first job of task τ_1, and we denote by B_1^2 and B_2^2 the two segments of the first job of task τ_2. The total preference ordering is:

$$B_1^1 >_{pref} B_1^2 >_{pref} B_2^1 >_{pref} B_2^2 \tag{5.152}$$

Therefore, under the treatment with modified segment priorities, the online schedule will look exactly like the schedule without treatment in Figure 5.26c.

Since in the online schedule, the segment priority is replaced with the segment preference, we can observe the following.

Observation 5.77. *Let $B \in \mathbb{B}$ be a segment. The segment B finishes in the online schedule with priority modification, i.e., in $[\mathcal{S}_1^{sp}]$, at the lowest $t \in \mathbb{R}$ such that*

$$t \geq r_B^{[\mathcal{S}_1^{sp}],\omega_1} + W_B^{[\mathcal{S}_1^{sp}]}(r_B^{[\mathcal{S}_1^{sp}],\omega_1}, t) + c_B^{\omega_1}, \tag{5.153}$$

where $W_B^{[\mathcal{S}_1^{sp}]}(r_B^{[\mathcal{S}_1^{sp}],\omega_1}, t)$ is the total amount of time that higher-preference segments are executed during the interval $[r_B^{[\mathcal{S}_1^{sp}],\omega_1}, t)$ in $[\mathcal{S}_1^{sp}]$, i.e.,

$$W_B^{[\mathcal{S}_1^{sp}]}(r_B^{[\mathcal{S}_1^{sp}],\omega_1}, t) := \sum_{B' >_{pref} B} \text{exec}_{B'}^{[\mathcal{S}_1^{sp}]}(r_B^{[\mathcal{S}_1^{sp}],\omega_1}, t). \tag{5.154}$$

To prove that timing anomalies do not occur, we need to show that the interference from higher-preference segments in the online schedule is not higher than the interference from higher-priority segments in the nominal schedule. To achieve this, we first prove the following key ingredient.

Lemma 5.78. *For all segments $B \in \mathbb{B}$, the interference in the nominal schedule is lower bounded by:*

$$W_B^{[\mathcal{S}_0]}(s_B^{[\mathcal{S}_0],\omega_0}, f_B^{[\mathcal{S}_0],\omega_0}) \geq \sum_{\substack{B' \in \mathbb{B} \\ s_B^{[\mathcal{S}_0],\omega_0} < f_{B'}^{[\mathcal{S}_0],\omega_0} < f_B^{[\mathcal{S}_0],\omega_0}}} c_{B'}^{\omega_0} \tag{5.155}$$

Proof. We know that $W_B^{[\mathcal{S}_0]}(s_B^{[\mathcal{S}_0],\omega_0}, f_B^{[\mathcal{S}_0],\omega_0}) = \sum_{B' >_{prio} B} \text{exec}_{B'}^{[\mathcal{S}_0]}(s_B^{[\mathcal{S}_0],\omega_0}, f_B^{[\mathcal{S}_0],\omega_0})$ by definition in Equation (5.117). Moreover, if $B' \in \mathbb{B}$ fulfills $s_B^{[\mathcal{S}_0],\omega_0} < f_{B'}^{[\mathcal{S}_0],\omega_0} < f_B^{[\mathcal{S}_0],\omega_0}$, then $B' \neq B$, and B' must be executed during the interval $(s_B^{[\mathcal{S}_0],\omega_0}, f_B^{[\mathcal{S}_0],\omega_0})$, i.e., it must have higher priority than B. Therefore, we can bound $W_B^{[\mathcal{S}_0]}(s_B^{[\mathcal{S}_0],\omega_0}, f_B^{[\mathcal{S}_0],\omega_0})$ from below by

$$W_B^{[\mathcal{S}_0]}(s_B^{[\mathcal{S}_0],\omega_0}, f_B^{[\mathcal{S}_0],\omega_0}) \geq \sum_{\substack{B' \in \mathbb{B} \\ s_B^{[\mathcal{S}_0],\omega_0} < f_{B'}^{[\mathcal{S}_0],\omega_0} < f_B^{[\mathcal{S}_0],\omega_0}}} \text{exec}_{B'}^{[\mathcal{S}_0]}(s_B^{[\mathcal{S}_0],\omega_0}, f_B^{[\mathcal{S}_0],\omega_0}). \tag{5.156}$$

Now consider an arbitrary $B' \in \mathbb{B}$ with $s_B^{[\mathcal{S}_0],\omega_0} < f_{B'}^{[\mathcal{S}_0],\omega_0} < f_B^{[\mathcal{S}_0],\omega_0}$. As explained above, B' must have higher priority than B. If B' would be executed before $s_B^{[\mathcal{S}_0],\omega_0}$, then it would have to finish before B can start. In that case, $f_{B'}^{[\mathcal{S}_0],\omega_0} \leq s_B^{[\mathcal{S}_0],\omega_0}$, which contradicts $s_B^{[\mathcal{S}_0],\omega_0} < f_{B'}^{[\mathcal{S}_0],\omega_0}$. Hence, we know that B' can only start after $s_B^{[\mathcal{S}_0],\omega_0}$, i.e., we have $s_B^{[\mathcal{S}_0],\omega_0} < s_B^{[\mathcal{S}_0],\omega_0} \leq f_{B'}^{[\mathcal{S}_0],\omega_0} < f_B^{[\mathcal{S}_0],\omega_0}$. Therefore,

$$\text{exec}_{B'}^{[\mathcal{S}_0]}(s_B^{[\mathcal{S}_0],\omega_0}, f_B^{[\mathcal{S}_0],\omega_0}) \geq \text{exec}_{B'}^{[\mathcal{S}_0]}(s_{B'}^{[\mathcal{S}_0],\omega_0}, f_{B'}^{[\mathcal{S}_0],\omega_0}) = c_{B'}^{\omega_0}. \tag{5.157}$$

Equations (5.156) and (5.157) combined prove the lemma. □

The previous lemma allows us to prove that no timing anomalies occur with the treatment that modifies segment priorities, as formulated in the following theorem.

Theorem 5.79. *The finishing time of each segment in the online schedule $[\mathcal{S}_1^{sp}]$ with segment preference instead of segment priorities is* no larger *than the finishing time in the nominal schedule $[\mathcal{S}_0]$, i.e., $f_B^{[\mathcal{S}_0],\omega_0} \geq f_B^{[\mathcal{S}_1^{sp}],\omega_1}$ for all $B \in \mathbb{B}$.*

Proof. Similar to the proof of Theorem 5.76, let $Seg = (B_0, B_1, \dots)$ denote the list of all segments $\mathbb{B}$ ordered by their finishing time in the nominal schedule, i.e., $f_{B_0}^{[\mathcal{S}_0],\omega_0} < f_{B_1}^{[\mathcal{S}_0],\omega_0} < \dots$ holds. Please note that this ordering respects the segment preferences, i.e., $B_0 >_{pref} B_1 >_{pref} \dots$. We consider the online schedule $[\mathcal{S}_1^{sp}]$ obtained with segment preferences instead of segment priorities. By induction over the segments in Seg we show that for each B_n, with $n = 0, 1, \dots$, the inequality $f_{B_n}^{[\mathcal{S}_0],\omega_0} \geq f_{B_n}^{[\mathcal{S}_1^{sp}],\omega_1}$ holds.

Base case ($n = 0$): We prove $f_{B_0}^{[\mathcal{S}_0],\omega_0} \geq f_{B_0}^{[\mathcal{S}_1^{sp}],\omega_1}$ by contradiction, i.e., we assume $f_{B_0}^{[\mathcal{S}_0],\omega_0} < f_{B_0}^{[\mathcal{S}_1^{sp}],\omega_1}$. By Observation 5.77, we know that $f_{B_0}^{[\mathcal{S}_1^{sp}],\omega_1}$ is the lowest $t \in \mathbb{R}$ such that $t \geq r_{B_0}^{[\mathcal{S}_1^{sp}],\omega_1} + W_{B_0}^{[\mathcal{S}_1^{sp}]}(r_{B_0}^{[\mathcal{S}_1^{sp}],\omega_1}, t) + c_{B_0}^{\omega_1}$ holds. Since $f_{B_0}^{[\mathcal{S}_0],\omega_0} < f_{B_0}^{[\mathcal{S}_1^{sp}],\omega_1}$ by assumption, we have

$$f_{B_0}^{[\mathcal{S}_0],\omega_0} < r_{B_0}^{[\mathcal{S}_1^{sp}],\omega_1} + W_{B_0}^{[\mathcal{S}_1^{sp}]}(r_{B_0}^{[\mathcal{S}_1^{sp}],\omega_1}, f_{B_0}^{[\mathcal{S}_0],\omega_0}) + c_{B_0}^{\omega_1}. \tag{5.158}$$

Since B_0 has the earliest finishing time in $[\mathcal{S}_0]$ among all segments, it must be the first segment of its corresponding job J. Hence, B_0 is released at time $r_{B_0}^{[\mathcal{S}_1^{sp}],\omega_1} = r_J^{[\mathcal{S}_1^{sp}],\omega_1} = r_J = r_J^{[\mathcal{S}_0],\omega_0} = r_{B_0}^{[\mathcal{S}_0],\omega_0}$. Moreover, since B_0 has the highest preference, $W_{B_0}^{[\mathcal{S}_1^{sp}]}(r_{B_0}^{[\mathcal{S}_1^{sp}],\omega_1}, f_{B_0}^{[\mathcal{S}_0],\omega_0})$ is the sum over an empty set, i.e., $W_{B_0}^{[\mathcal{S}_1^{sp}]}(r_{B_0}^{[\mathcal{S}_1^{sp}],\omega_1}, f_{B_0}^{[\mathcal{S}_0],\omega_0}) = 0$. This leads us to

$$f_{B_0}^{[\mathcal{S}_0],\omega_0} < r_{B_0}^{[\mathcal{S}_0],\omega_0} + c_{B_0}^{\omega_1} \leq r_{B_0}^{[\mathcal{S}_0],\omega_0} + c_{B_0}^{\omega_0} \leq r_{B_0}^{[\mathcal{S}_0],\omega_0} + W_{B_0}^{[\mathcal{S}_0]}(r_{B_0}^{[\mathcal{S}_0],\omega_0}, f_{B_0}^{[\mathcal{S}_0],\omega_0}) + c_{B_0}^{\omega_0}. \tag{5.159}$$

Since $f_{B_0}^{[\mathcal{S}_0],\omega_0} \geq r_{B_0}^{[\mathcal{S}_0],\omega_0} + W_{B_0}^{[\mathcal{S}_0]}(r_{B_0}^{[\mathcal{S}_0],\omega_0}, f_{B_0}^{[\mathcal{S}_0],\omega_0}) + c_{B_0}^{\omega_0}$ holds by Observation 5.71, we obtain a contradiction. This proves $f_{B_0}^{[\mathcal{S}_0],\omega_0} \geq f_{B_0}^{[\mathcal{S}_1^{sp}],\omega_1}$.

Induction Step ($n - 1 \mapsto n$): By the induction hypothesis, we know that $f_{B_j}^{[\mathcal{S}_1^{sp}],\omega_1} \leq f_{B_j}^{[\mathcal{S}_0],\omega_0}$ holds for all the previous segments $B_j \in Seg_n := (B_0, B_1, \dots, B_{n-1})$. In the following, we show that $f_{B_n}^{[\mathcal{S}_1^{sp}],\omega_1} \leq f_{B_n}^{[\mathcal{S}_0],\omega_0}$. To achieve this, we first prove that

$$W_{B_n}^{[\mathcal{S}_1^{sp}]}(s_{B_n}^{[\mathcal{S}_0],\omega_0}, f_{B_n}^{[\mathcal{S}_0],\omega_0}) \leq W_{B_n}^{[\mathcal{S}_0]}(s_{B_n}^{[\mathcal{S}_0],\omega_0}, f_{B_n}^{[\mathcal{S}_0],\omega_0}). \tag{5.160}$$

We already know that $W_{B_n}^{[\mathcal{S}_0]}(s_{B_n}^{[\mathcal{S}_0],\omega_0}, f_{B_n}^{[\mathcal{S}_0],\omega_0}) \geq \sum_{\substack{B \in \mathbb{B} \\ s_{B_n}^{[\mathcal{S}_0],\omega_0} < f_B^{[\mathcal{S}_0],\omega_0} < f_{B_n}^{[\mathcal{S}_0],\omega_0}}} c_B^{\omega_0}$ holds by Lemma 5.78. Therefore, it is left to show that

$$W_{B_n}^{[\mathcal{S}_1^{sp}]}(s_{B_n}^{[\mathcal{S}_0],\omega_0}, f_{B_n}^{[\mathcal{S}_0],\omega_0}) \overset{!}{\leq} \sum_{\substack{B \in \mathbb{B} \\ s_{B_n}^{[\mathcal{S}_0],\omega_0} < f_B^{[\mathcal{S}_0],\omega_0} < f_{B_n}^{[\mathcal{S}_0],\omega_0}}} c_B^{\omega_0}. \tag{5.161}$$

We know that $W_{B_n}^{[\mathcal{S}_1^{sp}]}(s_{B_n}^{[\mathcal{S}_0],\omega_0}, f_{B_n}^{[\mathcal{S}_0],\omega_0}) := \sum_{B >_{pref} B_n} \text{exec}_B^{[\mathcal{S}_1^{sp}]}(s_{B_n}^{[\mathcal{S}_0],\omega_0}, f_{B_n}^{[\mathcal{S}_0],\omega_0})$, by definition of $W_{B_n}^{[\mathcal{S}_1^{sp}]}$. In that definition, the condition $B >_{pref} B_n$ is equivalent to $f_B^{[\mathcal{S}_0],\omega_0} < f_{B_n}^{[\mathcal{S}_0],\omega_0}$. Moreover, if $\text{exec}_B^{[\mathcal{S}_1^{sp}]}(s_{B_n}^{[\mathcal{S}_0],\omega_0}, f_{B_n}^{[\mathcal{S}_0],\omega_0}) \neq 0$, then $f_B^{[\mathcal{S}_1^{sp}],\omega_1} > s_{B_n}^{[\mathcal{S}_0],\omega_0}$ must hold. Since $B \in Seg_n$, we have $f_B^{[\mathcal{S}_0],\omega_0} \geq f_B^{[\mathcal{S}_1^{sp}],\omega_1} > s_{B_n}^{[\mathcal{S}_0],\omega_0}$ by induction. We obtain

$$W_{B_n}^{[\mathcal{S}_1^{sp}]}(s_{B_n}^{[\mathcal{S}_0],\omega_0}, f_{B_n}^{[\mathcal{S}_0],\omega_0}) \leq \sum_{\substack{B \in \mathbb{B} \\ s_{B_n}^{[\mathcal{S}_0],\omega_0} < f_B^{[\mathcal{S}_0],\omega_0} < f_{B_n}^{[\mathcal{S}_0],\omega_0}}} \text{exec}_B^{[\mathcal{S}_1^{sp}]}(s_{B_n}^{[\mathcal{S}_0],\omega_0}, f_{B_n}^{[\mathcal{S}_0],\omega_0}) \tag{5.162}$$

$$\leq \sum_{\substack{B \in \mathbb{B} \\ s_{B_n}^{[\mathcal{S}_0],\omega_0} < f_B^{[\mathcal{S}_0],\omega_0} < f_{B_n}^{[\mathcal{S}_0],\omega_0}}} c_B^{\omega_1} \leq \sum_{\substack{B \in \mathbb{B} \\ s_{B_n}^{[\mathcal{S}_0],\omega_0} < f_B^{[\mathcal{S}_0],\omega_0} < f_{B_n}^{[\mathcal{S}_0],\omega_0}}} c_B^{\omega_0}. \tag{5.163}$$

This proves Equation (5.161) and therefore, also Equation (5.160) is proven.

We show $f_{B_n}^{[\mathcal{S}_0],\omega_0} \geq f_{B_n}^{[\mathcal{S}_1^{sp}],\omega_1}$ by contradiction, i.e., we assume that $f_{B_n}^{[\mathcal{S}_0],\omega_0} < f_{B_n}^{[\mathcal{S}_1^{sp}],\omega_1}$. By Observation 5.77, we know that $f_{B_n}^{[\mathcal{S}_1^{sp}],\omega_1}$ is the lowest $t \in \mathbb{R}$ such that $t \geq r_{B_n}^{[\mathcal{S}_1^{sp}],\omega_1} + W_{B_n}^{[\mathcal{S}_1^{sp}]}(r_{B_n}^{[\mathcal{S}_1^{sp}],\omega_1}, t) + c_{B_n}^{\omega_1}$ holds. Since $f_{B_n}^{[\mathcal{S}_0],\omega_0} < f_{B_n}^{[\mathcal{S}_1^{sp}],\omega_1}$, we have

$$f_{B_n}^{[\mathcal{S}_0],\omega_0} < r_{B_n}^{[\mathcal{S}_1^{sp}],\omega_1} + W_{B_n}^{[\mathcal{S}_1^{sp}]}(r_{B_n}^{[\mathcal{S}_1^{sp}],\omega_1}, f_{B_n}^{[\mathcal{S}_0],\omega_0}) + c_{B_n}^{\omega_1}. \tag{5.164}$$

If B_n is the first segment of a job J, then similar to the base case it is released at the release of J and $r_{B_n}^{[\mathcal{S}_1^{sp}],\omega_1} = r_J = r_{B_n}^{[\mathcal{S}_0],\omega_0}$ holds. Otherwise, the segment $B \in \mathbb{B}$ prior to B_n in its corresponding job J is in Seg_n. By induction $f_B^{[\mathcal{S}_1^{sp}],\omega_1} \leq f_B^{[\mathcal{S}_0],\omega_0}$, and therefore the segment B_n is released in $[\mathcal{S}_1^{sp}]$ no later than in $[\mathcal{S}_0]$, i.e., $r_{B_n}^{[\mathcal{S}_1^{sp}],\omega_1} \leq r_{B_n}^{[\mathcal{S}_0],\omega_0} \leq s_{B_n}^{[\mathcal{S}_0],\omega_0}$. We obtain

$$f_{B_n}^{[\mathcal{S}_0],\omega_0} < r_{B_n}^{[\mathcal{S}_1^{sp}],\omega_1} + W_{B_n}^{[\mathcal{S}_1^{sp}]}(r_{B_n}^{[\mathcal{S}_1^{sp}],\omega_1}, f_{B_n}^{[\mathcal{S}_0],\omega_0}) + c_{B_n}^{\omega_1} \tag{5.165}$$

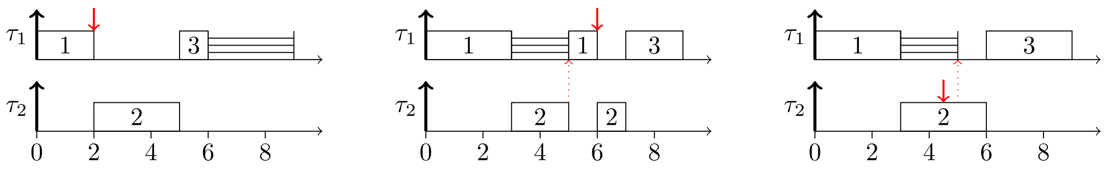
$$\leq r_{B_n}^{[\mathcal{S}_1^{sp}],\omega_1} + (s_{B_n}^{[\mathcal{S}_0],\omega_0} - r_{B_n}^{[\mathcal{S}_1^{sp}],\omega_1}) + W_{B_n}^{[\mathcal{S}_1^{sp}]}(s_{B_n}^{[\mathcal{S}_0],\omega_0}, f_{B_n}^{[\mathcal{S}_0],\omega_0}) + c_{B_n}^{\omega_1} \tag{5.166}$$

$$= s_{B_n}^{[\mathcal{S}_0],\omega_0} + W_{B_n}^{[\mathcal{S}_1^{sp}]}(s_{B_n}^{[\mathcal{S}_0],\omega_0}, f_{B_n}^{[\mathcal{S}_0],\omega_0}) + c_{B_n}^{\omega_1} \tag{5.167}$$

$$\leq s_{B_n}^{[\mathcal{S}_0],\omega_0} + W_{B_n}^{[\mathcal{S}_0]}(s_{B_n}^{[\mathcal{S}_0],\omega_0}, f_{B_n}^{[\mathcal{S}_0],\omega_0}) + c_{B_n}^{\omega_0}. \tag{5.168}$$

Since $f_{B_n}^{[\mathcal{S}_0],\omega_0} \geq s_{B_n}^{[\mathcal{S}_0],\omega_0} + W_{B_n}^{[\mathcal{S}_0]}(s_{B_n}^{[\mathcal{S}_0],\omega_0}, f_{B_n}^{[\mathcal{S}_0],\omega_0}) + c_{B_n}^{\omega_0}$ holds by Observation 5.72, we obtain a contradiction. Hence, $f_{B_n}^{[\mathcal{S}_0],\omega_0} \geq f_{B_n}^{[\mathcal{S}_1^{sp}],\omega_1}$ holds. This concludes the induction step and therefore the theorem is proven. $\square$

Implementation in RTEMS We implemented a S-FP scheduling mechanism in RTEMS [RTE23], an open-source RTOS, to demonstrate the applicability of the treatment *segment priority modification*. In the following, we first introduce the key APIs for



(a) Before. (b) After. (c) During.

Figure 5.27.: Impact of the time to perform priority modification. The red arrow indicates the time point that the priority is modified. Only option (c) leads to the desired behavior.

implementing the S-FP scheduling. Afterwards, we showcase the validity of the treatment in RTEMS with a working example.

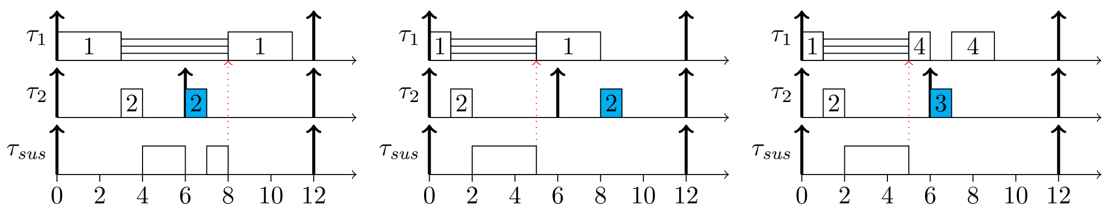
There are three major functionalities to be taken into consideration while designing the S-FP scheduling mechanism in RTEMS: (i) self-suspension of a task, (ii) resuming a self-suspended task, and (iii) modifying task priority. In our current design, we introduce a middle layer which wraps all the required functions provided by RTEMS at the API layer without modifying the underlying kernel. To perform self-suspension, the current executing task calls the function `rtems_task_suspend()` with its own task ID `RTEMS_SELF` at the end of a segment execution, except for the last segment. Since a task cannot resume itself after suspension, the resume of a suspended task must be triggered by other sources, e.g., another task.

In the S-FP scheduling, segments from the same task are allowed to have different priorities. However, RTEMS only supports task-level priority assignment in the current version. Therefore, we adjusted the priority for each segment by calling the function `rtems_task_set_priority()`.

There are three possible moments for modifying the priority: *before*, *after*, and *during* the suspension of the previous segment, as shown in Figure 5.27. If the priority is modified *before* the suspension starts, i.e., during the execution of the previous segment, the remaining execution of the previous segment can be preempted by another job (Figure 5.27a). Alternatively, modifying the priority *after* the segment starts can result in unexpected preemptions (Figure 5.27b). This is due to the gap between calling the function `rtems_task_set_priority()` and issuing the priority modification. Therefore, the ideal solution is to perform priority modification *during* the suspension, as shown in Figure 5.27c.

Considering the priority modification during suspension, we introduce a customized resume mechanism. Given the priority of each segment as inputs, we keep a lookup table in the middle layer. Each time before a suspended task is about to be resumed, the controlling task first calls the function `rtems_task_set_priority()` to assign the new priority to the task according to the lookup table, then it calls `rtems_task_resume()` with the ID of the task to be resumed.

We validated the proposed treatment, *segment priority modification*, in RTEMS with our S-FP implementation using the following example. Given a task set $\mathbb{T} = \{\tau_1, \tau_2\}$ of



(a) T-FP WCET. (b) T-FP with early completion. (c) S-FP with early completion.

Figure 5.28.: The working example implemented in RTEMS. With early completion of the first segment of τ_1, the finishing time of the second job from τ_2 (marked in blue) increases under T-FP scheduling (T-FP with early completion). By applying the priorities from the treatment *segment priority modification* on S-FP scheduling (S-FP with early completion), the finishing time remains the same as in the nominal schedule (T-FP WCET).

two tasks to be scheduled on a uniprocessor system, with:

- $\tau_1 \in \text{PerOff}(12, 0) \cap \text{SegSS}(3, 5, 3)$
- $\tau_2 \in \text{PerOff}(6, 0) \cap \text{SegSS}(1)$

Since a task cannot resume itself after suspension, we add one additional lowest priority task $\tau_{sus} \in \text{PerOff}(12, 0) \cap \text{SegSS}(3)$, which resumes τ_1 when it finishes execution. We consider three scenarios: (a) **T-FP WCET**, (b) **T-FP with early completion**, and (c) **S-FP with early completion**. In **T-FP WCET** and **T-FP with early completion**, the tasks are scheduled using T-FP scheduling. Each segment is executed for its WCET, and then suspended for the maximum length of the suspension interval in **T-FP WCET**. In **T-FP with early completion**, the segments and the suspension interval can finish earlier. More specifically, we decrease the execution time of the first segment in τ_1. In **S-FP with early completion**, the tasks are scheduled using S-FP scheduling. The priorities of the segments follow the treatment *segment priority modification*, which use the schedule generated by T-FP scheduling as the nominal schedule.

Figure 5.28 demonstrates the schedules generated in the three scenarios. The numbers in the segments are their priorities. A lower number indicates a higher priority. We observe that in Figure 5.28b, the first segment of τ_1 finishes earlier, which delays the second job of τ_2. With priorities generated from the treatment *segment priority modification*, the second job of τ_2 is not affected, i.e., no timing anomalies occur.

5.2.4 Evaluation of Proposed Mechanisms

We compare the proposed treatments to state-of-the-art scheduling algorithms in terms of schedulability on synthetic task sets. First, we describe how the task sets are synthesized, and briefly introduce the comparing algorithms. Afterwards, we report the acceptance ratios of the algorithms under different task set configurations, and propose an approach based on a combination of scheduling algorithms to achieve a higher acceptance ratio. The source code that is used to conduct the experiments is released on Github [TU 24].

Task Sets and Algorithms In our evaluation, we focus on periodic synchronous tasks with segmented self-suspension and implicit deadlines. The synthetic task sets are generated as follows. First, we consider different total utilization settings of a task set, ranging from 0 % to 100 % in a 5 % step. For each total utilization setting, we generate 100 task sets, each with 10 tasks $\{\tau_1, \dots, \tau_{10}\}$. Given the total utilization of a task set, we apply the Dirichlet-Rescale (DRS) algorithm [GBD20] to determine the utilization U_{τ_i} of each individual task τ_i. T_{τ_i}, the period of task τ_i, is selected uniformly at random from a set of semi-harmonic periods $T_{\tau_i} \in \{1, 2, 5, 10, 20, 50, 100, 200, 1\,000\}$, which is used in automotive systems [Brü+17; Ham+17; Tob+16]. Each task τ_i has an implicit deadline $D_\tau = T_\tau$. With the utilization U_{τ_i} and period T_{τ_i}, the total execution time of task τ_i is calculated accordingly, i.e., $C_{\tau_i} = U_{\tau_i} \cdot T_{\tau_i}$.

Next, we divide the total execution time C_{τ_i} into the m_{τ_i} segments. In our evaluation, m_{τ_i} is selected from the set $\{2\ (\textit{Rare}), 5\ (\textit{Moderate}), 8\ (\textit{Frequent})\}$ based on the configuration of the task set. The number of suspension intervals is set to $m_{\tau_i} - 1$ accordingly. The total suspension length of task τ_i is generated according to a uniform distribution in one of the following three ranges, as suggested in [BHC17; HC16]:

- Short suspension: $[0.01(T_{\tau_i} - C_{\tau_i}), 0.1(T_{\tau_i} - C_{\tau_i})]$
- Medium suspension: $[0.1(T_{\tau_i} - C_{\tau_i}), 0.3(T_{\tau_i} - C_{\tau_i})]$
- Long suspension: $[0.3(T_{\tau_i} - C_{\tau_i}), 0.6(T_{\tau_i} - C_{\tau_i})]$

Having the number of computation segments m_{τ_i}, the total execution time C_{τ_i}, and the total suspension length, we apply the DRS algorithm to determine the execution time of each computation segment and the length of each suspension interval, thus constructing the execution behavior $\mathrm{SegSS}(C^1_{\tau_i}, S^1_{\tau_i}, \dots, C^{m_{\tau_i}}_{\tau_i})$ for task τ_i.

We consider the following segmented self-suspending scheduling algorithms:[11]

- **NOM-EDF**: Our approach, the nominal schedules are generated using EDF.
- **NOM-RM**: Our approach, the nominal schedules are generated using RM.
- **SCAIR-OPA** [Sch+18b]: A pseudo-polynomial time schedulability test under Audsley's Optimal Priority assignment [Aud01].
- **SCAIR-RM** [Sch+18b]: A pseudo-polynomial time schedulability test under RM priority assignment.
- **EDAGMF-OPA** [HC16]: A T-FP equal deadline assignment scheduling with Audsley's Optimal Priority assignment.

[11]The evaluation framework for self-suspending task systems, i.e., SSSEvaluation [Gün+21b] (see Section 5.1.4.1), is applied for evaluating SCAIR-OPA, SCAIR-RM, and EDAGMF-OPA.

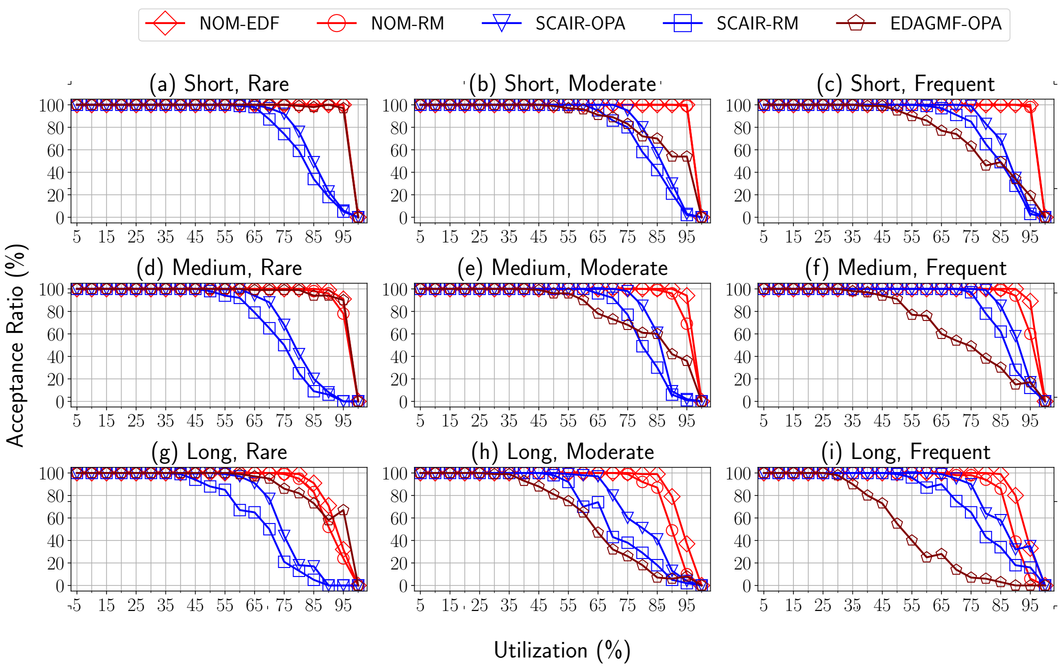


Figure 5.29.: Acceptance ratio of the approaches under different task set configurations.

Recall that the proposed treatments depend on information such as the nominal release times of segments and the total preference order in a nominal schedule. **NOM-EDF** and **NOM-RM** generate a nominal schedule by simulating the execution of the given task set based on the WCET and maximum suspension time of the segments using EDF/RM scheduling, respectively. As discussed in Remark 5.68, our treatments derive a feasible schedule whenever the nominal schedule simulated over one hyperperiod is feasible. Note that, although we focus on synchronous periodic tasks to ensure the schedulability of the nominal schedule in our evaluation, **NOM-EDF** and **NOM-RM** can work on any task set with a repetitive release pattern, e.g., periodic tasks with different offsets, as long as the nominal schedule repeats.

Schedulability under Different Task Set Configurations We compare the *acceptance ratio* between the evaluated scheduling algorithms. Figure 5.29 demonstrates the results on task sets with different configurations, i.e., number of segments in a task and total suspension length. We observe that in almost all the evaluated configurations, our **NOM-EDF** approach outperforms all the state of the arts. The reason is that in order to eliminate timing anomalies for scheduling sporadic tasks, the existing methods over-approximate the WCRTs of tasks, which leads to overly pessimistic results. An exception appears in Figure 5.29 (g), where **EDAGMF-OPA** has the highest acceptance ratio among all algorithms when the total utilization reaches 95 % for task sets with long suspension intervals and only two segments. We conclude that under certain configurations, priority assignment approaches such as **EDAGMF-OPA** can

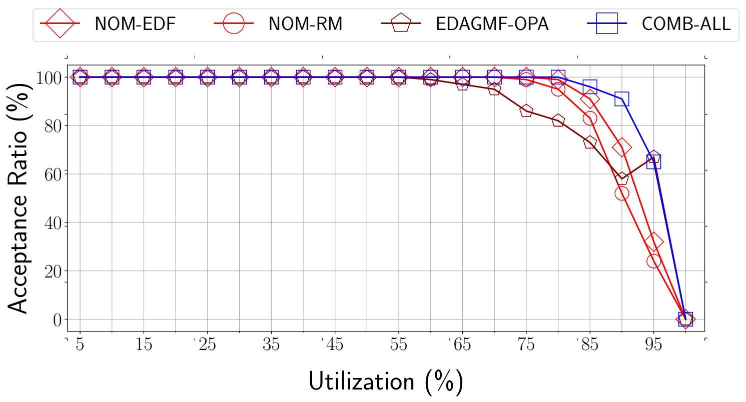


Figure 5.30.: Acceptance ratios under the *Long, Rare* configuration while considering the **COMB-ALL** approach.

significantly improve the performance, i.e., schedulability, of T-FP scheduling. Still, our proposed treatments show high acceptance ratios under all the other configurations.

Although we achieved high acceptance ratios with **NOM-RM** and **NOM-EDF** in almost all the evaluated configurations, we would like to point out that the proposed treatments do not bind to any specific scheduling algorithms for generating a nominal schedule. Given a feasible nominal schedule of a segmented self-suspension periodic task set generated by any T-FP scheduling algorithm, *segment release time enforcement* and *segment priority modification* eliminate timing anomalies and guarantee the schedulability, as proven in Sections 5.2.2 and 5.2.3. Therefore, we propose a new approach **COMB-ALL**, which applies several scheduling algorithms to a task set, and returns a nominal schedule if the task set is feasible by any of these algorithms. In our current design, we consider **NOM-EDF**, **NOM-RM**, and **EDAGMF-OPA** in **COMB-ALL** since these approaches in general outperform the others in Figure 5.29. Figure 5.30 demonstrates the acceptance ratios of **COMB-ALL** and those of the three approaches individually under the same configuration in Figure 5.29 (g). We observe that **COMB-ALL** has the highest acceptance ratio among the anomaly-free approaches.

5.3 Summary and Open Problems

In this chapter, we investigated the timing behavior of self-suspending tasks under preemptive uniprocessor scheduling. To that end, we first examined different *schedulability analyses* in Section 5.1. Second, in Section 5.2, we considered one specific reason for the analytical pessimism inherent to self-suspending tasks in more detail: the presence of *timing anomalies*.

Regarding schedulability analyses, we considered dynamic self-suspending tasks under T-FP, EDF and EL scheduling. Under T-FP scheduling, we provided the first suspension-aware analysis for arbitrary-deadline sporadic real-time tasks and tasks abstracted with arrival curves. Under EDF scheduling, we presented two novel analyses, namely a response-time analysis and an analysis that quantifies redundant self-suspension. We introduced a unifying response-time analysis for all EL scheduling algorithms, applicable to arbitrary-deadline sporadic tasks. Furthermore, we enhanced an analysis framework for schedulablility tests of self-suspending tasks by implementing state-of-the art schedulability tests we were aware of when the enhanced framework was published, and by providing additional features.

Regarding timing anomalies, we considered segmented self-suspending tasks with a fixed arrival pattern scheduled under S-FP scheduling. We detailed the reason for timing anomalies and used that observation to develop two mechanisms that avoid timing anomalies, namely segment release time enforcement and segment priority modification. By using these mechanisms, WCRTs of tasks occur when all tasks execute their WCET and suspend for their maximum suspension time. Hence, exact WCRT analysis for synchronous periodic tasks using our mechanisms can be conducted by simulating the schedule over one hyperperiod.

Despite our contributions of this chapter, there remain many open questions to be resolved and future work to be conducted, detailed as follows. Regarding the schedulability analysis under T-FP, we utilized heuristics from related work to partition $\mathbb{T}$ into $\mathbb{T}_1$ and $\mathbb{T}_2$. While any partitioning can be used to conduct the analysis, finding partitioning heuristics with mathematical reasoning tailored to our proposed schedulability test remains an open problem. Regarding the evaluation framework, currently only uniprocessor tests are integrated. To integrate multiprocessor tests as well, reasonable benchmarks to generate synthetic task sets for multiprocessor scenarios need to be explored. In that regard, the DRS algorithm [GBD20] could be used to make the generation process more efficient. Furthermore, since the publication of the enhanced evaluation framework, some novel analyses have not been integrated into the framework yet, and we should pursue to keep the framework up-to-date. Besides these immediate open problems, we emphasize the following research directions to be explored further:

- **Necessary tests for self-suspending tasks:** Although we made significant improvements in the analysis of self-suspending tasks, it remains unclear how close our results are to exact results. The reason is that for dynamic self-suspending tasks there exist no necessary schedulability conditions besides the simulation of specific analysis scenarios. In future work, necessary conditions should be analyzed in more detail to identify the margin for analytical improvement.

- **Exploration of self-suspension models:** While abstraction of self-suspending tasks using dynamic self-suspension is always possible, it may lead to potentially pessimistic results. Hence, the schedulability analysis in this work, although widely applicable, may be unsuitable in specific use cases. Besides dynamic self-suspension there exist some analyses for abstraction to segmented or hybrid self-suspension. However, the general problem of finding abstractions of self-suspension which give a tight description of actual use cases while being easily analyzable remains open. Instead of limiting the consideration to a few predefined task models, future work should pursue property-based abstractions to enable new analytical approaches and tight description of the preconditions of analytical results.

- **Extensions to multiprocessor scheduling:** Although the examination of this chapter focuses on uniprocessor scheduling, multicore systems are an essential part of distributed systems. While the results of this chapter are directly applicable to partitioned multiprocessor scheduling, by considering each processor individually, extensions to global multiprocessor scheduling are more involving. It remains unclear to what extent the insights of this chapter are applicable to global scheduling in multicore systems.
- **Generalized examination of timing anomalies:** While we identified two mechanisms to avoid timing anomalies for self-suspending tasks, these are limited to segmented self-suspending tasks with fixed release pattern. Furthermore, timing anomalies do not only occur for self-suspending tasks, as detailed in Section 3.1. It is unclear if similar mechanisms can be developed for more general task abstractions of self-suspending tasks or for other scenarios in which timing anomalies occur. In future work, timing anomalies should be examined in a more generalized and unified manner.

6 End-to-End Latency of Cause-Effect Chains

Most cyber-physical control systems must react to external activities with an appropriate actuation. They are often safety-critical, since not behaving as expected may lead to dire consequences for the system, humans, or the environment. A common characteristic of such control systems is that they perform the same functionality over and over again—frequently checking for an external activity and providing an actuation when needed. Figure 6.1 depicts an automatic break system where the car should automatically be stopped when a pedestrian is observed in front of the car. In this simplified example, three steps (each handled by a related task) must be performed: (i) A camera image is taken, (ii) image recognition marks pedestrians in the picture, and (iii) danger analysis determines whether the car must be stopped to not endanger pedestrians. This example also shows why safety-critical control systems often have timing requirements: It is not sufficient to calculate the correct result based on the observed situation, i.e., braking. Instead, the required actuation must also be performed within a certain time interval to avoid harming the pedestrian.

Prior to 2016, research primarily considered timing for individual tasks. For instance, a Worst-Case Response Time (WCRT) analysis for a specific task determines the maximum time interval between the release of a task instance and its completion. Such a WCRT analysis focuses only on the worst-case execution behavior of the task and the interference of other tasks. However, if a functionality is achieved by the interplay between tasks, as in the example in Figure 6.1, then the data propagation must be considered. Specifically, we are interested in the timing behavior of a chain that is composed of multiple tasks that must communicate the necessary data to their successor in the chain and may be executed on different processing units. We call a related analysis that considers a chain of communicating tasks and determines the time needed to process data through the complete chain an *end-to-end analysis*. This chapter follows the notation and terminology introduced in Section 3.2.

Starting in 2007 and more frequently since 2016, the end-to-end latency of such data flows has been analyzed. However, most results are tied to specific task models and one specific end-to-end latency, and the generalization of different concepts and approaches has mostly been neglected. In this chapter, we tackle this gap by (i) proving fundamental

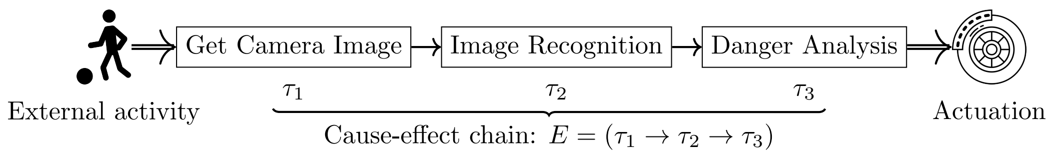


Figure 6.1.: Simplified automatic break system.

properties, that are applicable to almost any system model, and by (ii) developing a frame that can be used as a backbone to derive most analytical literature results. More specifically, this chapter is organized as follows:

- We present fundamental timing properties for end-to-end latency of cause-effect chains in Section 6.1. The fundamental properties are described with only a minimal set of restrictions and are therefore applicable to most system models.
- In Section 6.2, we discuss analytical bounds on the end-to-end latency of cause-effect chains. That includes the development of an analytical frame that can be applied universally to derive most analytical bounds from the literature. Furthermore, we present our novel analytical bounds.
- In Section 6.3, probabilistic metrics for the end-to-end latency are introduced and analyzed, considering two types of randomness: Response-time randomness and failure probabilities. A special focus is on the potential pitfalls that are inherent for the definition of probabilistic end-to-end latency.

6.1 Fundamental Properties

In this section, we detail two fundamental properties of the end-to-end latency. The first is a compositional property that allows to cut any cause-effect chain into smaller segments and analyze each of them separately. The second is the equivalence between the two metrics Maximum Reaction Time (MRT) and Maximum Data Age (MDA). Although those two metrics were believed to be inherently different, we show that they are actually the same. The mentioned properties are fundamental because they hold for any communication and scheduling mechanism that adheres to the two requirements (R1) and (R2) stated in Section 3.2.1. Hence, it covers periodic and sporadic tasks under implicit communication and Logical Execution Time (LET), and even tasks scheduled and communicating under more specialized mechanisms, like those using Robot Operating System 2 (ROS2).

For convenience, we introduce the notation $\preccurlyeq$ for the ordering of jobs of the same task, and use it to describe the fundamental behavior of immediate forward and immediate backward job chains. Specifically, Lemma 6.2 serves as a backbone for several analytical results in this chapter, especially in Sections 6.1.1, 6.1.2, 6.2.1 and 6.2.3.

Definition 6.1. *For two jobs $\tau(i)$ and $\tau(j)$ of the same task τ, we denote by $\tau(i) \preccurlyeq \tau(j)$ or $\tau(j) \succcurlyeq \tau(i)$ the case that $i \leq j$. Furthermore, $\tau(i) \prec \tau(j)$ or $\tau(j) \succ \tau(i)$ if $i < j$.*

Lemma 6.2. *Let $c = (J_1, \dots, J_n)$ and $c' = (J'_1, \dots, J'_n)$ be two job chains for E in ω.*

(i) If c is immediate forward and $J_i \preccurlyeq J'_i$ for some $i \in \{1, \dots, n\}$, then $J_j \preccurlyeq J'_j$ for all $j \in \{i, \dots, n\}$.

(ii) If c is immediate backward and $J_i \succcurlyeq J'_i$ for some $i \in \{1, \dots, n\}$, then $J_j \succcurlyeq J'_j$ for all $j \in \{1, \dots, i\}$.

Proof of Lemma 6.2. We prove (i) by induction over $j = i, \dots, n$.
Initial case ($j = i$): For $j = i$, $J_j \preccurlyeq J'_j$ by assumption.
Induction step ($j \mapsto j + 1$): If $J_j \preccurlyeq J'_j$ for $j \in \{i, \dots, n - 1\}$, then this means that the write-event of the job J_j is no later than the write-event of the job J'_j. Since the read-event of the job J'_{j+1} is no earlier than the write-event of J'_j (by definition of a job chain), the read-event of J'_{j+1} is also no earlier than the write-event of J_j. Since J_{j+1} is the earliest job with read-event no earlier than the write-event of J_j, we conclude that $J_{j+1} \preccurlyeq J'_{j+1}$.

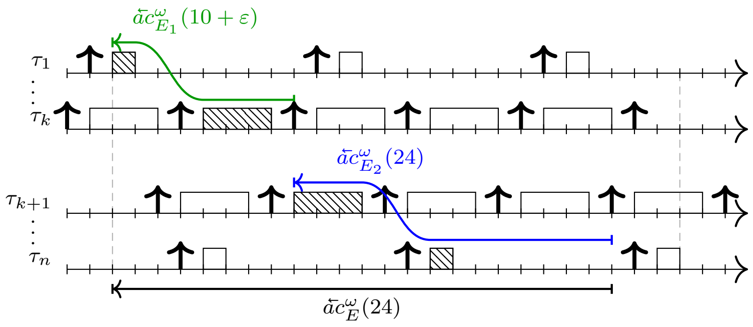
We prove (ii) by induction over $j = i, \dots, 1$.
Initial case ($j = i$): For $j = i$, $J'_j \preccurlyeq J_j$ by assumption.
Induction step ($j \mapsto j - 1$): If $J'_j \preccurlyeq J_j$ for $j \in \{i, \dots, 2\}$, then this means that the read-event of the job J'_j is no later than the read-event of the job J_j. Since the write-event of the job J'_{j-1} is no later than the read-event of J'_j (by definition of a job chain), the write-event of J'_{j-1} is also no later than the read-event of J_j. Since J_{j-1} is the latest job with write-event no later than the read-event of J_j, we conclude that $J'_{j-1} \preccurlyeq J_{j-1}$. □

6.1.1 Compositional Property

Usually, cause-effect chains are distributed over several components that require different treatments and analyses. The analysis of such potentially long cause-effect chains can be difficult and time-consuming. To mitigate this problem, in this section, we show that the end-to-end latency underlies a compositional property. More specifically, any cause-effect chain can be decomposed into smaller cause-effect chains, and adding up the end-to-end latency of all components yields a safe upper bound on the end-to-end latency of the full chain. After the decomposition of the cause-effect chain, the components only have to respect the local properties that they are executed on. Hence, by a smart decomposition the chain becomes much easier to analyze.

The results presented in this section appeared in RTAS 2021 [Gün+21a] and in a journal extension for ACM TECS in 2023 [Gün+23a]. However, the concept of *warm-up* has not been established at the date of publication. Instead, so-called *valid* job chains were used to constraint the observation window. The proofs of this section are adjusted to be compatible with the warm-up.

Theorem 6.3 (Cutting). *Let $E = (\tau_1 \to \dots \to \tau_n)$ be any cause-effect chain, decomposed into two components $E_1 = (\tau_1 \to \dots \to \tau_k)$ and $E_2 = (\tau_{k+1} \to \dots \to \tau_n)$. The end-to-end latency of the components E_1 and E_2 yields an upper bound on the end-to-end latency of*



$\overleftarrow{ac}^{\omega}_{E_1}(10+\varepsilon)$
τ_1
τ_k
$\overleftarrow{ac}^{\omega}_{E_2}(24)$
τ_{k+1}
τ_n
$\overleftarrow{ac}^{\omega}_{E}(24)$

Figure 6.2.: Cutting one immediate backward augmented job chain $\overleftarrow{ac}^{\omega}_{E}(24)$ into two as in the proof of Theorem 6.3 (Cutting).

E. That is,

$$\mathrm{MRT}(E,\omega) \leq \mathrm{MRT}(E_1,\omega) + \mathrm{MRT}(E_2,\omega) \tag{6.1}$$

$$\mathrm{MDA}(E,\omega) \leq \mathrm{MDA}(E_1,\omega) + \mathrm{MDA}(E_2,\omega) \tag{6.2}$$

for all $\omega \in \Omega$*. Consequently,* $\mathrm{MRT}(E) \leq \mathrm{MRT}(E_1) + \mathrm{MRT}(E_2)$ *and* $\mathrm{MDA}(E) \leq \mathrm{MDA}(E_1) + \mathrm{MDA}(E_2)$ *hold as well.*

The proof of the Cutting-Theorem relies on cutting immediate forward (backward, respectively) augmented job chains into smaller immediate forward (backward, respectively) augmented job chains, such that their combined length is at least the length of the initial immediate forward (backward, respectively) augmented job chain. Figure 6.2 shows the concept for immediate *backward* augmented job chains, assuming that jobs adhere to implicit communication. We see that $\ell(\overleftarrow{ac}^{\omega}_{E}(24)) \leq \ell(\overleftarrow{ac}^{\omega}_{E_1}(10+\varepsilon)) + \ell(\overleftarrow{ac}^{\omega}_{E_2}(24))$. The jobs that appear in the backward augmented job chain $\overleftarrow{ac}^{\omega}_{E}(24)$, marked with the pattern, are distributed among $\overleftarrow{ac}^{\omega}_{E_1}(10+\varepsilon)$ and $\overleftarrow{ac}^{\omega}_{E_2}(24)$. Only the events for external activity and actuation have to be determined properly. We note that the ε ensures that the immediate backward job chain of E_1 is considered in the calculation of the MDA, i.e., $z' > \mathrm{we}^{\omega}(\tau_k(W^{\omega}_{E,\tau_k}))$. For the decomposition of the immediate forward job chain we do not need this ε as we can see in the proof.

Proof. We start by proving Equation (6.1). Consider an immediate forward augmented job chain $\vec{ac}^{\omega}_{E}(z) = (z, J_1, \dots, J_n, \mathrm{we}^{\omega}(J_n))$ for cause-effect chain E in a system evolution $\omega \in \Omega$. We define the immediate forward augmented job chains:

$$\vec{ac}^{\omega}_{E_1}(z) = (z, J_1, \dots, J_k, \mathrm{we}^{\omega}(J_k)) \tag{6.3}$$

$$\vec{ac}^{\omega}_{E_2}(\mathrm{we}^{\omega}(J_k)) = (\mathrm{we}^{\omega}(J_k), J_{k+1}, \dots, J_n, \mathrm{we}^{\omega}(J_n)) \tag{6.4}$$

Then $\ell(\vec{ac}^{\omega}_{E}(z)) \leq \ell(\vec{ac}^{\omega}_{E_1}(z)) + \ell(\vec{ac}^{\omega}_{E_2}(\mathrm{we}^{\omega}(J_k)))$. We must check that if $\vec{ac}^{\omega}_{E}(z)$ is considered in the calculation of $\mathrm{MRT}(E,\omega)$, then also $\vec{ac}^{\omega}_{E_1}(z)$ and $\vec{ac}^{\omega}_{E_2}(\mathrm{we}^{\omega}(J_k))$ are considered in the calculation of $\mathrm{MRT}(E_1,\omega)$ and $\mathrm{MRT}(E_2,\omega)$, respectively. That is, if $z > \mathrm{re}^\omega(\tau_1(W^\omega_{E,\tau_1}))$, then

$$z > \mathrm{re}^\omega(\tau_1(W^\omega_{E_1,\tau_1})) \tag{6.5}$$
$$\mathrm{we}^\omega(J_k) > \mathrm{re}^\omega(\tau_{k+1}(W^\omega_{E_2,\tau_{k+1}})) \tag{6.6}$$

must hold as well.

For Equation (6.5): We know that $(\tau_1(W^\omega_{E,\tau_1}),\dots,\tau_n(W^\omega_{E,\tau_n}))$ is an immediate backward job chain for E in ω. Therefore, the first k jobs form an immediate backward job chain $(\tau_1(W^\omega_{E,\tau_1}),\dots,\tau_k(W^\omega_{E,\tau_k}))$ for E_1 in ω. Since by definition $\tau_1(W^\omega_{E_1,\tau_1})$ is the first job of the *first* immediate backward job chain of E_1 in ω, we know that $W^\omega_{E_1,\tau_1} \leq W^\omega_{E,\tau_1}$. Hence, $\mathrm{re}^\omega(W^\omega_{E_1,\tau_1}) \leq \mathrm{re}^\omega(W^\omega_{E,\tau_1}) < z$, i.e., Equation (6.5) holds.

For Equation (6.6): We prove that equation by contradiction. To that end, we assume that $\mathrm{we}^\omega(J_k) \leq \mathrm{re}^\omega(\tau_{k+1}(W^\omega_{E_2,\tau_{k+1}}))$. Since $(\tau_1(W^\omega_{E,\tau_1}),\dots,\tau_n(W^\omega_{E,\tau_n}))$ is an immediate backward job chain, we know that the last $(n-k)$ tasks form an immediate backward job chain $(\tau_{k+1}(W^\omega_{E,\tau_{k+1}}),\dots,\tau_n(W^\omega_{E,\tau_n}))$ for E_2 in ω. Since $\tau_{k+1}(W^\omega_{E_2,\tau_{k+1}})$ is the first job of the *first* immediate backward job chain for E_2 in ω, we know that $W^\omega_{E_2,\tau_{k+1}} \leq W^\omega_{E,\tau_{k+1}}$. Therefore, we have $\mathrm{we}^\omega(J_k) \leq \mathrm{re}^\omega(\tau_{k+1}(W^\omega_{E_2,\tau_{k+1}})) \leq \mathrm{re}^\omega(\tau_{k+1}(W^\omega_{E,\tau_{k+1}}))$. Since the job chain $(\tau_1(W^\omega_{E,\tau_1}),\dots,\tau_n(W^\omega_{E,\tau_n}))$ is immediate backward, we obtain that $\mathrm{we}^\omega(J_k) \leq \mathrm{we}^\omega(\tau_k(W^\omega_{E,\tau_k}))$, i.e., $J_k \preccurlyeq \tau_k(W^\omega_{E,\tau_k})$. By Lemma 6.2, also $J_1 \preccurlyeq \tau_1(W^\omega_{E,\tau_1})$. Since, $z \leq \mathrm{re}^\omega(J_1)$ by definition, we obtain $z \leq \mathrm{re}^\omega(\tau_1(W^\omega_{E,\tau_1}))$ which contradicts that $z > \mathrm{re}^\omega(\tau_1(W^\omega_{E,\tau_1}))$. Hence, Equation (6.6) holds.

We conclude $\ell(\vec{ac}^\omega_E(z)) \leq \ell(\vec{ac}^\omega_{E_1}(z)) + \ell(\vec{ac}^\omega_{E_2}(\mathrm{we}^\omega(J_k))) \leq \mathrm{MRT}(E_1,\omega) + \mathrm{MRT}(E_2,\omega)$. Since this holds for all $z > \mathrm{re}^\omega(\tau_1(W^\omega_{E,\tau_1}))$, it also holds for the supremum, i.e., we obtain $\mathrm{MRT}(E,\omega) \leq \mathrm{MRT}(E_1,\omega) + \mathrm{MRT}(E_2,\omega)$. □ Equation (6.1)

Equation (6.2) is proven analogously. Here, the length of any immediate backward augmented job chain $\overleftarrow{ac}^\omega_E(z') = (\mathrm{re}^\omega(J_1), J_1, \dots, J_n, z')$ for E is upper bounded by $\ell(\overleftarrow{ac}^\omega_E(z')) \leq \ell(\overleftarrow{ac}^\omega_{E_1}(\mathrm{re}^\omega(J_{k+1})+\varepsilon)) + \ell(\overleftarrow{ac}^\omega_{E_2}(z'))$, where

$$\overleftarrow{ac}^\omega_{E_1}(\mathrm{re}^\omega(J_{k+1})+\varepsilon) = (\mathrm{re}^\omega(J_1), J_1, \dots, J_k, \mathrm{re}^\omega(J_{k+1})+\varepsilon) \tag{6.7}$$
$$\overleftarrow{ac}^\omega_{E_2}(z') = (\mathrm{re}^\omega(J_{k+1}), J_{k+1}, \dots, J_n, z') \tag{6.8}$$

with $0 < \varepsilon$ sufficiently small to ensure that $(J_1,\dots,J_k)$ is part of $\overleftarrow{ac}^\omega_{E_1}(\mathrm{re}^\omega(J_{k+1})+\varepsilon)$, i.e., ε is chosen such that

$$\mathrm{re}^\omega(J_{k+1}) + \varepsilon < \min\{\mathrm{we}^\omega(\tau_k(j)) \mid j \in \mathbb{N},\ \mathrm{we}^\omega(\tau_k(j)) > \mathrm{re}^\omega(J_{k+1})\}. \tag{6.9}$$

We note that the minimum on the right-hand side exists due to restriction (R2). Again, we need to ensure that if $z' > \mathrm{we}^\omega(\tau_n(W^\omega_{E,\tau_n}))$, then

$$\mathrm{re}^\omega(J_{k+1}) + \varepsilon > \mathrm{we}^\omega(\tau_k(W^\omega_{E_1,\tau_k})) \tag{6.10}$$
$$z' > \mathrm{we}^\omega(\tau_n(W^\omega_{E_2,\tau_n})) \tag{6.11}$$

holds as well.

For Equation (6.10): Since $(J_1, \dots, J_n)$ is an immediate backward job chain for E in ω, we know that $(J_1, \dots, J_k)$ is an immediate backward job chain for E_1 in ω. Therefore, $\mathrm{we}^\omega(\tau_k(W^\omega_{E_1,\tau_k})) \leq \mathrm{we}^\omega(J_k)$. Moreover, $\mathrm{we}^\omega(J_k) \leq \mathrm{re}^\omega(J_{k+1}) < \mathrm{re}^\omega(J_{k+1}) + \varepsilon$. Hence, Equation (6.10) holds.

For Equation (6.11): Since $(\tau_1(W^\omega_{E,\tau_1}), \dots, \tau_n(W^\omega_{E,\tau_n}))$ is an immediate backward job chain for E in ω, the last $(n-k)$ jobs $(\tau_{k+1}(W^\omega_{E,\tau_{k+1}}), \dots, \tau_n(W^\omega_{E,\tau_n}))$ form an immediate backward job chain for E_2 in ω. Therefore, $\mathrm{we}^\omega(\tau_n(W^\omega_{E_2,\tau_n})) \leq \mathrm{we}^\omega(\tau_n(W^\omega_{E,\tau_n}))$. Moreover, $\mathrm{we}^\omega(\tau_n(W^\omega_{E,\tau_n})) < z$. Hence, Equation (6.11) holds.

We conclude that $\ell(\overleftarrow{ac}^\omega_E(z')) \leq \ell(\overleftarrow{ac}^\omega_{E_1}(\mathrm{re}^\omega(J_{k+1}) + \varepsilon)) + \ell(\overleftarrow{ac}^\omega_{E_2}(z')) \leq \mathrm{MDA}(E_1, \omega) + \mathrm{MDA}(E_2, \omega)$. Since this holds for all $z' > \mathrm{we}^\omega(\tau_n(W^\omega_{E,\tau_n}))$, it also holds for the supremum, i.e., $\mathrm{MDA}(E, \omega) \leq \mathrm{MDA}(E_1, \omega) + \mathrm{MDA}(E_2, \omega)$. □ Equation (6.2)

Applying the supremum over all $\omega \in \Omega$ also proves the bounds for overall latency. □

Although the theorem is only formulated for two components, applying the theorem several times allows a decomposition into more components. More specifically, if E can be decomposed into multiple cause-effect chains $E_1, \dots, E_N$, then

$$\mathrm{MRT}(E, \omega) \leq \sum_{j=1}^{N} \mathrm{MRT}(E_j, \omega) \qquad \text{and} \qquad \mathrm{MDA}(E, \omega) \leq \sum_{j=1}^{N} \mathrm{MDA}(E_j, \omega). \tag{6.12}$$

Consequently, $\mathrm{MRT}(E) \leq \sum_{j=1}^N \mathrm{MRT}(E_j)$ and $\mathrm{MDA}(E) \leq \sum_{j=1}^N \mathrm{MDA}(E_j)$.

The theorem can also be extended for comply with the Maximum Reduced Data Age (MRDA) and Maximum Reduced Reaction Time (MRRT). That is, by a similar proof, we obtain

$$\mathrm{MRDA}(E, \omega) \leq \mathrm{MDA}(E_1, \omega) + \mathrm{MRDA}(E_2, \omega) \tag{6.13}$$

for the MRDA, and

$$\mathrm{MRRT}(E, \omega) \leq \mathrm{MRRT}(E_1, \omega) + \mathrm{MRT}(E_2, \omega) \tag{6.14}$$

for the MRRT.

6.1.2 Equivalence of Latencies

Although multiple approaches to calculate or bound MRT or MDA have been provided [Bec+16a; Bec+16b; Ben+02; Dav+07; Dür+19; Fei+09; FBP17; Gün+21a; KBS18; Raj+10; Sch+18a], the relation between the MRT and MDA values is less analyzed. This is kind of surprising, since answering the questions *if, how, and in which scenarios the MRT and MDA values are related* is interesting—both from a practical perspective (since it may be sufficient to analyze one metric instead of two) and from a research perspective (since analysis methods for one metric may also be applied when analyzing the other).

Starting in 2019 the analytical relation between MRT and MDA has been further explored. More specifically, Dürr et al. [Dür+19] showed that an analytical bound for

MRT in sporadic systems also holds for the MRDA. Furthermore, Günzel et al. [Gün+21a; Gün+23a] showed that the exact MRT is an upper bound for the MDA under a general definition. Empirical observations suggest an even stronger relation. More specifically, the AUTOSAR Timing Extensions [AUT22] state that "*without over- and undersampling, age and reaction are the same*" [AUT22, Section 3.6.2, p. 114]. However, while this observation seems to imply that MRT and MDA can differ for systems with over- or undersampling, recent measurements in ROS2 show that the observed MRT and MDA always coincide [Tep+22]. Hence, it is unclear in which scenarios MRT and MDA coincide. Even more, while both observations suggest a strong relation between MRT and MDA, no proof for such a relation is provided.

In this section, we further investigate MRT and MDA through analytical discussion to determine *if, how, and in which scenarios the MRT and MDA values are related.* Specifically, we formally prove that they are equivalent after the warm-up period, i.e., after the data passes the complete cause-effect chain once. This insight allows the verification of timing constraints for both metrics at the same time. Moreover, analytical results in the literature for one metric can be utilized for the other one. The section is organized as follows:

- In Section 6.1.2.1, we define p-partitioned job chains, with $p \in \{1, \dots, n\}$, as a key ingredient to prove the equivalence of MRT and MDA. We show that MRT and MDA can be expressed as 1-partitioned job chains and n-partitioned job chains, respectively.
- In Section 6.1.2.2, we discuss the equivalence between MRT and MDA using partitioned job chains, and show that the timing behavior is independent of p, i.e., any arbitrary p can be chosen to compute MDA and MRT.
- The implication of our results in practice is discussed in Section 6.1.2.3.
- To validate our theoretical results, we examine MRT, MDA on a ROS2 system with non-periodic tasks under implicit communication in Section 6.1.2.4.

We show the equivalence of MRT and MDA while making only very few non-restrictive assumptions regarding tasks, communication, and scheduling model. Therefore, our results apply for a large variety of systems. Specifically, our results can be applied for but are not limited to periodic or sporadic tasks and implicit communication or LET. The underlying scheduler may be (i) a Job-level Fixed-Priority (J-FP) or Task-level Fixed-Priority (T-FP) scheduler (e.g., Earliest-Deadline-First (EDF) or Rate Monotonic (RM), respectively), (ii) work-conserving or non-work-conserving, and (iii) preemptive or non-preemptive. Furthermore, jobs may be executed on different processing units (e.g., on different Electronic Control Units (ECUs)). Due to the equivalence, no distinction between MRT and MDA is necessary. Hence, for the remainder of this dissertation

$$\mathrm{Lat}(E,\omega) \coloneqq \mathrm{MRT}(E,\omega) = \mathrm{MDA}(E,\omega) \tag{6.15}$$

$$\mathrm{Lat}(E) \coloneqq \mathrm{MRT}(E) = \mathrm{MDA}(E) \tag{6.16}$$

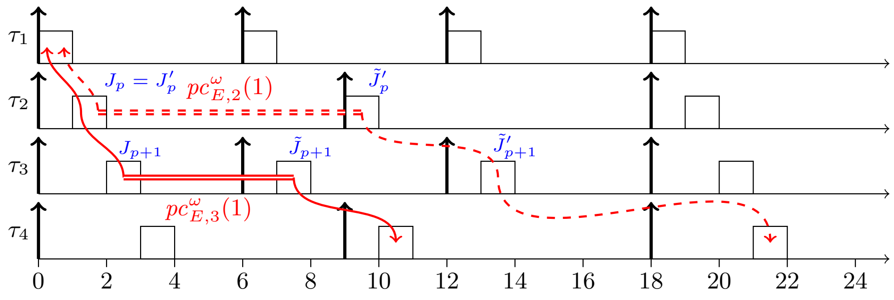


Figure 6.3.: Four tasks with cause-effect chain $(\tau_1 \to \tau_2 \to \tau_3 \to \tau_4)$ communicating via implicit communication. The red line depicts the chain $pc^{\omega}_{E,3}(1)$ whereas the dashed red line depicts $pc^{\omega}_{E,2}(1)$. The length of $pc^{\omega}_{E,3}(1)$ is upper bounded by the length of $pc^{\omega}_{E,2}(1)$. The blue job annotations illustrate the proof of Lemma 6.6 ($\geq$-relation).

denotes the end-to-end latency in general whenever the equivalence applies. The results presented in this section appeared in ECRTS 2023 [Gün+23b], where we received the *best paper award* for our work.

6.1.2.1 Partitioned Job chains

In this section, for a given cause-effect chain $E = (\tau_1 \to \cdots \to \tau_n)$, we define p-partitioned job chains ($p \in \{1, 2, \dots, n\}$). We show that these p-partitioned job chains allow MRT and MDA definitions that are equivalent to the ones based on augmented job chains stated in Definition 3.7. In particular, considering 1-partitioned job chains is equivalent to considering immediate forward augmented job chains, and considering n-partitioned job chain is equivalent to considering immediate backward augmented job chains. In Section 6.1.2.2, we utilize these p-partitioned job chains to show the equivalence between MRT and MDA by discussing the difference of p-partitioned and $(p+1)$-partitioned job chains for $p = 1, \dots, n-1$.

A p-partitioned job chain is a combination of (i) two subsequent jobs J_p and $\tilde{J}_p$ of task τ_p, (ii) an immediate backward job chain that starts at J_p, and (iii) an immediate forward job chain that starts at $\tilde{J}_p$. In Figure 6.3, the dashed chain $pc^{\omega}_{E,2}(1)$ is a 2-partitioned job chain comprising an immediate backward job chain with last entry $\tau_2(1)$, and an immediate forward job chain with first entry $\tau_2(2)$.

We start by formally defining partitioned job chains.

Definition 6.4 (Partitioned job chain)**.** *Let $p \in \{1, \dots, n\}$ and $m \in \mathbb{N}$. We define the m-th p-partitioned job chain $pc^{\omega}_{E,p}(m)$ by*

$$pc^{\omega}_{E,p}(m) = (J_1, \dots, J_p, \tilde{J}_p, \dots, \tilde{J}_n) \tag{6.17}$$

if the following conditions are satisfied:

- *J_p is the m-th job of τ_p, and $\tilde{J}_p$ is the $(m+1)$-th job of τ_p.*

- *$(J_1, \dots, J_p)$ is an immediate backward job chain for $(\tau_1 \to \dots \to \tau_p)$.*
- *$(\tilde{J}_p, \dots, \tilde{J}_n)$ is an immediate forward job chain for $(\tau_p \to \dots \to \tau_n)$.*

We note that $pc^\omega_{E,p}(m)$ exists if and only if the immediate backward job chain $(J_1, \dots, J_p)$ exists. The length of a partitioned job chain is $\ell(J_1, \dots, J_p, \tilde{J}_p, \dots, \tilde{J}_n) = \mathrm{we}(\tilde{J}_n) - \mathrm{re}(J_1)$.

Since $pc^\omega_{E,1}(m)$ is composed of the forward chain starting at $\tilde{J}_1 = \tau_1(m+1)$ and one additional job J_1 at the beginning (accounting for the external activity between $\mathrm{re}^\omega(\tau_1(m))$ and $\mathrm{re}^\omega(\tau_1(m+1))$), it is equivalent to $\vec{ac}^\omega_E(z)$ with $z \in (\mathrm{re}^\omega(\tau_1(m)), \mathrm{re}^\omega(\tau_1(m+1))]$. Since $pc^\omega_{E,n}(m)$ is composed of the backward job chain with last job $\tau_n(m)$ and one additional job at the end (accounting for the actuation between $\mathrm{we}^\omega(\tau_n(m))$ and $\mathrm{we}^\omega(\tau_n(m+1))$), it is equivalent to $\overleftarrow{ac}^\omega_E(z')$ with $z' \in [\mathrm{we}^\omega(\tau_n(m)), \mathrm{we}^\omega(\tau_n(m+1)))$. In the following we prove that the MRT and MDA can be expressed using partitioned job chains.

Theorem 6.5 (MRT and MDA by partitioned job chains)**.** *The MRT and MDA of cause-effect chain $E = (\tau_1 \to \dots \to \tau_n)$ in system evolution $\omega \in \Omega$ can be expressed by partitioned job chains as follows:*

$$\mathrm{MRT}(E, \omega) = \sup_{m \geq W^\omega_{E,\tau_1}} \ell(pc^\omega_{E,1}(m)) \tag{6.18}$$

$$\mathrm{MDA}(E, \omega) = \sup_{m \geq W^\omega_{E,\tau_n}} \ell(pc^\omega_{E,n}(m)) \tag{6.19}$$

Proof. By Definition 3.7, we know that $\mathrm{MRT}(E, \omega) = \sup_{z > \mathrm{re}^\omega(\tau_1(W^\omega_{E,\tau_1}))} \ell(\vec{ac}^\omega_E(z))$. This can be formulated as

$$\mathrm{MRT}(E, \omega) = \sup_{m \geq W^\omega_{E,\tau_1}} \sup_{z \in (\mathrm{re}^\omega(\tau_1(m)), \mathrm{re}^\omega(\tau_1(m+1))]} \ell(\vec{ac}^\omega_E(z)). \tag{6.20}$$

Since $\sup_{z \in (\mathrm{re}^\omega(\tau_1(m)), \mathrm{re}^\omega(\tau_1(m+1))]} \ell(\vec{ac}^\omega_E(z)) = \ell(pc^\omega_{E,1}(m))$, we obtain Equation (6.18).

By Definition 3.7, we know that $\mathrm{MDA}(E, \omega) = \sup_{z' > \mathrm{we}^\omega(\tau_n(W^\omega_{E,\tau_n}))} \ell(\overleftarrow{ac}^\omega_E(z'))$. Furthermore, for a sufficiently small $\varepsilon > 0$, we have

$$\ell(\overleftarrow{ac}^\omega_E(\mathrm{we}^\omega(\tau_n(W^\omega_{E,\tau_n})))) \leq \ell(\overleftarrow{ac}^\omega_E(\mathrm{we}^\omega(\tau_n(W^\omega_{E,\tau_n})) + \varepsilon)). \tag{6.21}$$

Therefore, including $z' = \mathrm{we}^\omega(\tau_n(W^\omega_{E,\tau_n}))$ in the supremum makes no difference, i.e., we obtain that $\mathrm{MDA}(E, \omega) = \sup_{z' \geq \mathrm{we}^\omega(\tau_n(W^\omega_{E,\tau_n}))} \ell(\overleftarrow{ac}^\omega_E(z'))$. We reformulate the MDA as

$$\mathrm{MDA}(E, \omega) = \sup_{m \geq W^\omega_{E,\tau_n}} \sup_{z' \in [\mathrm{we}^\omega(\tau_n(m)), \mathrm{we}^\omega(\tau_n(m+1)))} \ell(\overleftarrow{ac}^\omega_E(z')). \tag{6.22}$$

By using that $\sup_{z' \in [\mathrm{we}^\omega(\tau_n(m)), \mathrm{we}^\omega(\tau_n(m+1)))} \ell(\overleftarrow{ac}^\omega_E(z')) = \ell(pc^\omega_{E,n}(m))$, we obtain the result of Equation (6.19). □

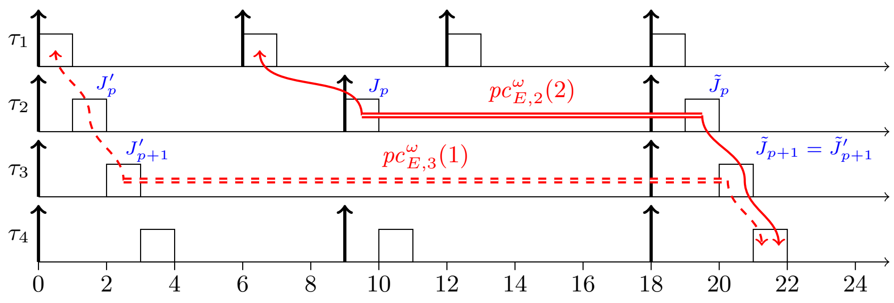


Figure 6.4.: Four tasks with cause-effect chain $(\tau_1 \to \tau_2 \to \tau_3 \to \tau_4)$ communicating via implicit communication. The red line depicts the chain $pc^{\omega}_{E,2}(2)$ whereas the dashed red line depicts $pc^{\omega}_{E,3}(1)$. The length of $pc^{\omega}_{E,2}(2)$ is upper bounded by the length of $pc^{\omega}_{E,3}(1)$. The blue job annotations illustrate the proof of Lemma 6.6 ($\geq$-relation).

6.1.2.2 Equivalence of Maximum Reaction Time and Maximum Data Age

In this section, we show the equivalence of Maximum Reaction Time (MRT) and Maximum Data Age (MDA). More precisely, we prove that, for a given cause-effect chain E, the maximum length of a p-partitioned job chain is the same for all $p \in \{1, \dots, n\}$. Since MRT and MDA can be expressed by 1-partitioned and by n-partitioned job chains, respectively, this directly shows the equivalence of MRT and MDA.

We prove that the maximum length of a p-partitioned job chain is the same for all $p \in \{1, \dots, n\}$ in two steps: (i) We show that for all $p \in \{n, \dots, 2\}$ the length of every p-partitioned job chain is upper bounded by the length of a $(p-1)$ partitioned job chain. This scenario is depicted in Figure 6.3, where the length of $pc^{\omega}_{E,3}(1)$ is upper bounded by the length of $pc^{\omega}_{E,2}(1)$. (ii) Conversely, we show that for all $p \in \{1, \dots, n-1\}$ the length of every p-partitioned job chain is upper bounded by the length of a $(p+1)$-partitioned job chain. This is depicted in Figure 6.4, where the length of $pc^{\omega}_{E,2}(2)$ is upper bounded by the length of $pc^{\omega}_{E,3}(1)$.

One problem we have to consider during step (ii) is that some $(p+1)$-partitioned job chains may not be fully constructible. For example, if the first job of τ_1 in Figure 6.4 would be missing, then $pc^{\omega}_{E,3}(1)$ could not be fully constructed and there would be no 3-partitioned job chain which provides an upper bound on the length of $pc^{\omega}_{E,2}(2)$. This, however, does not impact our result, since we show that this scenario never occurs for p-partitioned job chains $pc^{\omega}_{E,p}(m)$ after the warm-up, i.e., if $m \geq W^{\omega}_{E,\tau_p}$ with W^{ω}_{E,τ_p} from Definition 3.6.

The following lemma indicates that p-partitioned job chains for different p can be used equivalently for the computation of MDA and MRT according to their description by partitioned job chains from Theorem 6.5.

Lemma 6.6. *For all $p \in \{1, \dots, n-1\}$, we have*

$$\sup\left\{\ell(pc^{\omega}_{E,p}(m)) \,\middle|\, m \geq W^{\omega}_{E,\tau_p}\right\} = \sup\left\{\ell(pc^{\omega}_{E,p+1}(m)) \,\middle|\, m \geq W^{\omega}_{E,\tau_{p+1}}\right\}. \tag{6.23}$$

The proof relies on the fundamental properties of immediate forward and immediate backward job chains (which are part of partitioned job chains) formalized in Lemma 6.2. We split the proof of Lemma 6.6 into two steps, showing the $\geq$-relation and the $\leq$-relation, from which the equality directly follows.

To proof idea for the $\geq$-relation is as follows. First, we pick any $(p+1)$-partitioned and the related p-partitioned job chain that have the job of τ_p in common. Second, we show (i) that their immediate backward job chains both end at the same job of τ_1, and (ii) that the immediate forward job chain related to the $p+1$-partitioned job chain ends no later than the one related to the p-partitioned job chain. For instance, in Figure 6.3, $pc^{\omega}_{E,3}(1)$ and $pc^{\omega}_{E,2}(1)$ have job $\tau_2(1)$ in common, both immediate backwards job chains end at $\tau_1(1)$, and $pc^{\omega}_{E,3}(1)$ ends at $\tau_4(2)$ which is no later than the end of $pc^{\omega}_{E,2}(1)$ at $\tau_4(3)$. Hence, $\ell(pc^{\omega}_{E,2}(1)) \geq \ell(pc^{\omega}_{E,3}(1))$.

Proof of Lemma 6.6, $\geq$-relation. Let $m \geq W^{\omega}_{E,\tau_{p+1}}$. We denote the jobs of the partitioned job chain $pc^{\omega}_{E,p+1}(m)$ by $pc^{\omega}_{E,p+1}(m) = (J_1, \dots, J_p, J_{p+1}, \tilde{J}_{p+1}, \dots, \tilde{J}_n)$. Let $\xi \in \mathbb{N}$ such that J_p is the ξ-th job of τ_p, i.e., $\tau_p(\xi) = J_p$. We prove that the length of $pc^{\omega}_{E,p+1}(m)$ is upper bounded by the length of $pc^{\omega}_{E,p}(\xi) = (J'_1, \dots, J'_p, \tilde{J}'_p, \dots, \tilde{J}'_n)$ by showing $\mathrm{we}(\tilde{J}'_n) - \mathrm{re}(J'_1) \geq \mathrm{we}(\tilde{J}_n) - \mathrm{re}(J_1)$ in two substeps: First, we show that J_1 and J'_1 are the same (i.e., $J_1 = J'_1$), and second, that $pc^{\omega}_{E,p+1}(m)$ finishes no later than $pc^{\omega}_{E,p}(\xi)$ (i.e., $\tilde{J}_n \preccurlyeq \tilde{J}'_n$). Important jobs of this proof are illustrated in Figure 6.3.

Step 1 ($J_1 = J'_1$): By definition, $J_p = J'_p$. Since $(J_1, \dots, J_p)$ and $(J'_1, \dots, J'_p)$ are immediate backward job chains with the same last entry, they coincide. Hence, $J_1 = J'_1$.

Step 2 ($\tilde{J}_n \preccurlyeq \tilde{J}'_n$): Since $(J_1, \dots, J_{p+1})$ is an immediate backward job chain, this means that $J_p = \tau_p(\xi)$ is the latest job with write-event no later than the read-event of J_{p+1}. Therefore, the write-event of the subsequent job of the same task, which is $\tau_p(\xi+1) = \tilde{J}'_p$, must be after the read-event of J_{p+1}, i.e., $\mathrm{re}(J_{p+1}) < \mathrm{we}(\tilde{J}'_p)$.

The job $\tilde{J}_{p+1}$ of τ_{p+1} subsequent to J_{p+1} either has its read-event before $\mathrm{we}(\tilde{J}'_p)$ as well (i.e., $\mathrm{re}(\tilde{J}_{p+1}) < \mathrm{we}(\tilde{J}'_p)$) or is the earliest job of τ_{p+1} with $\mathrm{re}(\tilde{J}_{p+1}) \geq \mathrm{we}(\tilde{J}'_p)$. In both cases $\tilde{J}_{p+1} \preccurlyeq \tilde{J}'_{p+1}$ as $\mathrm{re}(\tilde{J}'_{p+1}) \geq \mathrm{we}(\tilde{J}'_p)$. Since $(\tilde{J}_{p+1}, \dots, \tilde{J}_n)$ and $(\tilde{J}'_{p+1}, \dots, \tilde{J}'_n)$ are both immediate forward job chains and $\tilde{J}_{p+1} \preccurlyeq \tilde{J}'_{p+1}$, we know $\tilde{J}_n \preccurlyeq \tilde{J}'_n$ by Lemma 6.2.

Combining Step 1 and 2, we get $\ell(pc^{\omega}_{E,p+1}(m)) = \mathrm{we}(\tilde{J}_n) - \mathrm{re}(J_1) \leq \mathrm{we}(\tilde{J}'_n) - \mathrm{re}(J'_1) = \ell(pc^{\omega}_{E,p}(\xi))$. Since $(J_1, \dots, J_{p+1})$ is an immediate backward job chain and $\tau_{p+1}(W^{\omega}_{E,\tau_{p+1}}) \preccurlyeq J_{p+1}$, by Lemma 6.2, $\tau_p(W^{\omega}_{E,\tau_p}) \preccurlyeq J_p$ holds as well. As Step 1 shows $J_p = J'_p = \tau_p(\xi)$, we conclude that $\xi \geq W^{\omega}_{E,\tau_p}$. Hence, $\ell(pc^{\omega}_{E,p+1}(m)) \leq \ell(pc^{\omega}_{E,p}(\xi)) \leq \sup\left\{\ell(pc^{\omega}_{E,p}(\eta)) \,\middle|\, \eta \geq W^{\omega}_{E,\tau_p}\right\}$.

Since $m \geq W^{\omega}_{E,\tau_{p+1}}$ is arbitrarily chosen, we have $\sup\left\{\ell(pc^{\omega}_{E,p+1}(m)) \,\middle|\, m \geq W^{\omega}_{E,\tau_{p+1}}\right\} \leq \sup\left\{\ell(pc^{\omega}_{E,p}(\eta)) \,\middle|\, \eta \geq W^{\omega}_{E,\tau_p}\right\}$, i.e., the relation $\geq$ holds for Equation (6.23) $\square$

Similarly, we show the $\leq$-relation by picking any p-partitioned and the related $(p+1)$-partitioned job chain that have the job of τ_{p+1} in common. Second, we show (i) that their immediate forward job chains both end at the same job of τ_n, and (ii) that the immediate backward job chain related to the $(p+1)$-partitioned job chain ends no later than the one related to the p-partitioned job chain. For instance, in Figure 6.4, $pc^{\omega}_{E,3}(1)$ and $pc^{\omega}_{E,2}(2)$ have job $\tau_3(2)$ in common, both immediate forward job chains end at $\tau_4(2)$, and $pc^{\omega}_{E,3}(1)$ begins at $\tau_1(1)$ which is no later than the beginning of $pc^{\omega}_{E,2}(2)$ at $\tau_1(2)$. Hence, $\ell(pc^{\omega}_{E,2}(2)) \leq \ell(pc^{\omega}_{E,3}(1))$.

Proof of Lemma 6.6, $\leq$-relation. Let $m \geq W^{\omega}_{E,\tau_p}$. We denote the jobs of the partitioned job chain $pc^{\omega}_{E,p}(m)$ by $pc^{\omega}_{E,p}(m) = (J_1, \dots, J_p, \tilde{J}_p, \tilde{J}_{p+1}, \dots, \tilde{J}_n)$. Let $\xi \in \mathbb{N}$ such that $\tilde{J}_{p+1}$ is the ξ-th job of τ_{p+1}, i.e., $\tau_{p+1}(\xi) = \tilde{J}_{p+1}$.

As an additional step, we must show that $\xi - 1 \geq W^{\omega}_{E,\tau_{p+1}}$ holds for the previous job of $\tau_{p+1}(\xi - 1)$. Assume for contradiction that $\xi - 1 < W^{\omega}_{E,\tau_{p+1}}$. Then $\xi \leq W^{\omega}_{E,\tau_{p+1}}$. Therefore, $\tilde{J}_{p+1} \preccurlyeq \tau_{p+1}(W^{\omega}_{E,\tau_{p+1}})$ holds. Since $(\tau_1(W^{\omega}_{E,\tau_1}), \dots, \tau_n(W^{\omega}_{E,\tau_n}))$ is an immediate backward job chain, by Lemma 6.2 we obtain that $\tilde{J}_p \preccurlyeq \tau_p(W^{\omega}_{E,\tau_p})$. Furthermore, since $\tilde{J}_p = \tau_p(m+1)$ by definition of $pc^{\omega}_{E,p}(m)$, we obtain $m + 1 \leq W^{\omega}_{E,\tau_p}$, i.e., $m < W^{\omega}_{E,\tau_p}$ which contradicts that $m \geq W^{\omega}_{E,\tau_p}$.

Since $\xi - 1 \geq W^{\omega}_{E,\tau_{p+1}}$ and for $W^{\omega}_{E,\tau_{p+1}}$ an immediate backward job chain exists, the immediate backward job chain of $(\tau_1 \to \dots \to \tau_{p+1})$ with last job $\tau_{p+1}(\xi - 1)$ can be fully constructed and $pc^{\omega}_{E,p+1}(\xi - 1)$ exists. We now prove that the length of $pc^{\omega}_{E,p}(m)$ is upper bounded by the length of $pc^{\omega}_{E,p+1}(\xi - 1) = (J'_1, \dots, J'_{p+1}, \tilde{J}'_{p+1}, \dots, \tilde{J}'_n)$, i.e., we show that $\mathrm{we}(\tilde{J}'_n) - \mathrm{re}(J'_1) \geq \mathrm{we}(\tilde{J}_n) - \mathrm{re}(J_1)$. Specifically, we show $\tilde{J}_n = \tilde{J}'_n$ and $J'_1 \preccurlyeq J_1$ in two individual steps. Important jobs of this proof are illustrated in Figure 6.4.

Step 1 ($\tilde{J}'_n = \tilde{J}_n$): By definition, we have $\tilde{J}'_{p+1} = \tau_{p+1}(\xi) = \tilde{J}_{p+1}$. Since the immediate forward job chains $(\tilde{J}'_{p+1}, \dots, \tilde{J}'_n)$ and $(\tilde{J}_{p+1}, \dots, \tilde{J}_n)$ both have the same job as the first entry, these job chains coincide. In particular, $\tilde{J}'_n = \tilde{J}_n$.

Step 2 ($J'_1 \preccurlyeq J_1$): Since $(\tilde{J}_p, \dots, \tilde{J}_n)$ is an immediate forward job chain, the job $\tilde{J}_{p+1} = \tau_{p+1}(\xi)$ is the earliest job with read-event no earlier than $\mathrm{we}(\tilde{J}_p)$. Thus, the read-event of $\tau_{p+1}(\xi - 1) = J'_{p+1}$ must be before the write-event of $\tilde{J}_p$, i.e., $\mathrm{we}(\tilde{J}_p) > \mathrm{re}(J'_{p+1})$. For the job of τ_p previous to $\tilde{J}_p$, which is J_p, we either have $\mathrm{we}(J_p) > \mathrm{re}(J'_{p+1})$ as well, or J_p is the latest job of τ_p with $\mathrm{we}(J_p) \leq \mathrm{re}(J'_{p+1})$. In both cases $J'_p \preccurlyeq J_p$ because $\mathrm{we}(J'_p) \leq \mathrm{re}(J'_{p+1})$. Since $(J'_1, \dots, J'_p)$ and $(J_1, \dots, J_p)$ are both immediate backward job chains and $J'_p \preccurlyeq J_p$, we have $J'_1 \preccurlyeq J_1$ as well by Lemma 6.2.

We obtain $\ell(pc^{\omega}_{E,p}(m)) = \mathrm{we}(\tilde{J}_n) - \mathrm{re}(J_1) \leq \mathrm{we}(\tilde{J}'_n) - \mathrm{re}(J'_1) = \ell(pc^{\omega}_{E,p+1}(\xi - 1))$. We already showed that $\xi - 1 \geq W^{\omega}_{E,\tau_{p+1}}$, hence $\ell(pc^{\omega}_{E,p}(m)) \leq \ell(pc^{\omega}_{E,p+1}(\xi - 1)) \leq \sup\left\{\ell(pc^{\omega}_{E,p+1}(\eta)) \,\middle|\, \eta \geq W^{\omega}_{E,\tau_{p+1}}\right\}$.

Since $m \geq W^{\omega}_{E,\tau_p}$ is arbitrarily chosen, the relation $\leq$ holds for Equation (6.23) as $\sup\left\{\ell(pc^{\omega}_{E,p}(m)) \,\middle|\, m \geq W^{\omega}_{E,\tau_p}\right\} \leq \sup\left\{\ell(pc^{\omega}_{E,p+1}(\eta)) \,\middle|\, \eta \geq W^{\omega}_{E,\tau_{p+1}}\right\}$. □

We have shown that p-partitioned job chains for different p can be utilized equivalently. The equivalence of MRT and MDA follows directly by applying Lemma 6.6 multiple times.

Theorem 6.7 (Equivalence). *The MRT and MDA are equivalent, i.e.,*

$$\mathrm{MRT}(E,\omega) = \sup\left\{\ell(pc^{\omega}_{E,p}(m)) \,\middle|\, m \geq W^{\omega}_{E,\tau_p}\right\} = \mathrm{MDA}(E,\omega) \tag{6.24}$$

for all $p \in \{1,\dots,n\}$ *and all* $\omega \in \Omega$*. Furthermore,* $\mathrm{MRT}(E) = \mathrm{MDA}(E)$*.*

Proof. This is achieved by applying Lemma 6.6 $n-1$ times, i.e., for $p = 1,\dots,n-1$. □

The result of Theorem 6.7 can be applied to the MRDA and MRRT, introduced in Section 3.2.2, by using Lemma 3.8. In particular, for periodic or sporadic systems $\mathrm{MRT}(E,\omega) = \mathrm{MDA}(E,\omega) > \mathrm{MRDA}(E,\omega)$ holds, i.e., $\mathrm{MRT}(E,\omega) \neq \mathrm{MRDA}(E,\omega)$ in all system evolutions $\omega \in \Omega$. This means that, AUTOSAR considers MDA (instead of MRDA) since they observe that "*without over- and undersampling, age and reaction are the same*" [AUT22, Section 3.6.2, p. 114].

6.1.2.3 Implication in Practice

The result of Theorem 6.7 is much stronger than the observation made in the AUTOSAR timing specification [AUT22, Section 3.6.2, p. 114]. Specifically, our theorem shows that MRT and MDA are *always* the same, not only without over- and undersampling. Since we only assume that each task releases a countably infinite number of jobs, this equivalence holds for a variety of scenarios. This covers, for example:

- Systems with over- and undersampling
- Implicit communication or communication with LET
- T-FP or Task-level Dynamic-Priority (T-DP) schedulers (e.g., EDF and RM, respectively)
- Preemptive or non-preemptive scheduling
- Work-conserving or non-work-conserving scheduling
- Periodic or sporadic task systems
- Synchronized or asynchronized distributed systems
- Tasks scheduled by ROS2, as demonstrated in Section 6.1.2.4

This implies that for many industrial applications the MRT and the MDA can be used and analyzed equivalently. In particular, any guarantee for one metric holds for the other one as well. Furthermore, end-to-end timing specification in industrial systems only needs

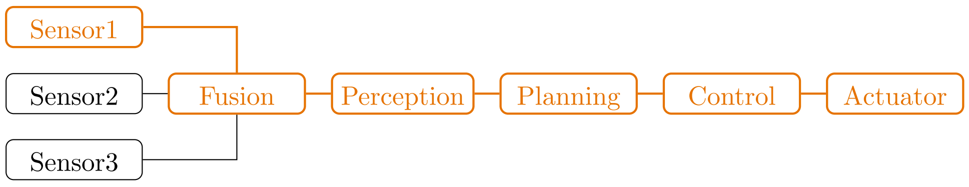


Figure 6.5.: ROS2 basic navigation system.

to consider one latency instead of two different latencies. This eases the verification of timing constraints.

Besides the equivalence of MRT and MDA, Theorem 6.7 also proposed a novel description of end-to-end latencies via p-partitioned job chains. This description can be used to improve the analysis of end-to-end latencies. For instance, a significant speedup of the latency calculation of periodic tasks communicating under LET can be obtained by choosing an advantageous $p \in \{1, \dots, n\}$, as demonstrated in Section 6.2.3.

6.1.2.4 Case Study: Robot Operating System 2

In this section, we validate the equivalence of MRT and MDA formulated in Theorem 6.7. To that end, we consider a basic navigation system, as shown in Figure 6.5, and apply the scheduling mechanism of the Robot Operating System 2 (ROS2) [Ope22] on a single ECU. The navigation system includes three sensors, whose data is combined and processed for the perception of the environment, planning the route, controlling the vehicle, and sending the output to the vehicle interfaces via an actuator.

A system in ROS2 consists of nodes and topics. Each node represents one component of the system. Nodes can communicate via topics, that implement a publish-subscribe architecture. Nodes are represented by tasks and each execution of a node can be considered as the execution of a job. The nodes follow an implicit communication policy, i.e., the read-event of a node is at its start and the write-event of a node is at its finish. Nodes are either time-triggered and event-triggered, i.e., some tasks have an aperiodic behavior. ROS2 has a non-standard custom scheduler that executes tasks instances under a round-robin scheduling approach. Specifically, the scheduler repeatedly collects at most one job of each task for the round, after which it executes all collected jobs according to their priority. The equivalence theorem can be applied since the basic assumptions (R1) and (R2) for the read- and write-events, stated in Section 3.2.1, are met.

The system depicted in Figure 6.5 has three sensors whose data is combined in the fusion node. The perception, planning, and control node process the data and supply the actuator node with instructions. For the simulation of the ROS2 scheduling behavior we assume that all jobs have a fixed execution time. Table 6.1 gives an overview of the Worst-Case Execution Time (WCET) of each component and of the period of the time-triggered components.

For the first chain $E = (\text{Sensor1} \to \dots \to \text{Actuator})$, marked in orange in Figure 6.5, we measured the MRT, MDA, MRRT, and MRDA by determining the longest immediate

Component	Type	Period	WCET
Sensor	time-triggered	100 ms	10 ms
Fusion	time-triggered	100 ms	100 ms
Fusion	event-triggered	-	5 ms
Perception	event-triggered	-	200 ms
Planning	event-triggered	-	100 ms
Control	event-triggered	-	50 ms
Actuator	event-triggered	-	5 ms

Table 6.1.: Periods and WCET of the nodes.

$\mathrm{MRT}(E, \omega)$	4.0	$\rho_{+}^{\omega}(\tau_1)$	0.5
$\mathrm{MDA}(E, \omega)$	4.0	$\rho_{-}^{\omega}(\tau_1)$	0.5
$\mathrm{MRRT}(E, \omega)$	3.5	$\rho_{+}^{\omega}(\tau_n)$	0.5
$\mathrm{MRDA}(E, \omega)$	3.5	$\rho_{-}^{\omega}(\tau_n)$	0.5

Table 6.2.: Measurements of the ROS2 system in seconds.

forward and immediate backward augmented job chains. The measured values are summarized in Table 6.2. We observe that Theorem 6.7 holds since the MRT and the MDA coincide. Furthermore, the measurements validate the results of Lemma 3.8 about the difference of MRT and MRRT, and the difference of MDA and MRDA.

6.2 Analysis of Worst-Case End-to-End Latency

While the previous section deals with fundamental properties of the end-to-end latency, in this section we derive bounds on the end-to-end latency, to validate whether end-to-end latency requirements are met. Specifically, this section aims to provide a property-based approach for time-triggered tasks scheduled under a T-FP scheduling algorithm $\mathcal{A}$. To that end, we first delve into bounds for cause-effect chains scheduled only on a single ECU (so-called *local* cause-effect chains). Afterwards, explain how these bounds can be applied cause-effect chains distributed among several (potentially heterogeneous and asynchronized) ECUs (so-called *interconnected* cause-effect chains). In particular, this section is organized as follows:

- In Section 6.2.1, we present a formal analysis framework, uncovering the principles that current end-to-end latencies are made from. Specifically, we use these principles to derive typical bounds for local cause-effect chains.

- In Sections 6.2.2 and 6.2.3, we provide safe bounds on the end-to-end latency of local cause-effect chains of periodic tasks, based on simulation over a hyperperiod. Specifically, in Section 6.2.2, we use *abstract integer representations* to derive the end-to-end latency if tasks are abstracted with Best-Case Execution Time (BCET) and WCET using implicit communication. In Section 6.2.3, we use *partitioned job* *chains* (introduced in Section 6.1.2) to speed up the calculation of the end-to-end latency under LET.

- In Section 6.2.4, we extend bounds for local cause-effect chains to interconnected cause-effect chains, using the cutting theorem from Section 6.1.1.
- In Section 6.2.5, we use the cutting theorem on local cause-effect chains to deal with heterogeneous setups of sporadic and periodic tasks, as well as implicit communication and LET in the same chain.

This section is only concerned with an upper bound on the *worst-case* end-to-end latency. *Probabilistic* bounds on the end-to-end latency are discussed in Section 6.3.

6.2.1 Property-Based Framework

In this section, we present the first formal analysis framework for end-to-end analysis of cause-effect chains. To that end, we uncover the principles that homogeneous analyses are built from in Section 6.2.1.1. These principles can be used as a construction kit to build the analyses for homogeneous systems in Sections 6.2.1.2 and 6.2.1.3. To the best of our knowledge this is the first time that such a generalized principle-based analysis is formulated. Although some analyses derived in this section coincide with analyses from the literature, the principle-based analysis framework presented in this work provides a unified perspective on end-to-end analyses, that may be applicable or transferable to future problem scenarios, such as novel communication means or different release policies (e.g., arrival curves).

For the analysis, we consider a cause-effect chain $E = (\tau_1 \to \dots \to \tau_n)$ and an arbitrary system evolution $\omega \in \Omega$. Tasks are abstracted as periodic or sporadic, i.e.,

$$\tau \in \mathrm{PerOff}(T_\tau, \phi_\tau) \qquad \vee \qquad \tau \in \mathrm{SporMinMax}(T_\tau^{\min}, T_\tau^{\max}). \tag{6.25}$$

Furthermore, tasks can use either implicit communication or LET to communicate. We assume that the WCRT $R_\tau^{\mathcal{A}} < \infty$ of all tasks is known beforehand. This can be achieved by performing, for instance, the busy-interval approach by Lehoczky [Leh90]. Furthermore, we assume that $R_\tau^{\mathcal{A}} < D_\tau$ for all tasks τ which have a relative deadline D_τ. The results presented in this section appeared in RTNS 2023 [Gün+23d]. However, the concept of *warm-up* has not been established at the date of publication. Instead, so-called *valid* job chains were used to constraint the observation window. The proofs of this section are adjusted to be compatible with the warm-up.

6.2.1.1 Latency Analysis Principles

In this section, we uncover the underlying principles in current end-to-end analyses. These principles are not only utilized to show the correctness of cause-effect chains in homogeneous analyses (see Sections 6.2.1.2 and 6.2.1.3), but can also be exploited when analyzing the heterogeneous case as shown in Section 6.2.5. We first detail the latency

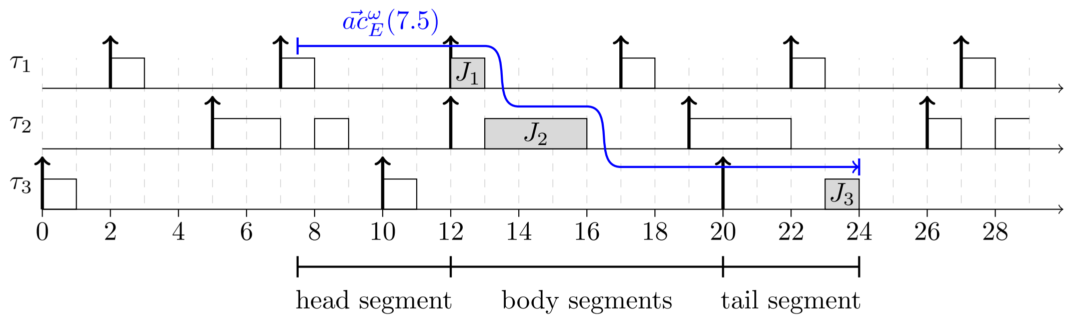


Figure 6.6.: Immediate forward augmented job chain $\vec{ac}_E^\omega(7.5) = (z, J_1, \ldots, J_3, z') = (7.5, \tau_1(3), \tau_2(2), \tau_3(3), 24)$ for the cause-effect chain $E = (\tau_1 \to \tau_2 \to \tau_3)$ under implicit communication. The segments of the job chain are introduced in Section 6.2.1.1.

analysis principles and then determine how the task features *LET, implicit communication, periodic,* and *sporadic* affect and simplify them. Please note that, although the analytical approach to analyze each task in the cause-effect chain individually is not completely new (e.g., [Dür+19; KBS18]) we are the first to formally define the principles that cover different communication policies at the same time.

We analyze the end-to-end latency from the perspective of the MRT, that is, we provide analytical bounds on the length of immediate forward augmented job chains. To that end, let $\vec{ac}_E^\omega(z) = (z, J_1, \ldots, J_n, z')$ be an immediate forward augmented job chain with $z > \mathrm{re}^\omega(\tau_1(W_{E,\tau_1}^\omega))$. We split this chain into three segments:

- **Head segment**: The time between z and the release of J_1.
- **Body segment(s)**: The time between the release of two consecutive jobs $r_{J_i}^\omega$ and $r_{J_{i+1}}^\omega$ for $i = 1, \ldots, n-1$.
- **Tail segment**: The time between $r_{J_n}^\omega$ and z'.

Example 6.8. *Figure 6.6 depicts the different segments for the immediate forward augmented job chain $\vec{ac}_E^\omega(7.5) = (7.5, \tau_1(3), \tau_2(2), \tau_3(3), 24)$. The jobs of the priority-indexed tasks*

- $\tau_1 \in \mathrm{PerOff}(5, 2) \cap \mathrm{WCET}(1)$
- $\tau_2 \in \mathrm{PerOff}(7, 5) \cap \mathrm{WCET}(3)$
- $\tau_3 \in \mathrm{PerOff}(10, 0) \cap \mathrm{WCET}(1)$

are scheduled on a single ECU using a preemptive T-FP scheduling policy. The head segment of $\vec{ac}_E^\omega(z)$ covers the 4.5 time units between z and the release of J_1. The first body segment has a length of 0 since it covers the time from the release of J_1, which is at time 12, until the release of J_2, which is at time 12 as well. The second body segment has *a length of* $20 - 12 = 8$ *since the job* J_3 *is released at time* 20. *The tail segment covers the time from the release of* J_3 *until* z', *that is* $24 - 20 = 4$ *time units.*

To formally express the principles for the segments, we use the following notation.

Definition 6.9 (ρ^ω_τ and ξ^ω_τ)**.** *Let* $\tau \in \mathbb{T}$ *be a task and let* $t \in \mathbb{R}$ *be a point in time. We denote by* $\rho^\omega_\tau(t)$ *the release time of the first job of* τ *that has its* read-event *at or after time point* t *in system evolution* ω, *i.e.,*

$$\rho^\omega_\tau(t) := \min\left\{ r^\omega_{\tau(j)} \,\middle|\, j \in \mathbb{N}, \mathrm{re}^\omega(\tau(j)) \geq t \right\}. \tag{6.26}$$

Moreover, we denote by $\xi^\omega_\tau(t)$ *the release time of the first job of* τ *that is* released *at or after time point* t *in system evolution* ω, *i.e.,*

$$\xi^\omega_\tau(t) := \min\left\{ r^\omega_{\tau(j)} \,\middle|\, j \in \mathbb{N}, r^\omega_{\tau(j)} \geq t \right\}. \tag{6.27}$$

In the following, we uncover one underlying analysis principle for each type of segments, starting with the head segment.

Principle 1: Since J_1 is by definition the earliest job of τ_1 with read-event at or after z, we have $r^\omega_{J_1} - z \leq \rho^\omega_{\tau_1}(z) - z$. Under LET and implicit communication, this simplifies to

$$r^\omega_{J_1} - z \leq \rho^\omega_{\tau_1}(z) - z \leq \xi^\omega_{\tau_1}(z) - z \leq \sup_{j\in\mathbb{N}}(r^\omega_{\tau_1(j+1)} - r^\omega_{\tau_1(j)}) \tag{P1}$$

since the read-event of each job is lower bounded by its release under both communication policies.

For the body segment, we observe the following principle.

Principle 2: For the time between the release of two consecutive jobs, we know that J_{i+1} is the first job of τ_{i+1} that has its read-event at or after the write-event of J_i.
Since under LET and under implicit communication each job reads no earlier than it is released, this leads to

$$r^\omega_{J_{i+1}} \leq \rho^\omega_{\tau_{i+1}}(\mathrm{we}^\omega(J_i)) \leq \xi^\omega_{\tau_{i+1}}(\mathrm{we}^\omega(J_i)) \tag{P2}$$

for all $i \in \{1, \dots, n-1\}$.

For the tail segment, we observe:

Principle 3: Since z' is at the write-event of the job J_n,

$$z' - r^\omega_{J_n} \leq \sup_{j\in\mathbb{N}}(\mathrm{we}^\omega(\tau_n(j)) - r^\omega_{\tau_n(j)}). \tag{P3}$$

In the remainder of this section we discuss how the task features *LET*, *implicit communication, periodic,* and *sporadic* affect and simplify these principles.

Sporadic Tasks For a sporadic task τ, the distance between two subsequent job releases is at most T_τ^{max} and Principle 1 simplifies to:

Lemma 6.10. *Let $\vec{ac}_E^\omega(z) = (z, J_1, \dots, J_n, z')$ be an immediate forward augmented job chain with $z > \mathrm{re}^\omega(\tau_1(W_{E,\tau_1}^\omega))$. If τ_1 is sporadic, then*

$$r_{J_1}^\omega - z \leq T_{\tau_1}^{max}. \tag{6.28}$$

Proof. This follows by applying $r_{\tau_1(j+1)}^\omega - r_{\tau_1(j)}^\omega \leq T_{\tau_1}^{max}$ to Equation (P1). □

Moreover, if task τ is already in the system at a time t, then it takes at most T_τ^{max} additional time units until the next job of τ is released. This simplifies Principle 2 as follows:

Lemma 6.11. *Let $\vec{ac}_E^\omega(z) = (z, J_1, \dots, J_n, z')$ be an immediate forward augmented job chain with $z > \mathrm{re}^\omega(\tau_1(W_{E,\tau_1}^\omega))$, and let $i \in \{1, \dots, n-1\}$. If τ_{i+1} is sporadic and $t \geq \mathrm{we}^\omega(J_i)$, then*

$$\xi_{\tau_{i+1}}^\omega(t) \leq t + T_{\tau_{i+1}}^{max}. \tag{6.29}$$

Proof. Since $z > \mathrm{re}^\omega(\tau_1(W_{E,\tau_1}^\omega))$, we know that $J_1 \succ \tau_1(W_{E,\tau_1}^\omega)$, i.e., J_1 is released *after* $\tau_1(W_{E,\tau_1}^\omega)$. By Lemma 6.2, also $J_{i+1} \succ \tau_{i+1}(W_{E,\tau_{i+1}}^\omega)$ must hold. Since J_{i+1} is the first job of τ_{i+1} with read-event at or after $\mathrm{we}^\omega(J_i)$, we know that $\tau_{i+1}(W_{E,\tau_{i+1}}^\omega)$ has its read-event before $\mathrm{we}^\omega(J_i)$. Consequently, $\tau_{i+1}(W_{E,\tau_{i+1}}^\omega)$ is released before $\mathrm{we}^\omega(J_i)$. Hence, there are already job releases of τ_{i+1} before t. The time until the next job release after t is therefore upper bounded by $T_{\tau_i+1}^{\max}$. □

Periodic Tasks For periodic tasks τ, the distance between two subsequent job releases is exactly T_τ and Principle 1 simplifies to:

Lemma 6.12. *Let $\vec{ac}_E^\omega(z) = (z, J_1, \dots, J_n, z')$ be an immediate forward augmented job chain with $z > \mathrm{re}^\omega(\tau_1(W_{E,\tau_1}^\omega))$. If τ_1 is periodic, then*

$$r_{J_1}^\omega - z \leq \phi_{\tau_1} + \left\lceil \frac{z - \phi_{\tau_1}}{T_{\tau_1}} \right\rceil T_{\tau_1} - z \tag{6.30}$$

Proof. If τ_1 is periodic, then $\xi_{\tau_1}^\omega(z) = \phi_{\tau_1} + \left\lceil \frac{z-\phi_{\tau_1}}{T_{\tau_1}} \right\rceil T_{\tau_1}$. Using that in Equation (P1), we obtain

$$r_{J_1}^\omega - z \leq \xi_{\tau_1}^\omega(z) - z = \phi_{\tau_1} + \left\lceil \frac{z - \phi_{\tau_1}}{T_{\tau_1}} \right\rceil T_{\tau_1} - z \tag{6.31}$$

which is the same as Equation (6.30). □

The release pattern of a periodic task τ is uniquely determined by its period and its phase. In particular, the k-th job of τ is released at time $\phi_\tau + (k-1) \cdot T_\tau$. This simplifies Principle 2 as follows:

Lemma 6.13. *Let $\vec{ac}^{\omega}_{E}(z) = (z, J_1, \dots, J_n, z')$ be an immediate forward augmented job chain with $z > \mathrm{re}^{\omega}(\tau_1(W^{\omega}_{E,\tau_1}))$, and let $i \in \{1, \dots, n-1\}$. If τ_{i+1} is periodic and $t \geq \mathrm{we}^{\omega}(J_i)$, then*

$$\xi^{\omega}_{\tau_{i+1}}(t) = \phi_{\tau_{i+1}} + \left\lceil \frac{t - \phi_{\tau_{i+1}}}{T_{\tau_{i+1}}} \right\rceil \cdot T_{\tau_{i+1}}. \tag{6.32}$$

Proof. A similar discussion as in the proof of Lemma 6.11, shows that $t \geq \phi_{\tau_{i+1}}$. By definition of the periodic release pattern, τ_{i+1} releases exactly $\left\lceil \frac{t-\phi_{\tau_{i+1}}}{T_{\tau_{i+1}}} \right\rceil$ jobs before time t. Therefore, the release time of the next job is described by Equation (6.32). □

If all tasks in E are periodic, then their release pattern repeats each hyperperiod. More specifically, let $H(E)$ be the least common multiple (lcm) of the periods of all tasks in E, then the release pattern in $[x, x+H(E)]$ is the same as in $[x+k\cdot H(E), x+(k+1)\cdot H(E)]$ for any $x \geq \Phi(E)$, $k \in \mathbb{N}$, where $\Phi(E)$ is the maximal phase of all tasks in E. Due to the repetitive behavior of the release pattern, it is sufficient to only bound the length of finitely many cause-effect chains as formulated in the following lemma.

Lemma 6.14. *Let $\vec{ac}^{\omega}_{E}(z)$ be an immediate forward augmented job chain with $z > \mathrm{re}^{\omega}(\tau_1(W^{\omega}_{E,\tau_1}))$. We assume that all tasks in E communicate under LET or implicit communication. Moreover, let $\ell^u(\vec{ac}^{\omega}_{E}(z))$ be an upper bound on the length of $\vec{ac}^{\omega}_{E}(z)$ that is only based on the following information:*

(i) The pattern of job releases at or after z.

(ii) The pattern of job deadlines at or after z.

(iii) The WCRT, WCET and priority of tasks in $\mathbb{T}$.

Then there exists $\tilde{z} \leq \Phi(E) + H(E)$ where $\Phi(E) := \max\left\{ r_{\tau(1)} \,\middle|\, \tau \in E \right\}$ is the maximal phase and $H(E) := \mathrm{lcm}(T_{\tau_1}, \dots, T_{\tau_n})$ is the hyperperiod, such that $\ell^u(\vec{ac}^{\omega}_{E}(z)) = \ell^u(\vec{ac}^{\omega}_{E}(\tilde{z}))$.

Proof. If $z \leq \Phi(E) + H(E)$, then we choose $\tilde{z} = z$. Otherwise, if $z > \Phi(E) + H(E)$, then let $k \in \mathbb{N}$ such that $z - k \cdot H(E) \in [\Phi(E), \Phi(E) + H(E)]$. Let $\tilde{z} = z - k \cdot H(E)$. Since the pattern of job releases and job deadlines repeats every hyperperiod after $\Phi(E)$, we have the same pattern after z and after $\tilde{z}$. Hence, $\ell^u(\vec{ac}^{\omega}_{E}(\tilde{z})) = \ell^u(\vec{ac}^{\omega}_{E}(z))$. □

Logical Execution Time (LET) Under LET, the difference between release and write-event of a job is exactly the relative deadline of the corresponding task, i.e., $\mathrm{we}^{\omega}(J) = r^{\omega}_{J} + D_{\tau}$ for each job J of task τ. This simplifies Principle 2 as follows:

Lemma 6.15. *Let $\vec{ac}^{\omega}_{E}(z) = (z, J_1, \dots, J_n, z')$ be an immediate forward augmented job chain with $z > \mathrm{re}^{\omega}(\tau_1(W^{\omega}_{E,\tau_1}))$ and let $i \in 1, \dots, n-1$. If τ_i communicates under LET, then*

$$r^{\omega}_{J_{i+1}} \leq \xi^{\omega}_{\tau_{i+1}}(r^{\omega}_{J_i} + D_{\tau_i}). \tag{6.33}$$

Proof. This follows from Equation (P2) and the property $\mathrm{we}^\omega(J_i) = r^\omega_{J_i} + D_{\tau_i}$ of LET tasks. $\square$

Moreover, Principle 3 simplifies as follows:

Lemma 6.16. *Let $\vec{ac}^\omega_E(z) = (z, J_1, \dots, J_n, z')$ be an immediate forward augmented job chain with $z > \mathrm{re}^\omega(\tau_1(W^\omega_{E,\tau_1}))$. If τ_n communicates under LET, then*

$$z' - r^\omega_{J_n} \leq D_{\tau_n}. \tag{6.34}$$

Proof. This follows by applying $\mathrm{we}^\omega(\tau_n(j)) - r^\omega_{\tau_n(j)} = D_{\tau_n}$ to Equation (P3). $\square$

Implicit Communication Under implicit communication, each job J of a task τ writes data at its finish $f_J^{\mathcal{A}(\omega),\omega}$. Since $r^\omega_J + R^\mathcal{A}_\tau$ is an upper bound on the finish of J, we have $\mathrm{we}^\omega(J) \leq r^\omega_J + R^\mathcal{A}_\tau$ for each job J of task τ. This simplifies Principle 2 in the following way:

Lemma 6.17. *Let $\vec{ac}^\omega_E(z) = (z, J_1, \dots, J_n, z')$ be an immediate forward augmented job chain with $z > \mathrm{re}^\omega(\tau_1(W^\omega_{E,\tau_1}))$, and let $i \in 1, \dots, n-1$. If τ_i communicates by implicit communication, then*

$$r^\omega_{J_{i+1}} \leq \xi^\omega_{\tau_{i+1}}(r^\omega_{J_i} + R^\mathcal{A}_{\tau_i}) \tag{6.35}$$

holds. Moreover, if τ_i and τ_{i+1} both communicate by implicit communication, are executed on the same ECU, and τ_i has higher priority than τ_{i+1}, then

$$r^\omega_{J_{i+1}} \leq \xi^\omega_{\tau_{i+1}}(r^\omega_{J_i}). \tag{6.36}$$

Proof. For implicit communication tasks $\mathrm{we}^\omega(J_i) \leq r^\omega_{J_i} + R^\mathcal{A}_{\tau_i}$ holds. Applying this to Equation (P2) leads to the statement from Equation (6.35).

If τ_i has higher priority than τ_{i+1} and a job J of τ_{i+1} is released no earlier than at time $r^\omega_{J_i}$, then J does not start before J_i finishes. In particular, if J_i and J both communicate with implicit communication, then J has its read-event after the write-event of J_i. Since J_{i+1} is the earliest job of τ_{i+1} that reads the data written by J_i, this proves Equation (6.36). $\square$

Principle 3 simplifies to the following:

Lemma 6.18. *Let $\vec{ac}^\omega_E(z) = (z, J_1, \dots, J_n, z')$ be an immediate forward augmented job chain with $z > \mathrm{re}^\omega(\tau_1(W^\omega_{E,\tau_1}))$. If τ_n communicates by implicit communication, then*

$$z' - r^\omega_{J_n} \leq R^\mathcal{A}_{\tau_n}. \tag{6.37}$$

Proof. Applying $\mathrm{we}^\omega(\tau_n(j)) - r^\omega_{\tau_n(j)} \leq R^\mathcal{A}_{\tau_n}$ to Equation (P3) leads to the result of this lemma. $\square$

	Comm.	**Our result**	**Coinciding literature analysis**	**Literature assumptions**
Spor.	LET	Theorem 6.19	Hamann et al. [Ham+17, Sec. 4.1.3]	Constr. DL ($D_\tau \leq T_\tau^{min}$)
	impl.	Theorem 6.20	Davare et al. [Dav+07]	$R_\tau^\mathcal{A} \leq T_\tau$ [1]
		Theorem 6.21	Dürr et al. [Dür+19] (only similar)	$R_\tau^\mathcal{A} \leq T_\tau$
Per.	LET	Theorem 6.22	(only similar approaches)	
	impl.	Theorem 6.23	(only similar approaches)	

[1] This assumption is unstated but used for the computation of the WCRT $R_\tau^\mathcal{A}$. In addition, they assume periodic tasks in their paper. However, the strict periodicity of the jobs of a task is never used in their analysis and their method is later usually referred in the literature for being applicable to sporadic systems.

Table 6.3.: Overview over the homogeneous analyses in Sections 6.2.1.2 and 6.2.1.3.

In the following, we consider homogeneous systems, where either all tasks communicate under LET or all tasks communicate by implicit communication, and where either all tasks are periodic or all tasks are sporadic. We apply the principles provided in this section to formulate dedicated analyses of the end-to-end latency for the four different cases. For the two cases with sporadic tasks the analytical bounds coincide with results from the literature (reformulated to match our notation). For the two cases with periodic tasks, these analytical bounds have not been provided before. As summarized in Table 6.3 the proofs from the related literature may require additional assumptions, however, we prove the bounds without additional requirements.

6.2.1.2 Bounds for Sporadic Tasks

In this section, we exploit the latency analysis principles from Section 6.2.1.1 to derive bounds for the end-to-end latency of a cause-effect chain of sporadic tasks. To that end, we consider a local cause-effect chain $E = (\tau_1 \to \cdots \to \tau_n)$. We assume that either all tasks of the chain communicate under LET or all tasks of the chain communicate using implicit communication.

The following is the most pessimistic bound, as it is dominated by all the following analyses. It is mentioned for instance in the work by Hamann et al. [Ham+17].

Theorem 6.19 (Sporadic, LET; coincides with [Ham+17])**.** *If all tasks of E are sporadic and all of them communicate under LET, then*

$$\mathrm{Lat}(E) \leq \sum_{i=1}^{n} (T_{\tau_i}^{max} + D_{\tau_i}) \tag{6.38}$$

is an upper bound on the end-to-end latency of E.

Proof. Let $\vec{ac}_E^{\omega}(z) = (z, J_1, \dots, J_n, z')$ be any immediate forward augmented job chain with $z > \mathrm{re}^\omega(\tau_1(W_{E,\tau_1}^\omega))$. Then we can decompose the length of $\vec{ac}_E^{\omega}(z)$ into the length of its head segment, body segments and tail segment as described in Section 6.2.1.1, i.e.,

$$\begin{aligned}\ell(\vec{ac}_E^\omega(z)) = (r_{J_1}^\omega - z) && \text{(head)} && (6.39)\\ + \sum_{i=1}^{n-1}(r_{J_{i+1}}^\omega - r_{J_i}^\omega) && \text{(body)} && (6.40)\\ + (z' - r_{J_n}^\omega). && \text{(tail)} && (6.41)\end{aligned}$$

Head: By Lemma 6.10, the head segment is bounded by

$$(r_{J_1}^\omega - z) \leq T_{\tau_1}^{max}. \qquad (6.42)$$

Body: First, we apply Lemma 6.15 to obtain $r_{J_{i+1}}^\omega \leq \xi_{\tau_{i+1}}^\omega(r_{J_i}^\omega + D_{\tau_i})$ for all $i \in \{1, \dots, n-1\}$. Second, Lemma 6.11 with $t = r_{J_i}^\omega + D_{\tau_i}$ yields $\xi_{\tau_{i+1}}^\omega(r_{J_i}^\omega + D_{\tau_i}) \leq r_{J_i}^\omega + D_{\tau_i} + T_{\tau_{i+1}}^{max}$. We conclude

$$(r_{J_{i+1}}^\omega - r_{J_i}^\omega) \leq T_{\tau_{i+1}}^{max} + D_{\tau_i} \qquad (6.43)$$

for the body segments.
Tail: By Lemma 6.16, the tail segment is bounded by

$$(z' - r_{J_n}^\omega) \leq D_{\tau_n}. \qquad (6.44)$$

The bound on the length of $\vec{ac}_E^\omega(z)$ is a combination of Equations (6.42), (6.43) and (6.44), i.e., we have

$$\ell(\vec{ac}_E^\omega(z)) \leq T_{\tau_1}^{max} + \sum_{i=1}^{n-1}(T_{\tau_{i+1}}^{max} + D_{\tau_i}) + D_{\tau_n}. \qquad (6.45)$$

By reordering the sum, we obtain

$$\ell(\vec{ac}_E^\omega(z)) \leq \sum_{i=1}^{n}(T_{\tau_i}^{max} + D_{\tau_i}). \qquad (6.46)$$

Since the upper bound holds for all immediate forward augmented job chains $\vec{ac}_E^\omega(z)$ with $z > \mathrm{re}^\omega(\tau_1(W_{E,\tau_1}^\omega))$ and $\omega \in \Omega$, this proves the bound on the end-to-end latency presented in the lemma. □

The bound for sporadic tasks using implicit communication coincides with the bound provided by Davare et al. [Dav+07] in 2007.

Theorem 6.20 (Sporadic, implicit communication; coincides with [Dav+07])**.** *If all tasks of E are sporadic and all of them communicate by implicit communication, then*

$$\mathrm{Lat}(E) \leq \sum_{i=1}^{n}(T_{\tau_i}^{max} + R_{\tau_i}^{\mathcal{A}}) \qquad (6.47)$$

is an upper bound on the end-to-end latency.

Proof. The same procedure as in the proof of Theorem 6.19 can be applied. In particular, for any immediate forward augmented job chain $\vec{ac}_E^\omega(z) = (z, J_1, \dots, J_n, z')$ with $z > \mathrm{re}^\omega(\tau_1(W_{E,\tau_1}^\omega))$ we achieve this bound as follows. We use Lemma 6.10 ($r_{J_1}^\omega - z \leq T_{\tau_1}^{max}$) to bound the head segment of $\vec{ac}_E^\omega(z)$. We apply Lemma 6.11 and Equation (6.35) of Lemma 6.17 to bound the body segments $((r_{J_{i+1}}^\omega - r_{J_i}^\omega) \leq T_{\tau_{i+1}}^{max} + R_{\tau_i}^{\mathcal{A}})$. We bound the tail segment of $\vec{ac}_E^\omega(z)$ by Lemma 6.18 ($z' - r_{J_n}^\omega \leq R_{\tau_n}^{\mathcal{A}}$). □

We derive an improved bound for the same case by applying the tightened bound for the body segment from Equation (6.36). Dürr et al. [Dür+19] achieved a similar bound in 2019 (cf. Theorem 3.10). However, they only remove $\min(R_{\tau_i}^{\mathcal{A}}, T_{\tau_{i+1}}^{\max})$ if $[P_i] = 1$. We show in the following that even a full $T_{\tau_{i+1}}^{\max}$ can be removed safely.[1]

Theorem 6.21 (Sporadic, implicit communication; similar to [Dür+19])**.** *If all tasks of E are sporadic and all of them communicate by implicit communication, then*

$$\mathrm{Lat}(E) \leq \sum_{i=1}^{n} (T_{\tau_i}^{\max} + R_{\tau_i}^{\mathcal{A}}) - \sum_{i=1}^{n-1} [P_i] \cdot R_{\tau_i}^{\mathcal{A}} \tag{6.48}$$

is an upper bound on the end-to-end latency, where $[P_i] = 1$ if τ_i and τ_{i+1} run on the same ECU and τ_i has higher priority than τ_{i+1}, and $[P_i] = 0$ otherwise.

Proof. For any immediate forward augmented job chain $\vec{ac}_E^\omega(z) = (z, J_1, \dots, J_n, z')$ with $z > \mathrm{re}^\omega(\tau_1(W_{E,\tau_1}^\omega))$ we bound the head segment by $r_{J_1}^\omega - z \leq T_{\tau_1}^{max}$ from Lemma 6.10 and we bound the tail segment by $z' - r_{J_n}^\omega \leq R_{\tau_n}^{\mathcal{A}}$ from Lemma 6.18. We apply Lemma 6.17 to get a bound for the body segments $r_{J_{i+1}}^\omega - r_{J_i}^\omega$. In particular, if τ_i has lower priority than τ_{i+1}, then Equation (6.35) together with Lemma 6.11 provides an upper bound $r_{J_{i+1}}^\omega - r_{J_i}^\omega \leq T_{\tau_{i+1}}^{max} + R_{\tau_i}^{\mathcal{A}}$. However, if τ_i has higher priority than τ_{i+1}, then Equation (6.36) together with Lemma 6.11 yields a tighter bound $r_{J_{i+1}}^\omega - r_{J_i}^\omega \leq T_{\tau_{i+1}}^{max}$. We obtain the bound in Equation (6.48) by combining the above bounds and ordering the summands accordingly. □

6.2.1.3 Bounds for Periodic Tasks

In this section, we exploit the latency analysis principles from Section 6.2.1.1 to derive bounds for the end-to-end latency of a cause-effect chain of *periodic* tasks. We consider a local cause-effect chain $E = (\tau_1 \to \cdots \to \tau_n)$, and assume that either all tasks of the chain communicate under LET or all tasks of the chain communicate using implicit communication.

As discussed in Section 6.2.1.1 for periodic tasks, the release pattern of the tasks repeats. Therefore, it is sufficient to examine a bounded time interval. To the best of our knowledge, the following two analyses are not presented in the literature, and only

[1] In our published work [Gün+23d], we considered *valid* augmented job chains instead of augmented job chains *after the warm-up*. Only as a result of that immature starting condition we achieved the same bound as [Dür+19]. Considering augmented job chains after the warm-up, we achieve the bound stated in this dissertation.

Algorithm 5 Compute MRT bound for periodic tasks under LET.

```
1:  $lengths = [\,]$
2:  for $m = 1, 2, 3, \dots$ do
3:      $z = \phi_{\tau_1} + (m-1) \cdot T_{\tau_1}$
4:      $r^\omega_{J_1} = \phi_{\tau_1} + m \cdot T_{\tau_1}$
5:      if $z > \Phi(E) + H(E)$ then
6:          break
7:      for $i = 1, 2, \dots, n-1$ do
8:          Compute $r^\omega_{J_{i+1}}$ with Equations (6.32) and (6.33).
9:      $z' = r^\omega_{J_n} + D_{\tau_n}$
10:     $lengths.append(z' - z)$
11: return $\max(lengths)$
```

similar approaches have been provided, e.g., [Ham+17; KBS18; KT20; MSB20]. The work by Kloda et al. [KBS18] is closest to our analysis. However, their result is limited to synchronous task sets, i.e., periodic tasks with offset 0, under implicit communication.

Theorem 6.22 (Periodic, LET)**.** *If all tasks of E are periodic and all of them communicate under LET, then an end-to-end latency upper bound is obtained by Algorithm 5.*

Proof. By definition, we have an upper bound

$$\mathrm{Lat}(E,\omega) = \sup_{z > \mathrm{re}^\omega(\tau_1(W^\omega_{E,\tau_1}))} \ell(\vec{ac}^\omega_E(z)) \leq \sup_{m\in\mathbb{N}} \sup_{z\in(r^\omega_{\tau_1(m)}, r^\omega_{\tau_1(m+1)}]} \ell(\vec{ac}^\omega_E(z)) \tag{6.49}$$

on the end-to-end latency. In each step $m = 1, 2, 3, \dots$ Algorithm 5 provides an upper bound on $A_m := \sup_{z\in(r^\omega_{\tau_1(m)}, r^\omega_{\tau_1(m+1)}]} \ell(\vec{ac}^\omega_E(z))$. In the following, we consider an arbitrary $\vec{ac}^\omega_E(z)$ with $z \in (r^\omega_{\tau_1(m)}, r^\omega_{\tau_1(m+1)}]$ and $z > \mathrm{re}^\omega(\tau_1(W^\omega_{E,\tau_1}))$.

In line 3, z is lower bounded by $r^\omega_{\tau_1(m)} = \phi_{\tau_1} + (m-1) \cdot T_{\tau_1}$. In line 4, the release of J_1 is upper bounded by using Lemma 6.12, i.e., $r^\omega_{J_1} \leq \phi_{\tau_1} + \left\lceil \frac{z-\phi_{\tau_1}}{T_{\tau_1}} \right\rceil \cdot T_{\tau_1} = \phi_{\tau_1} + m \cdot T_{\tau_1}$. In lines 7–8, the body segments are bounded by Lemmas 6.13 and 6.15. In line 9 the bound for the tail segment from Lemma 6.16 is applied. Since the bound obtained by the algorithm is only based on (i)–(iii) of Lemma 6.14, we know that the upper bound on A_m with $r^\omega_{\tau_1(m)} > \Phi(E) + H(E)$ is the same as the upper bound on $A_{m - \frac{H(E)}{T_{\tau_1}}}$. Hence, breaking the for-loop in lines 5–6 is safe. □

Theorem 6.23 (Periodic, implicit communication)**.** *If all tasks of E are periodic and all of them communicate by implicit communication, then an end-to-end latency upper bound is obtained by Algorithm 6.*

Algorithm 6 Compute MRT bound for periodic tasks under implicit communication.

```
1: $lengths = [\,]$
2: for $m = 1, 2, 3, \dots$ do
3:     $z = \phi_{\tau_1} + (m-1) \cdot T_{\tau_1}$
4:     $r^{\omega}_{J_1} = \phi_{\tau_1} + m \cdot T_{\tau_1}$
5:     if $z > \Phi(E) + H(E)$ then
6:         break
7:     for $i = 1, 2, \dots, n-1$ do
8:         Compute $r^{\omega}_{J_{i+1}}$ with Equations (6.32), (6.35) and (6.36).
9:     $z' = r^{\omega}_{J_n} + R^{\mathcal{A}}_{\tau_n}$
10:    $lengths.append(z' - z)$
11: return $\max(lengths)$
```

Proof. As in Algorithm 5, in each step $m = 1, 2, 3, \dots$ Algorithm 6 provides an upper bound on $A_m := \sup_{z \in (r^{\omega}_{\tau_1(m)}, r^{\omega}_{\tau_1(m+1)}]} \ell(\vec{ac}^{\omega}_E(z))$. In the following, we consider an arbitrary $\vec{ac}^{\omega}_E(z)$ with $z \in (r^{\omega}_{\tau_1(m)}, r^{\omega}_{\tau_1(m+1)}]$ and $z > \mathrm{re}^{\omega}(\tau_1(W^{\omega}_{E,\tau_1}))$.

The bounds in lines 3–4 coincide with the bounds of Algorithm 5 since we apply again Lemma 6.12. In lines 7–8, the body segments are bounded by Lemmas 6.13 and 6.17. In line 9, the bound for the tail segment from Lemma 6.18 is applied. Since the bound obtained by the algorithm is only based on (i)–(iii) of Lemma 6.14, we know that the upper bound on A_m with $r^{\omega}_{\tau_1(m)} > \Phi(E) + H(E)$ is the same as the upper bound on $A_{m - \frac{H(E)}{T_{\tau_1}}}$. Hence, breaking the for-loop in lines 5–6 is safe. □

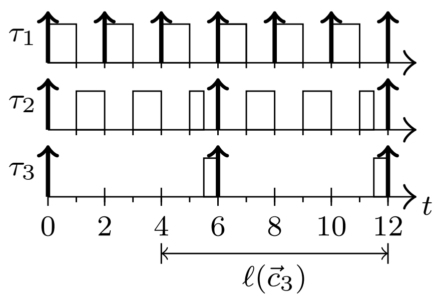
6.2.2 Abstract Integer Representations

Considering periodic tasks with offset, i.e.,

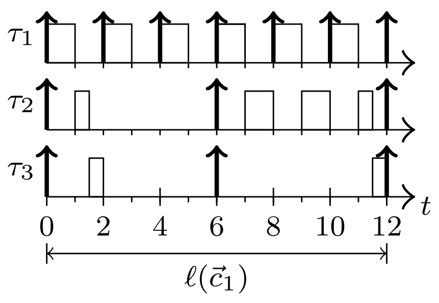
$$\tau \in \mathrm{PerOff}(T_\tau, \phi_\tau), \tag{6.50}$$

we know that the release pattern of all jobs repeats each hyperperiod after the maximal phase $\Phi(\mathbb{T}) := \max_{\tau \in \mathbb{T}} \phi_\tau$. In the previous section, we have already exploited that property by discussing upper bounds that rely solely on the release pattern, the pattern of deadlines, WCRT and WCET (cf. Lemma 6.14). This section aims to provide tighter bounds under implicit communication by examining the pattern of read- and write-events.

However, the read- and write-events of jobs depend significantly on the execution demand of the jobs under analysis. Due to this behavior, the pattern of read- and write-events does not repeat after $\Phi(\mathbb{T}) + H(\mathbb{T})$. Moreover, it is not sufficient to simulate the end-to-end latency for the system evolution where each job executes the WCET, as depicted in Figure 6.7. Instead, we determine the minimal and maximal read- and write-events for each job and utilize a concept called *abstract integer representations* to bound immediate forward and immediate backward augmented job chains. We show that the pattern of minimal and maximal read- and write-events repeats after $\Phi(\mathbb{T}) + 2H(\mathbb{T})$,



(a) Every task executes its WCET.



(b) Early completion of task τ_2.

Figure 6.7.: Two schedules of jobs released periodically by 3 tasks. For $E = (\tau_1 \to \tau_3)$, early completion of the first job of τ_2 leads to a larger immediate forward augmented job chain.

and that it can be determined by simulating the two schedules with the BCET and with the WCET, i.e., we assume that tasks are abstracted as

$$\tau \in \mathrm{BCET}(C_\tau^{\min}) \cap \mathrm{WCET}(C_\tau^{\max}). \tag{6.51}$$

Our analysis requires three steps:

(i) We calculate upper and lower bounds for both the read- and write-events of each job by simulating the schedule two times, once with the tasks WCETs and once with the BCETs.

(ii) Based on these upper and lower bounds, we bound the length of all immediate forward and immediate backward augmented job chains using *abstract integer representations*.

(iii) We bound the end-to-end latency.

Corollary 6.34 shows that our new analysis is exact for the special case that BCET and WCET coincide. We assume that the cause-effect chain $E = (\tau_1 \to \cdots \to \tau_n)$ under analysis is local, i.e., it contains only tasks on one (synchronized) ECU. Moreover, the tasks adhere to the implicit communication policy and all schedules are obtained by a T-FP preemptive scheduling algorithm $\mathcal{A}$ with fixed task priorities. For the sake of simplicity, we consider $\mathbb{T}$ to contain only tasks from one ECU as well, as the tasks from other ECUs have no impact on the scheduling behavior of the tasks on that ECU. The total utilization $U_\mathbb{T} = \sum_{\tau \in \mathbb{T}} \frac{C_\tau^{\max}}{T_\tau}$ of the ECU is assumed to be at most 1.

This section is organized as follows:

- In Sections 6.2.2.1–6.2.2.3, we derive the analysis of the end-to-end latency. This is achieved by following the three steps required for the analysis, as described above by (i)–(iii).

- In Section 6.2.2.4, we extend the analysis to the MRDA.
- In Section 6.2.2.5, we evaluate our analysis, comparing it do analytical bounds of the literature.

The results presented in this section appeared in RTAS 2021 [Gün+21a] and in a journal extension for ACM TECS in 2023 [Gün+23a]. While the conference version was limited to the case that BCET and WCET coincide, the journal extension allows that BCET and WCET differ.

6.2.2.1 Step 1: Obtain Bounds for Read- and Write-Events

Since under implicit communication the read- and write-events coincide with start and finish of the jobs, it is sufficient to examine those. Intuitively if the execution time of any job is increased, the interference on the other jobs does not decrease, and therefore the start and finish of all jobs cannot be decreased as well. Hence, the latest (earliest, respectively) start and finish of a job is achieved if all jobs execute their WCET (BCET, respectively). Formally, we state the following two Propositions.

Proposition 6.24. *Let $\omega_{\max} \in \Omega$ be the system evolution where each job executes its WCET, i.e., $\omega_{\max} = ((r_{\tau(j)})_{\tau(j)\in\mathbb{J}}, (C_\tau^{\max})_{\tau(j)\in\mathbb{J}})$.*[2] *Then*

$$s_{\tau(j)}^{\mathcal{A}(\omega),\omega} \leq s_{\tau(j)}^{\mathcal{A}(\omega_{\max}),\omega_{\max}} \qquad \textit{and} \qquad f_{\tau(j)}^{\mathcal{A}(\omega),\omega} \leq f_{\tau(j)}^{\mathcal{A}(\omega_{\max}),\omega_{\max}} \tag{6.52}$$

holds for all jobs $\tau(j) \in \mathbb{J}$ and all system evolutions $\omega \in \Omega$.

Proposition 6.25. *Let $\omega_{\min} \in \Omega$ be the system evolution where each job executes its BCET, i.e., $\omega_{\min} = ((r_{\tau(j)})_{\tau(j)\in\mathbb{J}}, (C_\tau^{\min})_{\tau(j)\in\mathbb{J}})$.*[3] *Then*

$$s_{\tau(j)}^{\mathcal{A}(\omega),\omega} \geq s_{\tau(j)}^{\mathcal{A}(\omega_{\min}),\omega_{\min}} \qquad \textit{and} \qquad f_{\tau(j)}^{\mathcal{A}(\omega),\omega} \geq f_{\tau(j)}^{\mathcal{A}(\omega_{\min}),\omega_{\min}} \tag{6.53}$$

holds for all jobs $\tau(j) \in \mathbb{J}$ and all system evolutions $\omega \in \Omega$.

Proof of Proposition 6.24 and 6.25. Let ω be any system evolution. It is sufficient to show that for any job $J \in \mathbb{J}$ the following two properties hold:

Property P1 Whenever the job execution of any other job $J' \in \mathbb{J}$ is increased, then the starting time of J does not decrease.

Property P2 Whenever the job execution of any other job $J' \in \mathbb{J}$ is increased, then the finishing time of J does not decrease. Or equivalenty, whenever the job execution of any other job $J' \in \mathbb{J}$ is *reduced*, then the finishing time of J does not increase.

[2] If the system evolution $\omega_{\max}$ does not exist in Ω, we add it artificially.
[3] If the system evolution $\omega_{\min}$ does not exist in Ω, we add it artificially.

The following proof is based on contradiction. In particular, we assume that P1 or P2 does *not* hold for all jobs in $\mathbb{J}$. We denote by $\tilde{J}$ the job in $\mathbb{J}$ with the earliest finishing time in ω such that P1 or P2 does not hold. In particular, for all jobs finished before $\tilde{J}$ in ω both properties P1 and P2 hold. Let $\tilde{\tau}$ denote the task of $\tilde{J}$.

Without any impact on the execution behavior of $\tilde{J}$ we remove all jobs of tasks with lower priority than $\tilde{\tau}$. Moreover, we remove all jobs from $\tilde{\tau}$ that are released later than $\tilde{J}$.

We obtain a contradiction in two steps: First we show that P1 holds for $\tilde{J}$, meaning that P2 does not hold for $\tilde{J}$. In the second step we lead statement *P2 does not hold for* $\tilde{J}$ to a contradiction.

Step 1: Proof that P1 holds for $\tilde{J}$. Let $[g_1, h_1)$ be the largest interval, such that

- $r_{\tilde{J}} \in [g_1, h_1]$, and
- at all times during $[g_1, h_1)$ either jobs with higher priority than $\tilde{\tau}$ or jobs of $\tilde{\tau}$ released before $\tilde{J}$ are executed in $\mathcal{A}(\omega)$.

For this interval, the property $s_{\tilde{J}}^{\mathcal{A}(\omega),\omega} = h_1$ holds. Let $\mathbb{J}_1 \subseteq \mathbb{J}$ be the set of jobs that are executed during $[g_1, h_1)$ in $\mathcal{A}(\omega)$. Then during $[g_1, h_1)$ there is always pending workload from jobs of $\mathbb{J}_1$, i.e., for all $t \in [g_1, h_1)$ there exists some $J \in \mathbb{J}_1$ such that $t \in [r_J, f_J^{\mathcal{A}(\omega),\omega})$. Moreover, for the jobs in $\mathbb{J}_1$ Properties P1 and P2 hold since those jobs finish before $\tilde{J}$.

When increasing the execution time of any job, then $f_J^{\mathcal{A}(\omega),\omega}$ does not decrease for all $J \in \mathbb{J}_1$ by Property P2. As a result, during the interval $[g_1, h_1)$ there is still pending workload from jobs of $\mathbb{J}_1$ at all times, i.e., the ECU is blocked and cannot execute $\tilde{J}$. Hence, $s_{\tilde{J}}^{\mathcal{A}(\omega),\omega}$ is still $\geq h_1$, i.e., P1 holds for $\tilde{J}$. Therefore, P2 does not hold for $\tilde{J}$ by the initial assumption.

Step 2: Contradiction of the statement: *P2 does not hold for $\tilde{J}$*. Let $[g_2, h_2)$ be the largest interval, such that

(i) $\tilde{J}$ is executed at some time during the interval, i.e., $\mathrm{exec}_{\tilde{J}}^{\mathcal{A}(\omega)}(g_2, h_2) > 0$, and

(ii) at all times in the interval (g_2, h_2) there is pending workload, i.e., workload that was released before but is not already finished, from $\tilde{J}$, earlier jobs of $\tilde{\tau}$ or jobs with higher priority than $\tilde{\tau}$.

Let $\mathbb{J}_2 \subseteq \mathbb{J}$ be the jobs that are executed during the interval $[g_2, h_2)$. By definition, $\tilde{J} \in \mathbb{J}_2$. In particular, (ii) ensures that

$$g_2 + \sum_{\{J \,|\, J \in \mathbb{J}_2 \text{ and } r_J \in [g_2, e)\}} c_J^\omega > e \tag{6.54}$$

holds for all $e \in [g_2, h_2)$, and

$$g_2 + \sum_{\{J \,|\, J \in \mathbb{J}_2 \text{ and } r_J \in [g_2, h_2)\}} c_J^\omega = h_2. \tag{6.55}$$

Moreover, we observe the typical busy-interval properties $g_2 = \min_{J \in \mathbb{J}_2} r_J$, i.e., the busy-interval starts with a job release, and

$$\min_{J \in \mathbb{I}} r_J + \sum_{J \in \mathbb{I}} c_J^\omega \leq h_2 \tag{6.56}$$

for all subsets $\mathbb{I} \subseteq \mathbb{J}_2$, i.e., the whole workload can be finished until the end of the busy-interval.

We have $f_{\tilde{J}}^{\mathcal{A}(\omega),\omega} = h_2$ as the following consideration shows:

- $\tilde{J}$ is executed during $[g_2, h_2)$ by (i). Therefore, $\tilde{J}$ has pending workload during $[g_2, h_2)$, i.e., $(r_{\tilde{J}}, f_{\tilde{J}}^{\mathcal{A}(\omega),\omega}) \cap (g_2, h_2) \neq \emptyset$. Since $[g_2, h_2)$ is the *largest* interval with pending workload and there is pending workload during $(r_{\tilde{J}}, f_{\tilde{J}}^{\mathcal{A}(\omega),\omega})$, we conclude $(r_{\tilde{J}}, f_{\tilde{J}}^{\mathcal{A}(\omega),\omega}) \subseteq (g_2, h_2)$ and therefore $f_{\tilde{J}}^{\mathcal{A}(\omega),\omega} \leq h_2$.
- Assume $f_{\tilde{J}}^{\mathcal{A}(\omega),\omega} < h_2$ *for contradiction.* Let $\mathbb{I} \subseteq \mathbb{J}$ be the set of the jobs that are executed during the interval $[f_{\tilde{J}}^{\mathcal{A}(\omega),\omega}, h_2) \neq \emptyset$. By definition, $\tilde{J} \notin \mathbb{I}$ holds. All jobs of $\mathbb{I}$ have higher priority than $\tilde{J}$ by (ii). If a job of $\mathbb{I}$ would be released before $f_{\tilde{J}}^{\mathcal{A}(\omega),\omega}$, then it would finish before $f_{\tilde{J}}^{\mathcal{A}(\omega),\omega}$ due to its priority. Therefore, all jobs of $\mathbb{I}$ are released at or after $f_{\tilde{J}}^{\mathcal{A}(\omega),\omega}$. Hence, there is no pending workload at time $f_{\tilde{J}}^{\mathcal{A}(\omega),\omega}$ which contradicts (ii).

Since Property P2 does not hold for $\tilde{J}$, there is a scenario where the execution time of any job is reduced but the finishing time of $\tilde{J}$ is increased. We denote the system evolution with that modified execution time by ω'. Let $[g_2', h_2')$ be the busy-interval of $\tilde{J}$, i.e., the interval that fulfills (i) and (ii), in the new schedule $\mathcal{A}(\omega')$. Moreover, let $\mathbb{J}_2' \subseteq \mathbb{J}$ be the jobs that are executed during $[g_2', h_2')$ in the new schedule $\mathcal{A}(\omega')$.

Analogous to the discussion that $h_2 = f_{\tilde{J}}^{\mathcal{A}(\omega),\omega}$ in the original schedule $\mathcal{A}(\omega)$, $h_2' = f_{\tilde{J}}^{\mathcal{A}(\omega'),\omega'}$ holds in the new schedule $\mathcal{A}(\omega')$. Hence, by the assumption that Property P2 does not hold, we have $h_2' > h_2$. Moreover, Equations (6.54), (6.55), and (6.56) hold for the new scenario as well.

All jobs that are finished before g_2 in the original schedule $\mathcal{A}(\omega)$ fulfill Property P2. Therefore, they are also finished before g_2 in the new schedule $\mathcal{A}(\omega')$. Consequently, in $\mathcal{A}(\omega')$ there is no pending workload at time g_2, and $g_2' \geq g_2$ holds. In the following we show that both different cases $g_2' = g_2$ and $g_2' > g_2$ lead to a contradiction.

- Case $g_2' = g_2$: In the new schedule $\mathcal{A}(\omega')$ we have $g_2' + \sum_{J \in \mathbb{J}_2'} c_J^{\omega'} > h_2$ by Equation (6.54) when choosing $e = h_2 \in [g_2', h_2')$. However, in the original schedule $\mathcal{A}(\omega)$ we have $g_2 + \sum_{J \in \mathbb{J}_2} c_J^\omega = h_2$ by Equation (6.55). Since the execution time of jobs is only decreased, we have $g_2' + \sum_{J \in \mathbb{J}_2'} c_J^{\omega'} \leq g_2 + \sum_{J \in \mathbb{J}_2} c_J^\omega$, which leads to a contradiction that $h_2 < h_2$ as follows:

$$h_2 < g_2' + \sum_{J \in \mathbb{J}_2'} c_J^{\omega'} \leq g_2 + \sum_{J \in \mathbb{J}_2} c_J^\omega = h_2 \tag{6.57}$$

- Case $g_2' > g_2$: Let $\mathbb{I} \subseteq \mathbb{J}_2$ and $\mathbb{I}' \subseteq \mathbb{J}_2'$ be the jobs that are released during $[g_2', h_2)$ in the original schedule $\mathcal{A}(\omega)$ and in the new schedule $\mathcal{A}(\omega')$, respectively. Since $\mathbb{T}$ is periodic with offset, the release times in different system evolutions coincide, and we have $\mathbb{I} = \mathbb{I}'$. Hence, we have $\sum_{J \in \mathbb{I}} c_J^{\omega} \geq \sum_{J \in \mathbb{I}'} c_J^{\omega'}$. By Equation (6.56) in the original schedule we obtain $g_2' + \sum_{J \in \mathbb{I}} c_J^{\omega} \leq h_2$. By Equation (6.54) in the new schedule we obtain $g_2' + \sum_{J \in \mathbb{I}'} c_J^{\omega'} > h_2$. These two conditions lead to the fact that $\sum_{J \in \mathbb{I}} c_J^{\omega} < \sum_{J \in \mathbb{I}'} c_J^{\omega'}$ which contradicts $\sum_{J \in \mathbb{I}} c_J^{\omega} \geq \sum_{J \in \mathbb{I}'} c_J^{\omega'}$.

Since this leads both cases $g_2' = g_2$ and $g_2' > g_2$ to a contradiction, this concludes Step 2 of the proof. □

Under implicit communication, the starting times and the finishing times coincide with the read- and write-events, respectively. Hence, the lower and upper bounds for starting times and finishing times from Proposition 6.24 and Proposition 6.25 are lower and upper bounds for the read- and write-events as well. We conclude that for each job we can provide upper and lower bounds for the read- and write-events by simulating the schedule two times until the job finishes: Once when all jobs execute their WCET and once when all jobs execute their BCET.

Since for each task there are infinitely many jobs, we cannot simulate the schedule for every job. Therefore, in the following, we show that it is sufficient to simulate the schedule for a finite time window since the release pattern repeats. The following considerations are based on the work of Leung and Whitehead [LW82]. Their proofs cannot be used directly, since they create a new schedule (they call it a partial schedule) and show that this one repeats. We need to show that even the original schedule repeats.

In the following, let $\omega_{\mathit{fix}} \in \{\omega_{\min}, \omega_{\max}\}$ the system evolution, where the execution demand of all jobs of the same task is fixed to the same value (either $C_\tau^{\min}$ or $C_\tau^{\max}$). We consider the corresponding schedule $\mathcal{A}(\omega_{\mathit{fix}})$. We denote by $C_\tau \in \left\{C_\tau^{\min}, C_\tau^{\max}\right\}$ the execution time of the jobs of task τ. For a time instant $t \in \mathbb{R}$, we denote by $X(t)$ the tuple $(\chi_\tau(t))_{\tau \in \mathbb{T}}$ where each $\chi_\tau(t)$ is the amount of time that jobs of τ_i have been executed since their last release, i.e., $\chi_\tau(t) = \mathrm{exec}_\tau^{\omega_{\mathit{fix}}}\left(\max\left\{r_{\tau(j)} \,\middle|\, j \in \mathbb{N}, r_{\tau(j)} \leq t\right\}, t\right)$. Similar to the proof of Leung and Whitehead [LW82, Lemma 3.3], we show the following.

Lemma 6.26. *For all $t \geq \Phi(\mathbb{T})$ the relation $X(t) \geq X(t + H(\mathbb{T}))$ (component-wise) holds.*

Proof. For a proof by contradiction, we assume that there are some $t \geq \Phi(\mathbb{T})$ and $\tau \in \mathbb{T}$ such that $\chi_\tau(t) < \chi_\tau(t + H(\mathbb{T}))$. We show that in this case infinitely many tasks $\tau' \in \mathbb{T}$ have a time instant

$$t_{\tau'} \geq \phi_{\tau'} \text{ with } \chi_{\tau'}(t_{\tau'}) < \chi_{\tau'}(t_{\tau'} + H(\mathbb{T})). \tag{6.58}$$

This contradicts the fact that $\mathbb{T}$ is finite.

By assumption, there is at least one task with the property from Equation (6.58). Assume there are only finitely many tasks with this property and let τ' be the one of them with the highest priority. Since $\chi_{\tau'}(t_{\tau'}) < \chi_{\tau'}(t_{\tau'} + H(\mathbb{T}))$, there exists some $\tilde{t} \in [\phi_{\tau'}, t_{\tau'}]$, where τ' is not executing during $[\tilde{t}, \tilde{t}+\varepsilon]$ but during $[\tilde{t}+H(\mathbb{T}), \tilde{t}+H(\mathbb{T})+\varepsilon]$ for some $\varepsilon > 0$. Hence, there is some higher-priority task $\tilde{\tau}$ which executes during $[\tilde{t}, \tilde{t}+\varepsilon]$ but not during $[\tilde{t}+H(\mathbb{T}), \tilde{t}+H(\mathbb{T})+\varepsilon]$, i.e., $\chi_{\tilde{\tau}}(\tilde{t}) < \chi_{\tilde{\tau}}(\tilde{t}+H(\mathbb{T})) = C_{\tilde{\tau}}$ since all jobs of task $\tilde{\tau}$ have the same execution time. Since $\tilde{\tau}$ executes during $\tilde{t}$, we know that $\phi_{\tilde{\tau}} \leq \tilde{t}$. This contradicts the assumption that τ' is the highest priority task with the property from Equation (6.58). □

Furthermore, similar to Leung and Whitehead [LW82, Lemma 3.4], we use the preceding lemma to show that the schedule repeats after $\Phi(\mathbb{T}) + 2H(\mathbb{T})$; that is, the schedule in the interval $[\Phi(\mathbb{T}) + H(\mathbb{T}), \Phi(\mathbb{T}) + 2H(\mathbb{T}))$ coincides with the schedule in the intervals $[\Phi(\mathbb{T}) + 2H(\mathbb{T}), \Phi(\mathbb{T}) + 3H(\mathbb{T}))$, $[\Phi(\mathbb{T}) + 3H(\mathbb{T}), \Phi(\mathbb{T}) + 4H(\mathbb{T}))$, and so on. We only utilize that the total utilization $U_{\mathbb{T}} = \sum_{\tau\in\mathbb{T}} \frac{C_\tau}{T_\tau}$ of the system is at most 1.

Lemma 6.27. *When $U_{\mathbb{T}} \leq 1$, then $X(t) = X(t + H(\mathbb{T}))$ holds for all $t \geq \Phi(\mathbb{T}) + H(\mathbb{T})$.*

Proof. We assume that there is some $t \geq \Phi(\mathbb{T}) + H(\mathbb{T})$ with $X(t) \neq X(t + H(\mathbb{T}))$. Then, by Lemma 6.26, there is some task τ with $\chi_\tau(t) > \chi_\tau(t + H(\mathbb{T}))$. There are two cases. Either, (i) the ECU idles during some $[t', t'+\varepsilon] \subseteq [t, t+H(\mathbb{T})]$ for some $\varepsilon > 0$, or (ii) the ECU is busy during the whole interval $[t, t + H(\mathbb{T})]$.

For (i), by Lemma 6.26, $X(t') = (C_\tau)_{\tau\in\mathbb{T}} \leq X(t' - H(\mathbb{T}))$, i.e., the ECU also idles during $[t' - H(\mathbb{T}), t' - H(\mathbb{T}) + \varepsilon]$. Since the job releases are the same, the schedule coincides in the intervals $[t' - H(\mathbb{T}), t]$ and $[t', t + H(\mathbb{T})]$. Hence, $X(t) = X(t + H(\mathbb{T}))$.

For (ii), since $\chi_\tau(t) > \chi_\tau(t + H(\mathbb{T}))$ and $\chi_{\tau'}(t) \geq \chi_{\tau'}(t + H(\mathbb{T}))$ for all $\tau' \in \mathbb{T}$, by Lemma 6.26, there is less remaining workload by jobs in the ready queue at time t than at time $t + H(\mathbb{T})$. We conclude that there was more workload released during $(t, t + H(\mathbb{T})]$ than could be executed by the ECU. This means that $\sum_{i=1}^{n} C_i \frac{H(\mathbb{T})}{T_{\tau_i}} > H(\mathbb{T})$, which contradicts $\sum_{i=1}^{n} \frac{C_{\tau_i}}{T_{\tau_i}} \leq 1$. □

Based on Lemma 6.27, the schedule repeats after $\Phi(\mathbb{T}) + 2H(\mathbb{T})$. More precisely, the starting time and finishing time in system evolution $\omega_{\min}$ of any m-th job $\tau(m)$ of $\tau \in \mathbb{T}$ that is released not before $\Phi(\mathbb{T}) + 2H(\mathbb{T})$ can be recursively computed using the formulas:

- $s_{\tau(m)}^{\mathcal{A}(\omega_{\min}),\omega_{\min}} = s_{\tau(m-\frac{H(\mathbb{T})}{T_\tau})}^{\mathcal{A}(\omega_{\min}),\omega_{\min}} + H(\mathbb{T})$
- $f_{\tau(m)}^{\mathcal{A}(\omega_{\min}),\omega_{\min}} = f_{\tau(m-\frac{H(\mathbb{T})}{T_\tau})}^{\mathcal{A}(\omega_{\min}),\omega_{\min}} + H(\mathbb{T})$

Similarly, the starting time and finishing time in system evolution $\omega_{\max}$ of any m-th job $\tau(m)$ of $\tau \in \mathbb{T}$ that is released not before $\Phi(\mathbb{T}) + 2H(\mathbb{T})$ can be recursively computed using the formulas:

- $s_{\tau(m)}^{\mathcal{A}(\omega_{\max}),\omega_{\max}} = s_{\tau(m-\frac{H(\mathbb{T})}{T_\tau})}^{\mathcal{A}(\omega_{\max}),\omega_{\max}} + H(\mathbb{T})$
- $f_{\tau(m)}^{\mathcal{A}(\omega_{\max}),\omega_{\max}} = f_{\tau(m-\frac{H(\mathbb{T})}{T_\tau})}^{\mathcal{A}(\omega_{\max}),\omega_{\max}} + H(\mathbb{T})$

As a result, it is sufficient to generate a schedule for $\omega_{\min}$ and a schedule for $\omega_{\max}$ until all jobs that are released before $\Phi(\mathbb{T}) + 2H(\mathbb{T})$ are finished.

In particular, the results from the previous propositions and the fact that read- and write-events are set to the starting time and finishing time enables us to provide upper and lower bounds for the read- and write-events. Independent of the system evolution, they are formulated by functions $\mathrm{re}^{\min}, \mathrm{re}^{\max}, \mathrm{we}^{\min}, \mathrm{we}^{\max} \colon \mathbb{T} \times \mathbb{N} \to \mathbb{R}$ as follows.

Definition 6.28 ($\mathrm{re}^{\min}, \mathrm{re}^{\max}, \mathrm{we}^{\min}, \mathrm{we}^{\max}$)**.** *For a task set $\mathbb{T}$, we define functions $\mathrm{re}^{\min}, \mathrm{re}^{\max}, \mathrm{we}^{\min}, \mathrm{we}^{\max} \colon \mathbb{T} \times \mathbb{N} \to \mathbb{R}$ by:*

$$\mathrm{re}^{\min}(\tau, m) := s_{\tau(m)}^{\mathcal{A}(\omega_{\min}),\omega_{\min}} \qquad \mathrm{re}^{\max}(\tau, m) := s_{\tau(m)}^{\mathcal{A}(\omega_{\max}),\omega_{\max}} \tag{6.59}$$

$$\mathrm{we}^{\min}(\tau, m) := f_{\tau(m)}^{\mathcal{A}(\omega_{\min}),\omega_{\min}} \qquad \mathrm{we}^{\max}(\tau, m) := f_{\tau(m)}^{\mathcal{A}(\omega_{\max}),\omega_{\max}} \tag{6.60}$$

The values of the functions from Definition 6.28 at any $(\tau, m) \in \mathbb{T} \times \mathbb{N}$ are constructed as follows:

(i) We construct two T-FP preemptive schedules, one for $\omega_{\min}$ and one for $\omega_{\max}$, until all jobs released before $\Phi(\mathbb{T}) + 2H(\mathbb{T})$ are finished.

(ii) If $\phi_\tau + m \cdot T_\tau < \Phi(\mathbb{T}) + 2H(\mathbb{T})$, then the values of starting time and finishing time are directly taken from the corresponding schedules.

(iii) If $\phi_\tau + m \cdot T_\tau \geq \Phi(\mathbb{T}) + 2H(\mathbb{T})$, then we calculate the read- and write-events recursively as described after Lemma 6.27 and set the values accordingly.

By the results from Proposition 6.24 and Proposition 6.25, we obtain that

- $\mathrm{re}^{\min}(\tau, m) \leq \mathrm{re}^{\omega}(\tau(m)) \leq \mathrm{re}^{\max}(\tau, m)$ and
- $\mathrm{we}^{\min}(\tau, m) \leq \mathrm{we}^{\omega}(\tau(m)) \leq \mathrm{we}^{\max}(\tau, m)$

for all jobs $\tau(m) \in \mathbb{J}$ and for all system evolutions $\omega \in \Omega$.

6.2.2.2 Step 2: Bound Length of Augmented Job Chains

To examine the end-to-end latency, we take the perspective of the MRT. That is, the end-to-end latency can be described by a supremum over immediate forward augmented job chains. To provide a bound for the end-to-end latency, we need to bound immediate forward augmented job chains. We achieve this by constructing *abstract integer representations $\vec{I}_i^E$* for cause-effect chain $E = (\tau_1 \to \cdots \to \tau_n)$ and $i \in \mathbb{N}$ based on the bounds of read- and write-events provided in the previous section. Afterwards, in Section 6.2.2.2, we show that finitely many abstract integer representations are sufficient to bound all immediate forward augmented job chains.

Definition 6.29 (Construction of $\vec{I}_i^E$)**.** *Let $i \in \mathbb{N}$ be a natural number. The abstract integer representation $\vec{I}_i^E = (i_0, \dots, i_{n+1}) \in \mathbb{N}^{n+2}$ is an $(n+2)$-tuple with:*

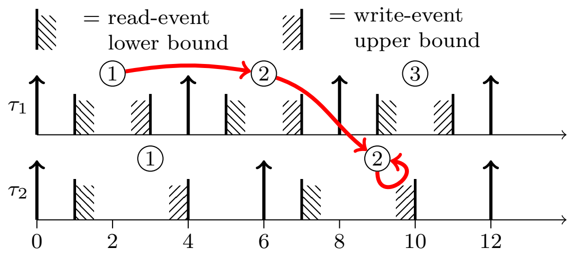


Figure 6.8.: Construction of $\vec{I}_1^E = (1,2,2,2)$ for $E = (\tau_1 \to \tau_2)$. We assume that the read-event lower bound and write-event upper bound for each job are given and that τ_2 has higher priority than τ_1.

- *$i_0 = i$ and $i_1 = i+1$.*
- *For $j \in \{2,\dots,n\}$ the entry i_j is the smallest number in $\mathbb{N}$ such that*
  (i) $\mathrm{re}^{\min}(\tau_j, i_j) \geq \mathrm{we}^{\max}(\tau_{j-1}, i_{j-1})$*, or*
  (ii) $\mathrm{re}^{\min}(\tau_j, i_j)$ *is no less than the i_{j-1}-th release of τ_{j-1} and τ_{j-1} has higher priority than τ_j.*
- *$i_{n+1} = i_n$.*

We define by $\ell(\vec{I}_i^E) := \mathrm{we}^{\max}(\tau_n, i_{n+1}) - \mathrm{re}^{\min}(\tau_1, i_0)$ the length of the abstract integer representation $\vec{I}_i^E$.

Please note that i_1 represents the $(i+1)$-th job of τ_1 and $i_2,\dots,i_n$ are chosen such that the i_j-th job of τ_j safely reads data after it was written by the i_{j-1}-th job of τ_{j-1} for all $j = 2,\dots,n$. Furthermore, i_0 and i_{n+1} represent the read-event of the i-th job of τ_1 (i.e., a lower bound for the external activity z) and the write-event of the i_n-th job of τ_n (i.e., an upper bound for z'), respectively. This description is similar to the description of immediate forward augmented job chains presented in Definition 3.4.

Example 6.30. *Figure 6.8 illustrates the construction of the abstract representation $\vec{I}_1^E = (1,2,2,2)$. The first entry is set to 1 due to the index of $\vec{I}_1^E$. The second entry is computed by $1+1$, which represents the second job of τ_1. The write-event upper bound of the second job of τ_1 is lower than the read-event of the second job of τ_2 under* any *system evolution, since the write-event upper bound $\mathrm{we}^{\max}(\tau_1, 2) = 7$, is no less than the read-event lower bound $\mathrm{re}^{\min}(\tau_2, 2) = 7$. The last entry of $\vec{I}_1^E = (1,2,2,2)$ is repeated.*

The abstract integer representation yields an upper bound on the immediate forward augmented job chains as stated in the following lemma.

Lemma 6.31 (Forward bound)**.** *Let $\vec{I}_i^E = (i_0,\dots,i_{n+1})$ be some abstract representation as specified in Definition 6.29. We have the following inequality*

$$\ell\left(\vec{ac}_E^{\omega}(z)\right) \leq \ell\left(\vec{I}_i^E\right) \tag{6.61}$$

for all system evolutions $\omega \in \Omega$ and $z \in (\mathrm{re}^{\omega}(\tau_1(i)), \mathrm{re}^{\omega}(\tau_1(i+1))]$.

Proof. We consider an immediate forward augmented job chain $\vec{ac}^{\omega}_{E}(z) = (z, J_1, \dots, J_n, z')$ with $z \in (\mathrm{re}^{\omega}(\tau_1(i)), \mathrm{re}^{\omega}(\tau_1(i+1))]$. In the following we show that $\ell(\vec{ac}^{\omega}_{E}(z)) = z' - z \leq \mathrm{we}^{\omega}(\tau_n(i_{n+1})) - \mathrm{re}^{\omega}(\tau_1(i_0))$.

By definition, $z \geq \mathrm{re}^{\omega}(\tau_1(i_0))$. Therefore, it is left to show that $z' \leq \mathrm{we}^{\omega}(\tau_n(i_{n+1}))$. We show that by proving that $J_j \preccurlyeq \tau_j(i_j)$ for all $j = 1, \dots, n$ by induction.

Base case ($j = 1$): By definition, J_1 is the first job of τ_1 with read-event $\geq z$. Hence, $J_1 = \tau_1(i_0 + 1) = \tau_1(i_1)$.

Induction step ($j \mapsto j + 1$): By definition, J_{j+1} is the first job of τ_{j+1} with read-event $\geq \mathrm{we}^{\omega}(J_j)$. Due to the induction hypothesis, $J_j \preccurlyeq \tau_j(i_j)$. Therefore, $\mathrm{we}^{\omega}(J_j) \leq \mathrm{we}^{\omega}(\tau_j(i_j)) \leq \mathrm{we}^{\max}(\tau_j, i_j)$. By construction of the abstract integer representation $\vec{I}^{E}_{i}$, we know that it is guaranteed that $\tau_{j+1}(i_{j+1})$ has read-event at or after $\mathrm{we}^{\max}(\tau_j, i_j)$ under any system evolution. Hence, the read-event of $\tau_{j+1}(i_{j+1})$ is $\geq \mathrm{we}^{\omega}(J_j)$. We conclude that $\tau_{j+1}(i_{j+1}) \succcurlyeq J_{j+1}$.

We obtain that $J_n \preccurlyeq \tau_n(i_n) = \tau_n(i_{n+1})$. Therefore, $\mathrm{we}^{\omega}(J_n) \leq \mathrm{we}^{\omega}(\tau_m(i_{n+1}))$ by requirement (R1) of Section 3.2.1. Hence, $z = \mathrm{we}^{\omega}(J_n) \leq \mathrm{we}^{\omega}(\tau_m(i_{n+1})) \leq \mathrm{we}^{\max}(\tau_m, i_{n+1})$. This proves the lemma. □

6.2.2.3 Step 3: Provide End-To-End Latency

By Lemma 6.31, the computation of a bound on the end-to-end latency is reduced to a construction of the abstract representations $\vec{I}^{E}_{i}$. As proven in Section 6.2.2.1, the upper and lower bounds on the read- and write-events repeat each hyperperiod after $\Phi(\mathbb{T}) + 2H(\mathbb{T})$. Therefore, the construction and also the length of the abstract integer representations $\vec{I}^{E}_{i}$ repeats as well and only a finite number of them has to be considered to provide a latency bound. The following lemma show that it is sufficient to consider abstract integer representations where the job that is described by the first entry is released until $\Phi(\mathbb{T}) + 2H(\mathbb{T})$.

Lemma 6.32. *Let $\vec{I}^{E}_{i} = (i_0, \dots, i_{n+1})$ be an abstract integer representation with $\phi_{\tau_1} + (i_0 - 1) \cdot T_{\tau_1} \geq \Phi(\mathbb{T}) + 2H(\mathbb{T})$. There exists $j \in \mathbb{N}$ such that for $\vec{I}^{E}_{j} = (j_0, \dots, j_{n+1})$:*

- $r_{\tau_1(j)} = \phi_{\tau_1} + (j-1)T_{\tau_1} \in [\Phi(\mathbb{T}) + H(\mathbb{T}), \Phi(\mathbb{T}) + 2H(\mathbb{T}))$
- $\ell\left(\vec{I}^{E}_{j}\right) = \ell\left(\vec{I}^{E}_{i}\right)$

Proof. Since $Z := [\Phi(\mathbb{T}) + H(\mathbb{T}), \Phi(\mathbb{T}) + 2H(\mathbb{T}))$ has length $H(\mathbb{T})$ and $w := \phi_{\tau_1} + (i_0 - 1) \cdot T_{\tau_1}$ is no less than the right boundary of Z, there exists some $\xi \in \mathbb{N}$ such that $w - \xi \cdot H(\mathbb{T}) \in Z$. We choose $j := i - \xi \cdot \frac{H(\mathbb{T})}{T_{\tau_1}}$ and consider $\vec{I}^{E}_{j}$. Since the maximal and minimal read- and write-events repeat each hyperperiod $H(\mathbb{T})$, the underlying job sequence of $\vec{I}^{E}_{j}$ is just the abstract representation of $\vec{I}^{E}_{i}$ shifted ξ hyperperiods to the left. Therefore, $\phi_{\tau_1} + (j_0 - 1) \cdot T_{\tau_1} = w - \xi \cdot H(\mathbb{T}) \in Z$ and $\ell\left(\vec{I}^{E}_{j}\right) = \ell\left(\vec{I}^{E}_{i}\right)$. □

Please note that since for $\vec{I}^{E}_{i}$ the first entry is $i_0 = i$, if i is increased, then i_0 is increased as well. As a consequence, to find all i such that $\phi_{\tau_1} + (i_0 - 1)T_{\tau_1} < \Phi(\mathbb{T}) + 2H(\mathbb{T})$ is

Algorithm 7 End-to-end latency bound.

```
1: Compute $\text{we}^{\max}$ and $\text{re}^{\min}$ by constructing concrete schedules.  ▷ After Definition 6.28
2: $\mathcal{I}^{fw} = \{\}$, $i = 1$
3: while $\phi_{\tau_1} + (i-1)T_{\tau_1} < \Phi(\mathbb{T}) + 2H(\mathbb{T})$ do
4:     Add $i$ to $\mathcal{I}^{fw}$.
5:     Construct $\vec{I}_i$.  ▷ Definition 6.29
6:     $i = i + 1$
7: Compute upper bound from Equation (6.62).
```

fulfilled, it is sufficient to consider $i = 1, 2, 3, \ldots$ and stop as soon as $\phi_{\tau_1} + (i_0 - 1)T_{\tau_1} \geq \Phi(\mathbb{T}) + 2H(\mathbb{T})$.

Under the assumption that the read- and write-event upper and lower bounds, as defined in Definition 6.28, are computed for all jobs, we can construct $\vec{I}_i^E$ for all $i = 1, 2, 3, \ldots$ until $\phi_{\tau_1} + (j-1)T_{\tau_1} \geq \Phi(\mathbb{T}) + 2H(\mathbb{T})$. The abstract representations $\vec{I}_i^E$ provide an upper bound on the length $\ell\left(\vec{ac}_E^{\omega}(z)\right)$ of the corresponding immediate forward augmented job chains with $z \in (\text{re}^{\omega}(\tau_1(i)), \text{re}^{\omega}(\tau_1(i+1))]$ by Lemma 6.31. By Lemma 6.32 this finite number of abstract representations is sufficient to bound the length of *all* immediate forward/backward augmented job chains. By taking the maximum $\max_i \ell\left(\vec{I}_i^E\right)$ of the length of these finitely many abstract representations, we obtain upper bounds on the end-to-end latency $\text{Lat}(E)$.

Theorem 6.33. *Let E be a cause-effect chain where all tasks are on one ECU and all tasks adhere to implicit communication. We denote by $\mathcal{I}^{fw}$ the set of all $i \in \mathbb{N}$ such that $\phi_{\tau_1} + (i-1)T_{\tau_1} < \Phi(\mathbb{T}) + 2H(\mathbb{T})$. The following equations hold:*

$$\text{Lat}(E) \leq \max_{i \in \mathcal{I}^{fw}} \ell\left(\vec{I}_i^E\right) \tag{6.62}$$

Proof. By Definition 3.7, the end-to-end latency is defined as

$$\text{Lat}(E) = \sup_{\omega \in \Omega} \sup_{z > \text{re}^{\omega}(\tau_1(W^{\omega}_{E,\tau_1}))} \ell(\vec{ac}_E^{\omega}(z)). \tag{6.63}$$

By Lemma 6.31, this can be upper bounded by $\text{Lat}(E) \leq \sup_{\omega \in \Omega} \sup_{i \in \mathbb{N}} \ell(\vec{I}_i^E)$. Applying Lemma 6.32, we obtain $\text{Lat}(E) \leq \sup_{\omega \in \Omega} \sup_{i \in \mathcal{I}^{fw}} \ell(\vec{I}_i^E)$. Since $\sup_{i \in \mathcal{I}^{fw}} \ell(\vec{I}_i^E)$ is independent of the choice of $\omega \in \Omega$, we obtain $\text{Lat}(E) \leq \sup_{i \in \mathcal{I}^{fw}} \ell(\vec{I}_i^E)$. □

The procedures to compute our bound on the end-to-end latency is shown in Algorithm 7.

Furthermore, we note that in the case that $C_\tau^u = C_\tau^\ell$ for all tasks $\tau \in \mathbb{T}$, the execution time of every job of each task is fixed. In this case there is only one system evolution $\omega = \omega_{\min} = \omega_{\max} \in \Omega$ to be considered, and the constructed minimal and maximal read- and write-events coincide with the actual read- and write-events. As a result, the abstract integer representations $\vec{I}_i^E$ constructed in Definition 6.29 provide an exact bound for $\sup_{z\in(\text{re}^\omega(\tau_1(i)),\text{re}^\omega(\tau_1(i+1))]} \ell(\vec{ac}^\omega_E(z))$. Therefore, when only considering $i \in \mathcal{I}^{fw}$ with $i \geq W^\omega_{E,\tau_1}$ or $\phi_{\tau_1} + (i-1)T_{\tau_1} \geq \Phi(\mathbb{T}) + H(\mathbb{T})$, the analysis using abstract integer representations becomes exact. We denote this subset of $\mathcal{I}^{fw}$ by $\mathcal{I}^{fw}_{exact}$

Corollary 6.34. *If $C^u_\tau = C^\ell_\tau$ for all tasks $\tau \in \mathbb{T}$, then* $\text{Lat}(E) = \max_{i\in\mathcal{I}^{fw}_{exact}} \ell\left(\vec{I}^E_i\right)$.

Proof. If $C^u_\tau = C^\ell_\tau$ for all tasks $\tau \in \mathbb{T}$, then there is only one system evolution $\Omega = \{\omega\}$. Moreover, the maximal and minimal read- and write-events utilized in the construction of $\vec{I}^E_i$ are the actual read- and write-events of the jobs, respectively. Hence,

$$\ell(\vec{I}^E_i) = \sup_{z\in(\text{re}^\omega(\tau_1(i)),\text{re}^\omega(\tau_1(i+1))]} \ell(\vec{ac}^\omega_E(z)). \tag{6.64}$$

We conclude that $\text{Lat}(E) = \sup_{i\geq W^\omega_{E,\tau_1}} \ell(\vec{I}^E_i) = \sup_{i\in\mathcal{I}^{fw}_{exact}} \ell(\vec{I}^E_i)$. □

Complexity: Two components play a decisive role for the time complexity of our analysis. We (i) create the schedule for the system evolutions $\omega_{\max}$ and $\omega_{\min}$ for the bounded time frame to obtain minimal and maximal read- and write-events for each job, and (ii) create and compare abstract integer representations based on the minimal and maximal read- and write-events. We examine the time complexity for a cause-effect chain E on an ECU with task set $\mathbb{T}$ with N tasks.

Since the schedule repeats after $\Phi(\mathbb{T}) + 2H(\mathbb{T})$, it suffices to schedule the jobs in the interval $[0, \Phi(\mathbb{T}) + 2H(\mathbb{T}))$. Hence, the time complexity for (i) is $\mathcal{O}\left(\frac{\Phi(\mathbb{T})+2H(\mathbb{T})}{T_{min}} \cdot n\right)$, where $T_{min} := \min_{\tau\in\mathbb{T}} T_\tau$ is the minimal period in $\mathbb{T}$.

For each abstract integer representation we have to determine $n+2$ integers. There is a cost of query[4] Q depending on the data structure for finding the next integer for the abstract representation. To compute the end-to-end latency bound, we have to simulate and compare up to $\frac{\Phi(\mathbb{T})+2H(\mathbb{T})}{T_{\tau_1}}$ abstract integer representations. Hence, the time complexity for component (ii) is $\mathcal{O}\left(n \cdot Q \cdot \frac{\Phi(\mathbb{T})+2H(\mathbb{T})}{T_{\tau_1}}\right)$.

We note that the time complexity for the method by Kloda et al. [KBS18] coincides with the complexity of component (ii) for our reaction time computation, except that they have to call a latency function for each job instead of determining the job itself. The methods by Dürr et al. [Dür+19] and by Davare [Dav+07] have complexity $\mathcal{O}(n)$. Since these methods all assume that the WCRTs are known, i.e., computed in advance, the time complexity of the WCRT computation should also be taken into account.

[4]The data structure can be designed such that $Q = \mathcal{O}(1)$. Let all jobs for each task τ be stored in a list l_τ, ordered by their release. If we want to find the first job of τ with minimal read-event after a time instant t and assume that all jobs finish their execution before the subsequent job release, then only those jobs in the list with index j between $\left\lceil\frac{t-\phi_\tau-T_\tau}{T_\tau}\right\rceil$ and $\left\lfloor\frac{t-\phi_\tau}{T_\tau}\right\rfloor$ are candidates to be checked, i.e., $\mathcal{O}(T_\tau/T_\tau)$ many jobs.

6.2.2.4 Analysis of Maximum Reduced Data Age

In this section we extend our bound to Maximum Reduced Data Age (MRDA) to compare with the bounds from [Bec+17a; Dür+19; KBS18]. By Definition 3.7, the MRDA is defined as

$$\sup_{\omega\in\Omega}\left(\sup\left\{\ell^\omega(J_1,\dots,J_n)\,\middle|\,\overleftarrow{ac}_E^\omega(z') = (z,J_1,\dots,J_n,z'),\ z' > \mathrm{we}^\omega(\tau_n(W_{E,\tau_n}^\omega))\right\}\right). \quad (6.65)$$

To provide a bound for the MRDA, we construct abstract integer representations $\overleftarrow{I}_i^E$ for immediate backward augmented job chains.

Definition 6.35 (Construction of $\overleftarrow{I}_i^E$). *Let $i \in \mathbb{N}$ be a natural number. The abstract integer representation $\overleftarrow{I}_i^E = (i_0,\dots,i_{n+1}) \in \mathbb{N}^{n+2}$ is a $(n+2)$-tuple with:*

- *$i_{n+1} = i$ and $i_n = i-1$.*
- *For $j \in \{n-1,\dots,1\}$ the entry i_j is the highest number in $\mathbb{N}$ such that*
  - *(i) $\mathrm{re}^{\min}(\tau_{j+1}, i_{j+1}) \geq \mathrm{we}^{\max}(\tau_j, i_j)$, or*
  - *(ii) τ_j has higher priority than τ_{j+1} and $\mathrm{re}^{\min}(\tau_{j+1}, i_{j+1})$ is no earlier than the release of the i_j-th job of τ_j.*
- *$i_0 = i_1$.*

We define by $\ell(\overleftarrow{I}_i^E) := \mathrm{we}^{\max}(\tau_n, i_{n+1}) - \mathrm{re}^{\min}(\tau_1, i_0)$ the length of the abstract integer representation $\overleftarrow{I}_i^E$.

If $\overleftarrow{I}_i^E = (i_0,\dots,i_{n+1})$ exists, then $\mathrm{we}^{\max}(\tau_n, i_n) - \mathrm{re}^{\min}(\tau_1, i_1)$ bounds $\ell^\omega(J_1,\dots,J_n)$ for any immediate backward augmented job chain $\overleftarrow{ac}_E^\omega(z')$ with $z' \in (\mathrm{re}^\omega(\tau_n(i-1)), \mathrm{re}^\omega(\tau_n(i))]$. This result can be proven similar to the proof of Theorem 6.31.

However, there might be cases where $\overleftarrow{ac}_E^\omega(z') = (z,J_1,\dots,J_n,z'),\ z' > \mathrm{we}^\omega(\tau_n(W_{E,\tau_n}^\omega))$ exists, but the corresponding $\overleftarrow{I}^E_{\left\lceil \frac{z'-\phi_{\tau_n}}{T_{\tau_n}} \right\rceil}$ does not. In that case, we use the upper bound $\Psi(i) := \mathrm{we}^{\max}(\tau_n, i) - \mathrm{re}^{\min}(\tau_1, 1)$ with $i = \left\lceil \frac{z'-\phi_{\tau_n}}{T_{\tau_n}} \right\rceil$ instead. Hence, we obtain a bound for the MRDA as follows:

(i) Construct $\mathrm{we}^{\max}$ and $\mathrm{re}^{\min}$ using concrete schedules for all jobs that are released before $\Phi(\mathbb{T}) + 2H(\mathbb{T})$

(ii) For $i = 1, 2, \dots$ construct $\overleftarrow{I}_i^E = (i_0,\dots,i_{n+1})$ until $r_{\tau_1(i_0)} \geq \Phi(\mathbb{T}) + 2H(\mathbb{T})$.

(iii) Compute the bound

$$\mathrm{MRDA}(E) \leq \max_i \begin{cases} \ell\left(\overleftarrow{I}_i^E\right) & ,\overleftarrow{I}_i^E \text{ exists} \\ \Psi(i) & ,\textit{else} \end{cases} \quad (6.66)$$

6.2.2.5 Evaluation

A relevant industrial use-case of the presented end-to-end latency analyses is the timing verification of cause-effect chains in the automotive domain. To assess the practical benefit of our proposed analyses, we evaluate it using synthesized task sets and cause-effect chains that adhere to the details described in *Automotive Benchmarks For Free* [KZH15]. Furthermore, we generate task sets with the UUnifast algorithm [BB05] to assess the performance for general task parameters. We consider periodic task sets and implicit job communication to apply our analysis results from Theorem 6.33 (for the end-to-end latency (Lat)) and Section 6.2.2.4 (for MRDA). The following setup is considered in the evaluation, which is released on Github [TU 22b].

We use the method by Davare et al. [Dav+07] to normalize all other end-to-end bounds, since this method yields the most pessimistic result. We define the *latency reduction* $LR(\mathcal{M})$ of an analysis method $\mathcal{M}$ with respect to an evaluated bound $B(\cdot)$, i.e., end-to-end latency or MRDA, by

$$LR(\mathcal{M}) := (B(Davare) - B(\mathcal{M}))/B(Davare). \tag{6.67}$$

Additionally, for the case BCET equals WCET, an exact analysis of end-to-end latency and MDA is given by Corollary 6.34. Since this is a valid execution scenario for the general case where $C_\tau^{\min} \leq C_\tau^{\max}$, Corollary 6.34 provides a lower bound for the case $C_\tau^{\min} \leq C_\tau^{\max}$. Therefore, we also consider the *gap reduction* $GR(\mathcal{M})$, defined by

$$GR(\mathcal{M}) := (B(Davare) - B(\mathcal{M}))/(B(Davare) - B(\text{Corollary } 6.34)). \tag{6.68}$$

In particular, the closer the gap reduction is to 1, the tighter that bound is to the lower bound, and therefore the tighter that value is to the exact latency bound.

Task and Task Set Generation A task τ is described by the WCET C_τ, period T_τ, and phase ϕ_τ. Furthermore, $U_\tau = C_\tau / T_\tau$ is the utilization of task τ. We consider the following two benchmarks for the generation of task sets.

Automotive benchmark [KZH15]: A task τ is generated as follows:[5]

(i) The period T_τ in ms is drawn from the set $T = \{1, 2, 5, 10, 20, 50, 100, 200, 1000\}$ according to the related share[6] of [KZH15, Table III, IV, and V].

(ii) The Average-Case Execution Time (ACET) of a task is generated based on a Weibull distribution that fulfills the properties in [KZH15, Table III, IV, and V].

[5]In the automotive benchmark [KZH15], atomic software components contain *runnables* subject to scheduling. Multiple runnables with the same period are afterwards grouped together as a task. Since communication usually happens on the runnable level, we set up the experiments accordingly, and denote each runnable as a task to match the common notation in real-time systems research and the cause-effect chain model in the literature.

[6]The sum of the probabilities in [KZH15, Table III, IV, and V] is only 85 %. The remaining 15 % are reserved for angle-synchronous tasks that we do not consider. Hence, all share values are divided by 0.85 in the generation process.

(iii) The task's WCET (WCET) is determined by multiplying its ACET with its WCET factor, which is drawn equally distributed from an interval $[f_{min}, f_{max}]$ [KZH15].

We generate 1 000 *automotive task sets* for each cumulative task set utilization of $U = 50\,\%$, $60\,\%$, $70\,\%$, $80\,\%$, and $90\,\%$. Since the tasks' utilizations are determined by the WCET and the automotive-specific semi-harmonic periods, we use a fully-polynomial approximation scheme to solve the subset-sum problem to select a subset of tasks within a candidate task set such that the cumulative utilization satisfies the above requirements. We initially generate $\mathcal{T}$, a set of 1 000 to 1 500 tasks, and then select a subset $\mathcal{T}'$ of tasks using the subset-sum approximation algorithm to reach the targeted utilization within 1 percentage point error bounds, i.e., $|(\sum_{\mathcal{T}'} U_\tau) - U| \leq 0.01$. On average, the generated task sets consist of around 50 tasks.

Uniform task set generation [BB05]: For user-specified values $N \in \mathbb{N}$ and $0 < U^* \leq 1$, the UUniFast algorithm [BB05] generates a task set $\mathbb{T}$ of cardinality N as follows. Utilization values $(U_\tau)_{\tau \in \mathbb{T}}$ are drawn from $(0, 1]^N$ uniform at random under the constraint that $\sum_{\tau \in \mathbb{T}} U_\tau = U^*$. Due to the fact that the analyses are computationally tractable only for sufficiently small hyperperiods, we draw semi-harmonic periods based on the automotive benchmark. For each of the utilization values, i.e., $U^* = 50\,\%, 60\,\%, 70\,\%, 80\,\%$, and $90\,\%$, we generate 1 000 task sets with 50 tasks each. Each task's period is drawn from the interval $[1, 2\,000]$ according to a log-uniform distribution and rounded to the next smallest period in the set $\{1, 2, 5, 10, 20, 50, 100, 200, 500, 1\,000\}$. Given the periods and the utilizations, the WCET is set to $C_\tau = U_\tau \cdot T_\tau$.

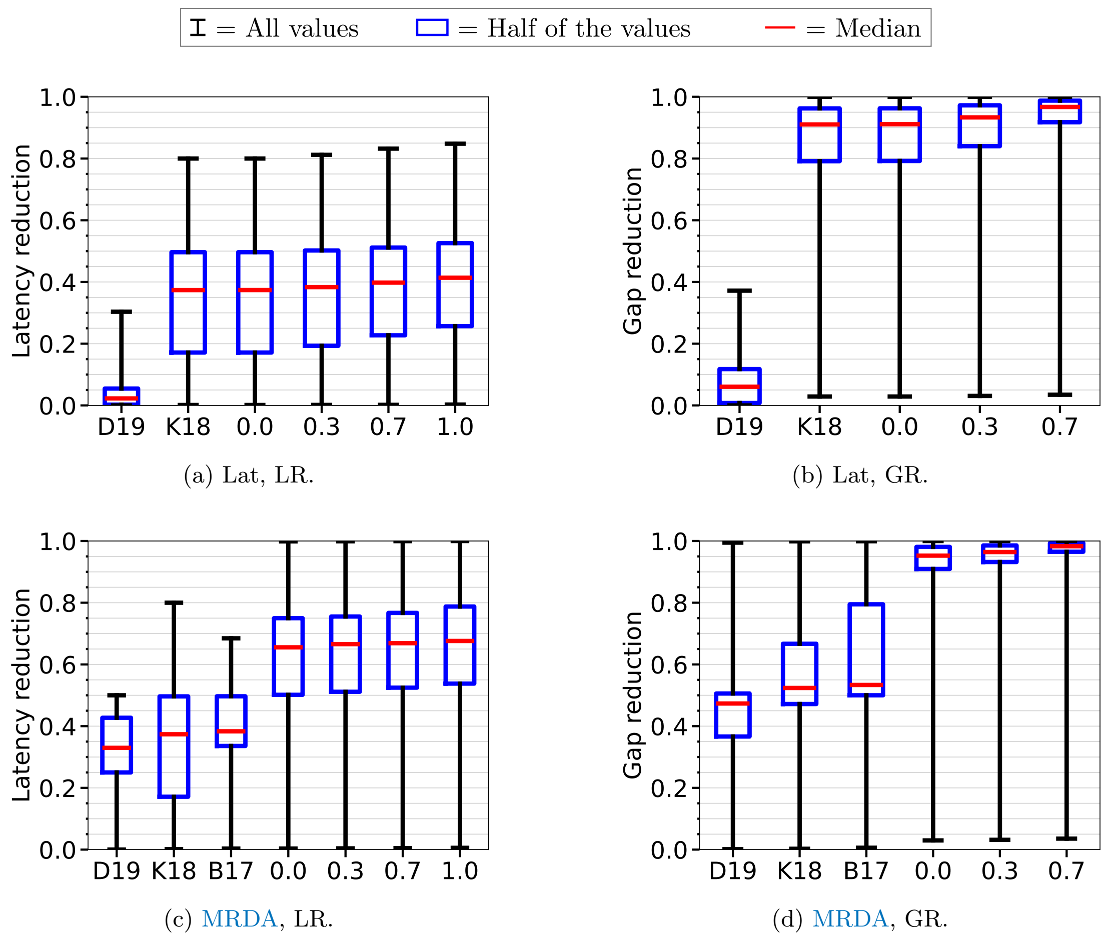
To the best of our knowledge, there are no benchmarks published that detail and reason how to experimentally set up an asynchronous release of tasks, i.e., what a task's phase value should be. Furthermore, the analysis by Kloda et al. [KBS18] is formulated only for cause-effect chains with synchronous tasks. Hence, we consider synchronous task sets (with $\phi_\tau = 0$) in this evaluation. All tasks are scheduled by preemptive RM scheduling.

Cause-Effect Chain Generation Given a generated task set, the set of cause-effect chains is generated according to the description in Section IV-E in [KZH15]. Namely, a set of cause-effect chains containing 30 to 60 cause-effect chains is generated from each task set as depicted in the following steps:

(i) The number of involved activation patterns $P_j \in \{1, 2, 3\}$, i.e., the number of unique periods of tasks in a generated cause-effect chain, is drawn according to the distribution shown in Table VI in [KZH15].

(ii) P_j unique periods are drawn from the task set from a uniform distribution without replacement. More specifically, this step yields a set $\mathbf{T}_j$ of P_j distinct periods.

(iii) For each period in $\mathbf{T}_j$ we draw 2 to 5 tasks at random (without replacement) according to the distribution in Table VII in [KZH15] from the tasks in the task set with the respective period.



(a) Lat, LR.

(b) Lat, GR.

(c) MRDA, LR.

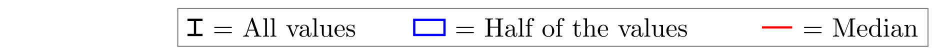
(d) MRDA, GR.

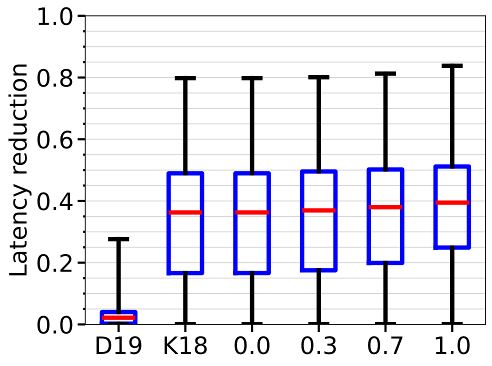
Figure 6.9.: Experiments for the **automotive benchmark**. Our method improves the state of the art, especially for the MRDA.

The resulting cause-effect chains consist of 2 to 15 tasks and no task occurs multiple times in the same cause-effect chain.

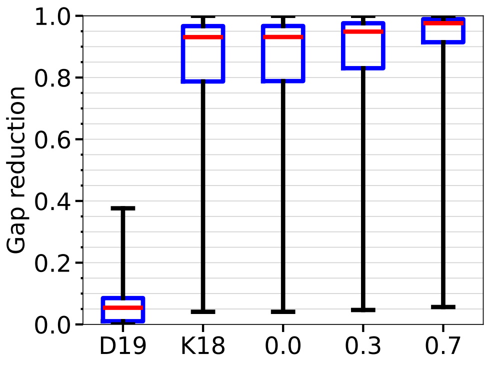
Evaluation Results In Figures 6.9 and 6.10, we show the evaluation results with automotive and uniform task generation, respectively. The boxplots display the evaluation results of the methods by Dürr et al. [Dür+19] **(D19)** and Kloda et al. [KBS18] **(K18)**, as well as of our method **(0.0,0.3,0.7,1.0)** where the BCET of the task is set to $0.0, 0.3, 0.7$ or 1.0 of the WCET. Since **1.0** is the bound given by Corollary 6.34, the gap reduction is always equal to 1 and therefore not reported. For the MRDA, the method by Becker et al. [Bec+17a][7] **(B17)** is displayed as well. The plots show the latency reduction (LR) from Equation (6.67) or the gap reduction (GR) from Equation (6.68). The method by Schlatow et al. [Sch+18a] is omitted since it is dominated by Davare's method for

[7]We apply the method where the WCRT is known (see [Bec+17a, Table 1]).

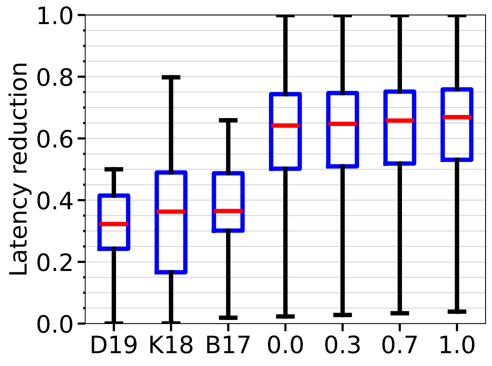




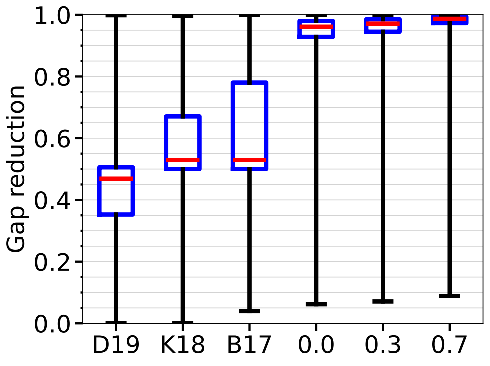
(a) Lat, LR.



(b) Lat, GR.



(c) MRDA, LR.



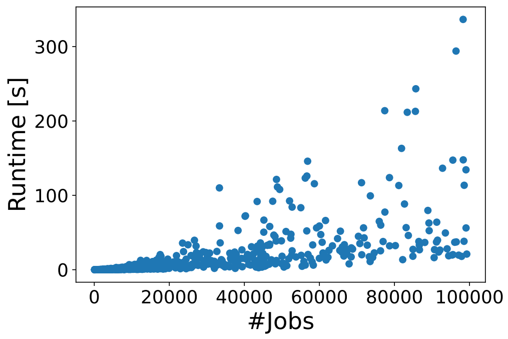
(d) MRDA, GR.

Figure 6.10.: Experiments for the **uniform benchmark**. Again, our method improves the state of the art, especially for the MRDA.

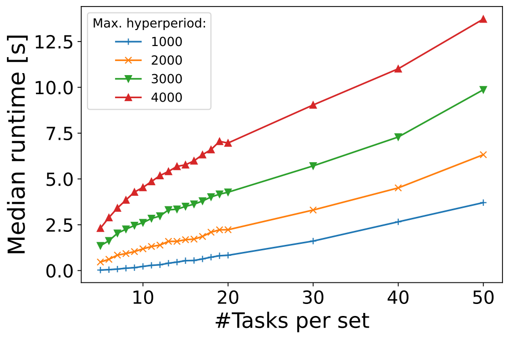
non-harmonic task systems.

Our analysis performs similar to K18 for the end-to-end latency ((a) and (b) in Figures 6.9 and 6.10) and outperforms the state of the art for MRDA ((c) and (d) in Figures 6.9 and 6.10). We observe that under both benchmarks the LR and GR of our method increases when the BCET is closer to the WCET. The GR ((b) and (d) in Figures 6.9 and 6.10) is in median over 90 percent for our methods and for K18, whereas D19 and B17 have a median GR of less than 55 percent in all scenarios.

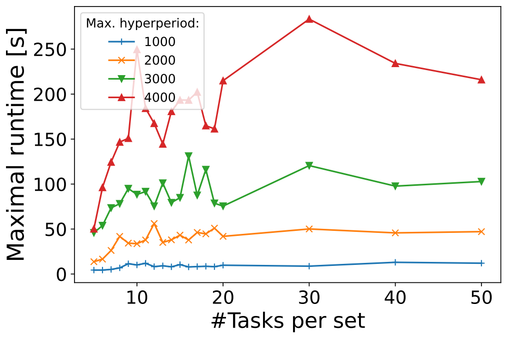
Runtime Evaluation In the following, we study the control parameters that regulate the runtime of our analysis. More specifically, we show that (i) the runtime of our single ECU algorithm is dependent on the number of jobs to be scheduled in the simulation, and (ii) the runtime can be controlled by bounding the hyperperiod of the task sets under analysis. For the measurements, we use a machine equipped with 2x AMD EPYC



(a) Dependency between number of jobs in the schedule and runtime of our analysis.



(b) Median runtime measurements of our analysis for different hyperperiod bounds.



(c) Maximal runtime measurements of our analysis for different hyperperiod bounds.

Figure 6.11.: Runtime Evaluation.

7742 running Linux, i.e., in total 256 threads with 2.25 GHz and 256 GB RAM. Each measurement runs on one independent thread and covers the time for simulation of the best-case and worst-case schedule and for deriving the end-to-end latency and MRDA. Both experiments rely on uniform task generation [BB05] with synchronous tasks. The total utilization is pulled uniformly from $[50, 90]$ [%] and task periods are pulled log-uniformly from the integers in $[1, 20]$. As a result, the hyperperiod of the task set is between 1 and $\mathrm{lcm}(1, 2, \dots, 20) = 232\,792\,560$. From each task set, we choose 5 tasks at random without replacement and combine them to a cause-effect chain.

For (i), we generate 1 000 task sets with 5 to 20 tasks each. The results are depicted in Figure 6.11a for task sets where the number of scheduled jobs is below 100 000. The number of scheduled jobs is upper bounded by $\sum_{\tau\in\mathbb{T}} \frac{2\cdot H(\mathbb{T})}{T_\tau} \leq \frac{2\cdot H(\mathbb{T})}{\min_{\tau\in\mathbb{T}} T_\tau} \cdot |\mathbb{T}|$. For a fixed number of tasks and a fixed range of periods, we can control the number of scheduled jobs by constraining the hyperperiod $H(\mathbb{T})$. For example, when the hyperperiod is 1 000 and the number of tasks is 20, then there are at most 40 000 scheduled jobs. The exact number of jobs may be lower since big hyperperiods require bigger periods T_τ.

In (ii), for a given number of tasks per set, we create 1 000 task sets with hyperperiod in the range from 0 to 1 000, 1 000 to 2 000, 2 000 to 3 000, and 3 000 to 4 000, each. The median and maximal runtimes sorted by hyperperiod bounds are depicted in Figure 6.11b and 6.11c, respectively. We observe that with a low hyperperiod, the maximum runtime can be controlled.

6.2.3 Partitioned Job Chains

In this section, we consider a cause-effect chain $E = (\tau_1 \to \cdots \to \tau_n)$ where all tasks τ_i are periodic tasks with offset, communicating under LET. That is, each task τ_i can be abstracted as

$$\tau_i \in \mathrm{PerOff}(T_{\tau_i}, \phi_{\tau_i}), \tag{6.69}$$

and the read-event and write-event of each job $\tau_i(j) \in \mathbb{J}$ is at the release $r^\omega_{\tau_i(j)}$ and deadline $r^\omega_{\tau_i(j)} + D_{\tau_i}$. Due to the fixed release pattern of periodic tasks with offsets, the read- and write-events repeat every hyperperiod. As a result, $\ell(\vec{ac}^\omega_E(z)) \leq \ell(\vec{ac}^\omega_E(z) + H(E))$, and it is sufficient to consider immediate forward augmented job chains $\vec{ac}^\omega_E(z)$ with $z \in (\mathrm{re}^\omega(\tau_1(W^\omega_{E,\tau_1})), \mathrm{re}^\omega(\tau_1(W^\omega_{E,\tau_1})) + H(E)]$. Furthermore, it is sufficient to only calculate $\vec{ac}^\omega_E(z)$ where z is right after a read-event of task τ_1, i.e. for $z = \phi_{\tau_1} + (k-1) \cdot T_{\tau_1} + \varepsilon$ with $k \in \mathbb{N}$, where $\varepsilon > 0$ is an infinitesimal number. This approach has been detailed for example in [Gün+21a]. Moreover, Kordon and Tang [KT20] compute the MRDA efficiently under LET, using this procedure as a backbone.

Still, to calculate the end-to-end latency, up to $\frac{\Phi(E)+H(E)}{T_{\tau_1}}$ immediate forward augmented job chains have to be considered. This can be time-consuming, especially if T_{τ_1} is small compared to the other task periods. We exploit the description of the end-to-end latency using p-partitioned job chains $pc^\omega_{E,p}(m)$, i.e.,

$$\mathrm{Lat}(E) = \sup_{\omega \in \Omega} \sup \left\{ \ell(pc^\omega_{E,p}(m)) \,\middle|\, m \geq W^\omega_{E,\tau_p} \right\}, \tag{6.70}$$

as proven in Theorem 6.7, to speed up the calculation of the end-to-end latency. In particular, choosing $p \in \{1, \dots, n\}$ which maximizes T_{τ_p}, only $\frac{\Phi(E)+H(E)}{T_{\tau_p}}$ partitioned job chains need to be constructed. Since task periods in automotive systems usually range over 3 orders of magnitude (cf. [KZH15]), the number of objects to be constructed can be reduced by a factor of up to 1 000. In the following, we evaluate the speed-up that is obtained by considering p-partitioned job chains instead of immediate forward augmented job chains. The results presented in this section appeared in ECRTS 2023 [Gün+23b].

6.2.3.1 Description of our Approach Using Partitioned Job Chains

Partitioned job chains have been introduced in Section 6.1.2.1 to prove the equivalence of MRT and MDA. As a side effect of the equivalence, in Theorem 6.7, the end-to-end latency has been described by partitioned job chains (see Equation (6.70)). Since the read- and write-events repeat every hyperperiod, also the partitioned job chains repeat.

Therefore, it is sufficient to consider only $\frac{H(E)}{T_{\tau_p}}$ partitioned job chains, as formalized by the following theorem.

Theorem 6.36. *The end-to-end latency of E can be computed by constructing $\frac{H(E)}{T_{\tau_p}}$ many p-partitioned job chains. More specifically,*

$$\operatorname{Lat}(E) = \sup_{\omega\in\Omega} \sup \left\{ \ell(pc^{\omega}_{E,p}(m)) \,\middle|\, W^{\omega}_{E,\tau_p} \leq m < W^{\omega}_{E,\tau_p} + \frac{H(E)}{T_{\tau_p}} \right\} \tag{6.71}$$

for any $p \in \{1, \ldots, n\}$.

Proof. Let $p \in \{1, \ldots, n\}$. By Theorem 6.7, the end-to-end latency $\operatorname{Lat}(E)$ is given by $\operatorname{Lat}(E) = \sup_{\omega\in\Omega} \operatorname{Lat}(E, \omega) = \sup_{\omega\in\Omega} \sup \left\{ \ell(pc^{\omega}_{E,p}(m)) \,\middle|\, m \geq W^{\omega}_{E,\tau_p} \right\}$. We consider an arbitrary $m \geq W^{\omega}_{E,\tau_p}$ and $m' = m + \frac{H(E)}{T_{\tau_p}}$. To prove the theorem it is sufficient to show that $\ell(pc^{\omega}_{E,p}(m)) \geq \ell(pc^{\omega}_{E,p}(m'))$.

We denote the jobs of the partitioned job chains as follows:

$$pc^{\omega}_{E,p}(m) = (\tau_1(m_1), \ldots, \tau_p(m_p), \tau_p(\tilde{m}_p), \ldots, \tau_n(\tilde{m}_n)) \tag{6.72}$$

$$pc^{\omega}_{E,p}(m') = (\tau_1(m'_1), \ldots, \tau_p(m'_p), \tau_p(\tilde{m}'_p), \ldots, \tau_n(\tilde{m}'_n)) \tag{6.73}$$

Furthermore, for convenience, we define $h_i := \frac{H(E)}{T_{\tau_i}}$ for all $i = 1, \ldots, n$. We know that $\ell(pc^{\omega}_{E,p}(m)) = \mathrm{we}^{\omega}(\tau_n(\tilde{m}_n)) - \mathrm{re}^{\omega}(\tau_1(m_1)) = \mathrm{we}^{\omega}(\tau_n(\tilde{m}_n + h_n)) - \mathrm{re}^{\omega}(\tau_1(m_1 + h_1))$. Therefore, to prove that $\ell(pc^{\omega}_{E,p}(m)) \geq \ell(pc^{\omega}_{E,p}(m'))$, it is sufficient to show that both $\tau_n(\tilde{m}_n + h_n) \succcurlyeq \tau_n(\tilde{m}'_n)$ and $\tau_1(\tilde{m}_1 + h_1) \preccurlyeq \tau_1(\tilde{m}'_1)$ hold.

By definition, $(\tau_1(m_1 + h_1), \ldots, \tau_p(m_p + h_p))$ is a job chain for $(\tau_1 \to \cdots \to \tau_p)$, and $(\tau_1(m'_1), \ldots, \tau_p(m'_p))$ is an *immediate backward* job chain. Furthermore, we have $\tau_p(m_p + h_p) = \tau_p(m'_p)$. Therefore, $\tau_1(m_1 + h_1) \preccurlyeq \tau_1(m'_1)$ by Lemma 6.2.

Similarly, $(\tau_p(\tilde{m}_p + h_p), \ldots, \tau_n(\tilde{m}_n + h_n))$ is a job chain, and $(\tau_p(\tilde{m}'_p), \ldots, \tau_n(\tilde{m}'_n))$ is an immediate forward job chain with $\tau_p(\tilde{m}_p + h_p) = \tau_p(\tilde{m}'_p)$. Hence, $\tau_n(\tilde{m}_n + h_n) \succcurlyeq \tau_n(\tilde{m}'_n)$ by Lemma 6.2. □

We note that the construction of $pc^{\omega}_{E,p}(m)$ is independent of the system evolution ω, since the release pattern does not depend on ω and therefore the read- and write-events do not depend on ω as well. Hence, $\sup_{\omega\in\Omega}$ can be removed in Equation (6.71). By choosing $p \in \{1, \ldots, n\}$ with the maximum T_{τ_p}, the amount of partitioned job chains to be constructed is minimized. Our approach can be formalized as follows.

Our approach (Our) using partitioned job chains:

(i) Choose $p \in \{1, \ldots, n\}$ with the maximum T_{τ_p}.

(ii) Calculate W^{ω}_{E,τ_p} by finding the first immediate backward job chain that exists.

(iii) Construct p-partitioned job chains $pc^{\omega}_{E,p}(m)$ for $m = W^{\omega}_{E,\tau_p}, W^{\omega}_{E,\tau_p} + 1, \ldots, W^{\omega}_{E,\tau_p} + \frac{H(E)}{T_{\tau_p}} - 1$.

In the next section we evaluate the speedup that can be achieved by our proposed approach.

6.2.3.2 Evaluation

In the following, we compare the time consumption of our approach (Our), defined in the previous section, with the approach detailed in [Gün+21a].

Approach by Günzel et al. [Gün+21a] (G21):

(i) Calculate W^{ω}_{E,τ_p} by finding the first immediate backward job chain that exists.

(ii) Construct immediate forward augmented job chains $\vec{ac}^{\omega}_{E}(r^{\omega}_{\tau_1(m)} + \varepsilon)$ for $m = W^{\omega}_{E,\tau_p}, W^{\omega}_{E,\tau_1} + 1, \dots, W^{\omega}_{E,\tau_1} + \frac{H(E)}{T_{\tau_1}} - 1$, where ε is the infinitesimal number.

Task set generation. We randomly generate 10 000 task sets $\mathbb{T}$, each with a random cardinality of $n_{\mathbb{T}} \in [50, 100] \cap \mathbb{N}$. To evaluate the approaches with tasks similar to a real-world application, we generate tasks according to the automotive benchmark by Kramer et al. [KZH15]. In particular, for each task τ we draw a period T_τ from the set $\{1, 2, 5, 10, 20, 50, 100, 200, 1\,000\}$ according to the related share[8] of these periods in [KZH15, Table III, IV and V]. We assume implicit deadlines, i.e., the relative deadline D_τ is set to the period T_τ, and the phase ϕ_τ is set to 0 for all tasks. Since the read- and write-events under LET are independent of the execution behavior, no execution time of the task is synthesized.

Cause-effect chain generation. For each task set $\mathbb{T}$, we generate a cause-effect chain E according to Kramer et al. [KZH15, Section IV-E]. In particular, we apply the following steps:

(i) The number of involved activation patterns $P_E \in \{1, 2, 3\}$, i.e., the number of unique periods of the tasks in the cause-effect chain E, is drawn according to the distribution in [KZH15, Table VI].

(ii) A set S_E of P_E unique periods is uniformly drawn from the periods in $\mathbb{T}$.

(iii) For each period in S_E, we draw 2 to 5 tasks at random (without replacement) according to the distribution in [KZH15, Table VII] from the tasks in $\mathbb{T}$ with the respective period. The cause-effect chain E consists of these tasks in random order.

If there are no sufficient tasks with the required period in Step (iii), we discard the set and randomly draw another task set until a cause-effect chain is successfully created.

Evaluation results. For each cause-effect chain, we apply **(G21)** and **(Our)** to compute the end-to-end latency. For the runtime measurements, we use a machine equipped with 2x AMD EPYC 7742 running Linux, i.e., in total 256 threads with 2.25 GHz and 256 GB RAM. Each measurement runs on one independent thread.

[8]The sum of probabilities in [KZH15] is only 85 %. The remaining 15 % are reserved for angle-synchronous tasks that we do not consider here. Therefore, all share values are divided by 0.85.

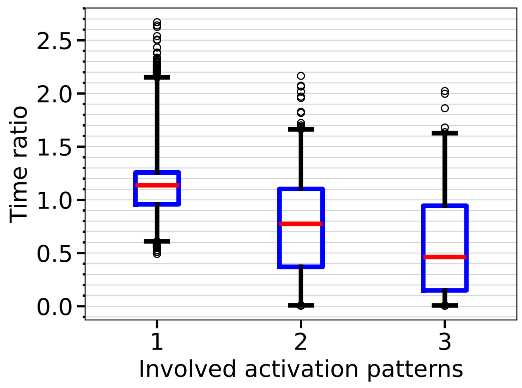


Figure 6.12.: The time_ratio for different number of involved activation patterns. Red line depicts median, blue box 50 % of the data, and the whiskers all data except the highest and the lowest 1 %.

involved patterns	1	2	3
number values	8 109	1 436	455
min speed-up	0.37	0.46	0.49
median speed-up	0.88	1.29	2.16
mean speed-up	0.92	5.43	8.82
max speed-up	2.03	180.93	234.92

Table 6.4.: Speed-up for different number of activation patterns.

Observation 1: For all 10 000 cause-effect chains all latency values obtained by **(Our)** coincide with the corresponding values obtained by **(G21)**.

Observation 2: Our results can reduce the required time for computing the end-to-end latencies significantly. Specifically, Figure 6.12 depicts the time ratio, defined by $\frac{\text{time\_our}}{\text{time\_g21}}$, where time_our and time_g21 is the time needed by **(Our)** and **(G21)**, respectively, to derive the end-to-end latency (considering the minimal runtime over 10 000 runs for each of the 10 000 cause effect chains). On the x-axis the number of different activation patterns, i.e., the number of different periods in the cause-effect chain, is shown. Additionally, Table 6.4 shows the speed-up, defined as $\frac{\text{time\_g21}}{\text{time\_our}}$.

We observe that **(Our)** reduces the required time significantly in most cases compared to **(G21)** if there are tasks with different periods in the cause-effect chain. Specifically, when there is more than one activation pattern, then **(Our)** reduces the number of constructed chains by choosing p such that $E(p)$ has the largest period, and a much larger speed-up is observed such cases. However, we note that if the first task has already the highest period, like for the case with only one activation pattern, then determining the task with the highest period leads to an overhead which has negative impact on the runtime. Due to this overhead, **(Our)** does not always outperform **(G21)**.

This observation is supported by Figure 6.13, which shows the absolute time for the

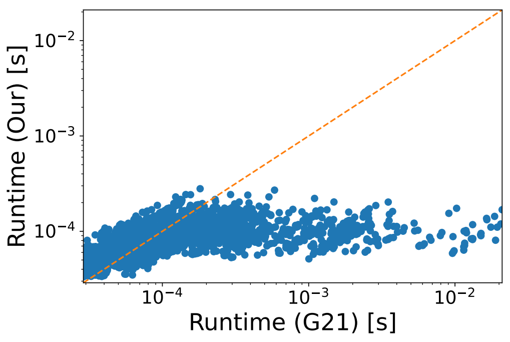


Figure 6.13.: Absolute time for computation of end-to-end latency in log scale. Results for **(G21)** are on the x-axis, and results for **(Our)** are on the y-axis.

computation of the end-to-end latency for **(G21)**, on the x-axis, and for **(Our)**, on the y-axis. As depicted, when **(G21)** performs already well, then **(Our)** might perform slightly worse due to the overhead for choosing p. However, if the runtime of **(G21)** is long, **(Our)** can reduce the runtime significantly. In all evaluated cases, the runtime of **(Our)** to obtain the end-to-end latency is bounded by $3 \cdot 10^{-4}$ s, while for **(G21)** a runtime of up to $2 \cdot 10^{-2}$ s can be observed.

6.2.4 Distributed Setups

Whereas Sections 6.2.1–6.2.3 focus on local cause-effect chains, i.e., cause-effect chains only involving tasks scheduled on one ECU, in this section we consider interconnected cause-effect chains, i.e., cause-effect chains distributed among several ECUs. Whereas tasks on a single ECU are scheduled using one synchronized clock, the clocks among different ECUs are usually not synchronized. Such systems are of high practical relevance. For instance, according to the Flexray standard [Fle05], the data communication cycle is divided into static segments (i.e., synchronous time-triggered communication) and dynamic segments (i.e., asynchronous event-driven communication).

There are mechanisms to achieve synchronization of several ECUs. That is, to enable a coherent collaboration among multiple distributed devices, clock synchronization techniques [KS22] enable one global system clock for all devices; a so-called globally synchronized system. Globally synchronized systems are usually easier to be optimized and analyzed, but the synchronization mechanism imposes strong dependencies (e.g., on the master node for clock synchronization), which makes the distributed system fragile against fault tolerance [Ben+02]. Therefore, accepting globally asynchronous clocks with offset and jitter enables distributed embedded systems that are *robust against imperfect synchronization of the architecture*. We call such systems Globally Asynchronized Locally Synchronized (GALS) systems.

GALS systems provoke new challenges for the analysis. That is, we have to account for asynchronous clocks for different tasks in the cause-effect chains. Moreover, multiple single ECUs are connected by an inter-communication infrastructure, e.g., Controller Area Network (CAN) [Bos91] or FlexRay [Fle05], which has to be accounted for. This can be done by applying the compositional property of cause-effect chains, proven in Theorem 6.3. In the following, we discuss how to achieve the analysis for interconnected cause-effect chains. Afterwards, we evaluate our findings. The results of this section appeared in RTAS 2021 [Gün+21a] and its journal extension [Gün+23a].

6.2.4.1 Our Approach Using Cutting Theorem

Considering an interconnected cause-effect chain $E = (\tau_1 \to \cdots \to \tau_n)$, the compositional property, formalized as *Cutting Theorem* in Theorem 6.3, can be utilized to bound the end-to-end latency as follows:

$$\mathrm{Lat}(E) \leq \mathrm{Lat}(E_1) + \mathrm{Lat}(E_2) \tag{6.74}$$

where $E_1 = (\tau_1 \to \cdots \to \tau_k)$ and $E_2 = (\tau_{k+1} \to \cdots \to \tau_n)$ for some $k \in \{1, \dots, n\}$. The intuition behind our approach is to cut the cause-effect chain whenever the ECU changes to obtain a set of local cause-effect chains that can be analyzed using the results from the previous sections. To achieve this, we assume that each task is assigned to a single ECU, i.e., partitioned scheduling.

To account for the communication infrastructure between different ECUs, we model the communication by an additional *communication tasks* τ^c. Each communication task has the purpose of transferring data from one specific ECU to another ECU. More specifically, jobs released by communication tasks read data from a shared resource of one ECU and write it to a shared resource of another ECU. We assume that the communication tasks are executed on dedicated components and do not impair job execution of the non-communication tasks. For notational convenience those dedicated components are treated as communication ECUs. This abstraction is valid for common inter-communication infrastructures like CAN [Bos91] or FlexRay [Fle05]. Our estimations utilize only knowledge about the WCRT R_{τ^c} and maximum inter-arrival time $T_{\tau^c}^{max}$, i.e., maximum time between two recurrent job releases, of each communication task τ^c.

In particular, we assume that each interconnected cause-effect chain E consists of local cause-effect chains $E_1, \dots, E_m$ and communication tasks $\tau_1^c, \dots, \tau_{m-1}^c$. That is, each E_i is executed on one ECU and τ_i^c transfers data from the ECU of E_i to the ECU of E_{i+1}. Hence, the cause-effect chain including communication tasks is

$$E = (E_1 \to \tau_1^c \to E_2 \to \tau_2^c \to \cdots \to E_m) \tag{6.75}$$

where each E_i is a local cause-effect chain. Please note that we use cause-effect chains E_i as entries of E to indicate that all *tasks* of E_i are added to the chain. The chain E is not a sequence of cause-effect chains. We do this for the sake of readability.

By using the Cutting Theorem, we achieve the following.

Theorem 6.37 (Analysis of interconnected cause-effect chain with communication tasks)**.** *The end-to-end latency of E is upper bounded by:*

$$\mathrm{Lat}(E) \leq \sum_{i=1}^{m} \mathrm{Lat}(E_i) + \sum_{i=1}^{m-1} \mathrm{Lat}((\tau_i^c)) \tag{6.76}$$

Proof. This is achieved by cutting E into local parts using the compositional property from Theorem 6.3. □

For $\mathrm{Lat}(E_i)$, any bound on the end-to-end latency of the local cause-effect chain can be applied. Several such bounds are provided in the previous sections. For $\mathrm{Lat}((\tau_i^c))$, since the cause-effect chain consists of only one task, trivial upper bounds are $\mathrm{Lat}((\tau_i^c)) \leq T_{\tau_i^c} + R_{\tau_i^c}$ if τ_i^c uses implicit communication, and $\mathrm{Lat}((\tau_i^c)) \leq T_{\tau_i^c} + D_{\tau_i^c}$ if τ_i^c uses communication under LET.

Similar theorems can be stated for the MRDA and MRRT.

Theorem 6.38. *The MRDA and MRRT of E are upper bounded by:*

$$\mathrm{MRRT}(E) \leq \mathrm{MRRT}(E_1) + \sum_{i=2}^{m} \mathrm{MRRT}(E_i) + \sum_{i=1}^{m-1} \mathrm{MRRT}((\tau_i^c)) \tag{6.77}$$

$$\mathrm{MRDA}(E) \leq \mathrm{MRDA}(E_m) + \sum_{i=1}^{m-1} \mathrm{MRDA}(E_i) + \sum_{i=1}^{m-1} \mathrm{MRDA}((\tau_i^c)) \tag{6.78}$$

Proof. This is achieved by applying the extensions of the Cutting Theorem, formulated in Equations (6.13) and (6.14). □

6.2.4.2 Evaluation

We evaluate our findings by comparing the bounds for the end-to-end latency (Lat) and for the MRDA of synthetic interconnected cause-effect chains under implicit communication. The following approaches are considered:

- **Our approach:** Using Theorem 6.37 or Theorem 6.38 together with the bounds from Section 6.2.2 for the local cause-effect chains. We denote our method as **(0.0,0.3,0.7,1.0)** where the BCET of each task is set to $0.0, 0.3, 0.7$ or 1.0 of the WCET, respectively.
- **Dürr et al. [Dür+19] (D19):** Using the bounds stated in Theorem 3.10.
- **Davare et al. [Dav+07] (D07):** Using the bound stated in Theorem 6.20. This bound is utilized to normalize all other end-to-end bounds, since this method yields the most pessimistic result.

The method by Schlatow et al. [Sch+18a] is omitted since it is dominated by Davare's method for non-harmonic task systems.

We define the *latency reduction* $LR(\mathcal{M})$ of an analysis method $\mathcal{M}$ with respect to an evaluated bound $B(\cdot)$, e.g., MRT, by

$$LR(\mathcal{M}) := (B(D07) - B(\mathcal{M}))/B(D07). \tag{6.79}$$

The following setup is considered in the evaluation, which is released on Github [TU 22b].

Task and Task Set Generation We generate 1 000 *automotive task sets* for each cumulative task set utilization of $U = 50\,\%$, $60\,\%$, $70\,\%$, $80\,\%$, and $90\,\%$, following the procedure described in Section 6.2.2.5, either utilizing the automotive benchmark [KZH15] or uniform task set generation [BB05].

Furthermore, in order to evaluate interconnected cause-effect chains, we assume a T-FP communication fabric. Specifically, we draw the period of each message log-uniformly at random from the range $10\,ms$ to $10\,000\,ms$ and truncate the result to the next smallest integer to model the communication frequency. Furthermore, we assume that the transmission time of a message, i.e., execution time on the communication fabric, is a constant. In our evaluations, we utilize the constant time from standard 2.0A CAN-Bus with 1 Mbps bandwidth, where transmitting 8 bytes of data (along with its 66 bits overhead due to its header and tail) takes $C_i = 130 \cdot 10^{-3}\,ms$. Given the set of all messages and with random priority assignment, the WCRT of each communication task is calculated using Time-Demand Analysis (TDA) for non-preemptive tasks.

Cause-Effect Chain Generation First, we generate 30 to 60 cause-effect chains from each task set as described in Section 6.2.2.5. Afterwards, we generate 10 000 interconnected cause-effect chains by selecting 5 cause-effect chains of different task sets with the same utilization under a uniform distribution. For each selection, we create 20 communication tasks as described above. Among them, 4 are chosen randomly to connect the 5 communication tasks.

Evaluation Results Figures 6.14 and 6.15 show the evaluation results under automotive and uniform task generation, respectively. The plots show the latency reduction (LR) from Equation (6.79). Our method outperforms the state of the art significantly: Whereas our method shows a median LR of more than 30 percent in all cases, (D19) has a median LR of around 5 percent or less.

6.2.5 Heterogeneous Setups

As stated by Hamann et al. [Ham+17], automotive systems are composed of cause-effect chains that use heterogeneous communication mechanisms. While implicit communication and LET both ensure data consistency, tasks use LET communication if timing determinism has to be further pursued, and tasks use implicit communication if reduced end-to-end latency is the primary objective.

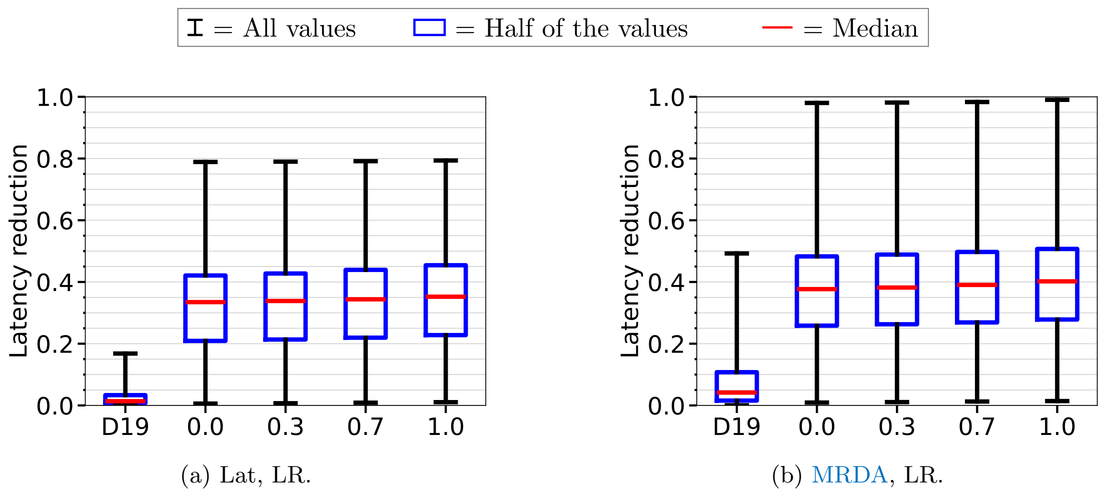


Figure 6.14.: **Inter-ECU** experiments for the **automotive benchmark**. Our method improves the state of the art for all setups.

However, the current state-of-the-art analyses (summarized in Section 3.3.2) only permit cause-effect chains with homogeneous communication mechanisms with either only implicit communication or only LET. That is, in case LET has to be applied for some tasks to improve the temporal determinism, all other tasks in the cause-effect chain must be converted to LET accordingly. As a task may be involved in multiple cause-effect chains and LET communication typically increases end-to-end latency, this conversion may have a domino effect with respect to timing, which is only due to the lack of proper analysis. To conquer such unnecessary conversion, having the possibility of LET and implicit communication in one cause-effect chain is practically relevant.

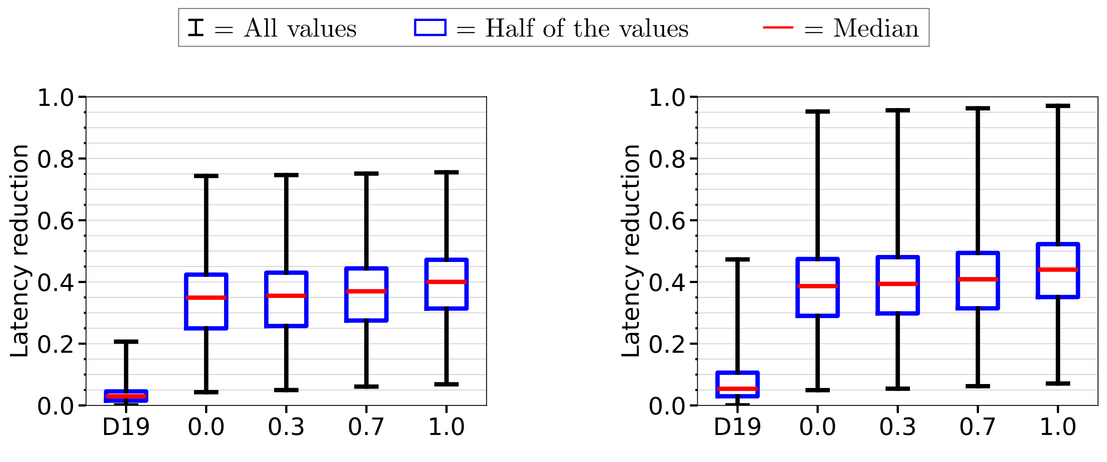
Similarly, the state of the art only allows all tasks of the cause-effect chain to have the same abstraction of job releases. Therefore, if only a single task of the cause-effect chain is not periodic, then only analyses for cause-effect chains of sporadic tasks are available. Hence, this leads to over-approximation along the whole cause-effect chain since the information about the release pattern cannot be exploited for the analysis.

In this work, we present the first formal analysis for end-to-end analysis of cause-effect chains $E = (\tau_1 \to \cdots \to \tau_n)$, scheduled by a T-FP scheduling algorithm $\mathcal{A}$, that allows heterogeneous types of recurrent tasks and different communication mechanisms, i.e., (i) a mixed setup of sporadic and periodic tasks that (ii) communicate by a mixed setup of LET and implicit communication mechanism. More specifically, each task is abstracted as periodic or sporadic, i.e.,

$$\tau \in \mathrm{PerOff}(T_\tau, \phi_\tau) \qquad \text{or} \qquad \tau \in \mathrm{SporMinMax}(T_\tau^{\min}, T_\tau^{\max}), \tag{6.80}$$

and each task in the system communicates using either *implicit communication* or *LET*. We use

$$\mathrm{comm}(\tau) \in \{LET, impl\} \tag{6.81}$$



(a) Lat, LR. (b) MRDA, LR.

Figure 6.15.: **Inter-ECU** experiments for the **uniform benchmark**. Again, our method improves the state of the art for all setups.

to denote the communication policy of task τ. We believe that the analysis of heterogeneous setups solves an open problem for industry driven systems.

For the analysis of heterogeneous chains, we apply Theorem 6.3 (Cutting) to divide the cause-effect chain into homogeneous subchains. We derive three analytical bounds:

A1 A baseline approach in Theorem 6.39, achieved by cutting the chain at each task.

A2 An analysis achieved by cutting the chains into largest homogeneous subchains, as summarized in Remark 6.40.

A3 An improved analysis which analyzes heterogeneous communication directly, using the analysis principles from Section 6.2.1, and which applies the Cutting Theorem only if the release policy changes. This approach is summarized in Remark 6.43.

Our analysis assumes a bounded response time $R_\tau^\mathcal{A} < \infty$ for tasks $\text{comm}(\tau) = \textit{impl}$, and $R_\tau^\mathcal{A} \leq D_\tau$ if $\text{comm}(\tau) = \textit{LET}$. Furthermore, we assume that, if a deadline is set of a task, that this task is arbitrary deadline, i.e., $D_\tau > T_\tau$ and $D_\tau > T_\tau^{\max}$ is allowed. While this section assumes that all tasks are executed on a single ECU, i.e., E is local, using the Cutting Theorem where the ECU changes, similar to Section 6.2.4, we can extend our approaches to interconnected cause-effect chains as well. This section is organized as follows:

- We explain how to use the Cutting Theorem for heterogeneous cause-effect chains to derive analytical bounds in Section 6.2.5.1. Specifically, in this section we derive analyses A1 and A2.
- In Section 6.2.5.2, we apply the analysis principles from Section 6.2.1 to tighten the analyis further. Specifically, in this section we derive analysis A3.

- We evaluate our approaches in Section 6.2.5.3, showing that for some systems the two more sophisticated approaches A2 and A3 outperform the baseline significantly, while for other systems the baseline A1 is already satisfactory.

The results of this section appeared in RTNS 2023 [Gün+23d].

6.2.5.1 Application of Cutting Theorem

In this section, we utilize Theorem 6.3 (Cutting) to derive analyses for heterogeneous setups. The Cutting Theorem was formulated for general communication means and release constraint, including heterogeneous systems. Given the cause-effect chain $E = (\tau_1 \to \cdots \to \tau_n)$, the Cutting Theorem can be formulated as

$$\mathrm{Lat}(E) \leq \mathrm{Lat}(E_1) + \mathrm{Lat}(E_2) \tag{6.82}$$

where $E_1 = (\tau_1 \to \cdots \to \tau_k)$ and $E_2 = (\tau_{k+1} \to \cdots \to \tau_n)$ for some $k \in \{1, \dots, n\}$. The strategy of this section is to apply the Cutting theorem several times until the problem is reduced to bounding the end-to-end latency of homogeneous cause-effect chains E_i. For each homogeneous cause-effect chain, results from the literature can be applied to obtain an upper bound for $\mathrm{Lat}(E)$.

The easiest way to achieve homogeneous cause-effect chains is to cut the chain E into chains of length 1. This leads to the following baseline for an upper bound on the end-to-end latency.

Theorem 6.39 (Baseline). *The end-to-end latency of cause-effect chain E is bounded by*

$$\mathrm{Lat}(E) \leq \sum_{i=1}^{n} \mathrm{Lat}((\tau_i)) \tag{6.83}$$

where $\mathrm{Lat}((\tau_i)) = T^{max}_{\tau_i} + R^{\mathcal{A}}_{\tau_i}$ *if* $\mathrm{comm}(\tau_i) = impl$ *and* $\mathrm{Lat}((\tau_i)) = T^{max}_{\tau_i} + D_{\tau_i}$ *if* $\mathrm{comm}(\tau_i) = LET$. *Please note that this baseline is applicable to periodic tasks as well since periodic tasks can be considered as sporadic tasks with* $T^{min}_{\tau_i} = T^{max}_{\tau_i} = T_{\tau_i}$.

Proof. We apply Theorem 6.3 (Cutting) to cut the cause-effect chain E after every task. This way, we obtain n subchains, each with a length of 1. For a cause-effect chain with only one task τ, the end-to-end latency is upper bounded by $T^{max}_{\tau} + R^{\mathcal{A}}_{\tau}$ if $\mathrm{comm}(\tau) = impl$ and by $T^{max}_{\tau} + D_{\tau}$ if $\mathrm{comm}(\tau) = LET$. □

While the provided baseline is simple to compute, in many cases this is only a pessimistic upper bound. However, instead of cutting E into chains of length 1, we can use the Cutting Theorem to cut E into larger homogeneous subchains. Specifically, we first determine the *longest* subsequences of E such that all tasks in the subsequence have the same communication policy (either LET or implicit communication) and the same release constraint (either periodic or sporadic). We denote these subsequences by $E_1, \dots, E_K$ for some $K \in \mathbb{N}$ and write $E = (E_1 \to \cdots \to E_K)$. By applying the Cutting Theorem, we obtain an upper bound on the end-to-end latency by analyzing the end-to-end latency of E_i for all $i \in \{1, \dots, K\}$, and summing up the results. To compute upper bounds for the homogeneous subchains E_i, with $i \in \{1, \dots, K\}$, we can apply any analysis for homogeneous cause-effect chains.

Remark 6.40 (Homogeneous Cutting Analysis)**.** *To conclude, the analysis of the heterogeneous case is achieved by cutting the cause-effect chain into the largest homogeneous subchains, i.e., all tasks of the subchain have the same communication policy and the same release constraint, and utilizing any upper bounds for the homogeneous subchains from the literature. We specifically consider the bounds provided in Section 6.2.1.1:*

- *Sporadic, LET: Theorem 6.19*
- *Sporadic, implicit communciation: Theorem 6.20 or 6.21*
- *Periodic, LET: Theorem 6.22*
- *Periodic, implicit communication: Theorem 6.23*

6.2.5.2 Application of Analysis Principles to Heterogeneous Communication

Reducing the analysis to homogeneous systems has substantial drawbacks when two subsequent tasks in one cause-effect chain adhere to different communication policies. Specifically, the periodic release pattern between tasks that communicate differently cannot be exploited. In this section, we take advantage of the release pattern by only cutting when the release constraint changes, i.e., from sporadic to periodic or from periodic to sporadic. For the resulting purely periodic and purely sporadic cause-effect chains with heterogeneous communication mechanisms, we provide an analysis derived from the principles presented in Section 6.2.1.1. We start by looking at chains comprised of sporadic tasks.

Theorem 6.41 (Sporadic, heterogeneous communication)**.** *If all tasks of the local cause-effect chain E are sporadic, then the end-to-end latency is bounded by*

$$\mathrm{Lat}(E) \leq \sum_{i=1}^{n} T_{\tau_i}^{max} + CX_i \tag{6.84}$$

where CX_i is defined as follows:

- $CX_i = D_{\tau_i}$*, if* $\mathrm{comm}(\tau_i) = LET$*,*
- $CX_i = 0$*, if* τ_{i+1} *exists,* $\mathrm{comm}(\tau_i) = impl = \mathrm{comm}(\tau_{i+1})$ *and* τ_i *has higher priority than* τ_{i+1}*,*
- *and* $CX_i = R_{\tau_i}^{\mathcal{A}}$*, otherwise.*

	$\text{comm}(\tau_{i+1}) = \textit{impl}$		$\text{comm}(\tau_{i+1}) = \textit{LET}$
$\text{comm}(\tau_i) = \textit{impl}$	$\xi_{\tau_{i+1}}(r^\omega_{J_i})$ $\xi_{\tau_{i+1}}(r^\omega_{J_i} + R^{\mathcal{A}}_{\tau_i})$	if $\pi(\tau_i) > \pi(\tau_{i+1})$ otherwise	$\xi_{\tau_{i+1}}(r^\omega_{J_i} + R^{\mathcal{A}}_{\tau_i})$
$\text{comm}(\tau_i) = \textit{LET}$	$\xi_{\tau_{i+1}}(r^\omega_{J_i} + D_{\tau_i})$		$\xi_{\tau_{i+1}}(r^\omega_{J_i} + D_{\tau_i})$

Table 6.5.: Upper bound on $r^\omega_{J_{i+1}}$ for different cases of communication for τ_i and τ_{i+1}.

Proof. Let $\vec{ac}^\omega_E(z) = (z, J_1, \dots, J_n, z')$ be any immediate forward augmented job chain with $z > \text{re}^\omega(\tau_1(W^\omega_{E,\tau_1}))$. For the analysis, we decompose $\vec{ac}^\omega_E(z)$ into head segment, body segments and tail segment (cf. Section 6.2.1.1).

Table 6.5 gives an overview of the upper bounds on $r^\omega_{J_{i+1}}$, $i = 1, 2, \dots, n-1$ provided in Lemmas 6.15 and 6.17. Please note that only for the bound in Equation (6.36) of Lemma 6.17 it is required that τ_i and τ_{i+1} both communicate in the same manner. Using the bound on $\xi_{\tau_{i+1}}$ from Lemma 6.11, we obtain bounds for the body segments.

Summing up the different bounds for the head segment $r^\omega_{J_1} - z$ from Lemma 6.10, for the tail segment $z' - r^\omega_{J_n}$ from Lemma 6.16 and Lemma 6.18, and for the body segments $r^\omega_{J_{i+1}} - r^\omega_{J_i}$, $i = 1, 2, \dots, n-1$, from the above discussion, proves Equation (6.84). □

Algorithm 8 shows how periodic release patterns can be exploited when considering heterogeneous communication.

Theorem 6.42 (Periodic, heterogeneous communication). *If all tasks of E are periodic, then an upper bound on the end-to-end latency is obtained by Algorithm 8.*

Algorithm 8 Compute latency bound for periodic tasks under heterogeneous communication means.

```
1: $\textit{lengths} = [\,]$
2: for $m = 1, 2, 3, \dots$ do
3:     $z = \phi_{\tau_1} + (m-1) \cdot T_{\tau_1}$
4:     $r^\omega_{J_1} = \phi_{\tau_1} + m \cdot T_{\tau_1}$
5:     if $z > \Phi(E) + H(E)$ then
6:         break
7:     for $i = 1, 2, \dots, n-1$ do
8:         Compute $r^\omega_{J_{i+1}}$ with Equation (6.32) and Table 6.5.
9:     if $\textit{comm}(\tau_n)$ is $\textit{LET}$ then
10:        $z' = r^\omega_{J_n} + D_{\tau_n}$
11:    else
12:        $z' = r^\omega_{J_n} + R^{\mathcal{A}}_{\tau_n}$
13:    $\textit{lengths.append}(z' - z)$
14: return $\max(\textit{lengths})$
```

Proof. The proof is similar to the proof of Theorem 6.22. In each step $m = 1, 2, 3, \dots$ Algorithm 8 provides an upper bound on $A_m := \sup_{z \in (r^\omega_{\tau_1(m)}, r^\omega_{\tau_1(m+1)}]} \ell(\vec{ac}^\omega_E(z))$. In the following, we consider an arbitrary immediate forward augmented job chain $\vec{ac}^\omega_E(z)$ with $z \in (r^\omega_{\tau_1(m)}, r^\omega_{\tau_1(m+1)}]$ and $z > \mathrm{re}^\omega(\tau_1(W^\omega_{E,\tau_1}))$.

In line 3, z is lower bounded by $r^\omega_{\tau_1(m)} = \phi_{\tau_1} + (m-1)T_{\tau_1}$. In line 4, the release of J_1 is upper bounded using Lemma 6.12, i.e., $r^\omega_{\tau_1(m)} \leq \phi_{\tau_1} + \left\lceil \frac{z - \phi_{\tau_1}}{T_{\tau_1}} \right\rceil \cdot T_{\tau_1} = \phi_{\tau_1} + m \cdot T_{\tau_1}$. In lines 7–8, the body segments are bounded. In lines 9–12, the bounds for the tail segment from Lemmas 6.16 and 6.18 are applied. Since the bound obtained by the algorithm is only based on (i)–(iii) of Lemma 6.14, we know that the upper bound on A_m with $r^\omega_{\tau_1(m)} > \Phi(E) + H(E)$ is the same as the upper bound on $A_{m - \frac{H(E)}{T_{\tau_1}}}$. Hence, breaking the for-loop in lines 5–6 is safe. □

Remark 6.43 (Improved Cutting Analysis)**.** *To conclude, an improved analysis of the heterogeneous case is achieved by cutting the cause-effect chain E into the largest subchains with the same release constraint, i.e., either all tasks of the subchain are periodic or all tasks of the subchain are sporadic, and utilizing the upper bounds provided by the following Theorems:*

- *Sporadic: Theorem 6.41*
- *Periodic: Theorem 6.42*

6.2.5.3 Evaluation

Since this is first work to handle cause-effect chains with heterogeneous communication policies and heterogeneous release policies, we compare the performance of our three analyses:

- **Baseline (Bas)**: The baseline in Theorem 6.39.
- **Homogeneous Cutting Analysis (Hom)**: The upper bound achieved by cutting the chain into the largest homogeneous subchains, as described in Remark 6.40.
- **Improved Cutting Analysis (Imp)**: The upper bound achieved by cutting only when the release constraint changes, as described in Remark 6.43.

We evaluate the latency reduction compared to (Bas), which is

$$\frac{B(Bas) - B(Hom)}{B(Bas)} \quad \text{and} \quad \frac{B(Bas) - B(Imp)}{B(Bas)} \tag{6.85}$$

where $B(X)$ is the bound obtained by applying X, for 5 000 randomized task sets based on the Real World Automotive Benchmarks by Kramer et al. [KZH15]. The source code of the evaluation is released on Github [TU 23].

Task Set Generation We randomly generate 5 000 task sets, 1 000 sets for each utilization in $\{50\,\%, 60\,\%, 70\,\%, 80\,\%, 90\,\%\}$, based on the parameters in Table III, IV, and V in [KZH15]. For each of the 5 000 individual task sets, we first generate a set of periodic tasks with implicit communication. Afterwards, depending on the setting, a share of these tasks is randomly chosen to be sporadic instead of periodic, and a share of these tasks is randomly chosen to use LET instead of implicit communication. In particular, each periodic task τ is generated as follows: (i) The task period T_τ is drawn at random according to the share in [KZH15][9] from the set $\{1, 2, 5, 10, 20, 50, 100, 200, 1\,000\}$. (ii) The ACET for τ is generated according to a Weibull distribution. (iii) The WCET C_τ results from multiplying ACET with a random factor drawn from the interval $[f_{min}, f_{max}]$. The phase ϕ_τ of τ is drawn uniformly at random in $[0, T_\tau]$.

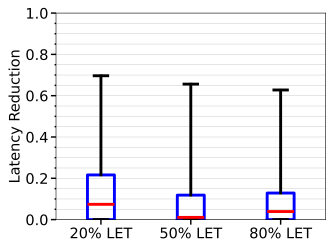
For each task set, we generate an initial set of 30 000 tasks and then select tasks from this initial set based on the subset-sum approximation algorithm until the task set utilization is within one percentage point of the targeted utilization. The generated task sets contain 50 tasks on average. The task priorities are ordered according to RM scheduling, i.e., a task with lower period has higher priority. Note that, assuming implicit deadlines, all task sets with utilization less than or equal to 90 % are by default schedulable under RM scheduling as shown by von der Brüggen et al. [Brü+17]. Therefore, the assumption $R_\tau^{\mathcal{A}} < \infty$ holds by default as well.

Each of these task sets is evaluated for all 9 combinations of (i) percentage of sporadic tasks in $\{20\,\%, 50\,\%, 80\,\%\}$, and (ii) percentage of LET tasks in $\{20\,\%, 50\,\%, 80\,\%\}$. That is, for instance, first 20 % of all tasks in the set are randomly drawn to be analyzed as sporadic (setting $T^{min} = T^{max} = T_\tau$) and then 80 % of all tasks in the set are randomly drawn to communicate using LET.

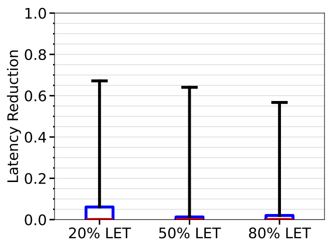
Cause-Effect Chain Generation We generate cause-effect chains based on Section IV-E in [KZH15]. In particular, each cause-effect chain comprised of 2 to 15 tasks for a task set $\mathbb{T}$ is generated as follows: (i) At first, the number of involved activation patterns P in the cause-effect chain, i.e., the number of unique periods, is drawn randomly from $\{1, 2, 3\}$ (see [KZH15, Table VI]). (ii) P different periods are drawn at random from the set $\{1, 2, 5, 10, 20, 50, 100, 200, 1\,000\}$. (iii) For each period, we draw 2 to 5 tasks at random without replacement from $\mathbb{T}$ according to the distribution given by Table VII in [KZH15]. If there are not enough tasks with the required period in $\mathbb{T}$ the cause-effect chain is discarded. For each $\mathbb{T}$, we generate 30 to 60 cause-effect chains.

Evaluation Results Figure 6.16 and Figure 6.17 show the evaluation results. The red line depicts median, blue box depicts 50 % of the data and the whiskers depict all data. We observe that both (Hom) and (Imp) improve (Bas) by up to 70 %, and that (Imp) improves (Hom) further. All these improvements come from a reduced number of cuttings, which would introduce additional pessimism into the analysis. This pessimism is the highest if the underlying tasks are periodic, i.e., the release pattern cannot be exploited

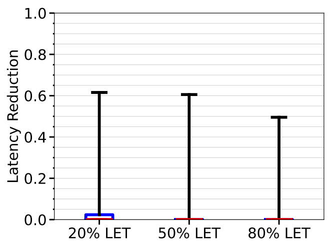
[9]Please note that the sum of the probabilities in their work is only 85 % because the remaining share is for angle-synchronous tasks. Therefore, all values from their table are divided by 0.85.



(a) 20 % sporadic, 80 % periodic.

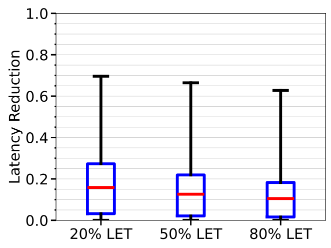


(b) 50 % sporadic, 50 % periodic.

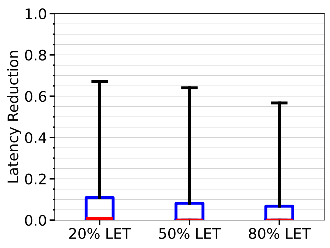


(c) 80 % sporadic, 20 % periodic.

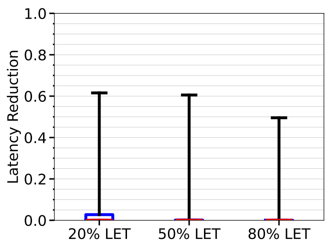
Figure 6.16.: Latency reduction of the homogeneous cutting analysis (Hom) over the baseline (Bas) for different ratio of sporadic and periodic tasks per task set, and different ratio of implicit communication and LET tasks per task set.



(a) 20 % sporadic, 80 % periodic.



(b) 50 % sporadic, 50 % periodic.



(c) 80 % sporadic, 20 % periodic.

Figure 6.17.: Latency reduction of the improved cutting analysis (Imp) over the baseline (Bas) for different ratio of sporadic and periodic tasks per task set, and different ratio of implicit communication and LET tasks per task set.

if the chain is cut. For the cases with 50 % tasks with implicit communication and 50 % tasks with LET in the task set, we observe a drop in the latency reduction of (Hom) but not for (Imp). That is because the number of cuttings of (Hom) is highest in this case, but for (Imp) there is no additional cutting between different communication means. This suggests that, if the number of necessary cutting is minimized by a smart choice of communication policy and release constraints of the tasks, the analytical end-to-end latency can be further optimized.

6.3 Analysis of Probabilistic End-to-End Latency

In the preceding section, we discuss approaches to bound the end-to-end latencies, considering the *worst-case* system behavior. However, in many cases worst-case guarantees may be pessimistic or even unachievable. For example, if data propagation is performed via communication over a network, data propagation is not certain and the worst-case end-to-end latency has to assume the (potentially unlikely) case that data propagation fails. For such cases, *probabilistic* guarantees on the reaction time are more practical.

In the literature, several probabilistic analyses have been derived for end-to-end latencies in the context of queuing networks, i.e., packages that travel through several nodes. Andrews [And00] analyzes the probability that a packet violates its delay bound when each server schedules packets with EDF. In the same context, Fidler [Fid06] establishes probabilistic network calculus with moment generating functions to derive performance bounds. Similarly, Scharbarg et al. [SRF09] focus on delay over links and switches using stochastic network calculus. However, the probability of failures is not considered in the communication and the analyzed probability is about the bursts of packet arrivals.

With a closer context to this chapter, Lee et al. [Lee+22] analyze the end-to-end latency of cause-effect chains for periodic task systems under fully partitioned EDF scheduling, assuming that the periods of cause-effect chain are sorted in non-decreasing order. However, a survey on industrial practice in real-time systems that was provided by Akesson et al. [Ake+20] in 2020 shows that sporadic activation with minimum inter-arrival time for tasks is a common industry practice. Specifically, sporadic tasks were part of 47 % of the investigated systems. Hence, it is highly relevant to consider the end-to-end latency of cause-effect chains for the sporadic task systems (cf. [Dür+19; Gün+21a; Gün+23a]) and to further take into account the randomness, i.e., response-time randomness and failure probabilities, for embedded systems subject to uncertainty (cf. [Hob+22; Lei+11]).

In this section, we discuss probabilistic end-to-end latencies and provide analytical guarantees that the reaction time does not exceed a certain threshold with (at least) a certain probability. To that end, we consider two types of randomness: Response-time randomness and failure probabilities. To the best of our knowledge, this is the first work that defines and analyzes probabilistic reaction time for cause-effect chains based on sporadic tasks. We do not restrict to any specific scheduling algorithm $\mathcal{A}$, except that jobs of the same task are executed in First In – First Out (FIFO) ordering. For the sake of readability, we assume that a task has a fixed arrival pattern $(r_J)_{J\in\mathbb{J}}$ and that this pattern can further be abstracted using minimum and maximum inter-arrival time, i.e.,

$$\tau \in \mathrm{FixRel}((r_{\tau(j)})_{j\in\mathbb{N}}) \cap \mathrm{SporMinMax}(T_\tau^{\min}, T_\tau^{\max}). \tag{6.86}$$

If several arrival patterns are possible, abstracted with the same minimum and maximum inter-arrival time, the minimal guarantee among all possible arrival patterns has to be considered. This section is organized as follows:

- In Section 6.3.1, we define probabilistic end-to-end latency metrics and discuss potential pitfalls for the definition.
- In Section 6.3.2, we introduce the probabilistic model that this section is focused on. Specifically, we present the model of response-time randomness and failure probabilities, and discuss assumptions on probabilistic independence.

- Our analytical results are presented in Section 6.3.3 for communication under LET, and in Section 6.3.4 for implicit communication.
- Efficient computation of our analytical bounds is discussed in Section 6.3.5.
- We evaluate our analysis in Section 6.3.6.

The results of this section appeared in EMSOFT 2023 [Gün+23c].

6.3.1 Definition and Potential Pitfalls

In this section, we define probabilistic end-to-end latencies and discuss potential pitfalls for the definition. To that end, we assume that the set of system evolutions Ω forms a probability space

$$(\Omega, \mathfrak{F}, \mathbb{P}). \tag{6.87}$$

More specifically, the *event space* is given by a σ-algebra $\mathfrak{F} \subseteq \mathcal{P}(\Omega)$, and the *probability measure* $\mathbb{P}\colon \mathfrak{F} \to [0,1]$ is a real-valued function characterizing the probability of events. If Ω is countable or finite, usually $\mathfrak{F} = \mathcal{P}(\Omega)$ is considered. However, if Ω is uncountable, for reasons going beyond the scope of this dissertation, $\mathfrak{F} = \mathcal{P}(\Omega)$ cannot always be chosen (cf. [Bor99, Chapter 2.1]). We assume that $\mathfrak{F}$ is chosen such that the probability $\mathbb{P}(A \leq x) := \mathbb{P}(\{\omega \in \Omega \mid A(\omega) \leq x\})$ is well-defined for all random variables $A\colon \Omega \to \mathbb{R} \cup \{\infty\}$ defined in this dissertation.

In the following, we extend the definition of end-to-end latency to the probabilistic case. Although the equivalence of MDA and MRT holds for the deterministic case, as proven in Section 6.1.2, we introduce probabilistic reaction time and probabilistic data age individually, and discuss the extension of the equivalence to the probabilistic definitions of data age and reaction time afterwards. We start be introducing probabilistic reaction time, pointing to potential pitfalls during its definition.

According to Definition 3.7, the MRT is defined as $\mathrm{MRT}(E) = \sup_{\omega\in\Omega} \mathrm{MRT}(E,\omega)$ with $\mathrm{MRT}(E,\omega) = \sup_{z > \mathrm{re}^\omega(\tau_1(W^\omega_{E,\tau_1}))} \ell(\vec{ac}^\omega_E(z))$. Intuitively, we would like to consider $\mathrm{MRT}(E,\bullet)\colon \Omega \to \mathbb{R}$ as a random variable. However, this random variable usually does not reveal any probabilistic information, as discussed below.

Pitfall 6.44 (MRT$(E,\bullet)$ as random variable)**.** *The random variable* $\mathrm{MRT}(E,\bullet)\colon \Omega \to \mathbb{R}$ *does not reveal any probabilistic information about the system in many cases, in the sense that* $\mathrm{MRT}(E,\bullet) = \mathrm{MRT}(E)$ *almost surely. To demonstrate this, we consider a task set* $\mathbb{T}$ *of periodic, synchronous, constrained-deadline tasks for which a hyperperiod* $H(\mathbb{T})$ *exists. We assume that the execution demand* c^ω_J *of all jobs are discrete, independent and identically distributed (iid) random variables.*

Assume there exist $\tilde\omega \in \Omega$ *and* $m \geq W^{\tilde\omega}_{E,\tau_1}$ *such that* $\mathrm{MRT}(E) = \ell(pc^{\tilde\omega}_{E,1})(m)$.[10] *By choosing* $k \in \mathbb{N}$ *large enough,* $(\ell(pc^\bullet_{E,1}(m + n \cdot k \cdot \frac{H(\mathbb{T})}{T_{\tau_1}})))_{n\in\mathbb{N}_0}$ *are iid random variables.*

[10] This is a common scenario when considering the description of MRT using partitioned job chains from Section 6.1.2.1.

As a direct consequence of the Borel-Cantelli Lemma (cf. [Kle20, Corollary 2.39]), we obtain that $\mathbb{P}\left(\limsup_{n\to\infty} \ell(pc^{\bullet}_{E,1}(m + n \cdot k \cdot H(\mathbb{T}))) = \mathrm{MRT}(E)\right) = 1$*. Hence, we also know that* $\mathbb{P}\left(\sup_{m \geq W^{\omega}_{E,\tau_1}} \ell(pc^{\bullet}_{E,1}(m)) = \mathrm{MRT}(E)\right) = 1$*, and consequently*

$$\mathbb{P}\left(\mathrm{MRT}(E,\bullet) = \mathrm{MRT}(E)\right) = 1 \tag{6.88}$$

holds. That is, $\mathrm{MRT}(E,\bullet)$ *is a random variable which results in the deterministic* $\mathrm{MRT}(E) \in \mathbb{R}$ *almost surely.*

Instead of examining the random variable $\mathrm{MRT}(E,\bullet)$, we consider the *reaction time* at a fixed time point z, i.e.,

$$\mathrm{RT}(E,\omega,z) := \ell(\vec{ac}^{\omega}_{E}(z)), \tag{6.89}$$

Intuitively, the reaction time at z is the time until an external activity at time z is fully processed. The reaction time can be considered as random variable $\mathrm{RT}(E,\bullet,z)$. The question we answer in this section is:

> Given an arbitrary z, then what is the probability that the reaction time $\mathrm{RT}(E,\bullet,z)$ does not exceed a predefined value $x \in \mathbb{R}$?

In particular, we define $\mathrm{PRTG}(E,x) = \inf_z \mathbb{P}(\mathrm{RT}(E,\bullet,z) \leq x)$ as the *probabilistic reaction time guarantee*. Given this value, it is guaranteed that $\mathbb{P}(\mathrm{RT}(E,\bullet,z) \leq x)$ is at least $\mathrm{PRTG}(E,x) \in [0,1]$ for all $z \in \mathbb{R}$. The second pitfall is the restriction of z in the infimum of the definition of $\mathrm{PRTG}(E,x)$.

Pitfall 6.45 (Restricting z using warm-up)**.** *When considering the probabilistic reaction time guarantee* $\mathrm{PRTG}(E,x) = \inf_z \mathbb{P}(\mathrm{RT}(E,\bullet,z) \leq x)$*, z must be restricted from below. Otherwise,* $\mathrm{RT}(E,\bullet,z) \to \infty$ *for* $z \to -\infty$*, and* $\mathrm{PRTG}(E,x) = 0$*. The intuitive approach is to only consider z after the warm-up, i.e.,* $z > \mathrm{re}^{\omega}(\tau_1(W^{\omega}_{E,\tau_1}))$*, following Definition 3.7. However, the problem is that* $\mathrm{re}^{\omega}(\tau_1(W^{\omega}_{E,\tau_1}))$ *depends on the system evolution. Hence, we cannot define* $\mathrm{PRTG}(E,x) = \inf_{z > \mathrm{re}^{\omega}(\tau_1(W^{\omega}_{E,\tau_1}))} \mathbb{P}(\mathrm{RT}(E,\bullet,z) \leq x)$ *because the bound on z depends on the probability space Ω. We can avoid this pitfall by bounding z using a deterministic value instead. In this work, we bound z by the maximum first release, i.e.,* $z \geq \Phi(E) := \max\left\{r_{\tau(1)} \,\middle|\, \tau \in E\right\}$*.*

This leads us to the following definition of the probabilistic reaction time guarantee.

Definition 6.46 (Probabilistic reaction time guarantee)**.** *The probabilistic reaction time guarantee of cause-effect chain E is defined by*

$$\mathrm{PRTG}(E,\bullet)\colon \qquad \mathbb{R} \longrightarrow [0,1] \tag{6.90}$$

$$x \longmapsto \inf_{z \geq \Phi(E)} \mathbb{P}(\mathrm{RT}(E,\bullet,z) \leq x) \tag{6.91}$$

with $\Phi(E) := \max\left\{r_{\tau(1)} \,\middle|\, \tau \in E\right\}$*.*

In the following, we introduce the probabilistic data age guarantee and discuss the equivalence between probabilistic data age and probabilistic reaction time. Similar to the reaction time, for a fixed z', we consider the data age random variable $\mathrm{DA}(E, \bullet, z')$, defined by:

$$\mathrm{DA}(E, \bullet, z') := \begin{cases} \ell(\overleftarrow{ac}^{\bullet}_E(z')) & \text{, if } \overleftarrow{ac}^{\bullet}_E(z') \text{ exists} \\ \infty & \text{, else} \end{cases} \tag{6.92}$$

Intuitively, $\mathrm{DA}(E, \bullet, z')$ is the age of the data that is used in an actuation at time z'. We note that we set the data age to ∞ if no $\overleftarrow{ac}^{\bullet}_E(z')$ exist, since in that case no data has been produced that could be utilized for an actuation at time z'.

If the reaction time at time point z is upper bounded by $x \in \mathbb{R}$, i.e., $\mathrm{RT}(E, \omega, z) \leq x$, then data that is used in an actuation at time $z + x$ cannot be older than $z + x - z = x$ time units, i.e., $\mathrm{DA}(E, \omega, z + x) \leq x$. Furthermore, if $\mathrm{DA}(E, \omega, z + x) \leq x$, then an external activity at time z cannot take longer than x time units to be fully processed, i.e., $\mathrm{RT}(E, \omega, z) \leq x$. Therefore, the probability of $\mathrm{RT}(E, \bullet, z)$ being at most x and the probability of $\mathrm{DA}(E, \bullet, z + x)$ being at most x are equal. We obtain

$$\mathbb{P}(\mathrm{RT}(E, \bullet, z) \leq x) = \mathbb{P}(\mathrm{DA}(E, \bullet, z + x) \leq x) \tag{6.93}$$

for any $x \in \mathbb{R}$. Hence, $\mathrm{RT}(E, \bullet, z)$ and $\mathrm{DA}(E, \bullet, z + x)$ are identically distributed (but not independent). We define the probabilistic data age guarantee as follows.

Definition 6.47 (Probabilistic data age guarantee)**.** *The probabilistic data age guarantee of cause-effect chain E is defined by:*

$$\mathrm{PDAG}(E, \bullet)\colon \qquad \mathbb{R} \longrightarrow [0, 1] \tag{6.94}$$

$$x \longmapsto \inf_{z \geq \Phi(E) + x} \mathbb{P}(\mathrm{DA}(E, \bullet, z) \leq x) \tag{6.95}$$

This definition of the probabilistic data age guarantee preserves the equivalence of data age and reaction time for probabilistic guarantees, as shown in the following theorem.

Theorem 6.48 (Equivalence of Probabilistic Guarantees)**.** *Let E be a cause-effect chain and ω a probability space of system evolutions. The probabilistic guarantees probabilistic reaction time guarantee and the probabilistic data age guarantee coincide, i.e.,*

$$\mathrm{PRTG}(E, x) = \mathrm{PDAG}(E, x) \tag{6.96}$$

for all $x \in \mathbb{R}$.

Proof. As a result of Lemma 6.2, the immediate forward augmented job chain $\overrightarrow{ac}^{\omega}_E(z)$ has a length of $\ell(\overrightarrow{ac}^{\omega}_E(z)) \leq x$ if and only if $\overleftarrow{ac}^{\omega}_E(z + x)$ exists and has a length of $\ell(\overleftarrow{ac}^{\omega}_E(z + x)) \leq x$. Therefore, $\mathrm{RT}(E, \omega, z) \leq x$ if and only if $\mathrm{DA}(E, \omega, z) \leq x$. We derive:

$$\begin{aligned} \mathrm{PDAG}(E, x) &= \inf_{z \geq \Phi(E) + x} \mathbb{P}(\mathrm{DA}(E, \bullet, z) \leq x) && (6.97) \\ &= \inf_{z \geq \Phi(E)} \mathbb{P}(\mathrm{DA}(E, \bullet, z + x) \leq x) && (6.98) \\ &= \inf_{z \geq \Phi(E)} \mathbb{P}(\mathrm{RT}(E, \bullet, z) \leq x) && (6.99) \end{aligned}$$

Hence, $\mathrm{PDAG}(E, x) = \mathrm{PRTG}(E, x)$. □

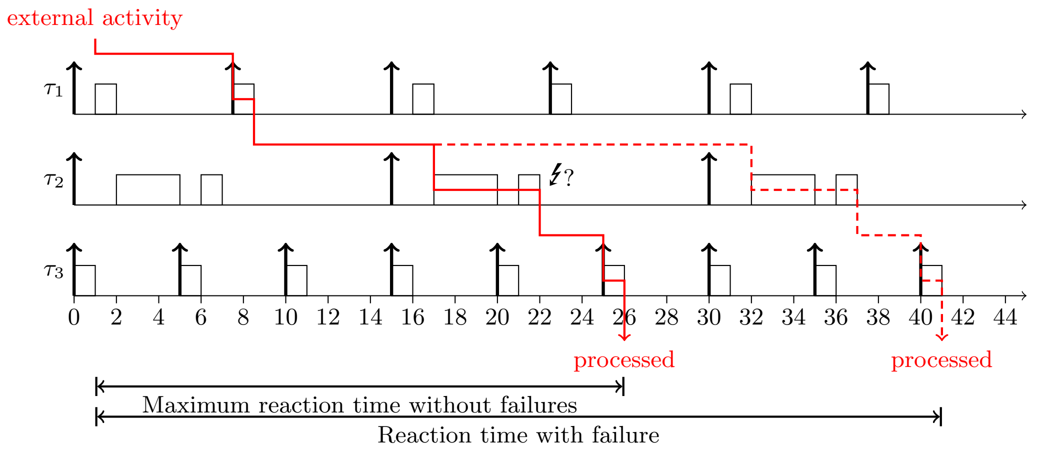


Figure 6.18.: Data propagation path and MRT for cause-effect chain $E = (\tau_1 \to \tau_2 \to \tau_3)$. If the second job of τ_2 fails, higher reaction time can be observed.

For the remainder of Section 6.3, we focus on the probabilistic reaction time and the probabilistic reaction time guarantee. However, the results can be applied to probabilistic data age using the equivalence discussed in this section.

6.3.2 Probabilistic Model

In this section, we introduce the probabilistic model that we focus on for the remainder of Section 6.3. Specifically, this examination considers two types of randomness: randomness in the response time, and failure probabilities. We introduce both types and describe how they are integrated in our model. Afterwards, we discuss assumptions on the probabilistic independence.

6.3.2.1 Response-Time Randomness

Considering the WCRT of every job may be very pessimistic. Especially, a very large but unlikely response time might lead to very high MRT, although the observed reaction time is much lower. For example, when considering Figure 6.18, if the second job of task τ_2 would finish with high probability before 20, then the data could be processed by the fifth job of τ_3 and the reaction time would be $21 - 1 = 20$ time units with high probability.

Naturally, the response time $R^{\mathcal{A}(\omega),\omega}_{\tau(j)}$ of a job $\tau(j)$ is a random variable:

$$R^{\mathcal{A}(\bullet),\bullet}_{\tau(j)}\colon \quad \Omega \to \mathbb{R} \tag{6.100}$$

The problem with utilizing the response time directly in the analysis is twofold:

- Usually, the response times of different jobs of the same task are not identically distributed. Due to the release pattern, the system may be more or less utilized, resulting in generally longer or shorter response times of jobs. Taking care of the actual response time of jobs is very involving.

- The response times of different jobs may not be independent. This complicates the analysis even further because dependencies have to be properly handled.

To avoid these complications for the analysis, we use over-approximation. The concept of over-approximating dependent random variables, is usually used in the context of execution-time [Boz+23; Cuc13; DC19]. Specifically, over-approximation "enables probabilistic independence to be assumed" [DC19] for execution times which are naturally dependent on common input parameters, cache states, and pipeline states. We omit further discussion and point the reader to the survey by Davis and Cucu-Grosjean [DC19] or to the work by Bozhko et al. [Boz+23] for details.

We assume that the response times $R_{\tau(j)}^{\mathcal{A}(\bullet),\bullet}$ are upper bounded by

$$\overline{R}_{\tau(j)}^{\mathcal{A}(\bullet),\bullet}\colon \quad \Omega \longrightarrow \mathbb{R} \tag{6.101}$$

for all $j \in \mathbb{N}$, and that $(\overline{R}_{\tau(j)}^{\mathcal{A}(\bullet),\bullet})_{j\in\mathbb{N}}$ are iid random variables. We note, that such over-approximation may reduce the gain provided by a probabilistic analysis compared to an analysis based on actual response times. Nevertheless, the probabilistic analysis is never worse than the analysis based on WCRT, since the WCRT is a trivial iid upper bound that can be considered as $\overline{R}_{\tau(j)}^{\mathcal{A}(\bullet),\bullet}$. The following example demonstrates how these response-time upper bounds can be achieved when scheduling jobs via a Time Division Multiple Access (TDMA) mechanism.

Example 6.49 (Iid response-time bounds with TDMA)**.** *We assume that the execution demand of task τ is abstracted as Probabilistic Worst-Case Execution Time (pWCET), specifically, $\tau \in \mathrm{pWCET}^{\mathrm{iid}}(D_\tau)$ with upper bounds $\overline{c}_{\tau(j)}^{\omega} \geq c_{\tau(j)}^{\omega}$ for all $j \in \mathbb{N}$ (cf. Section 2.4.1). During each TDMA cycle period of length T^c, each task is assigned a slot of length Q^τ where jobs of τ can be executed. The slot occurs at a fixed position during each cycle period and is reserved only for the particular task. If a job of task τ executes for $C \in \mathbb{R}$ time units, then it takes at most $\left\lceil \frac{C}{Q^\tau} \right\rceil$ cycle periods until the job is finished, and the response time of that job is upper bounded by $\left\lceil \frac{C}{Q^\tau} \right\rceil \cdot (T^c - Q^\tau) + C$. Therefore, since the execution time of job $\tau(j)$ is* upper bounded *by $\overline{c}_{\tau(j)}^{\omega}$, then*

$$\overline{R}_{\tau(j)}^{\mathcal{A}(\omega),\omega} := \left\lceil \frac{\overline{c}_{\tau(j)}^{\omega}}{Q^\tau} \right\rceil \cdot (T^c - Q^\tau) + \overline{c}_{\tau(j)}^{\omega} \tag{6.102}$$

is an upper bound *on the response time of job $\tau(j)$. Since the random variables $(\overline{c}_{\tau(j)}^{\bullet})_{j\in\mathbb{N}}$ are iid, $(\overline{R}_{\tau(j)}^{\mathcal{A}(\bullet),\bullet})_{j\in\mathbb{N}}$ are iid as well.*

While usually end-to-end analyses are based on the execution time of jobs, we exploit the probabilistic behavior of response times. The reason is that the analysis procedure actually uses response times instead of execution times, and the scheduling algorithm $\mathcal{A}$ that schedules the execution times to obtain response times is irrelevant for the analysis. In the previous example, we have demonstrated how to convert iid execution times to iid response-time upper bounds for TDMA scheduling. Computation of tight iid response-time upper bounds for more complex scheduling algorithms is an open problem to be considered in future work.

6.3.2.2 Failure Probabilities

Especially for network communication, a job may fail to read, process, or transmit the data. In such cases, the reaction time includes the waiting time until a job reads, processes, and transmits the data successfully. In the following, we account for such failures by extending the system evolution. Further, we describe the probabilistic model for failed jobs that we focus on in the subsequent sections. Please note that we consider only the failure of the *data propagation*, and that such failures do not lead to a failure of the whole system.

Each system evolution ω results in a set of failed jobs $\mathcal{F}^\omega \in \mathcal{P}(\mathbb{J})$. To account for the set of failed jobs, we extend the set Ω of system evolutions. More specifically, we define

$$\Omega = \{\omega = ((r_J^\omega)_{J\in\mathbb{J}}, (c_J^\omega)_{J\in\mathbb{J}}, \mathcal{F}^\omega)\} \tag{6.103}$$

to be the set of system evolutions. Furthermore, we extend the definition of immediate forward augmented job chains by tracking the corresponding non-failed job.

Definition 6.50 (Immediate forward augmented job chain with job failures.)**.** *Consider $z \in \mathbb{R}$ and a system evolution $\omega = ((r_J^\omega)_{J\in\mathbb{J}}, (c_J^\omega)_{J\in\mathbb{J}}, \mathcal{F}^\omega)$. The immediate forward augmented job chain for $E = (\tau_1 \to \cdots \to \tau_n)$ in ω at z is defined by*

$$\vec{ac}_E^\omega(z) = (z, (J_1, J_1'), \ldots, (J_n, J_n'), z') \tag{6.104}$$

where the following conditions hold:

- *J_1 is the job of τ_1 with the earliest read-event $\mathrm{re}^\omega(J_1) \geq z$, and J_1' is the earliest job of τ_1 without failure such that $J_1' \succcurlyeq J_1$, using the notation from Definition 6.1.*
- *J_i is the job of τ_i with the earliest read-event $\mathrm{re}^\omega(J_i) \geq \mathrm{we}^\omega(J_{i-1}')$, and J_i' is the earliest job of τ_i without failure such that $J_i' \succcurlyeq J_i$, for all $i = 2, \ldots, n$.*
- *$z' = \mathrm{we}^\omega(J_n')$*

This definition is similar to the definition of immediate forward augmented job chains in Definition 3.4, except that we track the jobs J_i' that successfully read, process, and transmit the data as well. We note that contrarily to the case without failures, immediate forward augmented job chains with job failures may not exist if all jobs fail after a certain time point. Hence, for preciseness, we need to slightly adjust the definition of reaction time:

$$\mathrm{RT}(E, \omega, z) = \begin{cases} \ell(\vec{ac}_E^\omega(z)) & \text{, if } \vec{ac}_E^\omega(z) \text{ exists} \\ \infty & \text{, else} \end{cases} \tag{6.105}$$

Figure 6.18 demonstrates that the reaction time is increased if the second job of τ_2 fails to propagate data.

We note that immediate backward augmented job chains can be adjusted to account for job failures similarly, i.e., by including the preceding job J_i' that has no job failure:

$$\overleftarrow{ac}^{\omega}_{E}(z') = (z, (J_1', J_1), \dots, (J_n', J_n), z') \tag{6.106}$$

The data age $\mathrm{DA}(E, \omega, z)$ from Equation (6.92) already accounts for the case that $\overleftarrow{ac}^{\omega}_{E}(z')$ does not exist, and therefore does not need to be adjusted. Moreover, we note that the equivalence of probabilistic guarantees, shown in Equation (6.96), extends to the case with job failures.

Theorem 6.51 (Equivalence of Probabilistic Guarantees with job failures)**.** *Let E be a cause-effect chain and Ω a probability space of system evolutions with job failures. The probabilistic guarantees probabilistic reaction time guarantee and the probabilistic data age guarantee coincide, i.e.,*

$$\mathrm{PRTG}(E, x) = \mathrm{PDAG}(E, x) \tag{6.107}$$

for all $x \in \mathbb{R}$.

Proof. First, we observe that the immediate forward augmented job chain $\overrightarrow{ac}^{\omega}_{E}(z) = (z, (J_1, J_1'), \dots, (J_n, J_n'), z')$ with job failures exists, if and only if $(z, J_1', \dots, J_n', z')$ is an immediate forward augmented job chain on the job set $\mathbb{J} \setminus \mathcal{F}^{\omega}$ without failures. Similarly, the immediate backward augmented job chain $\overleftarrow{ac}^{\omega}_{E}(z) = (z, (J_1', J_1), \dots, (J_n', J_n), z')$ with job failures exists, if and only if $(z, J_1', \dots, J_n', z')$ is an immediate backward augmented job chain on the job set $\mathbb{J} \setminus \mathcal{F}^{\omega}$ without failures. Therefore, the result of Lemma 6.2, formulated in the proof of Theorem 6.48 still holds. Particularly, $\ell(\overrightarrow{ac}^{\omega}_{E}(z)) \leq x$ if and only if $\ell(\overleftarrow{ac}^{\omega}_{E}(z + x)) \leq x$, and $\mathrm{RT}(E, \omega, z) \leq x$ if and only if $\mathrm{DA}(E, \omega, z) \leq x$. Similar to the proof of Theorem 6.48, we derive:

$$\begin{aligned} \mathrm{PDAG}(E, x) &= \inf_{z \geq \Phi(E) + x} \mathbb{P}(\mathrm{DA}(E, \bullet, z) \leq x) && (6.108)\\ &= \inf_{z \geq \Phi(E)} \mathbb{P}(\mathrm{DA}(E, \bullet, z + x) \leq x) && (6.109)\\ &= \inf_{z \geq \Phi(E)} \mathbb{P}(\mathrm{RT}(E, \bullet, z) \leq x) && (6.110) \end{aligned}$$

Hence, $\mathrm{PDAG}(E, x) = \mathrm{PRTG}(E, x)$. $\square$

In the context of job failures, probabilistic reaction time guarantees become particularly interesting. That is, if each job can experience a failure with a certain probability > 0, then $\mathrm{MRT}(E) = \sup_{\omega \in \Omega} \mathrm{MRT}(E, \omega)$ is unbounded since all jobs after a certain time point may experience a failure. In that case, probabilistic guarantees are more practical since they evaluate whether a high reaction time is common or only occurs in rare cases.

For the probabilistic behavior of job failures, we assume that each task τ_i of the cause-effect chain $E = (\tau_1 \to \dots \tau_n)$ is equipped with a *failure probability* $f_{\tau_i} \in [0, 1)$. This means that each job $\tau_i(j)$ of τ_i fails to propagate data with a probability of f_{τ_i}, i.e.,

$$\mathbb{P}(\tau_i(j) \in \mathcal{F}^\bullet) = f_{\tau_i}. \tag{6.111}$$

We assume that job failures occur independently, i.e., $(\{\omega \in \Omega \,|\, \tau_i(j) \in \mathcal{F}^\omega\})_{j\in\mathbb{N}}$ are iid events. Hence, the number of jobs of τ_i that it takes until the data propagation is successful starting from $\tau_i(j)$, denoted as random variable $S^\bullet_{\tau_i,j} \colon \Omega \to \mathbb{N}$, is geometrically distributed with

$$\mathbb{P}(S^\bullet_{\tau_i,j} = k) = (f_{\tau_i})^{k-1} \cdot (1 - f_{\tau_i}) \tag{6.112}$$

for all $k \in \mathbb{N}$. Specifically, for $k = 1$, $\mathbb{P}(S^\bullet_{\tau_i,j} = 1) = (1 - f_{\tau_i})$ is the probability that $\tau_i(j)$ is successful, and for $k = 2$, $\mathbb{P}(S^\bullet_{\tau_i,j} = 2) = f_{\tau_i} \cdot (1 - f_{\tau_i})$ is the probability that $\tau_i(j)$ fails data propagation and $\tau_i(j + 1)$ is successful.

6.3.2.3 Independence

We assume that all random variables are mutually independent. Independence is a fundamental property for our analytical results, similar to the analysis of probabilistic response-time analysis where most results consider independence (details can be found in the survey by Davis and Cucu-Grosjean [DC19]). We deem this assumption necessary for introducing the problem and providing an initial result future work can be built on. Specifically, in this work, we make the following two independence assumptions:

(i) The failure probability is the same for each job $\tau_i(j)$ of a task τ_i and is independent of the behavior of other jobs (of the same task τ_i and of tasks $\tau_{i'}$ with $i' \neq i$).

(ii) The distribution of the upper bound $\overline{R}^{\mathcal{A}(\bullet),\bullet}_{\tau_i(j)}$ on the response time is the same for each job $\tau_i(j)$ of a task τ_i and is independent of the behavior of other jobs (of the same task τ_i and of tasks $\tau_{i'}$ with $i' \neq i$).

This includes the independence of data-propagation failures of jobs and response-time distribution, i.e., larger or shorter response-time upper bound should not favor a data-propagation failure and vice versa. Moreover, this implies that $\tau_i \neq \tau_{i'}$ for all $i \neq i'$, i.e., there are no duplicate tasks in the cause-effect chain, since otherwise the jobs $\tau_i(j)$ and $\tau_{i'}(j)$ would be dependent for all $j \in \mathbb{N}$. Please note that our approach is not limited to cases with independent execution demand of jobs as long as independent response-time upper bounds can be derived.

We plan on removing independence assumptions through correlation-tolerant analysis in future work. Significant advancements in this regard have been made recently by [Mar+23] building upon Cantelli's inequality to achieve probabilistic response-time bounds using possibly dependent execution demands. Similar strategies could potentially be pursued for probabilistic end-to-end bounds using possibly dependent response-time bounds and failure probabilities as input.

6.3.3 Probabilistic Reaction Time Under Logical Execution Time

Under Logical Execution Time (LET), there is no impact of the response-time randomness because the read- and write-events are independent of the execution behavior. Hence, for that scenario the only source of randomness for the analysis is the failure probability. In this section, we first provide an upper bound on $\mathrm{RT}(E, \omega, z)$ under LET. Afterwards, we utilize that upper bound to analyze the probabilistic reaction time and the probabilistic reaction time guarantee.

For a fixed system evolution $\omega = ((r_J^\omega)_{J\in\mathbb{J}}, (c_J^\omega)_{J\in\mathbb{J}}, \mathcal{F}^\omega)$, we recursively construct upper bounds (to be specified in Equation (6.113)) for the immediate forward augmented job chain $\vec{ac}_E^\omega(z) = (z, (\tau_1(\xi_1), \tau_1(\zeta_1)), \dots, (\tau_n(\xi_n), \tau_n(\zeta_n)), z')$ as follows:

- $\bar{w}_0 := z$
- For $i = 1, \dots, n$:
  - $\bar{\xi}_i \in \mathbb{N}$ such that $\tau_i(\bar{\xi}_i)$ is the first job released at or after $\bar{w}_{i-1}$
  - $\bar{\zeta}_i = \bar{\xi}_i + S^\omega_{\tau_i, \bar{\xi}_i} - 1$
  - $\bar{r}_i := S^\omega_{\tau_i, \bar{\xi}_i} \cdot T^{\max}_{\tau_i} + \bar{w}_{i-1}$
  - $\bar{w}_i := \bar{r}_i + D_{\tau_i}$

where $S^\omega_{\tau_i, \bar{\xi}_i}$ is the number of jobs of τ_i until the first successful job beginning at $\tau_i(\bar{\xi}_i)$ according to $\mathcal{F}^\omega$. We can bound the length of $\vec{ac}_E^\omega(z)$ as follows:

Lemma 6.52. *Let $\vec{ac}_E^\omega(z) = (z, (\tau_1(\xi_1), \tau_1(\zeta_1)), \dots, (\tau_n(\xi_n), \tau_n(\zeta_n)), z')$ be an immediate forward augmented job chain with $z \geq \Phi(E)$. Then, for all $i = 1, \dots, n$:*

$$\xi_i \leq \bar{\xi}_i, \quad \zeta_i \leq \bar{\zeta}_i, \quad r^\omega_{\tau_i(\zeta_i)} \leq \bar{r}_i, \quad \text{and} \quad \mathrm{we}^\omega(\tau_i(\zeta_i)) \leq \bar{w}_i. \tag{6.113}$$

In particular,

$$\ell(\vec{ac}_E^\omega(z)) \leq \bar{w}_n - z = \sum_{i=1}^n S^\omega_{\tau_i, \bar{\xi}_i} \cdot T^{\max}_{\tau_i} + D_{\tau_i}. \tag{6.114}$$

Proof. We start by proving Equation (6.113) by induction over i.

Induction start ($i = 1$): The job $\tau_1(\xi_1)$ is the *first* job with read-event at or after z. The job $\tau_1(\bar{\xi}_1)$ is released at or after $\bar{w}_0 = z$. Therefore, its read-event is at or after z, and $\bar{\xi}_1 \geq \xi_1$ holds. $\tau_1(\zeta_1)$ is the *first* successful job at or after job $\tau_1(\xi_1)$. The job $\tau_1(\bar{\zeta}_1) = \tau_1(\bar{\xi}_1 + S^\omega_{\tau_1, \bar{\xi}_1} - 1)$ is a successful job at or after $\tau_1(\bar{\xi}_1)$, and therefore also at or after $\tau_1(\xi_1)$. This means that $\bar{\zeta}_1 \geq \zeta_1$. Since only $z \geq \Phi(E)$ is considered, the job $\tau_1(\bar{\zeta}_1)$ is released at most $S^\omega_{\tau_1, \bar{\xi}_1} \cdot T^{\max}_{\tau_1}$ time units after $z = \bar{w}_0$. Therefore, $r^\omega_{\tau_1(\zeta_1)} \leq S^\omega_{\tau_1, \bar{\xi}_1} \cdot T^{\max}_{\tau_1} + \bar{w}_0 = \bar{r}_1$. We have proven that the job $\tau_1(\zeta_1)$ is released no later than at $\bar{r}_1$. Therefore, its write-event is no later than at $\bar{r}_1 + D_{\tau_1}$. Hence, $\mathrm{we}^\omega(\tau_1(\zeta_1)) \leq \bar{r}_1 + D_{\tau_1} = \bar{w}_1$. This concludes the induction start.

Induction step ($i-1 \mapsto i$): The job $\tau_i(\xi_i)$ is the *first* job with read-event at or after $\mathrm{we}^\omega(\tau_{i-1}(\zeta_{i-1}))$. $\tau_i(\bar{\xi}_i)$ is released at or after $\bar{w}_{i-1}$, and therefore its read-event is at or after $\bar{w}_{i-1}$ as well. Since $\bar{w}_{i-1} \geq \mathrm{we}^\omega(\tau_{i-1}(\zeta_{i-1}))$ by induction, we obtain $\bar{\xi}_i \geq \xi_i$. $\tau_i(\zeta_i)$ is the *first* successful job at or after job $\tau_i(\xi_i)$. The job $\tau_i(\bar{\zeta}_i) = \tau_i(\bar{\xi}_i + S^\omega_{\tau_i,\bar{\xi}_i} - 1)$ is a successful job at or after $\tau_i(\bar{\xi}_i)$, and therefore also at or after $\tau_i(\xi_i)$. This means that $\bar{\zeta}_i \geq \zeta_i$. The job $\tau_i(\bar{\zeta}_i)$ is released at most $S^\omega_{\tau_i,\bar{\xi}_i} \cdot T^{\max}_{\tau_i}$ time units after $\bar{w}_{i-1}$. Therefore, $r^\omega_{\tau_i(\zeta_i)} \leq S^\omega_{\tau_i,\bar{\xi}_i} \cdot T^{\max}_{\tau_i} + \bar{w}_{i-1} = \bar{r}_i$. We have proven that the job $\tau_i(\zeta_i)$ is released no later than at $\bar{r}_i$. Therefore, its write-event is no later than at $\bar{r}_i + D_{\tau_i}$. Hence, $\mathrm{we}^\omega(\tau_i(\zeta_i)) \leq \bar{r}_i + D_{\tau_i} = \bar{w}_i$. This concludes the induction step and proves Equation (6.113).

Now we prove Equation (6.114). We have

$$\ell(\vec{ac}^\omega_E(z)) = z' - z = \mathrm{we}^\omega(\tau_n(\zeta_n)) - z \leq \bar{w}_n - \bar{w}_0 = \sum_{i=1}^{n} \bar{w}_i - \bar{w}_{i-1}. \tag{6.115}$$

Moreover, $\bar{w}_i - \bar{w}_{i-1} = S^\omega_{\tau_i,\bar{\xi}_i} \cdot T^{\max}_{\tau_i} + D_{\tau_i}$ by definition. This proves Equation (6.114). □

A bound on the reaction time follows directly.

Lemma 6.53. *For a given $\omega \in \Omega$ and $z \geq \Phi(E)$, the reaction time is upper bounded by*

$$\mathrm{RT}(E, \omega, z) \leq \sum_{i=1}^{n} S^\omega_{\tau_i,\bar{\xi}_i} \cdot T^{\max}_{\tau_i} + D_{\tau_i}. \tag{6.116}$$

Proof. If $\vec{ac}^\omega_E(z)$ exists, then $\mathrm{RT}(E,\omega,z) = \ell(\vec{ac}^\omega_E(z)) \leq \sum_{i=1}^n S^\omega_{\tau_i,\bar{\xi}_i} \cdot T^{\max}_{\tau_i} + D_{\tau_i}$ follows from the previous lemma. If $\vec{ac}^\omega_E(z)$ does not exist, then this means that $\vec{ac}^\omega_E(z)$ cannot be fully constructed. In particular, there exists an $i \in \{1, \dots, n\}$ such that $S^\omega_{\tau_i,\xi_i} = \infty$. In that case, $S^\omega_{\tau_i,\bar{\xi}_i}$ is also ∞, and $\mathrm{RT}(E,\omega,z) \leq \infty = \sum_{i=1}^n S^\omega_{\tau_i,\bar{\xi}_i} \cdot T^{\max}_{\tau_i} + D_{\tau_i}$ holds. □

The result in Lemma 6.53 also covers the deterministic case. Specifically, the analysis is identical to the known deterministic results if the failure probability is 0.

Remark 6.54. *If there are no failures, i.e., $f_{\tau_i} = 0$ for all i, then $S^\omega_{\tau_i,\bar{\xi}_i} = 1$ for all $\omega \in \Omega$. Hence, the upper bound on the reaction time from Lemma 6.53 simplifies to*

$$\mathrm{RT}(E, \omega, z) \leq \sum_{i=1}^{n} T^{\max}_{\tau_i} + D_{\tau_i} \tag{6.117}$$

which is the state of the art for sporadic tasks under LET (cf. Theorem 3.11).

Up to this point, no independence was utilized because we only analyzed the random variables on certain samples ω of Ω. However, we use independence in the following theorem to investigate the probability distribution of random variables. More specifically, we replace the random variable $S^\bullet_{\tau_i,\bar{\xi}_i}$ from Lemma 6.53 by $S^\bullet_{\tau_i,1}$ due to independence. This step is necessary, because $\bar{\xi}_i$ depends on ω, and therefore we do not know the probability distribution of $S^\bullet_{\tau_i,\bar{\xi}_i}$, we only know that $S^\bullet_{\tau_i,j}$ is geometrically distributed for any *fixed* value $j \in \mathbb{N}$.

For the purpose of over-approximation, we define a dominance relation of random variables X and Y as follows: If the probability distribution of the random variable X is upper bounded by the probability distribution of Y (in the sense that $\mathbb{P}(X > x) \leq \mathbb{P}(Y > x)$ for all $x \in \mathbb{R}$) we write $X \trianglelefteq Y$.

Theorem 6.55. *The probability distribution of the random variable* $\mathrm{RT}(E, \bullet, z)$ *under LET is upper bounded by the probability distribution of* $\sum_{i=1}^n S^\bullet_{\tau_i,1} \cdot T^{\max}_{\tau_i} + D_{\tau_i}$. *That is,*

$$\mathrm{RT}(E, \bullet, z) \trianglelefteq \sum_{i=1}^{n} S^\bullet_{\tau_i,1} \cdot T^{\max}_{\tau_i} + D_{\tau_i} \tag{6.118}$$

in the sense that $\mathbb{P}(\mathrm{RT}(E, \bullet, z) > x) \leq \mathbb{P}(\sum_{i=1}^n S^\bullet_{\tau_i,1} \cdot T^{\max}_{\tau_i} + D_{\tau_i} > x)$ *for all* $x \in \mathbb{R}$.

Proof. By Lemma 6.53, $\mathbb{P}(\mathrm{RT}(E, \bullet, z) > x) \leq \mathbb{P}\left(\sum_{i=1}^n S^\bullet_{\tau_i,\bar{\xi}_i} T^{\max}_{\tau_i} + D_{\tau_i} > x\right)$ for all $x \in \mathbb{R}$. In the remainder of the proof we show that

$$\mathbb{P}\left(\sum_{i=1}^{n} S^\bullet_{\tau_i,\bar{\xi}_i} \cdot T^{\max}_{\tau_i} + D_{\tau_i} > x\right) = \mathbb{P}\left(\sum_{i=1}^{n} S^\bullet_{\tau_i,1} \cdot T^{\max}_{\tau_i} + D_{\tau_i} > x\right) \tag{6.119}$$

for all $x \in \mathbb{R}$. We know that $S^\bullet_{\tau_i,j}$ and $S^\bullet_{\tau_i,1}$ are identically distributed for any fixed j. Therefore, also $S^\bullet_{\tau_i,j} \cdot T^{\max}_{\tau_j}$ and $S^\bullet_{\tau_i,1} \cdot T^{\max}_{\tau_j}$ are identically distributed for any fixed j. However, in the left-hand side of Equation (6.119), we have $\bar{\xi}_i$ which depends on the system evolution. Therefore, we fix $\bar{\xi}_i$ to a specific value using the law of total probabilities [Bor99, Chapter 2.4]. That is,

$$\mathbb{P}\left(\sum_{i=1}^{n} S^\bullet_{\tau_i,\bar{\xi}_i} \cdot T^{\max}_{\tau_i} > x\right) = \sum_{j \in \mathbb{N}} \mathbb{P}\left(\sum_{i=1}^{n} S^\bullet_{\tau_i,\bar{\xi}_i} \cdot T^{\max}_{\tau_i} > x \;\middle|\; \bar{\xi}_n = j\right) \cdot \mathbb{P}(\bar{\xi}_n = j). \tag{6.120}$$

If $\bar{\xi}_n$ is fixed to j, then $S^\bullet_{\tau_n,\bar{\xi}_n}$ is independent of $S^\bullet_{\tau_i,\bar{\xi}_i}$ for all $i < n$. Hence, we can replace $S^\bullet_{\tau_n,\bar{\xi}_n}$ by the identically distributed (and also independent) random variable $S^\bullet_{\tau_n,1}$. Applying again the law of total probabilities afterwards yields

$$\mathbb{P}\left(\sum_{i=1}^{n} S^\bullet_{\tau_i,\bar{\xi}_i} \cdot T^{\max}_{\tau_i} > x\right) = \mathbb{P}\left(\sum_{i=1}^{n-1} S^\bullet_{\tau_i,\bar{\xi}_i} \cdot T^{\max}_{\tau_i} + S^\bullet_{\tau_n,1} \cdot T^{\max}_{\tau_n} > x\right). \tag{6.121}$$

Replacing $S^\bullet_{\tau_k,\bar{\xi}_k} \cdot T^{\max}_{\tau_k}$ by $S^\bullet_{\tau_k,1} \cdot T^{\max}_{\tau_k}$ for $k = n-1, \dots, 1$ consecutively, following the same procedure, results in $\mathbb{P}\left(\sum_{i=1}^n S^\bullet_{\tau_i,\bar{\xi}_i} \cdot T^{\max}_{\tau_i} > x\right) = \mathbb{P}\left(\sum_{i=1}^n S^\bullet_{\tau_i,1} \cdot T^{\max}_{\tau_i} > x\right)$. By adding the constant value $\sum_{i=1}^n D_{\tau_i}$, we obtain Equation (6.119). $\square$

The upper bound on the probability distribution of $\mathrm{RT}(E, \bullet, z)$ is used to bound the expected value of the reaction time $\mathbb{E}(\mathrm{RT}(E, \bullet, z))$ and the probabilistic reaction time guarantee $\mathrm{PRTG}(x)$.

Corollary 6.56. *Since $S^{\bullet}_{\tau_i,1}$ is geometrically distributed with failure probability f_{τ_i}, the expected value of the probabilistic reaction time under LET is upper bounded by*

$$\mathbb{E}(\mathrm{RT}(E,\bullet,z)) \leq \sum_{i=1}^{n} \frac{1}{1-f_{\tau_i}} \cdot T^{\max}_{\tau_i} + D_{\tau_i}. \tag{6.122}$$

Proof. This follows from Theorem 6.55, the additivity of the expected value, and $\mathbb{E}(S^{\bullet}_{\tau_i,1}) = \frac{1}{1-f_{\tau_i}}$ for the geometrically distributed random variable $S^{\bullet}_{\tau_i,1}$. □

Corollary 6.57. *The probabilistic reaction time guarantee under LET is bounded by*

$$\mathrm{PRTG}(x) \geq \mathbb{P}\left(\sum_{i=1}^{n} S^{\bullet}_{\tau_i,1} \cdot T^{\max}_{\tau_i} + D_{\tau_i} \leq x\right) \tag{6.123}$$

Proof. By Definition 6.46, $\mathrm{PRTG}(x) = \inf_{z\geq\Phi(E)} \mathbb{P}(\mathrm{RT}(E,\bullet,z) \leq x)$. Moreover, we know that $\inf_{z\geq\Phi(E)} \mathbb{P}(\mathrm{RT}(E,\bullet,z) \leq x) \geq \inf_{z\geq\Phi(E)} \mathbb{P}(\sum_{i=1}^{n} S^{\bullet}_{\tau_i,1} \cdot T^{\max}_{\tau_i} + D_{\tau_i} \leq x)$ by Theorem 6.55. Since $\mathbb{P}(\sum_{i=1}^{n} S^{\bullet}_{\tau_i,1} \cdot T^{\max}_{\tau_i} + D_{\tau_i} \leq x)$ is independent of z, the infimum can be dropped. □

We discuss in Section 6.3.5 how to compute the safe bound on PRTG efficiently.

6.3.4 Probabilistic Reaction Time Under Implicit Communication

Under implicit communication, both the response-time randomness and failure probability play roles and the analysis should consider both of them. Still, the strategy for this section is similar to the one of the previous section: First, we provide an upper bound for $\mathrm{RT}(E,\omega,z)$, and afterwards, we utilize that upper bound to analyze the probabilistic reaction time and the probabilistic reaction time guarantee. However, now we consider the implicit-communication semantic, i.e., the read-event is at the start of each job and the write-event is at the finish of each job.

For a fixed system evolution $\omega = ((r^{\omega}_J)_{J\in\mathbb{J}}, (c^{\omega}_J)_{J\in\mathbb{J}}, \mathcal{F}^{\omega})$, we recursively construct upper bounds (to be specified in Equation (6.124)) for the immediate forward augmented job chain $\vec{ac}^{\omega}_E(z) = (z, (\tau_1(\xi_1), \tau_1(\zeta_1)), \dots, (\tau_n(\xi_n), \tau_n(\zeta_n)), z')$ as follows:

- $\bar{w}_0 := z$
- For $i = 1, \dots, n$:
  - $\bar{\xi}_i \in \mathbb{N}$ such that $\tau_i(\bar{\xi}_i)$ is the first job released at or after $\bar{w}_{i-1}$
  - $\bar{\zeta}_i = \bar{\xi}_i + S^{\omega}_{\tau_i,\bar{\xi}_i} - 1$
  - $\bar{r}_i := S^{\omega}_{\tau_i,\bar{\xi}_i} \cdot T^{\max}_{\tau_i} + \bar{w}_{i-1}$
  - $\bar{w}_i := \bar{r}_i + \overline{R}^{\mathcal{A}(\omega),\omega}_{\tau_i(\bar{\zeta}_i)}$

where $S^\omega_{\tau_i,\bar{\xi}_i}$ is the number of jobs of τ_i until the first successful job beginning at $\tau_i(\bar{\xi}_i)$ according to $\mathcal{F}^\omega$. We can bound the length of $\vec{ac}^\omega_E(z)$ as follows:

Lemma 6.58. *Let $\vec{ac}^\omega_E(z) = (z, (\tau_1(\xi_1), \tau_1(\zeta_1)), \dots, (\tau_n(\xi_n), \tau_n(\zeta_n)), z')$ be an immediate forward augmented job chain with $z \geq \Phi(E)$. Then, for all $i = 1, \dots, n$:*

$$\xi_i \leq \bar{\xi}_i, \quad \zeta_i \leq \bar{\zeta}_i, \quad r^\omega_{\tau_i(\zeta_i)} \leq \bar{r}_i, \quad \textit{and} \quad \mathrm{we}^\omega(\tau_i(\zeta_i)) \leq \bar{w}_i \tag{6.124}$$

In particular,

$$\ell(\vec{ac}^\omega_E(z)) \leq \bar{w}_n - z = \sum_{i=1}^n S^\omega_{\tau_i,\bar{\xi}_i} \cdot T^{\max}_{\tau_i} + \overline{R}^{\mathcal{A}(\omega),\omega}_{\tau_i(\bar{\zeta}_i)}. \tag{6.125}$$

Proof. The proof is analogous to the proof of Lemma 6.52 for LET, except that the write-event of $\tau_i(\zeta_i)$ is upper bounded by

$$\mathrm{we}^\omega(\tau_i(\zeta_i)) \leq \mathrm{we}^\omega(\tau_i(\bar{\zeta}_i)) \leq r^\omega_{\tau_i(\bar{\zeta}_i)} + \overline{R}^{\mathcal{A}(\omega),\omega}_{\tau_i(\bar{\zeta}_i)} \leq \bar{r}_i + \overline{R}^{\mathcal{A}(\omega),\omega}_{\tau_i(\bar{\zeta}_i)}, \tag{6.126}$$

instead of $\bar{r}_i + D_{\tau_i}$. □

A bound on the reaction time follows directly.

Lemma 6.59. *For a given $\omega \in \Omega$ and $z \geq \Phi(E)$, the reaction time is upper bounded by*

$$\mathrm{RT}(E, \omega, z) \leq \sum_{i=1}^n S^\omega_{\tau_i,\bar{\xi}_i} \cdot T^{\max}_{\tau_i} + \overline{R}^{\mathcal{A}(\omega),\omega}_{\tau_i(\bar{\zeta}_i)}. \tag{6.127}$$

Proof. If $\vec{ac}^\omega_E(z)$ exists, then the equation follows from the previous lemma. If $\vec{ac}^\omega_E(z)$ does not exist, then there must be some $i \in \{1, \dots, n\}$ such that $S^\omega_{\tau_i,\xi_i} = \infty$. In that case, $S^\omega_{\tau_i,\bar{\xi}_i}$ is also ∞, and $\mathrm{RT}(E, \omega, z) \leq \infty = \sum_{i=1}^n S^\omega_{\tau_i,\bar{\xi}_i} \cdot T^{\max}_{\tau_i} + \overline{R}^{\mathcal{A}(\omega),\omega}_{\tau_i(\bar{\zeta}_i)}$ holds. □

This result also covers the deterministic case where the failure probability is 0 and the response-time distribution is upper-bounded by the WCRT.

Remark 6.60. *If there are no failures, i.e., $f_{\tau_i} = 0$ for all i, then $S^\omega_{\tau_i,\bar{\xi}_i} = 1$. If further $\overline{R}^{\mathcal{A}(\omega),\omega}_{\tau_i(\bar{\zeta}_i)}$ is no random variable but a fixed value, e.g., the WCRT $R^{\mathcal{A}}_{\tau_i}$, then the upper bound on the probabilistic reaction time from Lemma 6.59 simplifies to*

$$\mathrm{RT}(E, \omega, z) \leq \sum_{i=1}^n T^{\max}_{\tau_i} + R^{\mathcal{A}}_{\tau_i}, \tag{6.128}$$

which is the popular bound from Davare (see Theorem 3.9) for sporadic tasks under implicit communication.

Up to this point no independence was utilized because we only analyzed the random variables on certain samples of Ω. However, we use independence in the following theorem to investigate the probability distribution of random variables. More specifically, we replace the random variables $S^\bullet_{\tau_i,\bar{\xi}_i}$ and $\overline{R}^{\mathcal{A}(\bullet),\bullet}_{\tau_i(\bar{\zeta}_i)}$ by $S^\bullet_{\tau_i,1}$ and $\overline{R}^{\mathcal{A}(\bullet),\bullet}_{\tau_i(1)}$, respectively. As for Theorem 6.55, to achieve the replacement of random variables, we have to take care of the random variables $\bar{\xi}_i$ and $\bar{\zeta}_i$.

Theorem 6.61. *The probability distribution of the random variable* $\mathrm{RT}(E, \bullet, z)$ *under implicit communication is upper bounded by the probability distribution of* $\sum_{i=1}^{n} S^{\bullet}_{\tau_i,1} \cdot T^{\max}_{\tau_i} + \overline{R}^{\mathcal{A}(\bullet),\bullet}_{\tau_i(1)}$. *That is,*

$$\mathrm{RT}(E, \bullet, z) \trianglelefteq \sum_{i=1}^{n} S^{\bullet}_{\tau_i,1} \cdot T^{\max}_{\tau_i} + \overline{R}^{\mathcal{A}(\bullet),\bullet}_{\tau_i(1)} \tag{6.129}$$

in the sense that $\mathbb{P}(\mathrm{RT}(E, \bullet, z) > x) \leq \mathbb{P}(\sum_{i=1}^{n} S^{\bullet}_{\tau_i,1} \cdot T^{\max}_{\tau_i} + \overline{R}^{\mathcal{A}(\bullet),\bullet}_{\tau_i(1)} > x)$ *for all* $x \in \mathbb{R}$.

Proof. This proof is similar to the proof of Theorem 6.55. First, by Lemma 6.59, we know that $\mathbb{P}(\mathrm{RT}(E, \bullet, z) > x) \leq \mathbb{P}\left(\sum_{i=1}^{n} S^{\bullet}_{\tau_i,\bar{\xi}_i} \cdot T^{\max}_{\tau_i} + \overline{R}^{\mathcal{A}(\omega),\omega}_{\tau_i(\bar{\zeta}_i)} > x\right)$ for all $x \in \mathbb{R}$. It is left to show that

$$\mathbb{P}\left(\sum_{i=1}^{n} S^{\bullet}_{\tau_i,\bar{\xi}_i} \cdot T^{\max}_{\tau_i} + \overline{R}^{\mathcal{A}(\bullet),\bullet}_{\tau_i(\bar{\zeta}_i)} > x\right) = \mathbb{P}\left(\sum_{i=1}^{n} S^{\bullet}_{\tau_i,1} \cdot T^{\max}_{\tau_i} + \overline{R}^{\mathcal{A}(\bullet),\bullet}_{\tau_i(1)} > x\right). \tag{6.130}$$

To achieve that, we again use the law of total probabilities. That is, we first fix $\bar{\zeta}_n$ to replace $\overline{R}^{\mathcal{A}(\bullet),\bullet}_{\tau_n(\bar{\zeta}_n)}$ by $\overline{R}^{\mathcal{A}(\bullet),\bullet}_{\tau_n(1)}$, and then we fix $\bar{\xi}_n$ to replace $S^{\bullet}_{\tau_n,\bar{\xi}_n}$ by $S^{\bullet}_{\tau_n,\bar{\xi}_n}$. Specifically,

$$\mathbb{P}\left(\sum_{i=1}^{n} (S^{\bullet}_{\tau_i,\bar{\xi}_i} \cdot T^{\max}_{\tau_i} + \overline{R}^{\mathcal{A}(\bullet),\bullet}_{\tau_i(\bar{\zeta}_i)}) > x\right) \tag{6.131}$$

$$= \sum_{j \in \mathbb{N}} \mathbb{P}\left(\sum_{i=1}^{n} (S^{\bullet}_{\tau_i,\bar{\xi}_i} \cdot T^{\max}_{\tau_i} + \overline{R}^{\mathcal{A}(\bullet),\bullet}_{\tau_i(\bar{\zeta}_i)}) > x \,\middle|\, \bar{\zeta}_n = j\right) \cdot \mathbb{P}(\bar{\zeta}_n = j) \tag{6.132}$$

$$= \sum_{j \in \mathbb{N}} \mathbb{P}\left(\sum_{i=1}^{n-1} (S^{\bullet}_{\tau_i,\bar{\xi}_i} \cdot T^{\max}_{\tau_i} + \overline{R}^{\mathcal{A}(\bullet),\bullet}_{\tau_i(\bar{\zeta}_i)}) + S^{\bullet}_{\tau_n,\bar{\xi}_n} \cdot T^{\max}_{\tau_n} + \overline{R}^{\mathcal{A}(\bullet),\bullet}_{\tau_n(1)} > x \,\middle|\, \bar{\zeta}_n = j\right) \cdot \mathbb{P}(\bar{\zeta}_n = j) \tag{6.133}$$

$$= \mathbb{P}\left(\sum_{i=1}^{n-1} (S^{\bullet}_{\tau_i,\bar{\xi}_i} \cdot T^{\max}_{\tau_i} + \overline{R}^{\mathcal{A}(\bullet),\bullet}_{\tau_i(\bar{\zeta}_i)}) + S^{\bullet}_{\tau_n,\bar{\xi}_n} \cdot T^{\max}_{\tau_n} + \overline{R}^{\mathcal{A}(\bullet),\bullet}_{\tau_n(1)} > x\right) \tag{6.134}$$

is achieved by fixing $\bar{\zeta}_n$. Fixing $\bar{\xi}_n$ afterwards, yields

$$\mathbb{P}\left(\sum_{i=1}^{n-1}(S^\bullet_{\tau_i,\bar{\xi}_i}\cdot T^{\max}_{\tau_i}+\overline{R}^{\mathcal{A}(\bullet),\bullet}_{\tau_i(\bar{\zeta}_i)})+S^\bullet_{\tau_n,\bar{\xi}_n}\cdot T^{\max}_{\tau_n}+\overline{R}^{\mathcal{A}(\bullet),\bullet}_{\tau_n(1)}>x\right) \tag{6.135}$$

$$=\sum_{j\in\mathbb{N}}\mathbb{P}\left(\sum_{i=1}^{n-1}(S^\bullet_{\tau_i,\bar{\xi}_i}\cdot T^{\max}_{\tau_i}+\overline{R}^{\mathcal{A}(\bullet),\bullet}_{\tau_i(\bar{\zeta}_i)})+S^\bullet_{\tau_n,\bar{\xi}_n}\cdot T^{\max}_{\tau_n}+\overline{R}^{\mathcal{A}(\bullet),\bullet}_{\tau_n(1)}>x\;\middle|\;\bar{\xi}_n=j\right)\cdot\mathbb{P}(\bar{\xi}_n=j) \tag{6.136}$$

$$=\sum_{j\in\mathbb{N}}\mathbb{P}\left(\sum_{i=1}^{n-1}(S^\bullet_{\tau_i,\bar{\xi}_i}\cdot T^{\max}_{\tau_i}+\overline{R}^{\mathcal{A}(\bullet),\bullet}_{\tau_i(\bar{\zeta}_i)})+S^\bullet_{\tau_n,1}\cdot T^{\max}_{\tau_n}+\overline{R}^{\mathcal{A}(\bullet),\bullet}_{\tau_n(1)}>x\;\middle|\;\bar{\xi}_n=j\right)\cdot\mathbb{P}(\bar{\xi}_n=j) \tag{6.137}$$

$$=\mathbb{P}\left(\sum_{i=1}^{n-1}(S^\bullet_{\tau_i,\bar{\xi}_i}\cdot T^{\max}_{\tau_i}+\overline{R}^{\mathcal{A}(\bullet),\bullet}_{\tau_i(\bar{\zeta}_i)})+S^\bullet_{\tau_n,1}\cdot T^{\max}_{\tau_n}+\overline{R}^{\mathcal{A}(\bullet),\bullet}_{\tau_n(1)}>x\right). \tag{6.138}$$

Repeating this procedure, replacing $\overline{R}^{\mathcal{A}(\bullet),\bullet}_{\tau_k(\bar{\zeta}_k)}$ by $\overline{R}^{\mathcal{A}(\bullet),\bullet}_{\tau_k(1)}$ and $S^\bullet_{\tau_k,\bar{\xi}_k}$ by $S^\bullet_{\tau_k,1}$, for $k=n-1,\dots,1$, results in Equation (6.130). □

The upper bound on the probability distribution of $\mathrm{RT}(E,\bullet,z)$ is used to bound the expected value of the reaction time $\mathbb{E}(\mathrm{RT}(E,\bullet,z))$ and the probabilistic reaction time guarantee $\mathrm{PRTG}(x)$.

Corollary 6.62. *Since $S^\bullet_{\tau_i,1}$ is geometrically distributed with failure probability f_{τ_i}, the expected value of the probabilistic reaction time under implicit communication is upper bounded by*

$$\mathbb{E}(\mathrm{RT}(E,\bullet,z))\leq\sum_{i=1}^{n}\frac{1}{1-f_{\tau_i}}\cdot T^{\max}_{\tau_i}+\mathbb{E}(\overline{R}^{\mathcal{A}(\bullet),\bullet}_{\tau_i(1)}). \tag{6.139}$$

Proof. This follows from Theorem 6.61, the additivity of the expected value, and $\mathbb{E}(S^\bullet_{\tau_i,1})=\frac{1}{1-f_{\tau_i}}$ for the geometrically distributed random variable $S^\bullet_{\tau_i,1}$. □

Corollary 6.63. *The probabilistic reaction time guarantee under implicit communication is bounded by*

$$\mathrm{PRTG}(x)\geq\mathbb{P}\left(\sum_{i=1}^{n}S^\bullet_{\tau_i,1}\cdot T^{\max}_{\tau_i}+\overline{R}^{\mathcal{A}(\bullet),\bullet}_{\tau_i(1)}\leq x\right) \tag{6.140}$$

Proof. By Definition 6.46, $\mathrm{PRTG}(x)=\inf_{z\geq\Phi(E)}\mathbb{P}(\mathrm{RT}(E,\bullet,z)\leq x)$. Moreover, we know that $\mathbb{P}(\mathrm{RT}(E,\bullet,z)\leq x)\geq\mathbb{P}(\sum_{i=1}^{n}S^\bullet_{\tau_i,1}\cdot T^{\max}_{\tau_i}+\overline{R}^{\mathcal{A}(\bullet),\bullet}_{\tau_i(1)}\leq x)$ by Theorem 6.61 for all $z\geq\Phi(E)$. Since $\mathbb{P}(\sum_{i=1}^{n}S^\bullet_{\tau_i,1}\cdot T^{\max}_{\tau_i}+\overline{R}^{\mathcal{A}(\bullet),\bullet}_{\tau_i(1)}\leq x)$ is independent of z, the infimum can be dropped. □

We discuss in Section 6.3.5 how to compute the safe bound on PRTG efficiently.

6.3.5 Computation

For efficient computation of the probabilistic reaction time guarantees presented in Corollaries 6.57 and 6.63, we adopt the idea by Chen and Chen [CC17] and apply the Chernoff bound to obtain a minimization problem over moment generating functions.

The Chernoff bound for a random variable X and a real number x is

$$\mathbb{P}(X \geq x) \leq \inf_{t>0} M_X(t) \cdot \mathrm{e}^{-tx} \tag{6.141}$$

where $M_X(t) := \mathbb{E}(\mathrm{e}^{tX})$ is the moment generating function of X. By applying the Chernoff bound to the lower bounds provided in Corollaries 6.57 and 6.63, we obtain the following lower bounds for PRTG.

Lemma 6.64. *For $x \in \mathbb{R}$, the following bounds hold:*

$$\mathrm{PRTG}(x) \geq \begin{cases} 1 - \inf_{t>0} \mathrm{e}^{-tx} \cdot \prod_{i=1}^n M_{S^\bullet_{\tau_i,1}}(T^{\max}_{\tau_i} t) \cdot \mathrm{e}^{D_{\tau_i} t} & \textit{under LET} \\ 1 - \inf_{t>0} \mathrm{e}^{-tx} \cdot \prod_{i=1}^n M_{S^\bullet_{\tau_i,1}}(T^{\max}_{\tau_i} t) \cdot M_{\overline{R}^{\mathcal{A}(\bullet),\bullet}_{\tau_i(1)}}(t) & \textit{under impl. comm.} \end{cases} \tag{6.142}$$

Proof. By Corollaries 6.57 and 6.63,

$$\mathrm{PRTG}(x) \geq \mathbb{P}(X \leq x) \tag{6.143}$$

with $X = \sum_{i=1}^n S^\bullet_{\tau_i,1} \cdot T^{\max}_{\tau_i} + D_{\tau_i}$ if tasks communicate according to LET, and $X = \sum_{i=1}^n S^\bullet_{\tau_i,1} \cdot T^{\max}_{\tau_i} + \overline{R}^{\mathcal{A}(\bullet),\bullet}_{\tau_i(1)}$ under implicit communication. The Chernoff-bound yields

$$\mathrm{PRTG}(x) \geq \mathbb{P}(X \leq x) \geq 1 - \mathbb{P}(X \geq x) \geq 1 - \inf_{t>0} M_X(t) \cdot \mathrm{e}^{-tx}. \tag{6.144}$$

Applying the following rules for the moment generating function yields Equation (6.142):

- $M_{Y+Z}(t) = M_Y(t) \cdot M_Z(t)$
- $M_{T \cdot Y}(t) = M_Y(T \cdot t)$
- $M_K(t) = \mathrm{e}^{K \cdot t}$

for all random variables Y, Z, all factors $T \in \mathbb{R}$, and all constants $K \in \mathbb{R}$. □

Remark 6.65. *The moment generating function of $S^\bullet_{\tau_i,1}$ (geometrically distributed) and $\overline{R}^{\mathcal{A}(\bullet),\bullet}_{\tau_i(1)}$ (discrete) is as follows:*

$$M_{S^\bullet_{\tau_i,1}}(T^{\max}_{\tau_i} t) = \frac{(1-f_{\tau_i})\mathrm{e}^{T^{\max}_{\tau_i} t}}{1 - f_{\tau_i}\mathrm{e}^{T^{\max}_{\tau_i} t}} \quad \textit{for all } t < \frac{-\ln(f_{\tau_i})}{T^{\max}_{\tau_i}} \tag{6.145}$$

$$M_{\overline{R}^{\mathcal{A}(\bullet),\bullet}_{\tau_i(1)}}(t) = \sum_v \mathrm{e}^{tv} \cdot \mathbb{P}(\overline{R}^{\mathcal{A}(\bullet),\bullet}_{\tau_i(1)} = v) \tag{6.146}$$

In this formula, v are the possible outcomes of the discrete random variable $\overline{R}^{\mathcal{A}(\bullet),\bullet}_{\tau_i(1)}$.

Since the moment generating function is log-convex, the infimum in Equation (6.142) can be efficiently computed. As suggested by Chen et al. [Che+19c], we use golden section search to calculate it.

6.3.6 Evaluation

Since this is the first analysis of end-to-end latency considering failure probabilities and response-time randomness for cause-effect chains based on sporadic tasks, comparison to other existing approaches is not possible unless there is no randomness at all. In that specific case without any randomness, when everything is deterministic (i.e., by investigating the worst-case under TDMA), then our result in Lemma 6.53 for LET (Lemma 6.59 for implicit communication, respectively) is identical to the state of the art as noted in Remark 6.54 (Remark 6.60, respectively). We further note that the analysis of implicit communication under T-FP preemptive scheduling by Dürr et al. [Dür+19] and Günzel et al. [Gün+21a] cannot be applied for TDMA. We demonstrate the impact of the probabilistic behavior on PRTG from Definition 6.46. The evaluation section is organized as follows:

- Section 6.3.6.1 specifies the benchmark used to generate task sets and cause-effect chains.
- Section 6.3.6.2 demonstrates the impact of probabilistic behavior for LET and for implicit communication.

More specifically, we examine the safe bounds on the *probabilistic reaction time guarantees* of cause-effect chains stated in Corollaries 6.57 (denoted as $\mathcal{L}_{LET}$) and 6.63 (denoted as $\mathcal{L}_{impl}$). The x-axis is normalized with the state-of-the-art for non-probabilistic behavior:

$$\text{MRT} = \sum_{i=1}^{n} T_{\tau_i}^{\max} + D_{\tau_i} \quad \text{under LET, i.e., Hamann et al. [Ham+17]} \tag{6.147}$$

$$\text{MRT} = \sum_{i=1}^{n} T_{\tau_i}^{\max} + R_{\tau_i}^{\mathcal{A}} \quad \text{under implicit communication, i.e., Davare et al. [Dav+07]} \tag{6.148}$$

That is, the function value at $x = 1.0$ is the probabilistic guarantee that the reaction time is at most the non-probabilistic bound (i.e., comparison to the state of the art for deterministic bounds), and the function value at $x = 2.0$ is the probabilistic guarantee that the reaction time is at most 2 times the non-probabilistic bound.

6.3.6.1 Task Set and Cause-Effect Chain Generation

In this section, we detail the generation process of sporadic task sets and corresponding cause-effect chains for the evaluation. We use the *free real world automotive benchmarks* provided by Kramer et al. [KZH15] in WATERS 2015 to generate periodic tasks and corresponding cause-effect chains. Afterwards, we translate the periodic tasks to sporadic tasks and inject probabilistic behavior.

We generate periodic task sets based on the parameters in Tables III, IV, and V in [KZH15]. In particular, for each generated task τ the following steps are carried out:

(i) A task period T_τ is drawn at random from the set $\{1, 2, 5, 10, 20, 50, 100, 200, 1000\}$ according to the share stated in [KZH15, Table III].[11]

(ii) The ACET of each task is generated according to a Weibull distribution based on the values in [KZH15, Table IV].

(iii) For the WCET, the ACET is multiplied with a factor drawn at random from the interval $[f_{min}, f_{max}]$ in [KZH15, Table V].

For each task set, we first generate 30 000 tasks and then use the subset-sum approximation algorithm until the total utilization is within 1 percentage point of the targeted utilization of 70 %. Each task set contains 82 tasks on average. We consider task sets distributed on 3 ECUs by generating one task set for each ECU and combining them afterwards.

We generate cause-effect chains based on Section IV-E in [KZH15]. Each cause-effect chain consists of 2 to 15 tasks. The cause-effect chain for a task set $\mathbb{T}$ is generated as follows:

(i) The number of involved activation patterns P is drawn randomly from $\{1, 2, 3\}$ according to [KZH15, Table VI].

(ii) P different periods are drawn at random from the set of periods in $\mathbb{T}$.

(iii) For each period, we draw 2 to 5 tasks at random without replacement in $\mathbb{T}$ according to [KZH15, Table VII].

If there are not enough tasks with the required period in $\mathbb{T}$, then the cause-effect chain and the task set are discarded. We repeat the procedure until 1 000 task sets and cause-effect chains for each experiment are successfully generated.

The tasks are made sporadic by setting $T_\tau^{\min} = T_\tau$ and $T_\tau^{\max} = 2T_\tau$. We use a TDMA-based scheduler to schedule the sporadic tasks. In particular, during each cycle with period $T^c := 0.1$, each task is assigned a certain slot with size $Q^\tau := 1.2 \cdot \frac{C_\tau}{T_\tau^{\min}} \cdot 0.1$. The slot is always at the same position of the cycle, which means that during any time interval of length $k \cdot T^c, k \in \mathbb{N}$, task τ can execute for $k \cdot Q^\tau$ time units and the WCRT $R_\tau^{\mathcal{A}}$ of task τ is upper bounded by $\left\lceil \frac{C_\tau}{Q^\tau} \right\rceil \cdot (T^c - Q^\tau) + C_\tau$.

Probabilities are injected as follows:

Failure probabilities: The failure probability f_τ for each task is drawn uniformly at random from a certain interval. We distinguish the following configurations:

[11] The sum of probabilities in [KZH15, Table III] is only 85 % because the remaining 15 % is for angle-synchronous tasks. Therefore, we divide all given share values by 0.85.

- Low failure probability: $f_\tau \in [0, 0.001]$
- Medium failure probability: $f_\tau \in [0.001, 0.01]$
- High failure probability: $f_\tau \in [0.01, 0.1]$

Response-time randomness: We assume that the response time is smaller than the WCRT in 90 % of cases. We distinguish different cases for how small the response time becomes:

- Slightly shorter response time: $0.8 \cdot R_\tau^{\mathcal{A}}$ in 90 % of cases
- Moderately shorter response time: $0.5 \cdot R_\tau^{\mathcal{A}}$ in 90 % of cases
- Immensely shorter response time: $0.2 \cdot R_\tau^{\mathcal{A}}$ in 90 % of cases

6.3.6.2 Evaluation Results

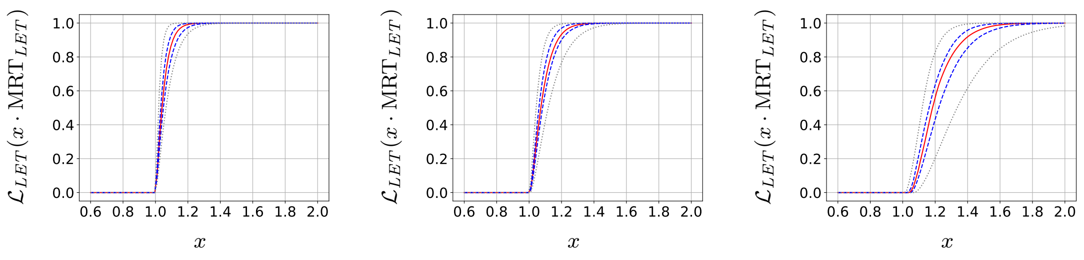
We evaluate the upper bound on the probabilistic reaction time guarantee. In the following figures, the red lines depict median, the blue dashed lines depict upper and lower quartile, and the dotted gray lines depict all values.

We first investigate the impact of failure probabilities under LET. As there is no impact of the response-time randomness because the read- and write-events are independent of the execution behavior under LET, the only source of randomness if the failure probability. In this case, if the failure probability is 0 %, our result is identical to the state of the art as noted in Remark 6.54. The impact of different failure probabilities under LET is illustrated in Figure 6.19, with MRT_{LET} being the MRT without any failures from Equation (6.147). We note that the state of the art cannot deal with any of these scenarios at all. As long as the failure probability is strictly more than 0 %, the probabilistic reaction time guarantee to be no more than MRT_{LET} is 0 %. As expected, the higher the failure probability, the lower are the probabilistic reaction time guarantees provided by our analysis. For the median case, there is a 99 % guarantee that the reaction time is no more than:

- $1.20 \cdot \text{MRT}_{LET}$ (with low failure probability)
- $1.30 \cdot \text{MRT}_{LET}$ (with medium failure probability)
- $1.62 \cdot \text{MRT}_{LET}$ (with high failure probability)

Our result shows that the failure probability has a direct impact on the probabilistic guarantee of the reaction time under LET. As long as the failure probability is small, a good probabilistic reaction time guarantee can still be provided.

The impact of different failure probabilities and the impact of the probabilistic WCRT under implicit communication is illustrated in Figure 6.20, with $\text{MRT}_{implicit}$ being the MRT without any failures from Equation (6.148). In this case, if the failure probability is 0 % and there is no randomness of response times, our result is identical to the state



(a) Low failure probability. (b) Medium failure probability. (c) High failure probability.

Figure 6.19.: Safe bound $\mathcal{L}_{LET}$ on the probabilistic reaction time guarantee under LET. Red line depicts median, blue dashed lines depict upper and lower quartile, and dotted gray lines depict all values.

of the art as noted in Remark 6.60. If there is randomness in the response time, even if the failure probability is more than 0 %, the probabilistic reaction time guarantee to be strictly less than $\mathrm{MRT}_{implicit}$ can be more than 0 %. For example, Figure 6.20g and Figure 6.20h show that the probability that the reaction time is strictly less than $\mathrm{MRT}_{implicit}$ is almost 99 % and 90 % for the scenarios with immensely shorter response times under low failure probability, respectively. We observe that with shorter response times the probabilistic reaction time guarantee increases. Similar to the LET scenarios, with higher failure probability, the probabilistic reaction time guarantee decreases.

6.4 Summary and Open Problems

In this chapter, we investigated the end-to-end latency of cause-effect chains. To that end, we first examined fundamental properties in Section 6.1. Subsequently, we analyzed the worst-case end-to-end latency in Section 6.2 and probabilistic guarantees for the end-to-end latency in Section 6.3.

Regarding fundamental properties, we developed a compositional property. That is, we showed that it is safe to cut a cause-effect chain into subchains and analyze them individually. Furthermore, we showed that MRT and MDA are equivalent. To that end, we developed a unified description of end-to-end latencies using partitioned job chains. The two properties only require very minor assumptions, making them applicable to basically all classical communication and scheduling mechanisms.

Regarding the analysis of the worst-case end-to-end latency, we uncovered the principles that current end-to-end latency bounds are built from, and used these to derive bounds for local cause-effect chains under different communication and release policies with a unified analysis framework. Furthermore, we used abstract integer representations and partitioned job chains to develop novel analysis approaches for local cause-effect chains under periodic release patterns. Subsequently, we used the compositional property to extend the bounds for local cause-effect chains to interconnected cause-effect chains and heterogeneous setups of sporadic and periodic tasks, as well as implicit communication and LET in the same chain.

Regarding the analysis of the probabilistic end-to-end latency, we defined the probabilistic reaction time and discussed potential pitfalls of the definition. Afterwards we provided analytical guarantees on the probabilistic reaction time under LET and implicit communication. For that, we considered two types of randomness: Response-time randomness and failure probabilities.

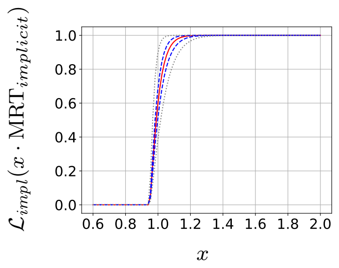
Despite our contributions of this chapter, we identify the following research directions to be explored further:

- **System design to meet requirements:** While this chapter deals with the problem of analyzing an already designed system, the proper design of systems to meet requirements on the end-to-end latency is even more challenging. That is, functionalities must be divided into tasks, tasks must be allocated to ECUs, and task parameters like period, phase and priority must be configured. While there have been some results to optimize the system towards meeting end-to-end latency requirements [Dav+07; Sch+18a; Wan+23], they are usually limited to specific task parameters. Hence, finding more complete solutions is still an open problem.

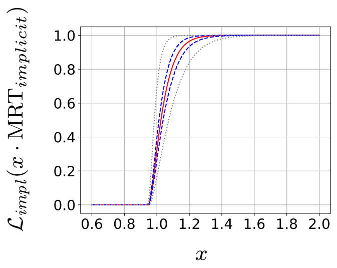
- **Triggering mechanisms:** While the analysis of this chapter has a special focus on time-triggered cause-effect chain, the relation towards other triggering mechanisms has to be further explored. Specifically, while releasing jobs in a time-triggered manner directly allows applying classical results from the literature on real-time systems, like WCRT bounds, it is also inefficient in the sense that many jobs may be released which do not take part in data transmission. Another approach is to use event-triggered tasks, which release jobs when specific events occur such as the arrival of data to the shared resource that the task reads from. While event-triggered tasks usually improve efficiency, they do that at the cost of predictability. Typical applications, e.g., those based on ROS2, allow the configuration of both time-triggered and event-triggered tasks. However, the trade-off between both mechanisms needs to be further explored and a unified view on different triggering mechanisms should be pursued.

- **Timing semantics:** We have shown that two metrics (MRT and MDA) are equivalent. Besides these, other timing semantics have been defined, namely MRRT, MRDA, First-to-Last, Last-to-First, and worst-case time disparity, as introduced in Section 3.2.2. This raises several questions to be explored: What is the intuitive meaning behind these metrics? Which metric should be specified in which use cases? Are there other important metrics?

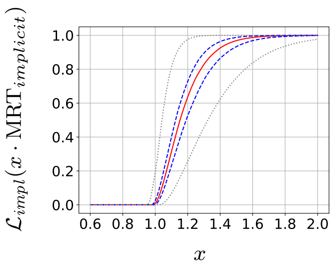
- **Probabilistic end-to-end latency:** While classical results analyze the worst-case behavior, soft real-time systems are common in industry practice as well [Ake+20]. Therefore, the analysis of probabilistic timing behavior has gained traction through recent developments [BBB21; Brü+21; Che+22a; DC19; Mar+23]. While this research direction is highly promising, in the context of end-to-end analysis it has been explored only to a limited extent so far. More specifically, in this chapter, we presented the first analysis of probabilistic reaction time. However, the analysis only considers randomness in the response time and failure probabilities, assuming that all random variables are iid and that the release pattern is fixed with lower and upper bounds on the inter-arrival time. How to compute iid response-time bounds is an open question for most scheduling algorithms. There are also approaches to drop the iid assumption [Mar+23], building upon Cantelli's inequality. However, the compatibility of such approaches with probabilistic end-to-end latency has to be further explored.



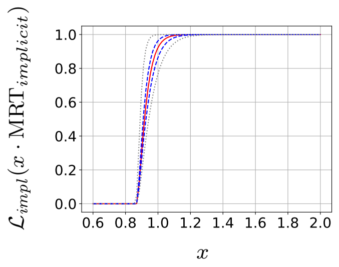
(a) Low failure probability, slightly shorter response time.



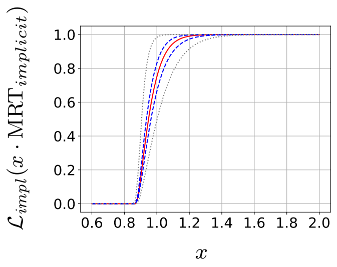
(b) Medium failure probability, slightly shorter response time.



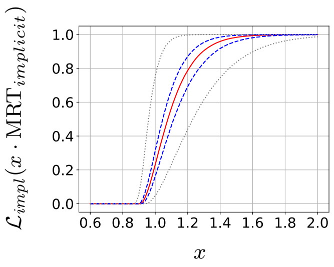
(c) High failure probability, slightly shorter response time.



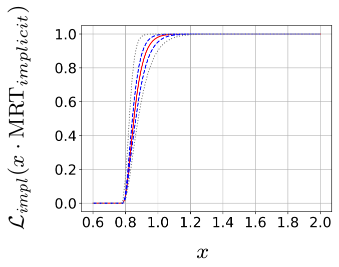
(d) Low failure probability, moderately shorter response time.



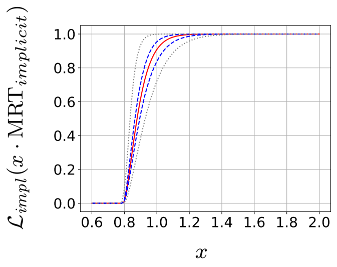
(e) Medium failure probability, moderately shorter response time.



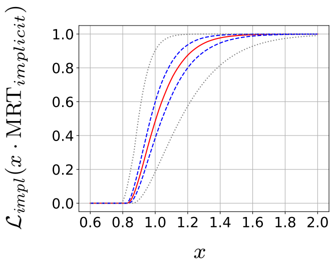
(f) High failure probability, moderately shorter response time.



(g) Low failure probability, immensely shorter response time.



(h) Medium failure probability, immensely shorter response time.



(i) High failure probability, immensely shorter response time.

Figure 6.20.: Safe bound $\mathcal{L}_{impl}$ on the probabilistic reaction time guarantee under implicit communication. Red line depicts median, blue dashed lines depict upper and lower quartile, and dotted gray lines depict all values.

7 Conclusion and Outlook

In this dissertation, we investigated distributed real-time systems. Specifically, we examined self-suspending tasks, which are common when jobs are executed in a distributed manner, and cause-effect chains, which are considered when a functionality is achieved by cooperating tasks. These examinations were conducted with a special focus on careful analysis based on fundamental properties of possible evolutions of the system. In this chapter, we conclude the dissertation by summarizing our contributions in Section 7.1. Afterwards we give an outlook on potential future work and research direction to be further explored in Section 7.2.

7.1 Summary of Contributions

To allow for careful, property-based analysis, in this dissertation we introduced scheduling concepts in a bottom-up strategy in Chapter 2. Specifically, while the classical top-down approach concretizes task models only if necessary, in this dissertation we started by considering possible system evolutions and abstracted task properties if they were utilized in the analysis. This approach is more sound than the top-down strategy, which led to a series of flawed results and missing or unnecessary assumptions. We emphasized the need for more careful, property-based analysis by providing three counterexamples for well-established literature results in Chapter 4. The first two counterexamples close the unresolved issues of the review paper on self-suspending tasks by Chen et al. [Che+19b], concluding their investigation on the literature of self-suspending tasks before 2019. The third counterexample disproves the classical extension of the Critical Instant Theorem (CIT) to probabilistic scenarios. However, we also provided two safe approaches to extend the CIT, namely carry-in and inflation, that were used as a backbone for further exploration [MNP22; Mar+23].

In Chapter 5, we investigated the timing behavior of self-suspending tasks. We developed novel sufficient schedulability analyses for dynamic self-suspending tasks under Task-level Fixed-Priority (T-FP), Earliest-Deadline-First (EDF) and EDF-Like (EL) scheduling. Moreover, we enhanced an analysis framework for schedulability tests of self-suspending tasks. We also examined timing anomalies under Segment-level Fixed-Priority (S-FP) scheduling and developed two mechanisms to avoid timing anomalies, namely segment release time enforcement and segment priority modification. While the presented analytical results improve the state of the art (in performance or applicability), preventing timing anomalies enables exact analysis of the Worst-Case Response Time (WCRT) by simulation of a single system evolution.

In Chapter 6, we investigated the end-to-end latency of cause-effect chains. First, we examined fundamental properties, namely a compositional property and the equivalence of Maximum Reaction Time (MRT) and Maximum Data Age (MDA). Second, we developed a series of analyses for the worst-case end-to-end latency. To that end, we uncovered principles of current end-to-end latency bounds, explored new analytical approaches using abstract integer representations and partitioned job chains, and applied the fundamental properties. Third, we examined probabilistic guarantees on the end-to-end latency. Specifically, we defined and analyzed probabilistic reaction time, and discussed potential pitfalls of the definition.

7.2 Future Work

While this dissertation contributes as summarized above, there are still open problems to be solved as detailed in Sections 5.3 and 6.4. Specifically, future work on self-suspending tasks should explore alternative self-suspension models and necessary schedulability tests. Furthermore, the examination on schedulability tests and timing anomalies should be generalized, e.g., towards multiprocessor systems. Regarding end-to-end latency of cause-effect chains, generalization of results to different triggering mechanisms and further exploration of probabilistic end-to-end latency would be useful from the industry perspective. Furthermore, specification of different timing requirements and system design to meet such requirements are challenging but essential problems.

With respect to the examination of distributed real-time systems, this dissertation focuses on specific aspects of the two proposed challenges: distributed execution of jobs, and interplay of distributed tasks. However, there are different aspects of these challenges to be further explored. Specifically, while we focus on self-suspending tasks, there are different task models like Directed Acyclic Graph (DAG) or Event-Driven Delay-Induced (EDD) that should be considered as well. For the interplay of distributed tasks, we considered the end-to-end latency assuming that communication is conducted through a shared resource with negligible time overhead or that it can be modeled by a communication task. The impact of different communication designs on the analytical end-to-end latency and their integration into existing analyses should be further discussed. Besides further examination of the two proposed challenges, we identify the following three research directions to be further explored:

- **Sound analysis:** Although in this dissertation we call for careful, property-based analysis, it is unclear how flawed results can be prevented reliably in future work. While we believe that the property-based bottom-up view on real-time systems discussed in this dissertation can be utilized in future work to develop more error-proof analyses even outside the context of self-suspending tasks and cause-effect chains, this does not solve the overall problem. Recently, proof assistants like Coq have gained traction in the real-time systems community [Bed+22; Boz+23; CSB16; Mar+23]. Such proof assistants are helpful, but only if used correctly. Specifically, they do not prevent flaws coming from errors during the modeling phase or missing assumptions in the form of incomplete axioms. Therefore, even results utilizing proof assistants need to be carefully checked in the review process. Besides prevention of flawed results, also detection of flawed results in the form of systematic generation of counterexamples would be desirable. However, how and if this can be achieved is an open problem.

- **Robustness against perturbations:** When analyzing a real-time system, the system needs to be transferred to a model. To control the analytical complexity, the model is usually a simplification of the real-world behavior. For example, overheads for scheduling or preemption and small delays within DAG tasks are neglected for the sake of simplicity. While such simplifications are usually not a problem, they can lead to rare unforeseen events, such as significant longer execution times, blocking or even job failures. These can then lead to a series of deadline misses if not treated correctly. Since such perturbations are usually unavoidable, it would be valuable to design systems that can handle perturbations with minor impact on schedulability, and to make analysis aware of perturbations. Specifically, future work should tackle the questions: How robust are existing analyses with respect to small perturbations? How do we design more robust analytical approaches? While there are some approaches in the literature to make analyses more robust (consideration of jitter, blocking and specific overheads) and to design robust systems against rare perturbations (e.g., mixed criticality), these are only limited yet and a unified, generalized view is missing.

- **Probabilistic analysis:** While classical results analyze hard real-time systems, soft real-time systems are common in industry as well. Therefore, a series of results regarding probabilistic behavior in the standard task model has been published recently [BBB21; Brü+21; Che+22a; DC19; Fri+23; Gün+23c; Mar+18; Mar+23]. While this dissertation contributes to this subject by examining extensions of the CIT to probabilistic scenarios and by analyzing the probabilistic reaction time, there are still open problems to be pursued. Specifically, while the current analyses focus on the standard task model, more complicated task models, like the EDD, DAG or self-suspension task model are still to be analyzed. Furthermore, probabilistic behavior in different parameters like in the release pattern, and removing iid assumptions of random variables are immediate problems to be solved.

While the overall problem of distributed real-time systems and providing careful property-based analysis remains challenging, this dissertation provides encouraging advancements and identifies promising open questions and research directions to be explored further.

A Mathematical Foundation

In this chapter, we detail mathematical notation and conventions that are used throughout this dissertation.

Classification of numbers The natural numbers are denoted by $\mathbb{N} = \{1, 2, 3, \ldots\}$. Please note that we follow the mathematical convention to not include 0 in $\mathbb{N}$. If 0 is to be considered in the set of natural numbers, we use $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$. The real numbers are denoted by $\mathbb{R}$. We use the notation $\mathbb{R}_{>0}$ and $\mathbb{R}_{\geq 0}$ to restrict to positive and non-negative real numbers, respectively. Specifically, $\mathbb{R}_{>0} = \{x \in \mathbb{R} \mid x > 0\}$ and $\mathbb{R}_{\geq 0} = \{x \in \mathbb{R} \mid x \geq 0\}$. If not stated otherwise, intervals $[a, b]$ with $a, b \in \mathbb{R}$ live in the space of real numbers, i.e., $[a, b] \subseteq \mathbb{R}$.

Set operations Given two sets A and B, we use $A \cup B$ and $A \cap B$ to denote their union and intersection, respectively. If we write $A \sqcup B$ instead of $A \cup B$, then the union of A and B is disjoint, i.e., $A \cap B = \emptyset$. The number of elements in a set A is denoted by $|A| \in \mathbb{N}_0 \cup \{\infty\}$. Moreover, we denote by $\mathcal{P}(A) = \{B \mid B \subseteq A\}$ the power set of A. Given a subset $A \subseteq \mathbb{R}$ of the real numbers, then we define the supremum of A as the lowest upper bound on A, denoted as $\sup A$. More specifically, let $U = \{u \in \mathbb{R} \cup \{+\infty, -\infty\} \mid u \geq a \ \forall a \in A\}$ be the set of upper bounds on A, then $\sup A \in U$ such that $u \geq \sup A$ for all $u \in U$. If the supremum is part of A, i.e., $\sup A \in A$, then we also call it the maximum of A, denoted by $\max A$. Similarly, given a subset $A \subseteq \mathbb{R}$ of the real numbers, then we define the infimum of A as the highest lower bound on A, denoted as $\inf A$. More specifically, let $L = \{\ell \in \mathbb{R} \cup \{+\infty, -\infty\} \mid \ell \leq a \ \forall a \in A\}$ be the set of lower bounds on A, then $\inf A \in L$ such that $\ell \leq \inf A$ for all $\ell \in L$. If the infimum is part of A, i.e., $\inf A \in A$, then we also call it the minimum of A, denoted by $\min A$.

Functions A function $f\colon A \to B$ of sets A and B assigns each element $a \in A$ to exactly one element $f(a) \in B$. The function is called *injective* if no two elements of A are mapped to the same element in B, i.e., $f(a) \neq f(a')$ for all $a \neq a' \in A$. The function is called *surjective* if it maps at least one element in A to each element in B, i.e., for all $b \in B$ there exists $a \in A$ such that $f(a) = b$. If f is injective and surjective, we call it *bijective*. For a bijective function f, its inverse function is $f^{-1}\colon B \to A$ mapping each $b \in B$ to the unique value $a \in A$ such that $f(a) = b$. We follow the mathematical convention to overload the notation f^{-1} to denote preimages of subsets of B. That is, we define $f^{-1}(B') := \{a \in A \mid f(a) \in B'\}$ for all subsets $B' \subseteq B$.

Measures Measures are used to quantify the size of objects. A typical example is the length of an interval $|[a, b]| = b - a$. Mathematically, this concept can be generalized to account for more complex subsets of measurable spaces (X, Σ). More specifically, given a set X and a σ-algebra $\Sigma \subseteq \mathcal{P}(X)$, then a function $\mu \colon \Sigma \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ is a measure if the following conditions hold:

- $\mu(\emptyset) = 0$
- σ-additivity: $\mu(\bigcup_{k\in\mathbb{N}} E_k) = \sum_{k\in\mathbb{N}} \mu(E_k)$ for any pairwise disjoint sets $(E_k)_{k\in\mathbb{N}}$ with $E_k \in \Sigma$

A classical example is the Lebesgue-measure λ on the space of real numbers $(\mathbb{R}, \mathfrak{L}(\mathbb{R}))$, that is utilized throughout this dissertation. Specifically, the Lebesgue-measure $\lambda(E)$ for $E \in \mathfrak{L} \subseteq \mathcal{P}(\mathbb{R})$ is defined as $\lambda(E) := \inf \left\{\sum_{k\in\mathbb{N}} |E_k| \,\middle|\, E \subseteq \bigcup_{k\in\mathbb{N}} E_k, E_k \subseteq \mathbb{R} \text{ interval}\right\}$. We refer to the book by Tao for a detailed introduction to Lebesgue-measures [Tao22]

Probability Probability is a mathematical model to quantify how likely events occur. This model is given in the form of a probability space $(\Omega, \mathfrak{F}, \mathbb{P})$, where Ω is the sample space, describing possible outcomes, $\mathfrak{F} \subseteq \mathcal{P}(\Omega)$ is the event space, describing events to be observed, and $\mathbb{P} \colon \mathfrak{F} \to [0, 1]$ defines the probability of events. Formally, $(\Omega, \mathfrak{F})$ is a measurable space, and $\mathbb{P}$ defines a measure on $(\Omega, \mathfrak{F})$, the so-called probablility measure. A random variable X is a function $X \colon \Omega \to \mathbb{R}$. The probability that the random variable X does not exceed a threshold $t \in \mathbb{R}$ is defined as $\mathbb{P}(X \leq t) := \mathbb{P}(\{\omega \in \Omega \mid X(w) \leq t\})$. Similarly, the probability that X exceeds t is $\mathbb{P}(X > t) := 1 - \mathbb{P}(X \leq t)$. The function $F_X := \mathbb{P}(X \leq \bullet) \colon \mathbb{R} \to [0, 1]$ is the cumulative distribution function of X. Two random variables X and Y are independent if their joint probability is equal to the product of their probabilities, i.e., $\mathbb{P}(X \leq t_1, Y \leq t_2) = \mathbb{P}(X \leq t_1) \cdot \mathbb{P}(Y \leq t_2)$ for all $t_1, t_2 \in \mathbb{R}$. If two independent random variables X and Y have the same cumulative distribution function, then we say that X and Y are independent and identically distributed (iid). For more detailed discussion on probability theory, we refer to the book from Borovkov [Bor99].

Directed Acyclic Graphs (DAGs) A directed graph (V, E) is defined by a set of vertices V and a set of directed edges $E \subseteq V \times V$. A path p in (V, E) is a sequence of edges, i.e., $p = (e_1, \ldots, e_k)$ with $k \in \mathbb{N}$, such that there exists a sequence of vertices $(v_1, \ldots, v_{k+1})$ with $e_i = (v_i, v_{i+1})$ for all $i \in \{1, \ldots, k\}$. A path forms a cycle if its start vertex and end vertex coincide, i.e., if $v_1 = v_{k+1}$. We call the directed graph (V, E) acyclic, if there exists no path that forms a cycle. A graph isomorphism $f \colon (V_1, E_1) \xrightarrow{\sim} (V_2, E_2)$ is a bijective function on the vertices $f \colon V_1 \to V_2$ that preserves the graph structure, i.e., $f \times f \colon E_1 \xrightarrow{\sim} E_2$ is a bijection. Two directed graphs (V_1, E_1) and (V_2, E_2) are isomorphic (denoted by $(V_1, E_1) \simeq (V_2, E_2)$) if there exists a graph isomorphism $f \colon (V_1, E_1) \xrightarrow{\sim} (V_2, E_2)$. If $(V_1, E_1) \simeq (V_2, E_2)$ and (V_1, E_1) is acyclic, then (V_2, E_2) is acyclic as well. A sequence can be considered as DAG isomorphic to $(\{1, 2, \ldots, m\}, \{(1, 2), (2, 3), \ldots, (n-1, m)\})$. Sometimes, in this dissertation, we denote sequence by $(1 \to 2 \to \cdots \to m)$, especially in the context of cause-effect chains (cf. Section 3.2) to follow the convention in that research area.

B

Appendix for Chapter 4

B.1 Appendix for Section 4.1

The following analysis consists of two parts. First, we derive the Worst-Case Response Time (WCRT) of τ_3 as foundation for the response-time analysis of τ_4. Second, we provide a bound on the response time of τ_4 which is sufficient for the counterexample in Section 4.1.

Response time of τ_3: To analyze the WCRT $R^{\mathcal{A}}_{\tau_3}$ of task τ_3, we consider the suspension-oblivious schedule where suspension is replaced by computation. In that case, we can utilize Time-Demand Analysis (TDA), as explained in Section 2.5. Using the time-demand function W_{τ_3} for this case yields a WCRT of $W_{\tau_3}(15+\delta) = (2+\delta) + \left\lceil \frac{15+\delta}{7} \right\rceil 1 + \left\lceil \frac{15+\delta}{24} \right\rceil 10 = 15+\delta$. This also bounds the WCRT of τ_3 in the case with suspension, i.e., $R^{\mathcal{A}}_{\tau_3} \leq 15+\delta$. The schedule in Figure B.1 shows a case where the response time is actually $15+\delta$. We conclude that $R^{\mathcal{A}}_{\tau_3} = 15+\delta$.

Response time of τ_4: To analyze the WCRT of τ_4, we consider a concrete schedule $[\mathcal{S}]$ of a system evolution ω. Suppose that the first job $\tau_4(1)$ of τ_4 is released at time $r^{\omega}_{\tau_4(1)}$ and finished at time $f^{[\mathcal{S}],\omega}_{\tau_4(1)}$. We bound the response time of $\tau_4(1)$ and prove that $f^{[\mathcal{S}],\omega}_{\tau_4(1)} - r^{\omega}_{\tau_4(1)} \leq T_{\tau_4}$. When this property holds, we can remove the first job of τ_4 in the schedule and use the same argument to bound the response time of every job of τ_4 inductively.

Let $t_0 \leq r^{\omega}_{\tau_4(1)}$ be the time point in $\mathbb{R}$ such that the schedule is busy from t_0 to $r^{\omega}_{\tau_4(1)}$ and the processor idles right prior to t_0. Since the job of τ_4 released at time $r^{\omega}_{\tau_4(1)}$ is not constrained by the inter-arrival time constraint T_{τ_4} of τ_4, we can move its release time to t_0. After this change of arrival time, the schedule remains unchanged, but the response time of the job $\tau_4(1)$ is increased. For notational brevity, we set t_0 to 0 in this proof.

As a fundamental tool for our analysis we use the time-demand function on each computation segment of τ_4. For a segment $B \in \mathbb{B}_{\tau_4(1)}$ of $\tau_4(1)$ let I be the interference of τ_3 during the segment. We define the time-demand function for that segment by

$$W^*_{\tau_4}(t, I) := 2 + \left\lceil \frac{t}{7} \right\rceil 1 + \left\lceil \frac{t}{24} \right\rceil 10 + I. \tag{B.1}$$

If there exists some $t \in [0, T_{\tau_4}]$ with $W^*_{\tau_4}(t, I) \leq t$, this is an upper bound on the response

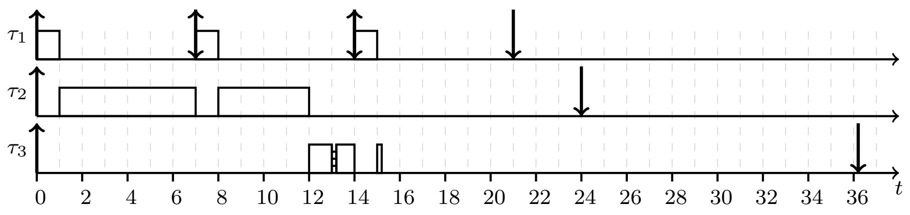


Figure B.1.: Worst-case schedule for the response time of τ_1, τ_2 and τ_3.

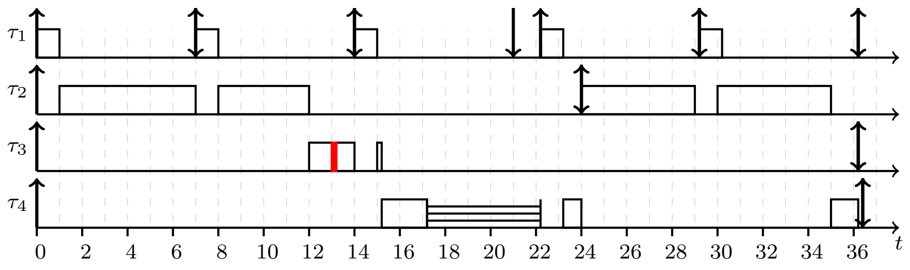


Figure B.2.: Task-level Fixed-Priority (T-FP) schedule of $\mathbb{T}$ for achieving an upper bound on the WCRT of τ_4 by replacing suspension of τ_3 by execution (in red).

time $R_B^{\mathcal{A}}(I)$ of that segment, i.e., $R_B^{\mathcal{A}}(I) \leq t$. Please note that in this notation of the response time we indicate the dependence on I.

To derive an upper bound on the WCRT of τ_4 which is sufficient for the counterexample, we fix the releases of the job segments of τ_4 and replace the suspension in τ_3 by execution, as depicted in Figure B.2. This conversion does not decrease the response time of $\tau_4(1)$. We call the new task τ_3^{obl} the suspension-oblivious τ_3 with Worst-Case Execution Time (WCET) $2 + \delta$. If there is some busy-interval $[x, 0]$ before 0 (choose the smallest x possible), then we move the release of $\tau_4(1)$ to x. This does not change the schedule and only increases the response time of $\tau_4(1)$. Moreover, after this procedure only jobs which are released at or after the release of $\tau_4(1)$ can interfere with $\tau_4(1)$. Therefore, we delete all jobs released before the release of $\tau_4(1)$ without changing the response time of $\tau_4(1)$.

The remaining analysis is to analyze the WCRT of τ_4 under the interference of three ordinary sporadic tasks $\tau_1, \tau_2, \tau_3^{obl}$, which can be achieved by adopting the response-time analysis in [Nel+15]. We use the time-demand function from Equation (B.1) on each segment of $\tau_4(1)$. If a job of τ_3 interferes with a segment B of $\tau_4(1)$, then the WCRT of that segment is

$$R_B^{\mathcal{A}}(2+\delta) \leq 17 + \delta \tag{B.2}$$

since $W_{\tau_4}^*(17+\delta, 2+\delta) = 17 + \delta$. If no job of τ_3 interferes with the segment B, then its WCRT is

$$R_B^{\mathcal{A}}(0) \leq 14 \tag{B.3}$$

since $W_{\tau_4}^*(14, 0) = 14$. If no job of τ_3 interferes with $\tau_4(1)$, then $\tau_4(1)$ is finished after at most $R_B^{\mathcal{A}}(0) + 5 + R_B^{\mathcal{A}}(0) \leq 33$ time units. If only the first job of τ_3 interferes with $\tau_4(1)$, then the total WCRT is at most $R_B^{\mathcal{A}}(2+\delta) + 5 + R_B^{\mathcal{A}}(0) \leq 36 + \delta$. We note that the second job of τ_3 cannot interfere with $\tau_4(1)$ since it is released when $\tau_4(1)$ is already finished.

B.2 Appendix for Section 4.3

Proof of Theorem 4.11. By Definition 4.6, the Worst-Case Response Time Exceedance Probability (WCRTEP) for the target value R is given by $\sup_{j\in\mathbb{N}}\{\mathbb{P}(R^{\mathcal{A}(\bullet),\bullet}_{\tau(j)} > R)\}$. In the following, we show that

$$\mathbb{P}(R^{\mathcal{A}(\bullet),\bullet}_{\tau(j)} > R) \overset{!}{\leq} \inf_{0<t\leq R} \mathbb{P}(S_t > t) \tag{B.4}$$

for all $\tau(j)$ with $j \in \mathbb{N}$. To achieve this, we consider one arbitrary job $\tau(\ell)$ of τ and one arbitrary but fixed release pattern, and show that $\mathbb{P}(R^{\mathcal{A}(\bullet),\bullet}_{\tau(\ell)} > R) \leq \inf_{0<t\leq R}\mathbb{P}(S_t > t)$ under this release pattern.

Equivalent Formulation: The following equivalence holds: $R^{\mathcal{A}(\omega),\omega}_{\tau(\ell)} > R$ if and only if $R^{\mathcal{A}(\omega),\omega}_{\tau(\ell)} > t$ for all $t \in (0, R]$. Hence,

$$\mathbb{P}(R^{\mathcal{A}(\bullet),\bullet}_{\tau(\ell)} > R) = \inf_{0<t\leq R} \mathbb{P}(R^{\mathcal{A}(\bullet),\bullet}_{\tau(\ell)} > t). \tag{B.5}$$

If the job $\tau(\ell)$ is not finished by time $r^{\omega}_{\tau(\ell)} + t$, then from $r^{\omega}_{\tau(\ell)}$ to $r^{\omega}_{\tau(\ell)} + t$ the processor either executes $\tau(\ell)$ or higher-priority jobs of the tasks in $hp(\tau)$ due to preemptive Task-level Fixed-Priority (T-FP) scheduling. Therefore, we know that $R^{\mathcal{A}(\omega),\omega}_{\tau(\ell)} > t$ if and only if $\tau(\ell)$ cannot be executed for $\bar{c}^{\omega}_{\tau(\ell)}$ time units in the interval $[r^{\omega}_{\tau(\ell)}, r^{\omega}_{\tau(\ell)} + t)$, i.e.,

$$\bar{c}^{\omega}_{\tau(\ell)} + \sum_{\tau'\in hp(\tau)} \mathrm{exec}^{\mathcal{A}(\omega)}_{\tau'}(r^{\omega}_{\tau(\ell)}, r^{\omega}_{\tau(\ell)} + t) > t. \tag{B.6}$$

We conclude the equivalent formulation for $\mathbb{P}(R^{\mathcal{A}(\bullet),\bullet}_{\tau(\ell)} > R)$:

$$\mathbb{P}(R^{\mathcal{A}(\bullet),\bullet}_{\tau(\ell)} > R) \quad = \inf_{0<t\leq R} \mathbb{P}\left(\bar{c}^{\bullet}_{\tau(\ell)} + \sum_{\tau'\in hp(\tau)} \mathrm{exec}^{\mathcal{A}(\bullet)}_{\tau'}(r^{\bullet}_{\tau(\ell)}, r^{\bullet}_{\tau(\ell)} + t) > t\right) \tag{B.7}$$

Quantification of Interference: In this paragraph, we provide an upper bound on $\mathrm{exec}^{\mathcal{A}(\omega)}_{\tau'}(r^{\omega}_{\tau(\ell)}, r^{\omega}_{\tau(\ell)} + t)$. To that end, we observe that any job of τ' that is released before $r^{\omega}_{\tau(\ell)} - D_{\tau'}$ is either aborted or finished at time $r^{\omega}_{\tau(\ell)}$. Therefore, only jobs of τ' that are released in $[r^{\omega}_{\tau(\ell)} - D_{\tau'}, r^{\omega}_{\tau(\ell)} + t)$ can be executed in $[r^{\omega}_{\tau(\ell)}, r^{\omega}_{\tau(\ell)} + t)$. The number of jobs of a sporadic task τ' released inside an interval of length $t + D_{\tau'}$ is at most $\left\lceil \frac{t+D_{\tau'}}{T_{\tau'}} \right\rceil$. Therefore, at most $\left\lceil \frac{t+D_{\tau'}}{T_{\tau'}} \right\rceil$ many jobs of τ' can be executed in $[r^{\omega}_{\tau(\ell)}, r^{\omega}_{\tau(\ell)} + t)$.

Let $\tau'(j_{\tau'})$ be the first job of τ' released at or after $r^{\omega}_{\tau(\ell)} - D_{\tau'}$.[1] Then only the jobs $\tau'(j_{\tau'}), \dots, \tau'(j_{\tau'} - 1 + \left\lceil \frac{t+D_{\tau'}}{T_{\tau'}} \right\rceil)$ of τ' can be executed in $[r^{\omega}_{\tau(\ell)}, r^{\omega}_{\tau(\ell)} + t)$. This means that

$$\operatorname{exec}^{\mathcal{A}(\omega)}_{\tau'}(r^{\omega}_{\tau(\ell)}, r^{\omega}_{\tau(\ell)} + t) \leq \sum_{q=j_{\tau'}}^{j_{\tau'}-1+\left\lceil \frac{t+D_{\tau'}}{T_{\tau'}} \right\rceil} \bar{c}^{\omega}_{\tau'(q)}. \tag{B.8}$$

We conclude that $\mathbb{P}(R^{\mathcal{A}(\bullet),\bullet}_{\tau(\ell)} > R)$ is upper bounded by

$$\inf_{0<t\leq R} \mathbb{P}\left(\bar{c}^{\bullet}_{\tau(\ell)} + \sum_{\tau'\in hp(\tau)} \sum_{q=j_{\tau'}}^{j_{\tau'}-1+\left\lceil \frac{t+D_{\tau'}}{T_{\tau'}} \right\rceil} \bar{c}^{\bullet}_{\tau'(q)} > t \right). \tag{B.9}$$

Exploit independent and identically distributed (iid): Since $\bar{c}^{\bullet}_{\tau(\ell)}$ has the same probability distribution as $\bar{c}^{\bullet}_{\tau(1)}$, we can replace $\bar{c}^{\bullet}_{\tau(\ell)}$ by $\bar{c}^{\bullet}_{\tau(1)}$ in Equation (B.9). Moreover, $\bar{c}^{\bullet}_{\tau'(j_{\tau'})}, \dots, \bar{c}^{\bullet}_{\tau'(j_{\tau'}-1+\left\lceil \frac{t+D_{\tau'}}{T_{\tau'}} \right\rceil)}$ and $\bar{c}^{\bullet}_{\tau'(1)}, \dots, \bar{c}^{\bullet}_{\tau'(\left\lceil \frac{t+D_{\tau'}}{T_{\tau'}} \right\rceil)}$ are iid with the same probability distribution $\mathcal{D}_{\tau'}$. Therefore, $\sum_{q=j_{\tau'}}^{j_{\tau'}-1+\left\lceil \frac{t+D_{\tau'}}{T_{\tau'}} \right\rceil} \bar{c}^{\bullet}_{\tau'(q)}$ and $\sum_{q=1}^{\left\lceil \frac{t+D_{\tau'}}{T_{\tau'}} \right\rceil} \bar{c}^{\bullet}_{\tau'(q)}$ have the same probability distribution for all $\tau' \in hp(\tau)$. Hence, the random variables $\bar{c}^{\bullet}_{\tau(\ell)} + \sum_{\tau'\in hp(\tau)} \sum_{q=j_{\tau'}}^{j_{\tau'}-1+\left\lceil \frac{t+D_{\tau'}}{T_{\tau'}} \right\rceil} \bar{c}^{\bullet}_{\tau'(q)}$ and $S_t = \bar{c}^{\bullet}_{\tau(1)} + \sum_{\tau'\in hp(\tau)} \sum_{q=1}^{\left\lceil \frac{t+D_{\tau'}}{T_{\tau'}} \right\rceil} \bar{c}^{\bullet}_{\tau'(q)}$ have the same probability distribution, and $\mathbb{P}(\bar{c}^{\bullet}_{\tau(\ell)} + \sum_{\tau'\in hp(\tau)} \sum_{q=j_{\tau'}}^{j_{\tau'}-1+\left\lceil \frac{t+D_{\tau'}}{T_{\tau'}} \right\rceil} \bar{c}^{\bullet}_{\tau'(q)} > t)$ coincides with $\mathbb{P}(S_t > t)$. We apply this to Equation (B.9) to obtain

$$\mathbb{P}(R^{\mathcal{A}(\bullet),\bullet}_{\tau(\ell)} > R) \leq \inf_{0<t\leq R} \mathbb{P}(S_t > t) \tag{B.10}$$

Closing remarks: We have shown that Equation (B.10) holds for an arbitrary release pattern, and therefore it holds for all possible release patterns. We can apply $\sup_{\ell\in\mathbb{N}}$ to obtain the statement from Theorem 4.11. $\square$

Proof of Theorem 4.14. The proof of this theorem is very similar to the proof of Theorem 4.11 by further exploiting the concept of *Interval Extension* and *Release Time Modification* used in the proof of the Critical Instant Theorem (CIT) in Theorem 2.21. Under the probabilistic setting, after the interval extension and release time modification, we show how to safely quantify the interference by using the concept of sample and inflate.

By Definition 4.6, the WCRTEP for R is given by $\sup_{j\in\mathbb{N}}\{\mathbb{P}(R^{\mathcal{A}(\bullet),\bullet}_{\tau(j)} > R)\}$. In the following we show that $\mathbb{P}(R^{\mathcal{A}(\bullet),\bullet}_{\tau(j)} > R)$ is upper bounded by Equation (4.18) for all $j \in \mathbb{N}$.

[1]Please note that $j_{\tau'}$ is only dependent on the release pattern and not on the probabilistic behavior.

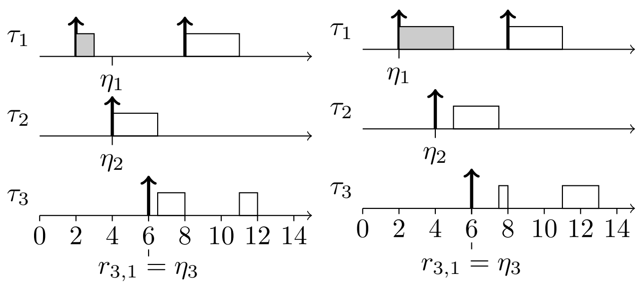


Figure B.3.: Interval extension in the proof of Theorem 4.14 for two different execution scenarios (left: $C_{1,1} = 1$, right: $C_{1,1} = 3$). We observe that the changes in the random variable η_1 are dependent on the sample under analysis.

To achieve this, we consider one arbitrary job $\tau(\ell)$ of τ and one arbitrary but fixed release pattern, and show that $\mathbb{P}(R^{\mathcal{A}(\bullet),\bullet}_{\tau(j)} > R)$ is upper bounded by Equation (4.18) under that release pattern.

Equivalent Formulation: As in the proof of Theorem 4.11, the probability for $R^{\mathcal{A}(\bullet),\bullet}_{\tau(j)} > R$ is equivalently formulated as

$$\mathbb{P}(R^{\mathcal{A}(\bullet),\bullet}_{\tau(\ell)} > R) = \inf_{0<t\leq R} \mathbb{P}\left(\bar{c}^{\bullet}_{\tau(\ell)} + \sum_{\tau' \in hp(\tau)} \text{exec}^{\mathcal{A}(\bullet)}_{\tau'}(r^{\bullet}_{\tau(\ell)}, r^{\bullet}_{\tau(\ell)} + t) > t \right). \tag{B.11}$$

Interval Extension: Similar to the proof of the WCRT in Theorem 2.21, we extend the analysis interval to the left-hand side. To that end, let $hp(\tau) = \{\tau_1, \dots, \tau_k\}$ with τ_i ordered by priority, i.e., τ_1 has the highest priority and τ_k has the lowest priority. We define a time point η_1 such that all jobs released before η_1 are not relevant for the analysis. More specifically, we construct the time points $\eta_1, \dots, \eta_k$ in an iterative manner, starting from $i = k, k-1, \dots, 1$. Let η_{k+1} be $r^{\omega}_{\tau(\ell)}$. After η_{i+1} is determined, we define η_i as follows:

$$\eta_i := \min(\eta_{i+1}, r^{\omega}_{\tau_i(\phi_i)}), \tag{B.12}$$

where $\tau_i(\phi_i)$ the first job of τ_i that is executed after η_{i+1}, i.e.,

$$\phi_i := \min\left\{ j \in \mathbb{N} \,\middle|\, \text{exec}^{\mathcal{A}(\omega)}_{\tau_i(j)}(\eta_{i+1}, \infty) > 0 \right\}. \tag{B.13}$$

In Figure B.3 the interval extension is presented for a task set with three tasks. Note that $\eta_1, \eta_2, \dots, \eta_k$ are random variables that depend on the execution behavior. With this definition of $\eta_1, \dots, \eta_{k+1}$ the following properties hold:

P1 During $[\eta_1, \eta_{i+1})$ the processor is busy executing jobs of $hep(\tau_i)$. (That is because if $\eta_i < \eta_{i+1}$, then the processor executes jobs of $hep(\tau_i)$ during $[\eta_i, \eta_{i+1})$.)

P2 None of the jobs of $\tau_i \in hp(\tau)$ released before time point η_i is executed after η_i. (If $\eta_i = \eta_{i+1}$, then there are by construction no jobs of τ_i released before η_{i+1} and executed after η_{i+1}. If $\eta_i = r^{\omega}_{\tau_i(\phi_i)}$, then under the constrained-deadline setup all previous jobs of τ_i have their deadline at η_i or earlier. Hence, they must be finished or aborted by η_i.)

P3 $\eta_i + D_i > \eta_{i+1}$ due to the assumption that a job is aborted after its deadline miss.

Please note that none of the jobs of $\tau_i \in hp(\tau)$ released before η_i is executed after η_1. (Otherwise, they have to finish by η_i because of Property P2. However, the jobs of τ_i cannot be executed in $[\eta_1, \eta_i)$ because of Property P1.) Therefore, jobs released before η_1 are not relevant for the analysis.

Release Time Modification: In the following we move the release of $\tau(\ell)$ to η_1 for the analysis, i.e., we show that

$$\sum_{\tau' \in hp(\tau)} \text{exec}^{\mathcal{A}(\omega)}_{\tau'}(r^{\omega}_{\tau(\ell)}, r^{\omega}_{\tau(\ell)} + t) \overset{!}{\leq} \sum_{\tau' \in hp(\tau)} \text{exec}^{\mathcal{A}(\omega)}_{\tau'}(\eta_1, \eta_1 + t). \tag{B.14}$$

According to Property P1, during $[\eta_1, r^{\omega}_{\tau(\ell)}) = [\eta_1, \eta_{k+1})$ the processor is busy executing jobs of tasks of $hep(\tau_k) = hp(\tau)$. Therefore, $\sum_{\tau' \in hp(\tau)} \text{exec}^{\mathcal{A}(\omega)}_{\tau'}(\eta_1, \eta_{k+1}) = \eta_{k+1} - \eta_1$ holds. Moreover, $\eta_{k+1} - \eta_1 \geq \sum_{\tau' \in hp(\tau)} \text{exec}^{\mathcal{A}(\omega)}_{\tau'}(\eta_1 + t, \eta_{k+1} + t)$ since the processor can only execute jobs for at most $\eta_{k+1} - \eta_1$ time units in the interval $[\eta_1 + t, \eta_{k+1} + t)$. Hence,

$$\sum_{\tau' \in hp(\tau)} \text{exec}^{\mathcal{A}(\omega)}_{\tau'}(\eta_1, \eta_{k+1}) = \eta_{k+1} - \eta_1 \geq \sum_{\tau' \in hp(\tau)} \text{exec}^{\mathcal{A}(\omega)}_{\tau'}(\eta_1 + t, \eta_{k+1} + t) \tag{B.15}$$

holds. Since by Equation (B.15) $\sum_{\tau' \in hp(\tau)} \text{exec}^{\mathcal{A}(\omega)}_{\tau'}(\eta_1, \eta_{k+1}) - \sum_{\tau' \in hp(\tau)} \text{exec}^{\mathcal{A}(\omega)}_{\tau'}(\eta_1 + t, \eta_{k+1} + t) \geq 0$, we obtain the following:

$$\sum_{\tau' \in hp(\tau)} \text{exec}^{\mathcal{A}(\omega)}_{\tau'}(r^{\omega}_{\tau(\ell)}, r^{\omega}_{\tau(\ell)} + t) = \sum_{\tau' \in hp(\tau)} \text{exec}^{\mathcal{A}(\omega)}_{\tau'}(\eta_{k+1}, \eta_{k+1} + t) \tag{B.16}$$

$$\leq \left(\begin{array}{l} \sum_{\tau' \in hp(\tau)} \text{exec}^{\mathcal{A}(\omega)}_{\tau'}(\eta_{k+1}, \eta_{k+1} + t) + \sum_{\tau' \in hp(\tau)} \text{exec}^{\mathcal{A}(\omega)}_{\tau'}(\eta_1, \eta_{k+1}) \\ - \sum_{\tau' \in hp(\tau)} \text{exec}^{\mathcal{A}(\omega)}_{\tau'}(\eta_1 + t, \eta_{k+1} + t) \end{array} \right) \tag{B.17}$$

$$= \sum_{\tau' \in hp(\tau)} \text{exec}^{\mathcal{A}(\omega)}_{\tau'}(\eta_{k+1}, \eta_{k+1} + t) + \text{exec}^{\mathcal{A}(\omega)}_{\tau'}(\eta_1, \eta_{k+1}) - \text{exec}^{\mathcal{A}(\omega)}_{\tau'}(\eta_1 + t, \eta_{k+1} + t) \tag{B.18}$$

$$= \sum_{\tau' \in hp(\tau)} \text{exec}^{\mathcal{A}(\omega)}_{\tau'}(\eta_1, \eta_{k+1} + t) - \text{exec}^{\mathcal{A}(\omega)}_{\tau'}(\eta_1 + t, \eta_{k+1} + t) \tag{B.19}$$

$$= \sum_{\tau' \in hp(\tau)} \text{exec}^{\mathcal{A}(\omega)}_{\tau'}(\eta_1, \eta_1 + t) \tag{B.20}$$

Note that the conversion from (B.18) to (B.19) and from (B.19) to (B.20) are both due to the fact that $\text{exec}^{\mathcal{A}(\omega)}_{\tau'}(a, b) + \text{exec}^{\mathcal{A}(\omega)}_{\tau'}(b, c) = \text{exec}^{\mathcal{A}(\omega)}_{\tau'}(a, c)$ for all real numbers $a \leq b \leq c$. This proves Equation (B.14).

Quantification of Interference: In the following we quantify the interference from higher-priority task $\tau' \in hp(\tau)$, which is $\text{exec}_{\tau'}^{\mathcal{A}(\omega)}(\eta_1, \eta_1 + t)$. For all $\tau_i \in hp(\tau)$, during $[\eta_1, \eta_i)$ only jobs of $hp(\tau_i)$ are executed by Property P1. Therefore, $\text{exec}_{\tau_i}^{\mathcal{A}(\omega)}(\eta_1, \eta_i) = 0$ and

$$\text{exec}_{\tau_i}^{\mathcal{A}(\omega)}(\eta_1, \eta_1 + t) \leq \text{exec}_{\tau_i}^{\mathcal{A}(\omega)}(\eta_1, \eta_i + t) = \text{exec}_{\tau_i}^{\mathcal{A}(\omega)}(\eta_i, \eta_i + t). \tag{B.21}$$

By Property P2 only those jobs of τ_i that are released in $[\eta_i, \eta_i + t)$ can be executed in $[\eta_i, \eta_i + t)$. These are at most $\left\lceil \frac{t}{T_{\tau_i}} \right\rceil$ jobs. However, since η_i is a random variable dependent on the execution behavior, it is not clear *which* jobs have to be included in the analysis.[2]

In the following we derive a set $\mathbb{J}^i$ of all jobs that may be released in $[\eta_i, \eta_i + t)$ under any execution behavior, i.e., the set is only dependent on the release pattern. By Property P3, η_i is lower bounded by $r_{\tau(\ell)}^{\omega} - \sum_{\tau'' \in hp(\tau) \setminus hp(\tau_i)} D_{\tau''}$. Hence, the interval $[\eta_i, \eta_i + t)$ is a subset of the interval $[r_{\tau(\ell)}^{\omega} - \sum_{\tau'' \in hp(\tau) \setminus hp(\tau_i)} D_{\tau_i}, r_{\tau(\ell)}^{\omega} + t)$. Let $\tau_i(j_{\tau_i})$ be the first job of τ_i that is released at or after $r_{\tau(\ell)}^{\omega} - \sum_{\tau'' \in hp(\tau) \setminus hp(\tau_i)} D_{\tau''}$. Please note that j_{τ_i} is only dependent on the release pattern and independent of the execution behavior. Since at most $\lambda_{\tau_i}^t = \left\lceil \frac{t + \sum_{\tau'' \in hp(\tau) \setminus hp(\tau)} D_{\tau''}}{T_{\tau_i}} \right\rceil$ jobs of τ_i are released in $[r_{\tau(\ell)}^{\omega} - \sum_{\tau'' \in hp(\tau) \setminus hp(\tau_i)} D_{\tau''}, r_{\tau(\ell)}^{\omega} + t)$, only the jobs

$$\mathbb{J}^i = \left\{ \tau_i(j_{\tau_i}), \ldots, \tau_i(j_{\tau_i} - 1 + \lambda_{\tau_i}^t) \right\} \tag{B.22}$$

can be released in $[r_{\tau(\ell)}^{\omega} - \sum_{\tau'' \in hp(\tau) \setminus hp(\tau_i)} D_{\tau''}, r_{\tau(\ell)}^{\omega} + t)$. Hence, only jobs of $\mathbb{J}^i$ may be released in $[\eta_i, \eta_i + t)$.

As discussed earlier, by Property P2 only jobs of τ_i that are released in $[\eta_i, \eta_i + t)$ can be executed in $[\eta_i, \eta_i + t)$, i.e., at most $\left\lceil \frac{t}{T_{\tau_i}} \right\rceil$ jobs. Hence, only the execution time of $\left\lceil \frac{t}{T_{\tau_i}} \right\rceil$ jobs in $\mathbb{J}^i$ has to be considered for the interference. We conclude

$$\text{exec}_{\tau_i}^{\mathcal{A}(\omega)}(\eta_i, \eta_i + t) \leq MAX_i^t \tag{B.23}$$

where MAX_i^t is the random variable that returns the sum of the $\left\lceil \frac{t}{T_{\tau_i}} \right\rceil$ maximal execution time upper bounds of the jobs in $\mathbb{J}^i$. After the quantification step, we obtain that

$$\mathbb{P}(R_{\tau(\ell)}^{\mathcal{A}(\bullet), \bullet} > R) \leq \inf_{0 < t \leq R} \mathbb{P}\Big(\bar{c}_{\tau(\ell)}^{\bullet} + \sum_{\tau_i \in hp(\tau)} MAX_i^t \Big), \tag{B.24}$$

which is a combination of Equations (B.11), (B.14), (B.21) and (B.23).

Exploit iid: Since the random variable $\bar{c}_{\tau(\ell)}^{\bullet}$ has the same probability distribution as $\bar{c}_{\tau(1)}^{\bullet}$, we can replace $\bar{c}_{\tau(\ell)}^{\bullet}$ by $\bar{c}_{\tau(1)}^{\bullet}$ in Equation (B.24). Moreover, the random variables $\bar{c}_{\tau_i(j_{\tau_i})}^{\bullet}, \ldots, \bar{c}_{\tau_i(j_{\tau_i} - 1 + \lambda_{\tau_i}^t)}^{\bullet}$ and $\bar{c}_{\tau_i(1)}^{\bullet}, \ldots, \bar{c}_{\tau_i(\lambda_{\tau_i}^t)}^{\bullet}$ are iid. Therefore, MAX_i^t and

[2]Please note that in the proof of Theorem 4.11 the jobs $\tau'(j_{\tau'}), \ldots, \tau'(j_{\tau'} - 1 + \left\lceil \frac{t + D_{\tau'}}{T_{\tau'}} \right\rceil)$ in the analysis can be determined directly from the release pattern. In this proof, the release time modification, i.e., moving the analysis interval to η_i, is dependent on the execution behavior.

$SAI(\tau_i, \left\lceil \frac{t}{T_{\tau_i}} \right\rceil, \lambda^t_{\tau_i})$ have the same probability distribution for all $\tau_i \in hp(\tau)$. We conclude that $\bar{c}^\bullet_{\tau(\ell)} + \sum_{\tau_i \in hp(\tau)} MAX^t_i$ and $\bar{c}^\bullet_{\tau(1)} + \sum_{\tau_i \in hp(\tau)} SAI(\tau_i, \left\lceil \frac{t}{T_{\tau_i}} \right\rceil, \lambda^t_{\tau_i})$ have the same probability distribution, and therefore

$$\mathbb{P}(\bar{c}^\bullet_{\tau(\ell)} + \sum_{\tau_i \in hp(\tau)} MAX^t_i) = \mathbb{P}(\bar{c}^\bullet_{\tau(1)} + \sum_{\tau_i \in hp(\tau)} SAI(\tau_i, \left\lceil \frac{t}{T_{\tau_i}} \right\rceil, \lambda^t_{\tau_i})) \tag{B.25}$$

holds. Applying this to Equation (B.24) yields

$$\mathbb{P}(R^{\mathcal{A}(\bullet),\bullet}_{\tau(\ell)} > R) \leq \inf_{0<t\leq R} \mathbb{P}\left(\bar{c}^\bullet_{\tau(1)} + \sum_{\tau_i \in hp(\tau)} SAI\left(\tau_i, \left\lceil \frac{t}{T_{\tau_i}} \right\rceil, \lambda^t_{\tau_i} \right) > t \right). \tag{B.26}$$

Closing remarks: We have shown that Equation (B.26) holds for an arbitrary release pattern, and therefore it holds for all possible release patterns. We can apply $\sup_{\ell \in \mathbb{N}}$ to obtain the statement from Theorem 4.14. □

Bibliography

[Ake+20] Benny Akesson, Mitra Nasri, Geoffrey Nelissen, Sebastian Altmeyer, and Robert I. Davis. “An Empirical Survey-based Study into Industry Practice in Real-time Systems.” In: *RTSS*. IEEE, 2020, pp. 3–11.

[Ake+22] Benny Akesson, Mitra Nasri, Geoffrey Nelissen, Sebastian Altmeyer, and Robert I. Davis. “A comprehensive survey of industry practice in real-time systems.” In: *Real Time Syst.* 58.3 (2022), pp. 358–398.

[And00] Matthew Andrews. “Probabilistic End-to-End Delay Bounds for Earliest Deadline First Scheduling.” In: *INFOCOM*. IEEE Computer Society, 2000, pp. 603–612.

[ABN22] Federico Aromolo, Alessandro Biondi, and Geoffrey Nelissen. “Response-Time Analysis for Self-Suspending Tasks Under EDF Scheduling.” In: *ECRTS*. Vol. 231. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 13:1–13:18.

[Aro+21] Federico Aromolo, Alessandro Biondi, Geoffrey Nelissen, and Giorgio C. Buttazzo. “Event-Driven Delay-Induced Tasks: Model, Analysis, and Applications.” In: *RTAS*. IEEE, 2021, pp. 53–65.

[Aud01] Neil C. Audsley. “On priority assignment in fixed priority scheduling.” In: *Inf. Process. Lett.* 79.1 (2001), pp. 39–44.

[AB04] Neil C. Audsley and Konstantinos Bletsas. “Fixed Priority Timing Analysis of Real-Time Systems with Limited Parallelism.” In: *ECRTS*. IEEE Computer Society, 2004, pp. 231–238.

[Aud+93] Neil C. Audsley, Alan Burns, Mike M. Richardson, Ken Tindell, and Andy J. Wellings. “Applying new scheduling theory to static priority pre-emptive scheduling.” In: *Softw. Eng. J.* 8.5 (1993), pp. 284–292.

[AUT22] AUTOSAR. *Specification of Timing Extensions (AUTOSAR CP R22-11)*. https://www.autosar.org/fileadmin/standards/R22-11/CP/AUTOSAR_TPS_TimingExtensions.pdf. Accessed: 2024-01-18. 2022.

[BCS12] Hyoungbu Back, Hoon Sung Chwa, and Insik Shin. “Schedulability Analysis and Priority Assignment for Global Job-Level Fixed-Priority Multiprocessor Scheduling.” In: *IEEE Real-Time and Embedded Technology and Applications Symposium.* IEEE Computer Society, 2012, pp. 297–306.

[Bak03] Theodore P. Baker. “Multiprocessor EDF and Deadline Monotonic Schedulability Analysis.” In: *RTSS.* IEEE Computer Society, 2003, pp. 120–129.

[BBD11] Sanjoy K. Baruah, Alan Burns, and Robert I. Davis. “Response-Time Analysis for Mixed Criticality Systems.” In: *RTSS.* IEEE Computer Society, 2011, pp. 34–43.

[BMR90] Sanjoy K. Baruah, Aloysius K. Mok, and Louis E. Rosier. “Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor.” In: *RTSS.* IEEE Computer Society, 1990, pp. 182–190.

[BRH90] Sanjoy K. Baruah, Louis E. Rosier, and Rodney R. Howell. “Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic, Real-Time Tasks on One Processor.” In: *Real Time Syst.* 2.4 (1990), pp. 301–324.

[Bec+16a] Matthias Becker, Dakshina Dasari, Saad Mubeen, Moris Behnam, and Thomas Nolte. “MECHAniSer — A timing analysis and synthesis tool for multi-rate effect chains with job-level dependencies.” In: *Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS).* Vol. 20. 2016.

[Bec+16b] Matthias Becker, Dakshina Dasari, Saad Mubeen, Moris Behnam, and Thomas Nolte. “Synthesizing Job-Level Dependencies for Automotive Multi-rate Effect Chains.” In: *RTCSA.* IEEE Computer Society, 2016, pp. 159–169.

[Bec+17a] Matthias Becker, Dakshina Dasari, Saad Mubeen, Moris Behnam, and Thomas Nolte. “End-to-end timing analysis of cause-effect chains in automotive embedded systems.” In: *J. Syst. Archit.* 80 (2017), pp. 104–113.

[Bec+17b] Matthias Becker, Saad Mubeen, Dakshina Dasari, Moris Behnam, and Thomas Nolte. “A generic framework facilitating early analysis of data propagation delays in multi-rate systems (Invited paper).” In: *RTCSA.* IEEE Computer Society, 2017, pp. 1–11.

[Bed+22] Kimaya Bedarkar, Mariam Vardishvili, Sergey Bozhko, Marco Maida, and Björn B. Brandenburg. “From Intuition to Coq: A Case Study in Verified Response-Time Analysis 1 of FIFO Scheduling.” In: *RTSS.* IEEE, 2022, pp. 197–210.

[BCM19] Slim Ben-Amor, Liliana Cucu-Grosjean, and Dorin Maxim. “Worst-case response time analysis for partitioned fixed-priority DAG tasks on identical processors.” In: *ETFA.* IEEE, 2019, pp. 1423–1426.

[Ben+02] Albert Benveniste, Paul Caspi, Paul Le Guernic, Hervé Marchand, Jean-Pierre Talpin, and Stavros Tripakis. "A Protocol for Loosely Time-Triggered Architectures." In: *EMSOFT*. Vol. 2491. Lecture Notes in Computer Science. Springer, 2002, pp. 252–265.

[BC07] Marko Bertogna and Michele Cirinei. "Response-Time Analysis for Globally Scheduled Symmetric Multiprocessor Platforms." In: *RTSS*. IEEE Computer Society, 2007, pp. 149–160.

[Bi+22] Ran Bi, Xinbin Liu, Jiankang Ren, Pengfei Wang, Huawei Lv, and Guozhen Tan. "Efficient maximum data age analysis for cause-effect chains in automotive systems." In: *DAC*. ACM, 2022, pp. 1243–1248.

[BB05] Enrico Bini and Giorgio C. Buttazzo. "Measuring the Performance of Schedulability Tests." In: *Real Time Syst.* 30.1-2 (2005), pp. 129–154.

[BBB01] Enrico Bini, Giorgio C. Buttazzo, and Giuseppe M. Buttazzo. "A Hyperbolic Bound for the Rate Monotonic Algorithm." In: *ECRTS*. IEEE Computer Society, 2001, pp. 59–66.

[Bio+16] Alessandro Biondi, Alessio Balsini, Marco Pagani, Enrico Rossi, Mauro Marinoni, and Giorgio C. Buttazzo. "A Framework for Supporting Real-Time Applications on Dynamic Reconfigurable FPGAs." In: *RTSS*. IEEE Computer Society, 2016, pp. 1–12.

[BA05] Konstantinos Bletsas and Neil C. Audsley. "Extended Analysis with Reduced Pessimism for Systems with Limited Parallelism." In: *RTCSA*. IEEE Computer Society, 2005, pp. 525–531.

[Bon+13] Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, Sebastian Stiller, and Andreas Wiese. "Feasibility Analysis in the Sporadic DAG Task Model." In: *ECRTS*. IEEE Computer Society, 2013, pp. 225–233.

[Bor99] Aleksandr Alekseevich Borovkov. *Probability theory*. CRC Press, 1999.

[Bos91] Bosch. *Controller Area Network specification 2.0*. `http://esd.cs.ucr.edu/webres/can20.pdf`. 1991.

[BT01] Jean-Yves Le Boudec and Patrick Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Vol. 2050. Lecture Notes in Computer Science. Springer, 2001.

[BBB21] Sergey Bozhko, Georg von der Brüggen, and Björn B. Brandenburg. "Monte Carlo Response-Time Analysis." In: *RTSS*. IEEE, 2021, pp. 342–355.

[Boz+23] Sergey Bozhko, Filip Markovic, Georg von der Brüggen, and Björn B. Brandenburg. "What Really is pWCET? A Rigorous Axiomatic Proposal." In: *RTSS*. IEEE, 2023, pp. 13–26.

[Bra19] Björn B. Brandenburg. "Multiprocessor Real-Time Locking Protocols: A Systematic Review." In: *CoRR* abs/1909.09600 (2019).

[Bra22] Björn B. Brandenburg. “Multiprocessor Real-Time Locking Protocols.” In: *Handbook of Real-Time Computing*. Springer, 2022, pp. 347–446.

[Bri04] R.J. Bril. “Real-time scheduling for media processing using conditionally guaranteed budgets.” English. Phd Thesis 2 (Research NOT TU/e / Graduation TU/e). Frits Philips Inst. Quality Management, 2004. ISBN: 90-74445-62-4. DOI: 10.6100/IR580445.

[BLV09] Reinder J. Bril, Johan J. Lukkien, and Wim F. J. Verhaegh. “Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption.” In: *Real Time Syst.* 42.1-3 (2009), pp. 63–119.

[Brü+22] Georg von der Brüggen, Alan Burns, Jian-Jia Chen, Robert I. Davis, and Jan Reineke. “On the Trade-offs between Generalization and Specialization in Real-Time Systems.” In: *RTCSA*. IEEE, 2022, pp. 148–159.

[BHC17] Georg von der Brüggen, Wen-Hung Huang, and Jian-Jia Chen. “Hybrid self-suspension models in real-time embedded systems.” In: *RTCSA*. IEEE Computer Society, 2017, pp. 1–9.

[Brü+16] Georg von der Brüggen, Wen-Hung Huang, Jian-Jia Chen, and Cong Liu. “Uniprocessor Scheduling Strategies for Self-Suspending Task Systems.” In: *RTNS*. ACM, 2016, pp. 119–128.

[Brü+18] Georg von der Brüggen, Nico Piatkowski, Kuan-Hsun Chen, Jian-Jia Chen, and Katharina Morik. “Efficiently Approximating the Probability of Deadline Misses in Real-Time Systems.” In: *ECRTS*. Vol. 106. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 6:1–6:22.

[Brü+21] Georg von der Brüggen, Nico Piatkowski, Kuan-Hsun Chen, Jian-Jia Chen, Katharina Morik, and Björn B. Brandenburg. “Efficiently Approximating the Worst-Case Deadline Failure Probability Under EDF.” In: *RTSS*. IEEE, 2021, pp. 214–226.

[Brü+17] Georg von der Brüggen, Niklas Ueter, Jian-Jia Chen, and Matthias Freier. “Parametric utilization bounds for implicit-deadline periodic tasks in automotive systems.” In: *RTNS*. ACM, 2017, pp. 108–117.

[BW94] A. Burns and A. J. Wellings. “Implementing analysable hard real-time sporadic tasks in Ada 9X.” In: *Ada Lett.* (1994), pp. 38–49. DOI: 10.1145/181492.181495. URL: https://doi.org/10.1145/181492.181495.

[BBW10] Giorgio C. Buttazzo, Enrico Bini, and Yifan Wu. “Partitioning Parallel Applications on Multiprocessor Reservations.” In: *ECRTS*. IEEE Computer Society, 2010, pp. 24–33.

[Cas+18] Daniel Casini, Alessandro Biondi, Geoffrey Nelissen, and Giorgio C. Buttazzo. “Partitioned Fixed-Priority Scheduling of Parallel Tasks Without Preemptions.” In: *RTSS*. IEEE Computer Society, 2018, pp. 421–433.

[CSB16] Felipe Cerqueira, Felix Stutz, and Björn B. Brandenburg. "PROSA: A Case for Readable Mechanized Schedulability Analysis." In: *ECRTS*. IEEE Computer Society, 2016, pp. 273–284.

[Che16] Jian-Jia Chen. "Computational Complexity and Speedup Factors Analyses for Self-Suspending Tasks." In: *RTSS*. IEEE Computer Society, 2016, pp. 327–338.

[CA14] Jian-Jia Chen and Kunal Agrawal. "Capacity augmentation bounds for parallel DAG tasks under G-EDF and G-RM." In: *Faculty for Informatik, TU Dortmund, Dortmund, Germany, Tech. Rep* 845 (2014).

[CB17] Jian-Jia Chen and Björn B. Brandenburg. "A Note on the Period Enforcer Algorithm for Self-Suspending Tasks." In: *Leibniz Trans. Embed. Syst.* 4.1 (2017), 01:1–01:22.

[Che+17] Jian-Jia Chen, Georg von der Brüggen, Wen-Hung Huang, and Cong Liu. "State of the art for scheduling and analyzing self-suspending sporadic real-time tasks." In: *RTCSA*. IEEE Computer Society, 2017, pp. 1–10.

[Che+19a] Jian-Jia Chen, Tobias Hahn, Ruben Hoeksma, Nicole Megow, and Georg von der Brüggen. "Scheduling Self-Suspending Tasks: New and Old Results." In: *ECRTS*. Vol. 133. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 16:1–16:23.

[CL14] Jian-Jia Chen and Cong Liu. "Fixed-Relative-Deadline Scheduling of Hard Real-Time Tasks with Self-Suspensions." In: *RTSS*. IEEE Computer Society, 2014, pp. 149–160.

[CNH16] Jian-Jia Chen, Geoffrey Nelissen, and Wen-Hung Huang. "A Unifying Response Time Analysis Framework for Dynamic Self-Suspending Tasks." In: *ECRTS*. IEEE Computer Society, 2016, pp. 61–71.

[Che+19b] Jian-Jia Chen, Geoffrey Nelissen, Wen-Hung Huang, Maolin Yang, Björn B. Brandenburg, Konstantinos Bletsas, Cong Liu, Pascal Richard, Frédéric Ridouard, Neil C. Audsley, Raj Rajkumar, Dionisio de Niz, and Georg von der Brüggen. "Many suspensions, many problems: a review of self-suspending tasks in real-time systems." In: *Real Time Syst.* 55.1 (2019), pp. 144–207.

[CBC18] Kuan-Hsun Chen, Georg von der Brüggen, and Jian-Jia Chen. "Analysis of Deadline Miss Rates for Uniprocessor Fixed-Priority Scheduling." In: *RTCSA*. IEEE Computer Society, 2018, pp. 168–178.

[CC17] Kuan-Hsun Chen and Jian-Jia Chen. "Probabilistic schedulability tests for uniprocessor fixed-priority scheduling under soft errors." In: *SIES*. IEEE, 2017, pp. 1–8.

[Che+22a] Kuan-Hsun Chen, Mario Günzel, Georg von der Brüggen, and Jian-Jia Chen. "Critical Instant for Probabilistic Timing Guarantees: Refuted and Revisited." In: *RTSS*. IEEE, 2022, pp. 145–157.

[Che+22b] Kuan-Hsun Chen, Mario Günzel, Boguslaw Jablkowski, Markus Buschhoff, and Jian-Jia Chen. "Unikernel-Based Real-Time Virtualization Under Deferrable Servers: Analysis and Realization." In: *ECRTS*. Vol. 231. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 6:1–6:22.

[Che+19c] Kuan-Hsun Chen, Niklas Ueter, Georg von der Brüggen, and Jian-Jia Chen. "Efficient Computation of Deadline-Miss Probability and Potential Pitfalls." In: *DATE*. IEEE, 2019, pp. 896–901.

[CKK20] Hyunjong Choi, Mohsen Karimi, and Hyoseung Kim. "Chain-Based Fixed-Priority Scheduling of Loosely-Dependent Tasks." In: *ICCD*. IEEE, 2020, pp. 631–639.

[Cuc13] Liliana Cucu-Grosjean. "Independence-a misunderstood property of and for probabilistic real-time systems." In: *In Real-Time Systems: the past, the present and the future* (2013), pp. 29–37.

[Das+22] Dakshina Dasari, Matthias Becker, Daniel Casini, and Tobias Blaß. "End-to-End Analysis of Event Chains under the QNX Adaptive Partitioning Scheduler." In: *RTAS*. IEEE, 2022, pp. 214–227.

[Dav+07] Abhijit Davare, Qi Zhu, Marco Di Natale, Claudio Pinello, Sri Kanajan, and Alberto L. Sangiovanni-Vincentelli. "Period Optimization for Hard Real-time Distributed Automotive Systems." In: *DAC*. IEEE, 2007, pp. 278–283.

[DC19] Robert I. Davis and Liliana Cucu-Grosjean. "A Survey of Probabilistic Schedulability Analysis Techniques for Real-Time Systems." In: *Leibniz Trans. Embed. Syst.* 6.1 (2019), 04:1–04:53.

[Dev03] UmaMaheswari C. Devi. "An Improved Schedulability Test for Uniprocessor Periodic Task Systems." In: *ECRTS*. IEEE Computer Society, 2003, p. 23.

[Día+02] José Luis Díaz, Daniel F. García, Kanghee Kim, Chang-Gun Lee, Lucia Lo Bello, José María López, Sang Lyul Min, and Orazio Mirabella. "Stochastic Analysis of Periodic Real-Time Systems." In: *RTSS*. IEEE Computer Society, 2002, pp. 289–300.

[DL16] Zheng Dong and Cong Liu. "Closing the Loop for the Selective Conversion Approach: A Utilization-Based Test for Hard Real-Time Suspending Task Systems." In: *RTSS*. IEEE Computer Society, 2016, pp. 339–350.

[Don+18] Zheng Dong, Cong Liu, Soroush Bateni, Kuan-Hsun Chen, Jian-Jia Chen, Georg von der Brüggen, and Junjie Shi. "Shared-Resource-Centric Limited Preemptive Scheduling: A Comprehensive Study of Suspension-Based Partitioning Approaches." In: *RTAS*. IEEE Computer Society, 2018, pp. 164–176.

[Dür+19] Marco Dürr, Georg von der Brüggen, Kuan-Hsun Chen, and Jian-Jia Chen. "End-to-End Timing Analysis of Sporadic Cause-Effect Chains in Distributed Systems." In: *ACM Trans. Embed. Comput. Syst.* 18.5s (2019), 58:1–58:24.

[ESD10] Paul Emberson, Roger Stafford, and Robert I Davis. “Techniques for the synthesis of multiprocessor tasksets.” In: *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*. 2010, pp. 6–11.

[Erc19] Kayhan Erciyes. *Distributed Real-Time Systems - Theory and Practice*. Computer Communications and Networks. Springer, 2019.

[EAG18] Rolf Ernst, Leonie Ahrendts, and Kai Bjorn Gemlau. “System Level LET: Mastering Cause-Effect Chains in Distributed Systems.” In: *IECON*. IEEE, 2018, pp. 4084–4089.

[Ern+18] Rolf Ernst, Stefan Kuntz, Sophie Quinton, and Martin Simons. “The Logical Execution Time Paradigm: New Perspectives for Multicore Systems (Dagstuhl Seminar 18092).” In: *Dagstuhl Reports* 8.2 (2018), pp. 122–149.

[Fei+09] Nico Feiertag, Kai Richter, Johan Nordlander, and Jan Jonsson. “A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics.” In: *Workshop on Compositional Theory and Technology for Real-Time Embedded Systems*. 2009.

[Fid06] Markus Fidler. “An End-to-End Probabilistic Network Calculus with Moment Generating Functions.” In: *IWQoS*. IEEE, 2006, pp. 261–270.

[Fle05] FlexRay Consortium. *FlexRay Communications System-Protocol Specification*. 2005.

[FNN17] José Carlos Fonseca, Geoffrey Nelissen, and Vincent Nélis. “Improved response time analysis of sporadic DAG tasks for global FP scheduling.” In: *RTNS*. ACM, 2017, pp. 28–37.

[Fon+16] José Carlos Fonseca, Geoffrey Nelissen, Vincent Nélis, and Luís Miguel Pinho. “Response time analysis of sporadic DAG tasks under partitioned scheduling.” In: *SIES*. IEEE, 2016, pp. 290–299.

[FBP17] Julien Forget, Frédéric Boniol, and Claire Pagetti. “Verifying end-to-end real-time constraints on multi-periodic models.” In: *ETFA*. IEEE, 2017, pp. 1–8.

[Fri+23] Anna Friebe, Filip Markovic, Alessandro Vittorio Papadopoulos, and Thomas Nolte. “Continuous-Emission Markov Models for Real-Time Applications: Bounding Deadline Miss Probabilities.” In: *RTAS*. IEEE, 2023, pp. 14–26.

[GL99] Mark K. Gardner and Jane W.-S. Liu. “Analyzing Stochastic Fixed-Priority Real-Time Systems.” In: *TACAS*. Vol. 1579. Lecture Notes in Computer Science. Springer, 1999, pp. 44–58.

[Gir+18] Alain Girault, Christophe Prévot, Sophie Quinton, Rafik Henia, and Nicolas Sordon. “Improving and Estimating the Precision of Bounds on the Worst-Case Latency of Task Chains.” In: *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 37.11 (2018), pp. 2578–2589.

[GNV22] Pourya Gohari, Mitra Nasri, and Jeroen Voeten. “Data-Age Analysis for Multi-Rate Task Chains under Timing Uncertainty.” In: *RTNS*. ACM, 2022, pp. 24–35.

[Gra69] Ronald L. Graham. “Bounds on Multiprocessing Timing Anomalies.” In: *SIAM Journal of Applied Mathematics* 17.2 (1969), pp. 416–429.

[GBD20] David Griffin, Iain Bate, and Robert I. Davis. “Generating Utilization Vectors for the Systematic Evaluation of Schedulability Tests.” In: *RTSS*. IEEE, 2020, pp. 76–88.

[GBC20] Mario Günzel, Georg von der Brüggen, and Jian-Jia Chen. “Suspension-Aware Earliest-Deadline-First Scheduling Analysis.” In: *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 39.11 (2020), pp. 4205–4216.

[Gün+22] Mario Günzel, Georg von der Brüggen, Kuan-Hsun Chen, and Jian-Jia Chen. “EDF-Like Scheduling for Self-Suspending Real-Time Tasks.” In: *RTSS*. IEEE, 2022, pp. 172–184.

[GC20] Mario Günzel and Jian-Jia Chen. “Correspondence Article: Counterexample for suspension-aware schedulability analysis of EDF scheduling.” In: *Real Time Syst.* 56.4 (2020), pp. 490–493.

[GC21] Mario Günzel and Jian-Jia Chen. “A note on slack enforcement mechanisms for self-suspending tasks.” In: *Real Time Syst.* 57.4 (2021), pp. 387–396.

[Gün+21a] Mario Günzel, Kuan-Hsun Chen, Niklas Ueter, Georg von der Brüggen, Marco Dürr, and Jian-Jia Chen. “Timing Analysis of Asynchronized Distributed Cause-Effect Chains.” In: *RTAS*. IEEE, 2021, pp. 40–52.

[Gün+23a] Mario Günzel, Kuan-Hsun Chen, Niklas Ueter, Georg von der Brüggen, Marco Dürr, and Jian-Jia Chen. “Compositional Timing Analysis of Asynchronized Distributed Cause-effect Chains.” In: *ACM Trans. Embed. Comput. Syst.* 22.4 (2023), 63:1–63:34.

[Gün+24] Mario Günzel, Harun Teper, Georg von der Brüggen, and Jian-Jia Chen. “Data Flow from Cause to Effect in Distributed Systems: A Tutorial.” Work submitted to ACM Transactions on Embedded Computing Systems. 2024.

[Gün+21b] Mario Günzel, Harun Teper, Kuan-Hsun Chen, Georg von der Brüggen, and Jian-Jia Chen. “Work-in-Progress: Evaluation Framework for Self-Suspending Schedulability Tests.” In: *RTSS*. IEEE, 2021, pp. 532–535.

[Gün+23b] Mario Günzel, Harun Teper, Kuan-Hsun Chen, Georg von der Brüggen, and Jian-Jia Chen. “On the Equivalence of Maximum Reaction Time and Maximum Data Age for Cause-Effect Chains.” In: *ECRTS*. Vol. 262. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, 10:1–10:22.

[GUC21] Mario Günzel, Niklas Ueter, and Jian-Jia Chen. “Suspension-Aware Fixed-Priority Schedulability Test with Arbitrary Deadlines and Arrival Curves.” In: *RTSS*. IEEE, 2021, pp. 418–430.

[Gün+23c] Mario Günzel, Niklas Ueter, Kuan-Hsun Chen, Georg von der Brüggen, and Jian-Jia Chen. “Probabilistic Reaction Time Analysis.” In: *ACM Trans. Embed. Comput. Syst.* 22.5s (2023), 143:1–143:22.

[Gün+23d] Mario Günzel, Niklas Ueter, Kuan-Hsun Chen, and Jian-Jia Chen. “Timing Analysis of Cause-Effect Chains with Heterogeneous Communication Mechanisms.” In: *RTNS.* ACM, 2023, pp. 224–234.

[Gur21] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual.* 2021. URL: https://www.gurobi.com.

[GH05] José C. Palencia Gutiérrez and Michael González Harbour. “Response time analysis of EDF distributed real-time systems.” In: *J. Embed. Comput.* 1.2 (2005), pp. 225–237.

[Ham+17] Arne Hamann, Dakshina Dasari, Simon Kramer, Michael Pressler, and Falk Wurst. “Communication Centric Design in Complex Automotive Embedded Systems.” In: *ECRTS.* Vol. 76. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 10:1–10:20.

[Ham+19] Arne Hamann, Dakshina Dasari, Falk Wurst, Ignacio Sañudo, Nicola Capodieci, Paolo Burgio, and Marko Bertogna. “WATERS Industrial Challenge 2019.” In: *Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS).* 2019.

[He+22] Qingqiang He, Nan Guan, Mingsong Lv, Xu Jiang, and Wanli Chang. “Bounding the Response Time of DAG Tasks Using Long Paths.” In: *RTSS.* IEEE, 2022, pp. 474–486.

[Hen+05] Rafik Henia, Arne Hamann, Marek Jersak, Razvan Racu, Kai Richter, and Rolf Ernst. “System level performance analysis–the SymTA/S approach.” In: *IEE Proceedings-Computers and Digital Techniques* 152.2 (2005), pp. 148–166.

[Hob+22] Clara Hobbs, Bineet Ghosh, Shengjie Xu, Parasara Sridhar Duggirala, and Samarjit Chakraborty. “Safety Analysis of Embedded Controllers Under Implementation Platform Timing Uncertainties.” In: *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 41.11 (2022), pp. 4016–4027.

[HAE17] Robin Hofmann, Leonie Ahrendts, and Rolf Ernst. “CPA: Compositional Performance Analysis.” In: *Handbook of Hardware/Software Codesign.* Springer, 2017, pp. 721–751.

[HC15] Wen-Hung Huang and Jian-Jia Chen. *Schedulability and Priority Assignment for Multi-Segment Self-Suspending Real-Time Tasks under Fixed-Priority Scheduling.* Tech. rep. Dortmund, Germany: TU Dortmund University, 2015.

[HC16] Wen-Hung Huang and Jian-Jia Chen. “Self-suspension real-time tasks under fixed-relative-deadline fixed-priority scheduling.” In: *DATE.* IEEE, 2016, pp. 1078–1083.

[HCR16] Wen-Hung Huang, Jian-Jia Chen, and Jan Reineke. "MIRROR: symmetric timing analysis for real-time tasks on multicore platforms with shared resources." In: *DAC*. ACM, 2016, 158:1–158:6.

[Hua+15] Wen-Hung Huang, Jian-Jia Chen, Husheng Zhou, and Cong Liu. "PASS: priority assignment of real-time tasks with dynamic suspending behavior under fixed-priority scheduling." In: *DAC*. ACM, 2015, 154:1–154:6.

[Int10] International Electrotechnical Commission (IEC). "Functional safety of electrical / electronic / programmable electronic safety-related systems ed2.0." In: 2010.

[Int00] International Organization for Standardization (ISO). "Iso/fdis26262: Road vehicles - functional safety." In: 2000.

[Jia+23] Xu Jiang, Xiantong Luo, Nan Guan, Zheng Dong, Shaoshan Liu, and Wang Yi. "Analysis and Optimization of Worst-Case Time Disparity in Cause-Effect Chains." In: *DATE*. IEEE, 2023, pp. 1–6.

[JP86] Mathai Joseph and Paritosh K. Pandya. "Finding Response Times in a Real-Time System." In: *Comput. J.* 29.5 (1986), pp. 390–395.

[Kan+07] Woochul Kang, Sang Hyuk Son, John A. Stankovic, and Mehdi Amirijoo. "I/O-Aware Deadline Miss Ratio Management in Real-Time Embedded Databases." In: *RTSS*. IEEE Computer Society, 2007, pp. 277–287.

[Kat+11] Shinpei Kato, Karthik Lakshmanan, Aman Kumar, Mihir Kelkar, Yutaka Ishikawa, and Ragunathan Rajkumar. "RGEM: A Responsive GPGPU Execution Model for Runtime Engines." In: *RTSS*. IEEE Computer Society, 2011, pp. 57–66.

[Kim+95] In-Guk Kim, Kyung-Hee Choi, Seung-Kyu Park, Dong-Yoon Kim, and Man-Pyo Hong. "Real-time scheduling of tasks that contain the external blocking intervals." In: *RTCSA*. IEEE Computer Society, 1995, pp. 54–59.

[Kim+13] Junsung Kim, Björn Andersson, Dionisio de Niz, and Ragunathan Rajkumar. "Segment-Fixed Priority Scheduling for Self-Suspending Real-Time Tasks." In: *RTSS*. IEEE Computer Society, 2013, pp. 246–257.

[KS12] Christoph M. Kirsch and Ana Sokolova. "The Logical Execution Time Paradigm." In: *Advances in Real-Time Systems*. Springer, 2012, pp. 103–120.

[Kla+18] Tobias Klaus, Florian Franzmann, Matthias Becker, and Peter Ulbrich. "Data Propagation Delay Constraints in Multi-Rate Systems: Deadlines vs. Job-Level Dependencies." In: *RTNS*. ACM, 2018, pp. 93–103.

[Kle20] Achim Klenke. *Probability theory: a comprehensive course*. Springer International Publishing, 2020.

[KBS18] Tomasz Kloda, Antoine Bertout, and Yves Sorel. "Latency analysis for data chains of real-time periodic tasks." In: *ETFA*. IEEE, 2018, pp. 360–367.

[Klo+22] Tomasz Kloda, Jiyang Chen, Antoine Bertout, Lui Sha, and Marco Caccamo. "Latency analysis of self-suspending task chains." In: *DATE*. IEEE, 2022, pp. 1299–1304.

[Köh+23] Leonie Köhler, Phil Hertha, Matthias Beckert, Alex Bendrick, and Rolf Ernst. "Robust Cause-Effect Chains with Bounded Execution Time and System-Level Logical Execution Time." In: *ACM Trans. Embed. Comput. Syst.* 22.3 (2023), 50:1–50:28.

[KS22] Hermann Kopetz and Wilfried Steiner. *Real-Time Systems - Design Principles for Distributed Embedded Applications, Third Edition*. Springer, 2022.

[KT20] Alix Munier Kordon and Ning Tang. "Evaluation of the Age Latency of a Real-Time Communicating System Using the LET Paradigm." In: *ECRTS*. Vol. 165. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 20:1–20:20.

[KZH15] Simon Kramer, Dirk Ziegenbein, and Arne Hamann. "Real world automotive benchmarks for free." In: *6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*. Vol. 130. 2015.

[LR10] Karthik Lakshmanan and Ragunathan Rajkumar. "Scheduling Self-Suspending Real-Time Tasks with Rate-Monotonic Priorities." In: *IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE Computer Society, 2010, pp. 3–12.

[Lee+22] Hyoeun Lee, Youngjoon Choi, Taeho Han, and Kanghee Kim. "Probabilistically Guaranteeing End-to-End Latencies in Autonomous Vehicle Computing Systems." In: *IEEE Trans. Computers* 71.12 (2022), pp. 3361–3374.

[Lee94] Suk Kyoon Lee. "On-line multiprocessor scheduling algorithms for real-time tasks." In: *Proceedings of TENCON'94-1994 IEEE Region 10's 9th Annual International Conference on:'Frontiers of Computer Technology'*. IEEE. 1994, pp. 607–611.

[Leh90] John P. Lehoczky. "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines." In: *RTSS*. IEEE Computer Society, 1990, pp. 201–209.

[LSD89] John P. Lehoczky, Lui Sha, and Y. Ding. "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior." In: *RTSS*. IEEE Computer Society, 1989, pp. 166–171.

[Lei+11] C Lei, EM Van Eenennaam, W Klein Wolterink, G Karagiannis, Geert Heijenk, and J Ploeg. "Impact of packet loss on CACC string stability performance." In: *2011 11th International Conference on ITS Telecommunications*. IEEE. 2011, pp. 381–386.

[LA07] Hennadiy Leontyev and James H. Anderson. “Generalized Tardiness Bounds for Global Multiprocessor Scheduling.” In: *RTSS*. IEEE Computer Society, 2007, pp. 413–422.

[Leu89] Joseph Y.-T. Leung. “A New Algorithm for Scheduling Periodic Real-Time Tasks.” In: *Algorithmica* 4.2 (1989), pp. 209–219.

[LW82] Joseph Y.-T. Leung and Jennifer Whitehead. “On the complexity of fixed-priority scheduling of periodic, real-time tasks.” In: *Perform. Evaluation* 2.4 (1982), pp. 237–250.

[Li+14] Jing Li, Jian-Jia Chen, Kunal Agrawal, Chenyang Lu, Christopher D. Gill, and Abusayeed Saifullah. “Analysis of Federated and Global Scheduling for Parallel Real-Time Tasks.” In: *ECRTS*. IEEE Computer Society, 2014, pp. 85–96.

[Lin+23] Ching-Chi Lin, Mario Günzel, Junjie Shi, Tristan Taylan Seidl, Kuan-Hsun Chen, and Jian-Jia Chen. “Scheduling Periodic Segmented Self-Suspending Tasks without Timing Anomalies.” In: *RTAS*. IEEE, 2023, pp. 161–173.

[LL73] C. L. Liu and James W. Layland. “Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment.” In: *J. ACM* 20.1 (1973), pp. 46–61.

[LA13] Cong Liu and James H. Anderson. “Suspension-Aware Analysis for Hard Real-Time Multiprocessor Scheduling.” In: *ECRTS*. IEEE Computer Society, 2013, pp. 271–281.

[LA15] Cong Liu and James H. Anderson. *Erratum to “Suspension-Aware Analysis for Hard Real-Time Multiprocessor Scheduling”*. https://cs.unc.edu/~anderson/papers/ecrts13e_erratum.pdf. 2015.

[LC14] Cong Liu and Jian-Jia Chen. “Bursty-Interference Analysis Techniques for Analyzing Complex Real-Time Task Models.” In: *RTSS*. IEEE Computer Society, 2014, pp. 173–183.

[Liu00] Jane W. S. Liu. *Real-Time Systems*. 1st. Prentice Hall PTR, 2000. ISBN: 0130996513.

[Liu+14] Wei Liu, Jian-Jia Chen, Anas Toma, Tei-Wei Kuo, and Qingxu Deng. “Computation Offloading by Using Timing Unreliable Components in Real-Time Systems.” In: *DAC*. ACM, 2014, 39:1–39:6.

[Lóp+08] José María López, José Luis Díaz, Joaquín Entrialgo, and Daniel F. García. “Stochastic analysis of real-time systems under preemptive priority-driven scheduling.” In: *Real Time Syst.* 40.2 (2008), pp. 180–207.

[Mar+18] Filip Markovic, Jan Carlson, Radu Dobrin, Björn Lisper, and Abhilash Thekkilakattil. “Probabilistic Response Time Analysis for Fixed Preemption Point Selection.” In: *SIES*. IEEE, 2018, pp. 1–10.

[MNP22] Filip Markovic, Thomas Nolte, and Alessandro Vittorio Papadopoulos. “Analytical Approximations in Probabilistic Analysis of Real-Time Systems.” In: *RTSS*. IEEE, 2022, pp. 158–171.

[MPN21] Filip Markovic, Alessandro Vittorio Papadopoulos, and Thomas Nolte. “On the Convolution Efficiency for Probabilistic Analysis of Real-Time Systems.” In: *ECRTS*. Vol. 196. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 16:1–16:22.

[Mar+23] Filip Markovic, Pierre Roux, Sergey Bozhko, Alessandro V. Papadopoulos, and Björn B. Brandenburg. “CTA: A Correlation-Tolerant Analysis of the Deadline-Failure Probability of Dependent Tasks.” In: *RTSS*. IEEE, 2023, pp. 317–330.

[MSB20] Jorge Martinez, Ignacio Sañudo, and Marko Bertogna. “End-to-end latency characterization of task communication models for automotive systems.” In: *Real Time Syst.* 56.3 (2020), pp. 315–347.

[Mar21] Peter Marwedel. *Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems, and the Internet of Things*. Springer International Publishing, 2021.

[MC13] Dorin Maxim and Liliana Cucu-Grosjean. “Response Time Analysis for Fixed-Priority Tasks with Multiple Probabilistic Parameters.” In: *RTSS*. IEEE Computer Society, 2013, pp. 224–235.

[Max+17] Dorin Maxim, Robert I. Davis, Liliana Cucu-Grosjean, and Arvind Easwaran. “Probabilistic analysis for mixed criticality systems using fixed priority preemptive scheduling.” In: *RTNS*. ACM, 2017, pp. 237–246.

[Max+12] Dorin Maxim, Mike Houston, Luca Santinelli, Guillem Bernat, Robert I. Davis, and Liliana Cucu-Grosjean. “Re-sampling for statistical timing analysis of real-time systems.” In: *RTNS*. ACM, 2012, pp. 111–120.

[MEY16] Morteza Mohaqeqi, Pontus Ekberg, and Wang Yi. “On Fixed-Priority Schedulability Analysis of Sporadic Tasks with Self-Suspension.” In: *RTNS*. ACM, 2016, pp. 109–118.

[MC97] Aloysius K. Mok and Deji Chen. “A Multiframe Model for Real-Time Tasks.” In: *IEEE Trans. Software Eng.* 23.10 (1997), pp. 635–645.

[Mok83] Aloysius Ka-Lau Mok. “Fundamental design problems of distributed systems for the hard-real-time environment.” PhD thesis. Massachusetts Institute of Technology, Cambridge, MA, USA, 1983.

[MMS12a] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. “Implementation of end-to-end latency analysis for component-based multi-rate real-time systems in Rubus-ICE.” In: *WFCS*. IEEE, 2012, pp. 165–168.

[MMS12b] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. “Translating end-to-end timing requirements to timing analysis model in component-based distributed real-time systems.” In: *SIGBED Rev.* 9.4 (2012), pp. 17–20.

[NNB19] Mitra Nasri, Geoffrey Nelissen, and Björn B. Brandenburg. "Response-Time Analysis of Limited-Preemptive Parallel DAG Tasks Under Global Scheduling." In: *ECRTS*. Vol. 133. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 21:1–21:23.

[Nel+17] Geoffrey Nelissen, José Fonseca, Gurulingesh Raravi, and Vincent Nélis. *Errata: Timing Analysis of Fixed Priority Self-Suspending Sporadic Tasks.* Tech. rep. CISTER-TR-170205. CISTER, ISEP, INESC-TEC, 2017.

[Nel+15] Geoffrey Nelissen, José Carlos Fonseca, Gurulingesh Raravi, and Vincent Nélis. "Timing Analysis of Fixed Priority Self-Suspending Sporadic Tasks." In: *ECRTS*. IEEE Computer Society, 2015, pp. 80–89.

[Nim+10] Yamini Nimmagadda, Karthik Kumar, Yung-Hsiang Lu, and C. S. George Lee. "Real-time moving object recognition and tracking using computation offloading." In: *IROS*. IEEE, 2010, pp. 2449–2455.

[Ope22] Open Robotics. *ROS 2 Documentation: Foxy.* `https://docs.ros.org/en/foxy`. May 2022.

[PM22] Paolo Pazzaglia and Martina Maggio. "Characterizing the Effect of Deadline Misses on Time-Triggered Task Chains." In: *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 41.11 (2022), pp. 3957–3968.

[PF16] Bo Peng and Nathan Fisher. "Parameter Adaption for Generalized Multiframe Tasks and Applications to Self-Suspending Tasks." In: *RTCSA*. IEEE Computer Society, 2016, pp. 49–58.

[PF17] Bo Peng and Nathan Fisher. "Parameter adaptation for generalized multiframe tasks: schedulability analysis, case study, and applications to self-suspending tasks." In: *Real Time Syst.* 53.6 (2017), pp. 957–986.

[Per21] PerceptIn. "RTSS Industry Challenge 2021." In: *Real-Time Systems Symposium (RTSS)*. 2021.

[Raj+10] A. C. Rajeev, Swarup Mohalik, Manoj G. Dixit, Devesh B. Chokshi, and S. Ramesh. "Schedulability and end-to-end latency in distributed ECU networks: formal modeling and precise estimation." In: *EMSOFT*. ACM, 2010, pp. 129–138.

[Raj91] R. Rajkumar. *Dealing with Suspending Periodic Tasks.* Tech. rep. IBM T. J. Watson Research Center, 1991.

[RSL88] Ragunathan Rajkumar, Lui Sha, and John P. Lehoczky. "Real-Time Synchronization Protocols for Multiprocessors." In: *RTSS*. IEEE Computer Society, 1988, pp. 259–269.

[RH10] Khaled S. Refaat and Pierre-Emmanuel Hladik. "Efficient Stochastic Analysis of Real-Time Systems via Random Sampling." In: *ECRTS*. IEEE Computer Society, 2010, pp. 175–183.

[Ren+15] Jiankang Ren, Guowei Wu, Xinjiao Li, Poria Pirozmand, and Mohammad S. Obaidat. “Probabilistic response-time analysis for real-time systems in body area sensor networks.” In: *Int. J. Commun. Syst.* 28.16 (2015), pp. 2145–2166.

[Ren+19] Jiankang Ren, Zichuan Xu, Chao Yu, Chi Lin, Guowei Wu, and Guozhen Tan. “Execution allowance based fixed priority scheduling for probabilistic real-time systems.” In: *J. Syst. Softw.* 152 (2019), pp. 120–133.

[RRC04] Frédéric Ridouard, Pascal Richard, and Francis Cottet. “Negative Results for Scheduling Independent Hard Real-Time Tasks with Self-Suspensions.” In: *RTSS*. IEEE Computer Society, 2004, pp. 47–56.

[RTE23] RTEMS Project and Contributors. *RTEMS User Manual*. 2023. URL: `https://www.rtems.org`.

[SRF09] Jean-Luc Scharbarg, Frédéric Ridouard, and Christian Fraboul. “A probabilistic analysis of end-to-end delays on an AFDX avionic network.” In: *IEEE transactions on industrial informatics* 5.1 (2009), pp. 38–49.

[SE16] Johannes Schlatow and Rolf Ernst. “Response-Time Analysis for Task Chains in Communicating Threads.” In: *RTAS*. IEEE Computer Society, 2016, pp. 245–254.

[Sch+18a] Johannes Schlatow, Mischa Möstl, Sebastian Tobuschat, Tasuku Ishigooka, and Rolf Ernst. “Data-Age Analysis and Optimisation for Cause-Effect Chains in Automotive Control Systems.” In: *SIES*. IEEE, 2018, pp. 1–9.

[Sch+18b] Lea Schönberger, Wen-Hung Huang, Georg von der Brüggen, Kuan-Hsun Chen, and Jian-Jia Chen. “Schedulability Analysis and Priority Assignment for Segmented Self-Suspending Tasks.” In: *RTCSA*. IEEE Computer Society, 2018, pp. 157–167.

[SR94] Kang G. Shin and Parameswaran Ramanathan. “Real-time computing: a new discipline of computer science and engineering.” In: *Proc. IEEE* 82.1 (1994), pp. 6–24.

[SL96] Jun Sun and Jane W.-S. Liu. “Synchronization Protocols in Distributed Real-Time Systems.” In: *ICDCS*. IEEE Computer Society, 1996, pp. 38–45.

[Tan+23a] Yue Tang, Nan Guan, Xu Jiang, Zheng Dong, and Wang Yi. “Reaction Time Analysis of Event-Triggered Processing Chains with Data Refreshing.” In: *DAC*. IEEE, 2023, pp. 1–6.

[TGY22] Yue Tang, Nan Guan, and Wang Yi. “Real-Time Task Models.” In: *Handbook of Real-Time Computing*. Springer, 2022, pp. 469–487.

[Tan+23b] Yue Tang, Xu Jiang, Nan Guan, Dong Ji, Xiantong Luo, and Wang Yi. “Comparing Communication Paradigms in Cause-Effect Chains.” In: *IEEE Trans. Computers* 72.1 (2023), pp. 82–96.

[Tao22] Terence Tao. “Lebesgue Measure.” In: *Analysis II*. Singapore: Springer Nature Singapore, 2022, pp. 145–166. ISBN: 978-981-19-7284-3. DOI: 10.1007/978-981-19-7284-3_7. URL: https://doi.org/10.1007/978-981-19-7284-3_7.

[Tep+22] Harun Teper, Mario Günzel, Niklas Ueter, Georg von der Brüggen, and Jian-Jia Chen. “End-To-End Timing Analysis in ROS2.” In: *RTSS*. IEEE, 2022, pp. 53–65.

[TCN00] Lothar Thiele, Samarjit Chakraborty, and Martin Naedele. “Real-time calculus for scheduling hard real-time systems.” In: *ISCAS*. IEEE, 2000, pp. 101–104.

[Tia+95] Too-Seng Tia, Zhong Deng, Mallikarjun Shankar, Matthew F. Storch, Jun Sun, L.-C. Wu, and Jane W.-S. Liu. “Probabilistic performance guarantee for real-time tasks with varying computation times.” In: *IEEE Real Time Technology and Applications Symposium*. IEEE Computer Society, 1995, pp. 164–173.

[Tob+16] Sebastian Tobuschat, Rolf Ernst, Arne Hamann, and Dirk Ziegenbein. “System-level timing feasibility test for cyber-physical automotive systems.” In: *SIES*. IEEE, 2016, pp. 121–130.

[TC13] Anas Toma and Jian-Jia Chen. “Computation Offloading for Frame-Based Real-Time Tasks with Resource Reservation Servers.” In: *ECRTS*. IEEE Computer Society, 2013, pp. 103–112.

[TU 21a] TU Dortmund LS12. *Evaluation Framework for Self-Suspending Task Systems*. https://github.com/tu-dortmund-ls12-rt/SSSEvaluation. 2021.

[TU 21b] TU Dortmund LS12. *Suspension-Aware Analysis for Arrival Curves*. https://github.com/tu-dortmund-ls12-rt/arr_curve. 2021.

[TU 22a] TU Dortmund LS12. *EDF-Like Scheduling Schedulability Evaluation*. https://github.com/tu-dortmund-ls12-rt/EDF-Like. 2022.

[TU 22b] TU Dortmund LS12. *End-To-End Timing Analysis*. https://github.com/tu-dortmund-ls12-rt/end-to-end_inter. 2022.

[TU 23] TU Dortmund LS12. *End-To-End Timing Analysis*. https://github.com/tu-dortmund-ls12-rt/end-to-end_mixed. 2023.

[TU 24] TU Dortmund LS12. *Segment-Level-Early-Execution*. https://github.com/tu-dortmund-ls12-rt/Segment-Level-Early-Execution. 2024.

[TU 13] TU Dortmund University, Department of Computer Science. *Promotionsordnung der Fakultät für Informatik der Technischen Universität Dortmund vom 29. August 2011 (Lesefassung einschließlich Erster Ordnung zur Änderung der Promotionsordnung vom 8. Januar 2013)*. https://cs.tu-dortmund.de/storages/cs/r/Informatik/Dateien/Gremien/

Promotionsordnungen_richtlinien/2011/Promotionsordnung-FK4-Lesefassung-1_-Aend0.pdf. 2013.

[Uet+20] Niklas Ueter, Jian-Jia Chen, Georg von der Brüggen, Vanchinathan Venkataramani, and Tulika Mitra. "Simultaneous Progressing Switching Protocols for Timing Predictable Real-Time Network-on-Chips." In: *RTCSA*. IEEE, 2020, pp. 1–10.

[Uet+21] Niklas Ueter, Mario Günzel, Georg von der Brüggen, and Jian-Jia Chen. "Hard Real-Time Stationary GANG-Scheduling." In: *ECRTS*. Vol. 196. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 10:1–10:19.

[UGC21] Niklas Ueter, Mario Günzel, and Jian-Jia Chen. "Response-Time Analysis and Optimization for Probabilistic Conditional Parallel DAG Tasks." In: *RTSS*. IEEE, 2021, pp. 380–392.

[Uth04] Patchrawat Uthaisombut. "The Optimal Online Algorithms for Minimizing Maximum Lateness." In: *SWAT*. Vol. 3111. Lecture Notes in Computer Science. Springer, 2004, pp. 420–430.

[vT17] Martinus Richardus van Steen and Andrew S. Tanenbaum. *Distributed Systems*. English. 3rd. Self-published, open publication. Feb. 2017. ISBN: 978-15-430573-8-6.

[von+19] Georg von der Brüggen, Milad Nayebi, Junjie Shi, Kuan-Hsun Chen, and Jian-Jia Chen. "Evaluation Framework for Self-Suspending Task Systems." In: *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*. 2019.

[Wan+23] Sen Wang, Dong Li, Ashrarul H. Sifat, Shaoyu Huang, Xuanliang Deng, Changhee Jung, Ryan K. Williams, and Haibo Zeng. "Optimizing Logical Execution Time Model for Both Determinism and Low Latency." In: *CoRR* abs/2310.19699 (2023).

[Wys+13] Rémy Wyss, Frédéric Boniol, Claire Pagetti, and Julien Forget. "End-to-end latency computation in a multi-periodic design." In: *SAC*. ACM, 2013, pp. 1682–1687.

[YNB19] Beyazit Yalcinkaya, Mitra Nasri, and Björn B. Brandenburg. "An Exact Schedulability Test for Non-Preemptive Self-Suspending Real-Time Tasks." In: *DATE*. IEEE, 2019, pp. 1228–1233.

Index

Notation

Real-Time Scheduling		
$\mathbb{T}$	Task set	Section 2.1
$\tau \in \mathbb{T}$	A task	Section 2.1
$\tau(j)$	j-th job of task τ	Section 2.1
$\mathbb{J}$	Job set	Section 2.1
$J \in \mathbb{J}$	A generic job	Section 2.1
D_τ	Relative deadline	Section 2.1
d_J	Absolute deadline	Section 2.1
Ω	Set of system evolutions	Section 2.2
$\omega \in \Omega$	A system evolution	Section 2.2
r_J^ω	Job release	Section 2.2
c_J^ω	Job execution demand	Section 2.2
$\mathcal{S}$	A discrete schedule	Section 2.3
$\mathbb{S}^{discr}$	Set of discrete schedules	Section 2.3
$s_J^{\mathcal{S},\omega}$	Start of job J in discrete schedule $\mathcal{S}$	Section 2.3
$f_J^{\mathcal{S},\omega}$	Finish of job J in discrete schedule $\mathcal{S}$	Section 2.3
$[\mathcal{S}]$	A continuous schedule	Section 2.3
$\mathbb{S}^{cont}$	Set of continuous schedules	Section 2.3
$s_J^{[\mathcal{S}],\omega}$	Start of job J in continuous schedule $[\mathcal{S}]$	Section 2.3
$f_J^{[\mathcal{S}],\omega}$	Finish of job J in continuous schedule $[\mathcal{S}]$	Section 2.3
$\mathrm{idle}^{[\mathcal{S}]}(t_1, t_2)$	Idle time in interval $[t_1, t_2]$	Section 2.3
$\mathrm{exec}_J^{[\mathcal{S}]}(t_1, t_2)$	Execution time of job J in interval $[t_1, t_2]$	Section 2.3
$\mathrm{exec}_\tau^{[\mathcal{S}]}(t_1, t_2)$	Execution time of task τ in interval $[t_1, t_2]$	Section 2.3

Table 1.: Notation used in this work for real-time scheduling.

Abstraction of Task Properties		
WCET(C_τ) or WCET($C_\tau^{\max}$)	Worst-Case Execution Time	Section 2.4.1
BCET($C_\tau^{\min}$)	Best-Case Execution Time	Section 2.4.1
pET($\mathcal{D}_\tau$)	Probabilistic Execution Time	Section 2.4.1
pET$^{\text{iid}}$($\mathcal{D}_\tau$)	Probabilistic Execution Time with iid assumption	Section 2.4.1
pWCET($\mathcal{D}_\tau$)	Probabilistic Worst-Case Execution Time	Section 2.4.1
pWCET$^{\text{iid}}$($\mathcal{D}_\tau$)	Probabilistic Worst-Case Execution Time with iid assumption	Section 2.4.1
FixRel($(r_{\tau(j)})_{j\in\mathbb{N}}$)	Fixed release pattern	Section 2.4.2
PerOff(T_τ, ϕ_τ)	Periodic tasks with offset	Section 2.4.2
Per(T_τ)	Periodic tasks	Section 2.4.2
Spor(T_τ)	Sporadic tasks	Section 2.4.2
SporMinMax($T_\tau^{\min}, T_\tau^{\max}$)	Sporadic tasks with minimum and maximum inter-arrival time	Section 2.4.2
Arr(α_τ)	Tasks with arrival curve	Section 2.4.2
$U_\tau := \frac{C_\tau}{T_\tau}$	Task utilization	Section 2.4.2
$U_\mathbb{T} = \sum_{\tau\in\mathbb{T}} U_\tau$	Task set utilization	Section 2.4.2
$H(\mathbb{T})$	Hyperperiod	Section 2.4.2

Table 2.: Notation used in this work for abstraction of task properties.

Scheduling Algorithms and Response Times		
$\mathcal{A}$	A scheduling algorithm	Section 2.4.3
$\mathcal{A}(\omega)$	Schedule derived from system evolution ω under scheduling algorithm $\mathcal{A}$	Section 2.4.3
Π_τ	Relative priority point for EDF-Like (EL) scheduling	Section 2.4.3
$R_{\tau(j)}^{\mathcal{A}(\omega),\omega}$	Response time of job $\tau(j)$	Section 2.5
$R_\tau^{\mathcal{A}}$	Worst-Case Response Time (WCRT)	Section 2.5
W_τ	Time-demand function (for TDA)	Section 2.5

Table 3.: Notation used in this work for scheduling algorithms and response times.

Distributed Execution of Jobs		
$\mathbb{B}_J^\omega$	Blocks of job J	Section 3.1
$\mathcal{P}_J^\omega$	Precedence constraints	Section 3.1
$\mathbb{B}^\omega$	Set of all blocks	Section 3.1
$\delta^\omega_{(B,B')}$	Delay between two blocks B and B'	Section 3.1
EDD$(\dots)$	Abstraction of EDD tasks	Section 3.1
DAG$(\dots)$	Abstraction of DAG tasks	Section 3.1
SegSS$(C_\tau^1, S_\tau^1, \dots, C_\tau^{m_\tau})$	Abstraction of segmented self-suspending tasks	Section 3.1
DynSS(C_τ, S_τ)	Abstraction of dynamic self-suspending tasks	Section 3.1

Table 4.: Notation used in this work for distributed execution of jobs and for self-suspension.

Interplay of Distributed Tasks		
$E = (\tau_1 \to \dots \to \tau_n)$	A cause-effect chain	Section 3.2
$\text{re}^\omega(J)$	Read-event	Section 3.2
$\text{we}^\omega(J)$	Write-event	Section 3.2
$c = (J_1, \dots, J_n)$	A job chain	Section 3.2
$\ell^\omega(c)$	Length of job chain c	Section 3.2
$\vec{ac}^\omega_E(z) = (z, J_1, \dots, J_n, z')$	Immediate forward augmented job chain	Section 3.2
$\ell(\vec{ac}^\omega_E(z))$	Length of $\vec{ac}^\omega_E(z)$	Section 3.2
$\overleftarrow{ac}^\omega_E(z') = (z, J_1, \dots, J_n, z')$	Immediate backward augmented job chain	Section 3.2
$\ell(\overleftarrow{ac}^\omega_E(z'))$	Length of $\overleftarrow{ac}^\omega_E(z')$	Section 3.2
W^ω_{E,τ_i}	Job index for warm-up	Section 3.2
MRT(E, ω) and MRT(E)	Maximum Reaction Time	Section 3.2
MDA(E, ω) and MDA(E)	Maximum Data Age	Section 3.2
MRRT(E, ω) and MRRT(E)	Maximum Reduced Reaction Time	Section 3.2
MRDA(E, ω) and MRDA(E)	Maximum Reduced Data Age	Section 3.2
Lat(E, ω) and Lat(E)	End-to-end latency	Section 6.1.2
$pc^\omega_{E,p}(m)$	m-th p-partitioned job chain	Section 6.1.2.1
$\vec{I}_i^E$	i-th abstract integer representation	Section 6.2.2.2
RT$(E, \bullet, z)$	Reaction time at z	Section 6.3.1
PRTG$(E, \bullet)$	Probabilistic reaction time guarantee	Section 6.3.1
DA$(E, \bullet, z)$	Data age at z	Section 6.3.1
PDAG$(E, \bullet)$	Probabilistic data age guarantee	Section 6.3.1

Table 5.: Notation used in this work for the interplay of distributed tasks and end-to-end analysis of cause-effect chains.

Acronyms

ACET Average-Case Execution Time, 211 f., 230, 250

APS Adaptive Partitioning Scheduler, 57

BCET Best-Case Execution Time, 23 f., 187, 199 f., 203, 211, 214, 222

CAN Controller Area Network, 2, 221, 223

CIT Critical Instant Theorem, 6, 11, 31–34, 37, 54, 59, 63, 67, 69, 71 ff., 75, 77 ff., 81 f., 257, 259, 268

CPA Compositional Performance Analysis, 26, 95 f., 102

DAG Directed Acyclic Graph, 38, 40 f., 81, 258 f., 262 f.

DBP Dynamic Buffering Protocol, 57

DFP Deadline Failure Probability, 68, 71, 77

DM Deadline Monotonic, 5, 29 ff., 100, 126, 143, 146, 153

DRS Dirichlet-Rescale, 168, 171

ECU Electronic Control Unit, 2 f., 9, 43, 46, 57 f., 179, 186 f., 189, 193, 196, 199, 201, 204, 208 f., 214, 220 f., 225, 250, 253

EDD Event-Driven Delay-Induced, 39–42, 258 f.

EDF Earliest-Deadline-First, ii, 5–8, 11, 29 ff., 35, 54, 65 f., 78, 82, 103–108, 110, 112 f., 117 ff., 121 f., 126 ff., 130, 143 f., 153, 168 ff., 179, 185, 232, 257

EDZL Earliest Deadline Zero Laxity, 29

EL EDF-Like, ii, 5, 7 f., 12, 29 f., 82, 126–131, 137, 139, 141, 143, 153, 170 f., 257

EQDF Earliest-Quasi-Deadline-First, 126, 128, 130, 143 f.

FCFS First-Come First-Served, 29 f.

FIFO First In – First Out, 29 f., 49, 126, 128, 130 f., 134, 232

FILO First In – Last Out, 49

FPGA Field Programmable Gate Array, 55

FRD Fixed-Relative-Deadline, 54

GALS Globally Asynchronized Locally Synchronized, 220

GUI Graphical User Interface, 7, 147, 151

iid independent and identically distributed, 24, 67, 233, 237 f., 240, 254, 259, 262, 268, 271

J-DP Job-level Dynamic-Priority, 29, 127

J-FP Job-level Fixed-Priority, 29 ff., 39, 58, 127 f., 130, 153, 179

lcm least common multiple, 28, 192

LET Logical Execution Time, 5, 9, 45 ff., 57 f., 174, 179, 185 f., 188 ff., 192 ff., 196 f., 216, 218, 222 ff., 226 f., 230 f., 233, 241–245, 248 f., 251 ff.

LIFO Last In – First Out, 49

LILO Last In – Last Out, 49

LLF Least-Laxity-First, 29

LST Least Slack Time, 29

MDA Maximum Data Age, 3, 44, 49 f., 52 f., 57, 174, 176, 178–182, 185 ff., 211, 216, 233, 252 f., 258

MILP Mixed-Integer Linear Programming, 40

MRDA Maximum Reduced Data Age, 49–53, 55, 57, 178 f., 185 ff., 200, 210 f., 213–216, 222, 253

MRRT Maximum Reduced Reaction Time, 50–53, 57, 178, 185 ff., 222, 253

MRT Maximum Reaction Time, 3, 44, 49 f., 52 f., 55, 57, 174, 178–182, 185 ff., 189, 205, 216, 223, 233, 236, 251 ff., 258

OEM Original Equipment Manufacturer, 46

pET Probabilistic Execution Time, 24, 34, 67

PTDA Probabilistic Time-Demand Analysis, 34

pWCET Probabilistic Worst-Case Execution Time, 24, 67, 69, 71, 73, 237

pWCRT probabilistic Worst-Case Response Time, 69

List of Figures