Rheinland-Pfälzische Technische Universität Kaiserslautern Landau

FB Elektrotechnik und Informationstechnik

Timing Analysis of Multi-Rate Cause-Effect Chains in Safety-Critical Real-Time Systems

Luiz Gonzaga Nunes Maia Neto

Dissertation 2026

Timing Analysis of Multi-Rate Cause-Effect Chains in Safety-Critical Real-Time Systems

vom
Fachbereich Elektrotechnik und Informationstechnik
der Rheinland-Pfälzische Technische Universität Kaiserslautern-Landau
zur Verleihung des akademischen Grades eines

Doktor der Ingenieurwissenschaften (Dr.-Ing.)

genehmigte Dissertation

von
Luiz Gonzaga Nunes Maia Neto
geboren in Pouso Alegre, Minas Gerais, Brasilien

D 386

Eingereicht am:	16.04.2026
Tag der mündlichen Prüfung:	28.05.2026
Dekan des Fachbereichs:	Prof. Dr.-Ing. Daniel Görges
Promotionskomission	
Vorsitzender:	Prof. Dr.-Ing. Wolfgang Kunz
Berichterstattende:	Prof. Dipl.-Ing. Dr. Gerhard Fohler
	Prof. Martina Maggio

Erklärung gem. § 6 Abs. 3 Promotionsordnung

Ich versichere, dass ich diese Dissertation selbst und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt und die aus den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Diese Dissertation wurde weder als Ganzes noch in Teilen als Prüfungsarbeit für eine staatliche oder andere wissenschaftliche Prüfung eingereicht. Es wurde weder diese noch eine andere Abhandlung bei einem anderen Fachbereich oder einer anderen Universität als Dissertation eingereicht.

(Ort, Datum) (Name)

Abstract

Over the past years, different sectors of the industry have been trying to cope with the continuous demand for more powerful, safe, and efficient safety-critical real-time systems. The ongoing industry shift from single-core to multi-core architecture adds an extra complexity layer to the problem of verifying and validating if safety-critical real-time systems are behaving according to their specification parameters. The multi-core architecture allows tasks to execute in parallel and issue memory requests concurrently, which can lead to unpredictable contention at the bus and/or memory level. In the *worst-case* scenario, contention can result in the deadline miss of a task and the consequent failure of the system, which for safety-critical systems means that the failure can lead to a catastrophic event that can endanger the lives of multiple humans.

Due to the increasing complexity of safety-critical functionalities in modern automotive and avionics systems, it is often required that multiple software applications have to constantly communicate with each other in order to achieve a given safety-critical functionality within the system. As a result, many software applications in those domains are often modeled by the system designers as ordered sequences of communicating tasks known in the literature as cause-effect chains. For those chained tasks, inter-task communication occurs by means of shared resources, with the output produced by one task serving as input for the next one. Since the correct behavior of a safety-critical functionality according to system's specification no longer solely depends on the execution of a single task, but rather on the time taken by data to propagate through the cause-effect chain, the complexity of verifying whether or not timing constraints are respected during runtime increases even further. Especially when the tasks forming a cause-effect chain have different activation periods, resulting in what is known as a multi-rate cause-effect chain.

For safety-critical applications, it is pivotal to ensure that the end-to-end latencies of a cause-effect chain modeling a given system functionality are always within a given range during runtime in order to prevent system failure. However, since in a multi-core platform tasks might be allocated to different cores, execute in parallel, and access shared resources in a non-deterministic manner, the problem of tracing data propagation and ensuring that the end-to-end latency of cause-effect chains are always within a given range turns into a non-trivial task to be solved.

In this dissertation, we study the challenges of performing end-to-end timing analysis in safety-critical real-time systems with multi-rate cause-effect chains and multi-core architecture. The main contribution of this dissertation consists of proposing methods to verify and improve the end-to-end latencies of safety-critical real-time applications with tasks applying the logical execution time (LET) communication paradigm. More specifically, we propose the following contributions: **(i)** a method to reduce end-to-end latencies of multi-rate cause-effect chains applying the LET paradigm by considering knowledge of the schedule in later design phases; **(ii)** a method to further reconfigure the communication intervals of LET tasks by establishing precedence constraints between specific task instances; **(iii)** a method to decrease system utilization by skipping the execution of task instances that do not affect end-to-end latencies of cause-effect chains; **(iv)** a method to increase the feasibility of multi-rate cause-effect chains to meet their end-to-end latency constraints in multi-execution mode systems. Evaluations using automotive benchmarks and synthetic task sets show the benefits of our methods under different end-to-end latency metrics such as reaction time and data age.

A fé na vitória tem que ser inabalável [1]

Dexter - Fênix

[1] English translation: Faith in victory must be unwavering.

Acknowledgments

First, I would like to thank God for giving me the strength and determination I needed to make it this far. Without His spiritual comfort, this journey would have been much more difficult. I am grateful to my parents for teaching me to have faith and hope. Speaking of them, I am immensely grateful for all the love and support that my father Diogo, my mother Lúcia, my brother Gabriel, and my grandmothers Nilcéia and Zezé have given me throughout all these years of my life. My family is everything to me. I thank my father for his daily messages and calls, my mother for her emotional support, my brother for his love and companionship, and my grandmothers for the life lessons they taught me.

I would also like to express my immense gratitude to Professor Gerhard Fohler, a person who, in addition to being my academic mentor, has always believed in my potential and encouraged me to keep moving forward. Thank you, Professor Fohler, for your support and for the life lessons that have helped me more than you can imagine. I also thank my former mentors, Dr. Rodrigo Coelho and MSc. Florian Heilmann, people who also taught me valuable lessons, especially about having more confidence in myself and my work. I would like to thank Professor Martina Maggio for accepting to be part of the committee that evaluated and reviewed my dissertation.

During my time as a PhD student in the Real-Time Systems department, I had the pleasure of working with my great colleagues Carlos Rodriguez, Gautam Gala, Marine Kadar, Kristin Krüger, Ibrahim Alkoudsi, Isser Kadusale, Allan Paz, and Shreya Kulhalli. Thank you for making my work environment a more pleasant place and for helping me to become a better researcher. I would like to thank all the researchers who worked with me on my scientific papers. Your contributions, comments, and feedback were essential to our joint work. A special thanks to Markus Müller and Stephanie Jung for helping me to better integrate into the German life and culture. I will always carry with me the tips and knowledge I learned from you. I am immensely grateful for all the times you two helped me. To my students, Katherine Sirois, Priya, Aaryaa Shekhar, and Vinayaka Kirani Srinatha, I would like to thank you for your valuable contributions and the academic discussions we had. Supervising you was a pleasure and a great opportunity for my own academic growth.

To my relatives in Brazil (uncles, aunts, and cousins), thank you so much for always thinking of me and staying in touch with me all these years. I will never forget your kindness and support. I would like to thank my true friends for all the emotional support

you have given me during these years in Germany. A special thanks to my friends Dr. Fabiana Silva Bittencourt and Isadora Pereira for offering me kindness and emotional support when I needed it most. I also thank my friends who no longer live in Germany; it was great to share life with you during the time we were together. I am especially grateful to those who taught me to be kinder to myself.

I would also like to thank all the creators of the mangas, games, and music that have had a positive impact on me during my PhD. Your artistic works touched me in ways I can't fully describe; they gave me strength, hope, and emotional comfort to keep going.

I thank God once again for always being by my side. May He continue to light my path through life.

Luiz Maia
Kaiserslautern, 16.04.2026

Publications

I have authored or co-authored the following publications:

Conference and refereed workshop papers

- Luiz Maia, Mohammad Ibrahim Alkoudsi, Gerhard Fohler, "Ensuring End-To-End Latencies of Multi-Rate Cause-Effect Chains in Multi-Mode Systems Through Migration", 13th European Congress on Embedded Real-Time Systems (ERTS) 2026
- Luiz Maia, Mario Günzel and Gerhard Fohler, "Safe Reconfiguration of LET Communication Intervals to Reduce End-to-End Latencies", 33rd International Conference on Real-Time Networks and Systems (RTNS) 2025
- Luiz Maia and Gerhard Fohler, "Decreasing Utilization of Systems with Multi-Rate Cause-Effect Chains While Reducing End-to-End Latencies", 28th IEEE International Symposium On Real-Time Distributed Computing (ISORC) 2025
- Luiz Maia and Gerhard Fohler, "Reducing End-to-End Latencies of Multi-Rate Cause-Effect Chains in Safety Critical Embedded Systems", 12th European Congress on Embedded Real-Time Systems (ERTS) 2024
- Gautam Gala, Tilmann Unte, Luiz Maia, Johannes Kühbacher, Isser Kadusale, Mohammad Ibrahim Alkoudsi, Gerhard Fohler, and Sebastian Altmeyer, "Safety-Critical Edge Robotics Architecture with Bounded End-to-End Latency", Real-time Cloud (RT-Cloud) workshop co-hosted with 36th Euromicro conference on Real-Time Systems (ECRTS) 2024

Contents

List of Figures

List of Tables

Chapter I

Introduction

Over the past decades different sectors of the industry have been trying to cope with the continuous demand for more powerful and efficient computing systems. From the smartphone in our pockets to the autopilot system in autonomous vehicles, complex embedded computing systems of different types and criticality levels became part of our everyday life. While the failure of a smartphone can be unpleasant to its user, the failure of a car's autopilot system can lead to a catastrophic event and endanger the lives of multiple humans, i.e., that is a *safety-critical* failure.

Before being deployed in vehicles, airplanes, trains, and sometimes even medical devices, safety-critical embedded systems have to go under strict certification processes to ensure that they behave in accordance to their specification parameters during runtime. For a *real-time system (RTS)*, however, behaving according to specification does not solely depends on the correct computation of output data (logical correctness), but also depends on when the output is available (temporal correctness).

The ongoing industry shift from single-core to multi-core architecture adds an extra complexity layer to the problem of verifying and validating the specification of safety-critical real-time systems, e.g., the *electronic control units (ECUs)* in the automotive domain. The multi-core architecture allows tasks to execute in parallel and issue memory requests concurrently, which can lead to unpredictable contention at the bus and/or memory level. In the *worst-case*, contention can result on the deadline miss of a task and the consequent failure of the safety-critical system.

Modern automotive ECUs run multiple control applications containing several tasks that are continuously communicating with each other to provide specific safety-critical functionalities. For those tasks, inter-task communication occurs by means of shared resources, with the output produced by one task serving as input for the next one, i.e., data propagates in a chained manner. As expected from a safety-critical control application, it is pivotal to ensure that the *end-to-end (E2E)* latency of each ECU — time required to read, process and output data — is within a given range even in scenarios where contention might happen. Failing to ensure E2E latency within a specific interval can directly compromise the control performance of an application and in some cases lead to more severe types failures within the control system [1].

The E2E latencies of an ECU are directly affected by the communication model and the scheduling mechanism adopted by the system unit. For most communication models, the points in time where inter-task communications occurs (accesses to shared resources) are non-deterministic since they depend on when tasks start and finish their execution, which in turn is indirectly determined by the adopted scheduling policy. Given that, during runtime, tasks that are allocated to different cores might run in parallel and their execution might vary from one instance to another, it is nearly impossible to verify the E2E latencies by considering all the possible execution scenarios. As a result, the problem of tracing data propagation and ensuring that the E2E latency of a given ECU is within a given range turns into a non-trivial task to be solved.

This dissertation studies the challenges of performing E2E timing analysis in safety-critical real-time systems with multi-core architecture. The main contribution of this dissertation consists of proposing methods to verify and improve the E2E latencies of safety-critical real-time systems applying the Logical Execution Time (LET) communication paradigm. In this dissertation, we focus our analysis in the automotive domain, but the methods and analyses presented throughout this dissertation can also be applied to other domains where the LET paradigm is present.

The remaining sections of this chapter introduces the main concepts and challenges discussed throughout this dissertation. In Section I.1, we present the main research area of this dissertation. In Section I.2, we discuss about the main scheduling policies and communication paradigms currently considered by the automotive industry. In Section I.4, we formulate our problem statement and briefly present the main contributions of this dissertation. In Section I.5, we present a complete overview of the remaining chapters of this dissertation.

I.1 Automotive Safety-Critical Systems

Over the past decade, the amount of electronic units present in a vehicle increased significantly, especially due to the fact that most of the safety-critical functionalities of a vehicle are now controlled by specific ECUs. Moreover, the fact that old hydraulic systems have been replaced by their modern drive-by-wire counterparts was another fact that contributed to this significant increase due to their improved control performance and safety. Although most of the electronic units present in a vehicle are related to safety-critical functionalities, not all of them have the same constraints. Some units might favor data throughput over respecting a given E2E latency constraint. An example of these difference in requirements are the *telematic control unit (TCU)* and *anti-lock breaking system (ABS)* systems. The former favors data throughput, while the latter favors shorter E2E latency.

Since 2003, applications in the automotive domain are developed in accordance to the *AUTomotive Open System ARchitecture (AUTOSAR)* standard [2]. As a global partnership between multiple automotive manufacturers, the AUTOSAR standard defines a specific software development methodology for designers, as well as a standardization of how the software architecture should be. As a result, AUTOSAR enables scalability allowing sustainable use of software across multiple platforms and vendors.

Following the AUTOSAR standard, designers structure modern automotive applications as *software components (SWCs)*, which consists of several *runnables* — smallest executable unit within a SWC — that are grouped into tasks depending on their execution period. The process of scheduling tasks is done by the *operating system (OS)* according to one of the policies available in AUTOSAR (See Section I.2). Designing safety-critical applications in embedded systems, such as in AUTOSAR, requires complex analysis for temporal properties, such as E2E latencies. Usually, during early design phases, designers abstract system semantics, e.g., scheduling algorithms, in order to reduce the complexity of the timing analysis. However, abstracting system semantics in this manner results in pessimistic E2E latencies as discussed in [3] by Matic and Henzinger.

When designing safety-critical applications in AUTOSAR, the timing analysis' complexity is directly dependent on the inter-task communication model adopted by the tasks (See Section I.2). That is, the complexity of computing the E2E latencies of an ECU depends on when tasks access shared resources during their execution. Since a safety-critical functionality might require data from different applications running on multiple ECUs, the problem of performing E2E timing analysis becomes even more complex.

Nowadays, a vehicle can be seen as a distributed system consisting of multiple ECUs connected via a network. As defined by Tanenbaum and Maarten [4], a distributed system represents a set of independent computing devices that appears as a single device to its end user. In the automotive domain, the controller area network (CAN) bus [5] is used as the main inter-communication medium for ECUs. Figure I.1 illustrates an automotive distributed safety-critical system consisting of three ECUs connected via a CAN network. As an introductory example, let us focus on $ECU1$ and its applications as shown in Figure I.2.

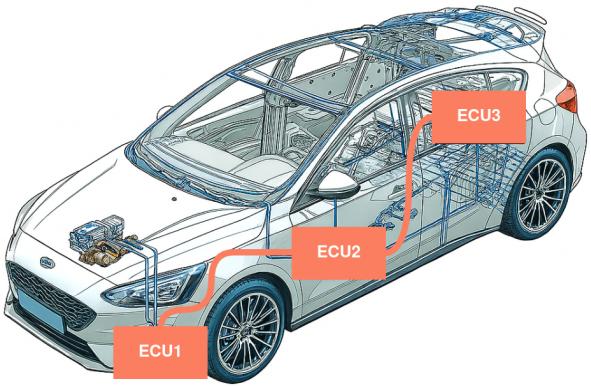


Figure I.1: *Distributed automotive system with three electronic control units*

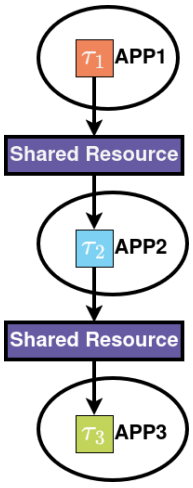


Figure I.2: *Set of applications within electronic control unit $ECU1$*

In general, a modern ECU has multiple cores that run a set of independent applications consisting of one or more tasks. In order to ease the understanding and explanation, let us assume $ECU1$ is a multi-core ECU containing three applications ($APP1, APP2, APP3$) consisting of a single tasks with real-time requirements. Let us also assume that each application is allocated to a dedicated core and that $APP1$ in $ECU1$ runs a sensory application for object detection. Task τ_1 in $APP1$ collects data of the environment around the vehicle and stores the data in a shared variable that is accessible by $APP2$, which runs a digital processing application. Task τ_2 in $APP2$ stores the outputs of its computations in a shared variable that is accessible by $APP3$, where an actuator task τ_3 is being executed. As a result, τ_3 is the task responsible for performing a specific functionality of the system based on the data sensed by task τ_1 and processed by τ_2, e.g., adaptive cruise control (ACC). This chained sequence of tasks executing and propagating data from one to another can be represented as a *task-chain* (See Section I.3). Note that although the tasks running on $ECU1$ are independent from each other and might respect their individual time constraints, from the system's functionality point of view that is not enough since the final system functionality depends on the time taken (E2E latency) for data to propagate through the entire task chain.

In order to verify whether or not system functionalities respects their E2E latency constraints, timing analysis methods should consider when tasks communicate with each other (access time to shared resources) and not only their individual deadlines. Although straightforward for the example described above, the problem of verifying E2E latency constraints becomes non-trivial once applications have multiple tasks that might be allocated to different cores and be executed in parallel. In Section I.2, we discuss about how scheduling and communication mechanisms can influence the timing analysis of automotive safety-critical systems.

I.2 Scheduling and Communication

According to the AUTOSAR standard, when designing real-time (RT) systems in the automotive domain, designers should consider specific scheduling properties and communication paradigms. In Chapter II, we discuss in detail the concepts of a real-time system (task, job) as well as other scheduling policies and communication paradigms that are commonly used in the industry.

I.2.1 Scheduling Policies and State Models

In AUTOSAR, tasks are scheduled by the OS according to fixed-priority. That is, each task has a static priority level [6]. During runtime, the task with the highest priority is selected by the OS to be executed (for more details about fixed-priority scheduling see Chapter II). Depending on the designer's needs, the OS can allow tasks to run for their completion, i.e., in a non preemptive manner, or, they can be preempted during execution by other higher priority tasks. Another policy presented in AUTOSAR is called *cooperative scheduling*, where a non-preemptive task informs the OS when it can be preempted if needed.

As a computing process, a task can be modeled as a finite-state machine (FSM) and have different states depending on the parameters of the system at a given time point. In AUTOSAR, a task can be classified as: (i) *basic task* or (ii) *extended task* [6]. Figure I.3 shows the possible states and transitions of a basic task.

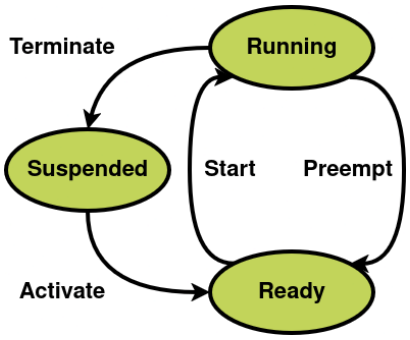


Figure I.3: *Finite-state machine of a basic task*

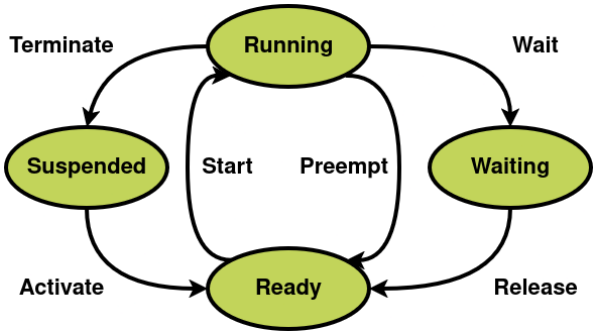


Figure I.4: *Finite-state machine of an extended task*

As shown in Figure I.3, a basic task has three states: (i) suspended, due to task's termination or waiting for the next activation, (ii) ready, waiting to have the highest priority, (iii) running, being executed by the OS. As shown in Figure I.4, an extended task has an additional *waiting* state compared to its other form. The addition of an extra state allows the task to wait for the occurrence of an specific event before resuming its execution.

In AUTOSAR, task can be activated by the OS in three ways: (i) periodically, (ii) sporadically, or (iii) single activation. As the name suggests, periodic tasks are activated by the OS after an specific time interval. In the context of AUTOSAR, sporadic tasks are directly related to specific angles and speed of the crankshaft [7]. The activation period is given by:

$$Period = \frac{120}{rpm * nCyl} \tag{I.1}$$

where rpm stands for rotations per minute and $nCyl$ represents the number of cylinders in the crankshaft. Single activation tasks are activated by OS only once, usually during system startup or termination.

Depending on the combination between priority and model, the execution time of a task may vary drastically during runtime, specially when other task are running in parallel and competing for shared resources. Since inter-task communication directly depends on when a task executes, the AUTOSAR standard adopted different communication paradigms over the years to ease the development and analysis of safety-critical applications.

I.2.2 Communication Paradigms

Inter-task communication represents the process of sharing data through a given medium between two or more tasks. In such process, a task can either act as a *writer* or as a *reader*. The current version of the AUTOSAR standard considers three possible communication paradigms: (i) explicit, (ii) implicit, and (iii) logical execution time [6]. The main differences between the three paradigm lie in how they handle their local data and on when they access their shared resources. Below, we briefly describe the three communication paradigms, while in Section III.1, we formally define them.

Explicit Communication

In the explicit communication paradigm, a task does not keep a local copy of any data that it might need during its execution. That is, every time the task needs to read or write (R/W) a given data, it has to access the global shared variable storing it, which may lead to data inconsistency. Figure I.5 illustrates the behavior of two tasks applying the explicit communication paradigm. Note that in Figure I.5, $Task1$ works on two different data values during its execution due to the write-event made by $Task2$ while $Task1$ was preempted.

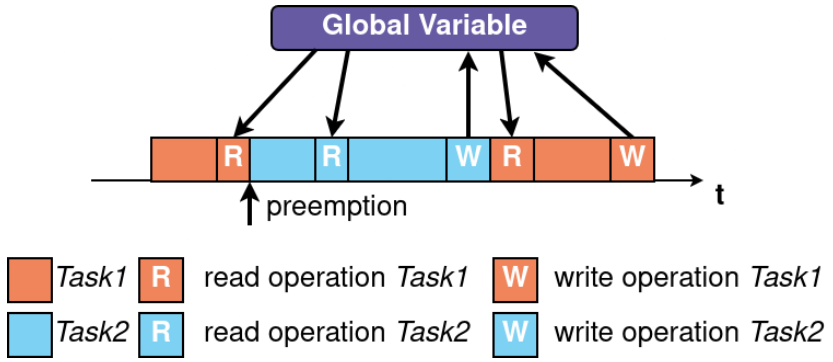


Figure I.5: *Explicit communication paradigm*

The main advantage of the explicit communication paradigm is its simplified logic and low implementation overhead. However, due to the unpredictable access pattern of tasks to global shared resources, the explicit paradigm is usually limited to scenarios where shared variable can only be read and not written.

Implicit Communication

The implicit communication paradigm, also known as *read-execution-write* paradigm, enforces that tasks copy at the start of their execution all the data they might require to complete their execution. That is, tasks have local copies of all the shared variables they access during runtime. By enforcing tasks to have local copies, the implicit communication paradigm solves the issue of data inconsistency present in the explicit

communication paradigm and reduces probability of suffering contention while accessing shared variables. Once tasks finish their execution, they copy their output from the local variable to the global. Figure I.6 illustrates the behavior of two tasks applying the implicit communication paradigm.

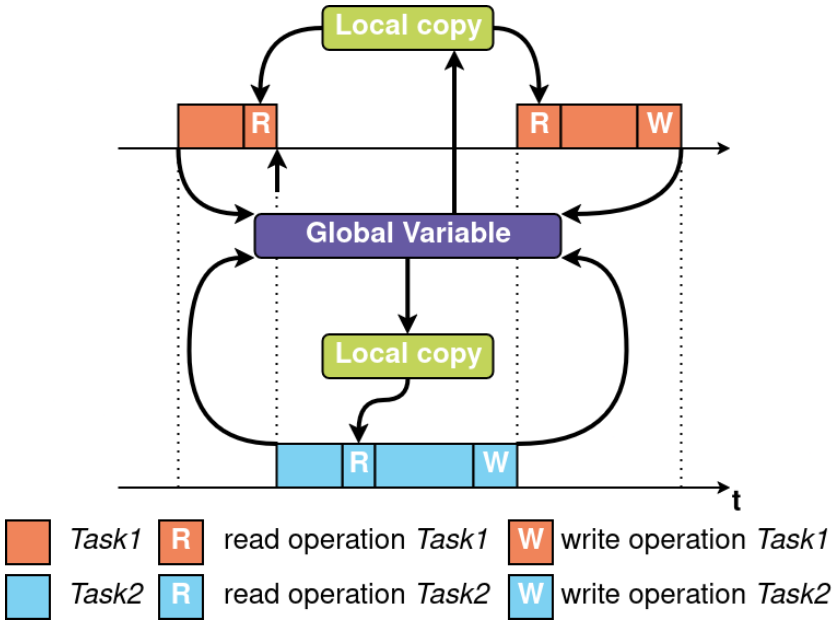


Figure I.6: *Implicit communication paradigm*

Note that in Figure I.5, even when preempted by $Task2$, the data value accessed by $Task1$ and used during its execution remains the same all the time, i.e., there is no data inconsistency during $Task1's$ execution. The main advantage of the implicit communication paradigm is that it solves the main issue present in the explicit communication paradigm, which was data inconsistency. However, since R/W-events from/to shared resources are directly dependent of when tasks start and end their execution, the E2E latencies between two communicating tasks remain non-deterministic, i.e., they might vary from one instance to another. That is, the E2E latencies can not be computed precisely and timing analysis methods can only provide a pessimistic upper bound.

Logical Execution Time - LET

The LET paradigm emerged originally as part of Giotto, a time triggered programming language proposed by Henzinger et al. [8] for embedded systems. In the LET paradigm, task's communication is decoupled from task's actual execution. That is, inter-task communication happens independently of when tasks start and end their execution, which enables timing and data-flow determinism during inter-task communication. In LET, tasks only perform their R/W-events at the boundaries of the so-called communication interval, whose length is considered equal to the period interval of the task. Figure I.6 illustrates the behavior of two tasks applying the LET paradigm.

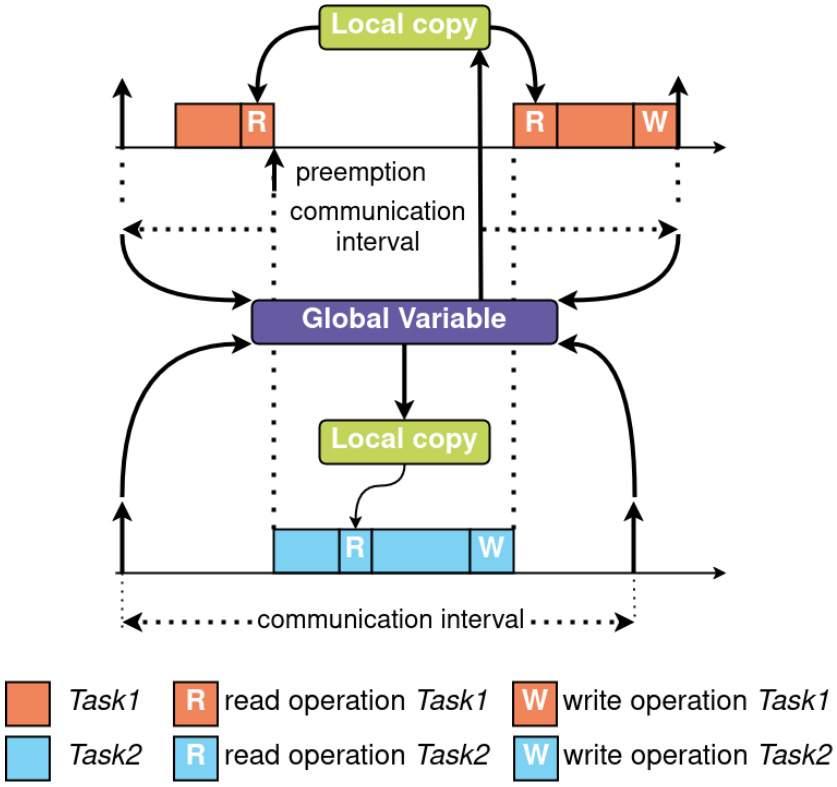


Figure I.7: *Logical execution time paradigm*

Note that in Figure I.7 due to the LET paradigm, it does not matter when or for how long $Task1$ and $Task2$ execute, the points in time when inter-task communication (access to shared resources) happens are always the same. By knowing exactly when shared resources are accessed by tasks, data can be easily traced and E2E latencies can be precisely computed. However, as observed by [3, 7], abstracting system semantics (scheduling choices) to ease timing analysis results in pessimistic E2E latencies, i.e., longer E2E latencies compared to the implicit communication paradigm.

In Section I.3, we discuss about how the different communication paradigms affect data propagation and consequently the E2E latencies of task-chains present in one or multiple ECUs.

I.3 Task Chains

As motivated during the introductory example in Section I.1, an automotive safety-critical system functionality can consist of multiple applications containing several tasks running on different cores. In order to achieve the desired system functionality, applications have to constantly communicate with each other by means of shared resources, which are constantly accessed by the tasks contained in each application. As data propagates from one task to another in an ordered manner, it creates a chained sequence of communicating tasks known in the RT literature as cause-effect chains (CECs).

A cause-effect chain CEC represents an ordered sequence of communications carried out between a finite set of tasks that are part of one or more applications. In the automotive domain, a sensor to actuator controlled application is a typical example of a CEC. It consists of a task that reads the sensor (*cause*), a task that processes the read value, and a task that writes to an actuator (*effect*). Since tasks might be part of multiple CECs, run in parallel, and access resources in a non deterministic manner, the analysis of whether or not the E2E latency between the cause and the effect fulfills required timing constraints is not trivial [7, 9]. The analysis' complexity increases even further as complex data dependencies may exist between tasks due to their different activation period rates, which results in a special type of CEC known as *multi-rate cause-effect chain*. Figure I.8 illustrates a multi-rate CEC consisting of three tasks applying the LET paradigm where the communication intervals are equal to task's period. The upside arrows in Figure I.8 represent a new activation of a task, while the other arrows represent data propagation paths through the CEC according to the LET paradigm. Note that in Figure I.8 each colored arrow represent a distinct data propagation path within the CEC. We formally define a CEC in Section III.1.1.

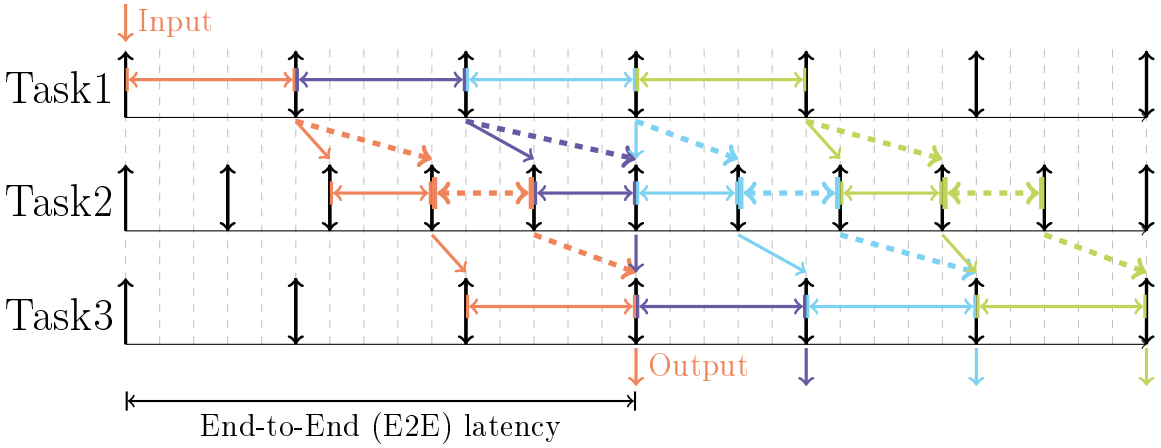


Figure I.8: *Multi-rate cause-effect chain with tasks applying the logical execution time paradigm*

Note that due to the multi-rate nature of the CEC in Figure I.8, data might be under/over-sampled while being propagated (e.g., the paths with dotted lines). As a result, not all instances of the tasks are involved in the process of data propagation, which in turn wastes processing resources during runtime.

In the automotive domain, the two most common E2E metrics analyzed when performing the timing analysis of a multi-rate CEC are: (i) *reaction time*; (ii) data age, also known as *First to First* and *Last to Last* semantics respectively in [10] (See Section III.1.2). The reaction time measures system's reactivity to a given input, that is, *how long does it take for an input to traverse the CEC for the first time assuming the maximum sampling delay.* The data age measures the *freshness* of the output data, that is, *for how long a given input influences the outputs.* Figure I.9 exemplifies the reaction time and data age metrics for a CEC consisting of two tasks (*Task1, Task2*), where both have a worst-case execution time of one time unit and apply the LET communication

paradigm. Note that in this example *Task2* has the highest priority. Although Figure I.9 only shows the reaction time and data age of the *orange* and *purple* data, each other color data has its corresponding reaction time and data age values.

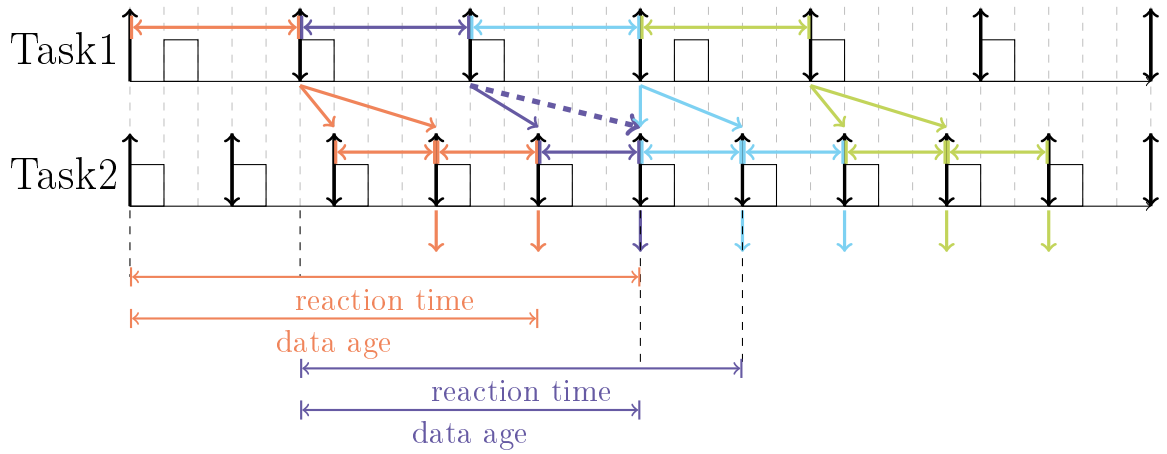


Figure I.9: *Graphical representation of the reaction time and data age latency metrics*

Figure I.9 shows that although both tasks have a short response time — time interval between release and completing execution — the output jitter caused by the LET paradigm results in pessimistic E2E values.

I.4 Problem Statement and Contributions

As discussed throughout this chapter, ensuring the correct functionality of safety-critical applications in a multi-core system requires more than just guaranteeing that individual task's deadlines are met during runtime. For safety-critical applications, the time required for data to traverse through a cause-effect chain is also essential and has to be taken into account during timing analysis since it directly affects the E2E latencies of the applications. Given that a safety-critical application may contain several tasks, which may have different activation periods (multi-rate) and might run in parallel, the problem of guaranteeing E2E latencies of multi-rate cause-effect chains becomes non-trivial.

Although LET facilitates the timing analysis of multi-rate cause-effect chains during early design stages, it also introduces unnecessary pessimism in the form of longer E2E latencies. As a result, the application's performance is severely compromised and sometimes requires applications to be fully redesigned in order to keep the E2E latencies below a given time constraint. Moreover, the redesign of complex multi-rate CECs might increase the side effects of data being under/over-sampled, which, in turn, might increase the waste of processing resources during runtime.

This dissertation focuses on proposing, from a theoretical and practical point of view, methods to reduce the long E2E latencies present in multi-rate cause-effect chains applying the LET paradigm, as well as the waste of processing resources during runtime.

The main contributions of this dissertation are listed in the sections below:

I.4.1 Safe Reconfiguration of Communication Intervals

By abstracting from system implementation the LET paradigm brings many benefits to the timing analysis of multi-rate CECs during early design phases. However, as explained in I.2.2, abstracting system semantics in this manner results in unnecessarily longer E2E latencies, which in some cases might require a complete re-design of the whole cause-effect chain. In Section IV.2, we show that by correctly configuring tasks' communication intervals, the E2E latency of multi-rate CECs can be significantly improved while maintaining all the benefits of the LET paradigm.

Our method extracts information from a feasible schedule to derive new boundaries for tasks' communication intervals. By postponing read-events and preponing write-events, our method allows data to propagate through different communication paths, which ultimately result in significantly shorter E2E latencies. In Section IV.2, we also shown that since tasks might belong to multiple CECs not all communication intervals have to be reconfigured in order to reduce the overall E2E latencies.

We propose an analytical method to safely reconfigure at once the communication intervals of tasks without comprising the E2E latencies of the CECs present in the system. By safe reconfiguration we mean that our method guarantees that task instances are always executing within their specified communication intervals. That is, E2E latencies are deterministic and can be easily verified at any point in time.

I.4.2 Timing Analysis of Asynchronous Cause-Effect Chains

As further discussed in Section III.2, the literature about performing timing analysis of asynchronous CECs applying the LET paradigm is scarce and limited to cases where the length of tasks' communication intervals are equal to tasks' period interval. As we propose in Section IV.2 a method to reconfigure the communication intervals of tasks applying the LET paradigm, a new set of analytical equations to compute E2E latencies have to derived since the ones available in the literature are no longer applicable (they don't consider the possibility of reconfigured intervals).

In Section IV.3, we demonstrate analytically and by means of theorems, how to compute precisely the points in time when data propagates throughout a multi-rate CEC containing tasks with reconfigured communication intervals. Based on the propagation points of a CEC, our method recursively computes the reaction time and data age of each propagation path present in the CEC. That is, our method extends the state of the art by providing more general equations that can be used to compute the reaction time and data age of (a)synchronous multi-rate CECs with(out) reconfigured communication intervals.

In Section IV.7, we present a framework where designers can perform the timing analysis of a task set and the CECs present in it. For each CEC, our framework lists all the possible propagation paths as well as the maximum reaction time and data age of each CEC.

I.4.3 Schedule Manipulation to Improve End-to-End Latencies

Although we demonstrate in Section IV.2 that the E2E latencies of multi-rate CECs applying the LET paradigm can be reduced, they are still not optimal and can be further reduced in some cases. As later discussed in Section III.2, the literature about optimizing the E2E latencies of tasks with reconfigured communication intervals was non-existent before the proposal of our method.

In Section IV.4, we demonstrate how minor manipulations on the schedule can further reduce the E2E latencies of CECs containing tasks with reconfigured communication intervals. We propose a method that uses precedence constraints to manipulate and control the execution pattern of a given set of tasks. By manipulating when specific task instances start and end their execution, our method modifies the parameters used in Section IV.2 to define boundaries of the communication intervals. As a result, new interval configuration are obtained depending on which precedence constraints our method establishes between the tasks.

We propose in Section IV.7 a framework where designers can specify which multi-rate CECs should have their E2E latencies further reduced by means of precedence constraints. The proposed framework also allows designers to limit the number and choose which tasks can receive precedence constraints. Our framework uses a searching heuristic function (See Section IV.8) to guide the search for the set of precedence constraints that best fits the designer's requirements.

I.4.4 Improving System Utilization

One of the benefits of the LET paradigm is that inter-task communications only happen at the boundaries of tasks' communication intervals. As a result, it is possible to trace data propagation in a deterministic manner throughout the multi-rate CECs. Given that the resources used for inter-task communication may suffer from data oversampling, not all outputs produced by task instances ultimately affect E2E latency values. That is, their output are overwritten before being consumed, i.e., read, by the next task in the chain. Since not every task instance affects the E2E latencies of multi-rate CECs, the execution of such instances only wastes processing resources (See Figure I.8). By skipping the execution of task instances that are not affecting the E2E latencies, it is possible to reduce system utilization without affecting the output, i.e., E2E, behavior of the multi-rate CEC.

In Section IV.5, we propose a set of methods to identify which task instances can be skipped. Since tasks may belong to more than one CEC and data propagation directly defines which task instances can be skipped, we propose a method that manipulates communication intervals in order to maximize the number of jobs that can be skipped and minimize system utilization. We use the framework proposed in Section IV.7 to find the set of communication intervals that best fits our objective of decreasing system utilization while potentially reducing E2E latencies.

I.4.5 Improving End-to-End Latency Feasibility in Multi-Execution Mode Systems

Safety-critical control applications often need to operate under multiple system modes during runtime. A transition from one mode to another may cause new tasks to join the system which might increase the E2E latency of certain CECs within the system. As later discussed in Section III.2, a research gap exists in the literature since neither the original LET model nor existing E2E optimization methods based on it (including our approaches in sections I.4.1 and I.4.3) account for system mode change during runtime.

In Section IV.6, we propose a method to increase the feasibility of meeting E2E latency requirements of multi-rate CECs in multi-execution mode systems. By means of process migration, our method selectively migrates task instances in order to validate the timing constraints of CECs that had their latencies affected after a mode change. For every CEC that violates its E2E latency requirements, our method identifies which tasks of that CEC must have their communication intervals reconfigured. The reconfiguration process takes place by migrating to candidate cores a set of specific task instances, which are selected based on the amount of interference they suffer in the current mode.

In Section IV.6, we show how to ensure that selected task instances completely execute within an specific interval and how we ensure that the E2E latencies of the tasks currently allocated in the candidate core remains unaffected.

I.5 Dissertation Outline

The remaining chapters of this dissertation are organized as follows:

- In Chapter II, we present the concepts and scheduling theory of single/multi-core real-time systems as well as the task models most used in the literature. We also present the concepts and properties of shared resources, offline scheduling approaches, as well as load balancing techniques such as task migration.

- In Chapter III, we do literature review in the broaden area of timing analysis of multi-core safety-critical real-time system. We discuss about the current state of the art and position our work with respect to it. In Chapter III, we also present the challenge that inspired most of the methods proposed in this dissertation.

- In Chapter IV, we present the main contributions of this dissertation. In Section IV.2 we present of our Schedule-Aware communication model, while in Section IV.3 we demonstrate how E2E latencies can be computed for the new method. In Section IV.4, we show how E2E latencies can be further reduced by adding specific precedence constraints to manipulate task's communication intervals. In Section IV.5, we show how our Schedule-Aware communication model can assist in minimizing system utilization by skipping the execution of task instances that don't affect the E2E latencies of CECs with tasks applying the LET paradigm. In Section IV.6, we demonstrate how to increase the feasibility of multi-rate CECs that apply the LET model to meet their E2E latency requirements after a mode

change. In Section IV.7, we propose a framework to assist designers on how to properly configure the communication intervals of LET tasks. In Section IV.8, we propose a heuristic function for our framework.

- In Chapter V, we evaluate our work using the automotive benchmark proposed by BOSCH [11] as well as synthetic task sets.
- In Chapter VI, we conclude this dissertation and present possible research directions to continue our work.

Chapter II

Real-Time Systems

For most types of computing systems, behaving according to definitions means that a correct computation value was produced by the system (logical correctness). However, for systems classified as real-time (RT), the point in time when the computed value was produced is also an important behavioral metric (temporal correctness). The quality of a RT system can be determined based on its compliance to the logical and temporal correctness of the data produced. That is, a RT system must deliver the expected logical value within defined timing constraints, i.e., before the assigned deadline.

The airbag system present in vehicles is a classic example of a RT system in the literature. During a collision, the airbag system must not only inflate the bag that will protect the driver's head, but it must also inflate at the right instant. For example, if the airbag inflates too early during the collision, it will not provide the necessary cushioning to protect the driver's head. If the airbag inflates too late, the driver will hit his head against the vehicle and the airbag becomes a useless protection feature.

In the literature, RT systems can be classified in two categories: (i) soft real-time, and (ii) hard real-time. In a soft RT system, *some* deadline are allowed to be missed without causing system's failure. However, the quality of the outputs produced by the system decreases significantly over time if those deadline misses continue to occur. In a hard RT system, *all* the deadlines and timing constraints have to be respected, otherwise the system fails and it might endanger human lives in case of a safety-critical system. In order to guarantee that no deadlines are missed during runtime, a schedule that enforces which and when tasks should execute has to be defined *offline* or *online*, i.e., before or during runtime.

Most RT systems define their schedules online according to a predetermined scheduling policy (See Section II.2). In such systems, known as event-triggered (ET) RT systems, significant events such as task's releases or completion trigger the activation of the operating system (OS)'s scheduler. In order to create the schedule that will be followed by all the tasks present in the system, the OS scheduler requires deep knowledge of task's properties such as how often they activate and for how long they might execute. Based on those properties and on the chosen scheduling policy, the scheduler decides which task should be executed at every time a significant event occurs in the system.

Unlike ET RT systems that define their schedules online, the RT systems known as time-triggered (TT) RT systems have their schedules defined during the design phase,

i.e., offline (see Section II.4). That is, during runtime, no scheduling decision is taken by the OS's scheduler, all the decisions were already planned and decided before system's deployment. Instead of making decisions online, a TT RT system follows a *scheduling table* containing all the necessary information regarding when and which task should be executed as time progresses. Unlike a ET RT system where significant events were responsible for triggering the OS's scheduler, a TT RT system adopts a global notion of time, where the progression of time is what triggers the OS's scheduler. As time progresses and matches the time points defined in the scheduling table, the OS's scheduler dispatches the task corresponding to a specific time point as described in the scheduling table.

Throughout the remainder of this chapter, we will properly define the concepts of tasks and present an overview of the scheduling theory for single/multi-core real-time systems. In Section II.1, we present the terminology for tasks and its instances (jobs), as well as the task models most used in the literature. In Section II.2, we discuss classical scheduling algorithms for single/multi-core systems. Section II.3 provides an overview around the concept of shared resources, and how they are managed in real-time systems.

II.1 Tasks and Jobs

Task is the elementary term used in the realm of RT systems to describe a process that contains a sequence of instructions to be executed by the system's processor. Usually, a real-time application consists of several tasks. This collection of tasks from one or multiple applications is known as *task set*. During runtime, a task can be activated multiple times and at each activation an instance of the tasks is created. Once activated, instances that are ready to execute stay in a queue and wait to be selected by the scheduler to be executed. In the realm of RT systems, the term *job* is used to describe an instance of a task that is released at a given time point and should be executed before its deadline.

Depending on their functionality and implementation, tasks might have different parameters. Each parameter specifies a specific characteristic of the task, e.g., release time, deadline, execution time, etc. Those parameters are used by the scheduler or designer to decide when each task should be executed depending on the RT system type (ET or TT). Below we list and describe some common task's parameters:

- *worst-case execution time (WCET)* : represents the *maximum* interval of time a task might require to execute completely without interruption over all possible input data. The WCET value may be platform dependent, since it is an upper bound of the execution time of all jobs in a given system platform. Therefore, its value might change significantly when measured across different hardware platforms.

- *best-case execution time (BCET)* : represents the *minimum* interval of time a task might require to execute completely without interruption over all possible input data. It is the counter part of WCET.

- *Phase* : represents the point in time when the first job of the task should be released and available to be scheduled by the scheduler.
- *Deadline* : represents a point in time relative to task's release. A deadline indicates the latest point in time a task can finish its execution without compromising its timing requirements. That is, after its release, each job should finish its execution by the defined deadline.
- *Priority* : represents the importance of a task from the scheduler's point of view, i.e., the higher the priority, the higher the chance of being chosen by the scheduler as the next task to execute. For ET systems, the scheduler's scheduling policy is the one responsible for assigning priorities to tasks (See Section II.2), while for TT systems, the priorities can be assigned offline by the designer.

II.1.1 Task Models

A task model represents an abstraction methodology used to describe temporal requirements of tasks. In the realm of RT systems, tasks can be classified into a task model according to their activation pattern. The most common task models present in the RT system literature are: (i) the *periodic* model, and (ii) the *sporadic* model. First defined by Liu and Layland in [12], the periodic task model assumes that tasks release jobs periodically according to a given activation rate. Therefore, during runtime, a infinite sequence of jobs are released by each task according to their activation rate. Below, we define the *Period* parameter of a task according to the periodic task model.

- *Period* : represents the minimum intervals of time between the release of two consecutive jobs by a given task.

The periodic task model proposed by Liu and Layland in [12] assumes that the deadline of a task is equal to its period. That is, all jobs should finish their execution no later than the release of the next job of the same task. It also assumes that tasks are independent from each other, do not overrun and can suspend themselves. In the literature, there are variations of this model with slightly differences such as when tasks have deadline different than their period.

The sporadic task model was first proposed by Mok in [13], and it assumes that the activation interval between two jobs of the same task is not fixed and can vary during runtime. That is, there is no upper bound on when the next job of a task might get released. However, the sporadic task model assumes a lower bound, i.e., a minimal time interval between the release of two consecutive jobs of the same task. For the sake of timing analysis, the maximum demand generated by a sporadic task will happen when the jobs are always released according to the lower bound. In this scenario, the sporadic task behaves as a periodic task, where the period is the minimum inter arrival time of two consecutive jobs.

Although not a proper task model given its nature, the release pattern of tasks can also be modeled as *aperiodic*. That is, they can be released at any point in time, there is no lower or upper bound to limit when new instances of the aperiodic task might arrive in

system during runtime. As further explained in Section II.2, aperiodic tasks are treated using a special procedure during runtime and are executed or dropped depending on capabilities of the scheduling policy applied by the scheduler.

II.2 Scheduling

In the context of RT systems, *scheduling* is a decision making process where the scheduler decides which job from a set of ready tasks should be executed according to a given scheduling policy. The end result of this decision making process is a *schedule*, which is a particular execution sequence of the tasks present in the system. In the literature, a scheduler can be classified as: (i) *preemptive*, or (ii) *non-preemptive* depending on its implementation. During runtime, a scheduler classified as preemptive *can* preempt the execution of a job in order to serve another job that just arrived in the ready queue. On the contrary, jobs executing under the control of a non-preemptive scheduler *cannot* be preempted, and as a result, those jobs must run until completion every time they start to execute. Once selected as the next job to be executed by the scheduler, all the resources required for the correct execution of the selected job are assigned by the OS. In Section II.3, we further discuss about the resources needed by jobs during their execution and how mutual accesses can be controlled.

According to the literature, for a periodic task model a schedule is said to be *feasible* if all tasks respect their timing constraints. In the same manner, a task set is said to be *schedulable*, if there is at least on scheduling policy that produces a feasible schedule during the hyperperiod (HP). The HP represents an interval of time in which the schedule starts to repeat itself, i.e., task's release pattern start to repeat. The length of a HP can be obtained by computing the least common multiple (LCM) of task's period.

Over the last decades, different scheduling algorithms were proposed in the RT system literature. A scheduling algorithm can be defined as a systematic approach that assign processing resources to computing tasks as time progresses. Initially, the only difference between those scheduling algorithms was how they assigned priority and if it was dynamic or static during runtime. Since the advent of multi-core systems, scheduling algorithms started to be classified according to the system platform they are targeting (single/multi-core).

In Section II.2.1, we discuss about the most common scheduling algorithms for single-core (uniprocessor) systems. In Section II.2.2, we expand our analysis to also include the most common scheduling algorithms for multi-core (multiprocessor) systems.

II.2.1 Uniprocessor Scheduling

The first real-time scheduling algorithms were proposed between the late 60's and early 70's [14, 15, 16, 12], being the work by Liu and Layland in [12] definitely the most prominent among all of them.

Rate-Monotonic

As mentioned in Section II.1.1, Liu and Layland [12] were the first to propose the periodic task model and methods to check task's schedulability. In [12], the authors propose a task-level fixed-priority preemptive scheduling algorithm named Rate-Monotonic (RM), which is still one of the most widely used scheduling policies given its efficiency and ease of implementation. The RM algorithm has its name due to how priorities are assigned to tasks by the scheduler. In RM, the priority of a task is directly related to its activation rate. That is, the shorter the period, the higher the priority of the task. In Appendix A.1, we provide an example of a task set scheduled according to the RM scheduling policy.

In order to check the schedulability of a task set according to a given scheduling algorithm, two conditions have to be checked. The first is known as the *necessary condition*, while the second one is known as the *sufficient condition*. According to the literature [12], if a task set fails the necessary condition of a given scheduling algorithm, that means the task set is definitely not schedulable according to that algorithm. If the task set passes the necessary test, it means the task set might or not be schedulable. The sufficient condition, is an additional test to check weather a task set is schedulable or not. If a task set passes the sufficient test, that means the task set is definitely schedulable. On the other hand, if the task set passed the necessary test, but failed the sufficient test, further investigation by means of other tests, e.g., the response-time analysis (RTA) test [17, 18], has to be done in order to verify task set's schedulability. The RTA test can also be used in scenarios where task's deadline is less than their period. The necessary and sufficient tests for a RM are shown in equations II.1 and II.2 respectively.

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \tag{II.1}$$

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{1/n} - 1) \tag{II.2}$$

The necessary test for RM checks whether the total utilization of the task set is less or equal to one. In Equation II.1, the term n represents the number of tasks in task set, while C_i represents the WCET of task i, and T_i its period. As shown by Liu and Layland [12], Equation II.2 provides an schedulability bound for RM according to the utilization of the task set. A less pessimistic bound for RM known as the *hyperbolic bound* was later proposed by Bini et al. [19, 20]. The new test (shown in Equation II.3) has the same complexity as the original, but provides a tighter bound allowing task sets that were originally reject to be accepted. Bini et al. [19, 20] showed that the new bound is optimal, meaning the no better bound can be achieved.

$$\prod_{i=1}^{n} (\frac{C_i}{T_i} + 1) \leq 2 \tag{II.3}$$

Deadline-Monotonic (DM) is a priority ordering scheduling algorithm with a similar concept to RM and it was initially proposed by Leung et al. in [21]. In DM, priorities are assigned to tasks by the scheduler inversely proportional to the length of their deadline. That is, the shorter the deadline, the higher the priority of the task. As shown by Leung et al. in [21], DM is an optimal scheduling algorithm for static priority task sets. The DM scheduling policy covers a wider range of task sets, but compared to RM, it requires a more complex implementation. In Appendix A.2, we provide an example of a task set scheduled according to the DM scheduling policy.

Earliest-Deadline First

Earliest Deadline First (EDF) is a well known and widely used job-level fixed-priority scheduling policy proposed by Liu and Layland [12]. In EDF, tasks' priority change during runtime depending on the absolute deadline of their jobs. For this reason, EDF is also known as a task-level dynamic-priority scheduling policy. According to EDF, upon scheduler invocation (due to the release or completion of a job), the job with the earliest absolute deadline will be chosen by the scheduler as the next job to execute. Unlike RM that has requires two different tests to check the schedulability of a task set, EDF only requires one test. The necessary and sufficient test for EDF is the same as the necessary test for RM, i.e., Equation II.1. In Appendix A.3, we provide an example of a task set scheduled according to the EDF scheduling policy.

As shown by Buttazzo in [22], performance wise, the EDF scheduling policy generates less runtime overhead than RM. Moreover, Buttazzo shows that if tasks' deadlines are equal to periods, for EDF, exact schedulabilty analysis can be performed in $O(n)$, while for RM it can be performed in pseudo-polynomial time. For scenarios where tasks' deadlines are less than periods, the analysis is pseudo- polynomial for both scheduling policies. For task sets with tasks which have their deadlines different from their periods, the necessary and sufficient test shown in Equation II.1 is no longer applicable. In [23], Baruah et al. proposed a processor demand test to verify the schedulability of task sets containing tasks with deadlines less than their period according to the EDF scheduling policy. The processor demand of a task within a given interval of time $[t1, t2]$ is the amount of processing time required its jobs that are activated and must be completely executed in $[t1, t2]$. Equation II.4 shows the processor demand required by a given task i within the interval $[t1, t2]$.

$$g_i(t1, t2) = \sum_{r_{i,k} \geq t1, d_{i,k} \leq t2} C_i \tag{II.4}$$

In Equation II.4, term $r_{i,k}$ represents the release time of the k^{th} instance of task i, where $k \in \mathbb{N}^+$, while $d_{i,k}$ represents the absolute deadline of the k^{th} instance of task i. Equation II.5 shows the processor demand for the whole task set.

$$g(t1, t2) = \sum_{i=1}^{n} g_i(t1, t2) \tag{II.5}$$

As shown by Baruah et al. [23], the schedulability of the task set can be guaranteed *if* and *only if* in any interval of time the processor demand generated by the task set is below the available processing time. That is,

$$\forall t1, t2 \quad g(t1, t2) \leq (t2 - t1) \tag{II.6}$$

Therefore, for a synchronous periodic task set, i.e., the phase of all tasks are equal to 0, the schedulability of the task set can be guaranteed if Equation II.7 is always respected.

$$\forall L > 0 \quad \sum_{i=1}^{n} \left\lfloor \frac{L + T_i - D_i}{T_i} \right\rfloor C_i \leq L \tag{II.7}$$

In [23], Baruah et al. showed that the feasibility test shown in Equation II.7 can be simplified by bounding the test interval and reducing the number of points in time that have to be checked. Baruah et al. showed demonstrated that for a synchronous periodic task set, Equation II.7 only has to be verified at time points equal to tasks' absolute deadlines and that are within the interval $0 < L < HP$.

II.2.2 Multiprocessor Scheduling

Before the advent of computing systems with multiple cores, uniprocessor scheduling was relatively easy and intuitive since the basic rule was that only one task could execute at a time in the available processing unit. However, as multi-core systems started to become more common in commercially available off-the-shelf (COTS) products, the challenges related to the development of real-time applications for those systems have also become more common and evident for the designers.

Although extra processing resources are available in multi-core systems, it is not easy and intuitive for the designers how those additional resources should be exploited in order to extract the maximum performance while guaranteeing real-time requirements. As stated by Liu in [24]:

> "Few of the results obtained for a single processor generalize directly to the multiple processor case; bringing in additional processors adds a new dimension to the scheduling problem. The simple fact that a task can use only one processor even when several processors are free at the same time adds a surprising amount of difficulty to the scheduling of multiple processors."

The direct applicability of well known real-time scheduling algorithms such as RM and EDF proved to be sub-optimal in terms of resource utilization when tasks are not statically allocated to a specific core [25]. The work done by Dhall and Liu in [25] showed that in the worst case, the utilization bound for global RM/ EDF on M processors is 1 and not M. This became known as the "*Dhall Effect*" and for decades lead to the belief that global scheduling was inferior to partitioned scheduling. The *PFair* algorithm proposed by Baruah et al. in [26] remains as the only known optimal method for scheduling periodic tasks on multi-core systems. In PFair, task progression occurs

proportionally to its utilization. That is, the scheduler divides the timeline into equal length quanta (slot), and at each quanta the scheduler selects tasks such that the accumulated processor time allocated to each task is proportionate fair [26]. Over the years, different authors proposed variations for the PFair algorithm [27, 28, 29], being the work done by Anderson and Srinivasan in [27] one of the most prominent since it extends the PFair algorithm to also handle sporadic tasks.

Traditionally, *global* and *partitioned* scheduling have been the two existing approaches to perform task allocation in multi-core systems. In global scheduling, there is a single priority-ordered queue that stores all the jobs ready to execute. Upon the release or completion of a job, the OS activates its scheduler, which selects the M jobs with the highest priority to be executed. Due to the priority reasons, e.g., the release of a higher priority job, some jobs might have their execution preempted in a given core and resumed in a different core later in time when their priority become high enough again to be selected by the scheduler. That is, each instance of a task might execute on a different core, e.g., instances can *migrate* from one core to another during runtime.

In the realm of RT systems, migration is the name given to the act of not tying all the instances of a task to only execute on a specific core. Depending of the migration technique applied in system, the instances of a task can migrate at any point in time (*full migration*) or only at the boundaries of their execution (*job-level migration*). When a task instance starts its execution on a given core and due to a preemption event resumes its execution on a different core, it is said that the instance suffered full migration during its execution. That is, the whole context of the task had to be moved by the OS from one core to another. Job-level migration is the name given when different instances of the same task are executed on different cores, but once started on a core, the execution of an instance is performed on that core until completion, even if a preemption event occurs.

In partitioning scheduling, tasks are statically allocated to a specific core meaning that all the jobs of those tasks are always executed in the same core. As a result, the multi-processor scheduling problem reduces to a standard uniprocessor scheduling problem. However, despite the fact that statically allocating tasks to specific cores facilitates scheduling problem, it also leads to two negative side effects. First, the static allocation of tasks to cores is a bin-packing problem, i.e., NP-hard in the strong sense. Second, there are task sets that are schedulable under global scheduling algorithms, but not under partitioned [30].

As shown by Carpenter et al. in [30], the maximum utilization bound for a given task set in a multi-core system depends on two factors: (i) the allocation algorithm, and (ii) the priority assignment algorithm. Figure II.1 shows the different utilization bounds that can be obtained depending on the algorithms applied to the task set, where M represents the number of cores present in the system.

Global Fixed-Task-Priority Scheduling

As mentioned in Section II.2.2, the work done by Dhall and Liu in [25] showed that the utilization bound for RM under global scheduling (*global RM*) is $1+\varepsilon$, where ε represent

	Static (e.g., RM)	Job-level dynamic (e.g., EDF)	Unrestricted dynamic (e.g., LFF)
Full migration	$\frac{M^2}{3M-2} \leq U \leq \frac{M+1}{2}$	$\frac{M^2}{3M-1} \leq U \leq \frac{M+1}{2}$	$U = M$
Job-level migration	$U \leq \frac{M+1}{2}$	$M - U_{max}(M-1) \leq U \leq \frac{M+1}{2}$	$M - U_{max}(M-1) \leq U \leq \frac{M+1}{2}$
No migration	$(\sqrt{2}-1)M \leq U \leq \frac{M+1}{1+2^{\frac{1}{M+1}}}$	$U = \frac{M+1}{2}$	$U = \frac{M+1}{2}$

Figure II.1: *Utilization bound according to allocation and priority assignment algorithms*

a very small utilization value. This phenomenon occurs when a system with M cores has a task set with $M+1$ tasks, where M tasks have short periods and small utilization, and a single task that has long period and an utilization that is close to 1. In that scenario, the M tasks with short periods (higher priority) will be scheduled first causing the task with long period (low priority) to miss its deadline.

In [29], Andersson et al. proposed the *RM-US[M/(3M-2)]*, which is a static priority global scheduling algorithm based on the RM policy. Unlike global RM, the method proposed by Andersson et al. [29] can schedule any task as long as the total utilization is not greater than $M^2/(3M-2)$ and the individual utilization is less or equal than $M/(3M-2)$. In [31], Bertogna et al. tightened the bound for DM under global scheduling (global DM) and showed that the maximum bound is

$$U_{total} \leq \frac{m}{2}(1 - U_{max}) + U_{max} \quad , \tag{II.8}$$

where U_{max} is highest utilization among the tasks present in the task set. Bertogna et al. [31] showed that their bound also works for task sets with sporadic tasks. In [32], Andersson extended his previous work in [29] and proposed the so-called *SM-US(2/(3+$\sqrt{5}$))* algorithm. Compared to his previous method, the new *slack monotonic* algorithm proposed in [32] assigns priority to tasks based on the difference (*slack*) between task's period and WCET. The author showed that SM-US(2/(3+$\sqrt{5}$)) performs better than RM-US[M/(3M-2)] on systems where $M > 2$.

Global Fixed-Job-Priority Scheduling

In the literature, most of the research related to global fixed-job-priority scheduling algorithm has focused on sequential tasks [33]. As a result, multiple schedulability tests for EDF under global scheduling (*global EDF*) have been proposed. In [29], Andersson et

al. showed that for periodic task sets with implicit deadlines, the maximum utilization a task set can have is $(M+1)/2$. Srinivasan and Baruah proposed in [34] a hybrid method to assign priorities to tasks under global fixed-job-priority scheduling. In their new method, called *EDF-US[ς]*, the task with an utilization value greater than the threshold defined by ς will receive the highest priority in the system, while the other tasks will be scheduled according to EDF. The authors showed that by making $\varsigma = M/(2M-1)$, the maximum utilization bound for the task set is $M^2/(2M-1)$.

Similar to [34], Goossens et al. proposed in [35] an algorithm based on EDF where the k tasks with the highest utilization in the system have the highest priority. The authors named their algorithm as *EDF(k)* and proposed a sufficient schedulability test for it. In [36, 37], Baker improved the work done by Srinivasan and Baruah in [34] and presented multiple sufficient feasibility tests for systems with M cores under preemptive global EDF considering periodic and sporadic tasks with arbitrary deadlines. Baker showed that for EDF-US[ς], the optimal threshold for ς is 1/2 and not $M/(2M-1)$. The new threshold proposed by Baker, which is a tight bound, shows that the maximum utilization for any task set scheduled according to global EDF is

$$U_{EDF-US[1/2]} = (M+1)/2 \tag{II.9}$$

In [37], Baker proposed an algorithm called *EDF(k_{min})*, which is a modified version of EDF(k) proposed by Goossens et al. in [35]. EDF(k_{min}) computes the minimum number (k) of tasks that should have the highest priority based on their utilization so that Equation II.9 evaluate to true. Although the bounds obtained by Baker for EDF-US[ς] and EDF(k_{min}) are tight, EDF(k_{min}) dominates EDF-US[ς] based on the number of task sets that each one of them can schedule.

Hybrid Scheduling

As discussed in Section II.2.2, global scheduling approaches can lead to very high overheads during runtime, which can be an impediment for some systems. In order to take the advantage of load balancing techniques without occurring in higher execution times caused by task migration, over the years different authors focused their efforts in proposing hybrid approaches that combine the advantages of partitioned and global scheduling. Among the different approaches available in the literature for hybrid scheduling, all of them can be classified either as: (i) *semipatitioned*, or (ii) *clustering* approaches.

Semipatitioned algorithms aim in reducing the fragmentation caused by fully patitioned systems by splitting a small number of tasks among the cores, so that the spare capacity available in those core could be better utilized. The *EDF with task splitting and k processors in a group (EKG)* algorithm proposed by Andersson and Tovar in [38] is one of the most prominent hybrid algorithms for multi-core systems. EKG is a scheduling algorithm that assumes periodic task sets with implicit deadlines. In EKG, some tasks are statically allocated to specific cores while others can be split into two parts that execute on different cores at a non overlapping point in time. By classifying tasks as *heavy* or *light* depending on a parameter k, Andersson and Tovar showed that periodic task sets with implicit deadlines running on a system with M cores under EKG

have an utilization bound of

$$U_{EKG} = \begin{cases} \frac{k}{(k+1)}, & k < M, \\ 1, & k. \end{cases} \tag{II.10}$$

Note that when $k = M$, the utilization bound is 100%. Andersson and Tovar also showed that EKG has a bounded number of preemptions of $2k$ per job over the hyperperiod. They showed that when $k = 2$, the utilization bound is around 66% and the average number of preemptions per job is 4. Over the years other semipartitioned algorithms were proposed in the literature [39, 40, 41, 42]. Deadline-Monotonic with Priority Migration (DM-PM) is a semipartitioned algorithm that considers periodic and sporadic tasks scheduled under fixed-task-priority policies like DM. DM-PM was proposed by Kato and Yamakashi in [43] and it fully dominates other fixed-task-priority partitioned scheduling algorithms. In DM-PM, migration can only occurs if a task does not fit entirely on a given core. When it happens, the highest priority in the system is assigned to the migrating task as it is portioned and allocated to different cores. DM-PM ensures that the task's portions are scheduled at non-overlapping time points so that the execution of a portion only starts when the previous has completed its execution. As showed by the authors, DM-PM has an utilization bound of 50%.

In [44], Kato et al. extended the work done by Kato and Yamakashi in [43] and proposed an algorithm called earliest deadline first with window-constrained migration (EDF-WM), which splits a task among all the available cores in case it does not fit entirely in one of them. By utilizing all the spare capacity of a core, the number of parts the task is split into depends on how much capacity is available in each core. The main idea is to assign execution windows based on pseudo-deadlines to each part of the split task. That is, each core schedules the individual parts according to EDF and the pseudo-deadlines, guaranteeing that two part never execute simultaneously. In [45], Lakshmanan et al. proposed a scheduling algorithm based on semi-patitioned fixed-task-priority for periodic and sporadic tasks with implicit or constrained deadlines. The authors presented an algorithm, called *PDMS_HPTS* that always splits the highest priority task in each available core regardless of whether that is a task that has already been split. By exploiting the execution benefits of fixed-task-priority policies, i.e., the response time = WCET, PDMS_HPTS can maximize the deadline of the subtasks created by splitting the highest priority task. The authors showed that depending on maximum utilization of a task and the order that they are allocated to cores, the utilization bound can vary from 60% to 69%.

Clustering is a special type of approach for scheduling tasks on multi-core systems. In clustering, cores are grouped into clusters in which tasks can be allocated. Each group of cores that form a cluster can apply a different scheduling algorithm, which might help to reduce capacity fragmentation inside each core during runtime. Depending on the system's architecture, cores in the cluster can share the same cache reducing the cost of performing migration among the cores inside the same cluster. In [46], Shin et al. proposed a technique to minimize processor utilization on individual cores based on global EDF for both intra/inter-cluster scheduling. Leontyev and Anderson proposed in [47] a multiprocessor scheduling scheme that supports hierarchical containers encap-

sulating sporadic soft and hard real-time tasks. For each container, a fixed amount of bandwidth is assigned providing temporal isolation among other containers.

II.3 Shared Resources

In the previous sections, we discussed about task models and scheduling algorithms that assume the execution of tasks are completely independent from each other. Although there are different type of dependencies that a task might have, e.g., precedence constraints, in this section, we discuss about the type of dependency related to resources that are accessed by multiple tasks during runtime. Competing for a resource can lead to different problems, where priority inversion, data inconsistency, deadlocks and increased response time are the most common problems.

Shared Resources in Uniprocessor Systems

In the literature, there are classical approaches based on lock-based schemes (semaphores, read/write locks, etc.) to ensure mutual exclusion for resources accessed by multiple tasks during runtime. The most prominent lock-based algorithms are: *PIP*, *PCP* and *SRP*. The priority inheritance protocol (PIP) proposed by Sha et al. in [48] is an algorithm designed to eliminate the phenomenon of priority inversion suffered by fixed-task-priority scheduling algorithms in uniprocessor systems. The phenomenon of priority inversion occurs when a low priority task blocks the execution of a high priority task due to a locked resource. In PIP, if a task gets blocked because a given resource is currently locked by a low priority task, the task locking the resource will inherit the priority of the blocked task. By increasing the priority of the task locking the resource, the waiting time of the blocked task reduces since the number of tasks that can preempt the execution of the task locking the resource decreases. However, PIP does not prevent the system from suffering from deadlocks and chainned blocking (the high priority task is blocked as many times as the number of resources it tries to access).

Priority ceiling protocol (PCP) is an enhanced version of PIP also proposed in [48] by Sha et al. In PCP, there is the concept of *resource ceiling*, which is a value assigned to each resource in the system accessed by more than one task. The resource ceiling value is defined based on the priority of the tasks accessing that resource. That is, the highest priority among the tasks accessing the given resource will define the resource ceiling value. In PCP, a task can lock a resource if two conditions are fulfilled: (i) the resource is free, (ii) the priority of the task attempting to lock the resource is higher than the ceiling of all the resources currently locked by other tasks. If one of these conditions fail, the task that caused the block inherent the priority of the blocked task.

Stack resource policy (SRP) is a resource scheduling algorithm proposed by Baker in [49]. Unlike PIP and PCP that were designed for task-fixed-priority scheduling policies, SRP is based on EDF, i.e., a job-fixed-priority policy. The main difference between SRP and PCP lies on when tasks are blocked. While in PCP tasks are blocked when attempting to lock a resource, in SRP tasks are blocked when attempting to preempt another task. By blocking a task before preempting another task, SRP reduces the

overall response time of the tasks accessing resources. In SRP, each task has a *preemption value*, which is inversely proportional to its relative deadline. Similar to PCP, SRP assigns resource ceiling values to all the resources accessed by more than one task. The value is based on highest preemption level among the tasks accessing each resource. The system level is a dynamic value used by SRP to decide whether a task may preempt the execution of another task. As a dynamic value, the system level varies during runtime and has its value equal to the highest resource ceiling currently locked by a task. In SRP, a task can only preempt another task if its preemption level is higher than the system level at that given point in time.

Shared Resources in Multiprocessor Systems

Over the years, different authors proposed lock-free approaches like Anderson et al. in [50], where they propose a method that allows immediate access, but later checks for possibles conflicts. Likewise, Kopetz and Reisinger proposed in [51] a protocol to ensure non-blocking accesses to shared memory in distributed systems. In multiprocessor systems with partitioned scheduling, Rajkumar et al. proposed a protocol called *MPCP* in [52], which is a version of standard PCP but for multiprocessor systems. The main difference lies in the fact that in MPCP, there are local and global resources. The resource ceiling for global resources are based on all tasks in the system that access that resource. In [53], Gai et al. proposed MSRP, a modified version of SRP for systems with partitioned scheduling. In MSRP, tasks have a bounded blocking and execution time. Unlike MPCP [52] that preempts a task when trying to access a global shared resource, in MSRP, tasks do *busy-wait* checking (called *spin lock* in [53]).

In multiprocessor systems with global scheduling, Devi et al. [54] proposed two approaches for systems scheduled under the global EDF policy. In the first approach, *spin-based queue locks*, tasks perform busy-wait on their own spin variable before accessing a global resource in the system. By updating the spin variables of the tasks according to a FIFO queue, Devi et al. showed that the maximum waiting time in a system with M processors is: $(\min(M, n) - 1)e$, where n is the number of tasks accessing the resource and e is the access time. The second approach, *lock-free synchronization*, accesses to global shared resources happen in a *trial and error* manner. In case of a contention, the task keeps trying until it can successfully access the resource. As shown by Devi et al. [54], the spin-based queue locks approach outperforms the lock-free synchronization method. In [55], Block et al. proposed another approach for systems scheduled under the global EDF policy. Their *Flexible Multiprocessor Locking Protocol* guarantees a job can only be preempted when being released or has its execution resumed. According to their protocol, resources are classified either as *long* or *short* access time. When attempting to access a resource classified as *short*, jobs become nonpreemptable and perform busy-wait. When attempting to access a resource classified as *long*, jobs get blocked and placed on a semaphore queue.

Shared Resources in Automotive Systems

In the case of AUTomotive Open System ARchitecture (AUTOSAR) and the Logical Execution Time (LET) paradigm, most of the authors proposed solutions based on *Point-To-Point (PTP)* communication [56, 57, 58]. In [57], Resmerita et al. proposed an approach to enforce the LET paradigm behavior using PTP communication in legacy automotive systems. Their approach introduces an efficient buffering mechanism that reconciles the performance-oriented concerns of the legacy system with the non-functional requirements of the LET paradigm. In [58], Resmerita et al. extended their previous work in [57] so it can also be applied in multi-core systems. The authors focused on the problem of minimizing the computational costs related to the additional buffering that is required to enforce the LET paradigm behavior during parallel executions.

In [59], Sofronis et al. proposed an optimal wait-free approach called *Dynamic Buffering Protocol (DBP)*, which preserves data consistency for preemptive tasks. The authors designed DBP as a single-writer multiple-reader communication protocol for priority driven tasks scheduled on single-core processors. By design, DBP is memory optimal since it only buffers the data necessary to preserve the data-flow during inter-task communication. Since in an automotive application a shared resource might be accessed by multiple writers, the directly applicability of DBP is not possible. Yip et al. proposed in [60] an approach called *Static Buffering Protocol (SBP)*, which is an extension of DBP but designed to work properly with the LET paradigm in multi-core systems. By exploiting the timing determinism of the LET paradigm (well defined read or write (R/W) -events), SBP derives static buffering schedules for one HP. The schedule derived by SBP specifies when tasks are allowed to access their buffers. The authors showed through evaluation that compared to PTP, SBP needs on average 60% less buffer memory and causes 77% less execution overhead.

II.4 Offline Scheduling

In offline scheduling, the scheduler initializes activities in the system according to a scheduling table, which is defined by the designers *offline*, i.e., before runtime. By taking complete information about the system, its requirements and available resources, an offline algorithm creates a single scheduling table representing a feasible solution for the task set under analysis. Since the table creation takes place offline, fairly complex task set can be handled, e.g., task sets with precedence constraints, mutual exclusion, communication over a given network medium, etc [61]. During runtime, i.e., *online*, a fairly simple dispatcher executes the activities stored in the table that was derived offline. For each invocation of the scheduler, an activity in the table is performed by the OS. Normally, a minimum time granularity is assumed between two consecutive invocations of the scheduler, this granularity is often refereed as *time slot*.

The major advantage of offline scheduling over online scheduling lies in the fact that system behavior is known before runtime. That is, it can assured that the schedule will meet and respect all system's time and precedence requirements during runtime. Naturally, malicious interference caused by external entities (hackers) can deviate the

system from its intended state behavior. In the literature different authors proposed methods to protect the scheduler and its scheduling table from external malicious entities [62, 63, 64, 65, 66].

A well known disadvantage of offline scheduling is the fact that it has a low degree of flexibility and composability, meaning that any changes to the system (task set, requirements, available resources, etc) requires the whole scheduling table to be reconfigured. In order to overcome those disadvantages, well establish methods such as the *Slot Shifting* algorithm [67] proposed by Fohler provide a certain level of flexibility to systems using offline schedules. The slot shifting algorithm consists of an offline and an online phase. In the offline phase, the TT schedule is analyzed by the algorithm in order to determine the amount of leeway available in the schedule. During the online phase, in the case that a non-periodic task arrives in the system, the slot shifting algorithm analyzes the leeway currently available in the system and performs a test to check if its possible to admit the non-periodic task. Passing the test means that there is enough leeway available in the system to accommodate the non-periodic task without affecting the time requirements of the other tasks present in the system.

Over the decades, multiple authors proposed methods to derive offline schedules and increase the flexibility of TT systems. Sandstrom et al. presented in [68] methods to combine static scheduling and online interrupt handling in real-time systems. Their methods take into account possible online interrupt requests during the schedule construction. In [69], Fohler et al. proposed a method to efficiently handle soft real-time tasks in offline scheduled RT systems using total bandwidth server [70, 71]. The method consists of a two phases strategy, where during the first phase (offline), the scheduler resolves all the complex constraints present in the model. During the second phase (online), the schedule constructed by the scheduler is translated into independent tasks on single nodes with start times and deadline constraints only. Once translated, the tasks are scheduled according to an EDF based bandwidth server. Isovic and Fohler proposed in [72] a method to handle a combination of mixed sets of tasks and constraints, i.e., periodic tasks with complex/simple constraints, soft and firm aperiodic/sporadic tasks. In their method, periodic and sporadic tasks are guaranteed offline, while during runtime (online phase) tasks scheduled offline are flexibly shifted to allow the feasible inclusion of dynamically arriving sporadic and aperiodic tasks. By reclaiming resources reserved, but not used by sporadic tasks, Isovic and Fohler showed that their method perform efficient aperiodic task handling during runtime compared to the previous work in [73].

The literature in terms of offline scheduling can be classified into five categories: (i) meta-heuristic methods (e.g., Simulated Annealing [74], Tabu Search [75]); (ii) formal scheduling approaches ([76]); (iii) constraint programming [77]; (iv) Satisfiability Modulo Theory [78]; (V) graph-based algorithms with searching heuristics [79, 80]. Meta-heuristic are stochastic processes and deeply dependent of the input parameters. As a result, if no measures are taken into account, the meta-heuristic method might explore the same/similar search space more than once. Formal scheduling approaches don't have such problem, but are limited to small task sets, since they do not scale well as the number of tasks per processing node increases. Methods based on constraint programming have a slighter better performance compared to formal scheduling approaches, but their

time to execute explodes for larger task set. That is why is often so complicate to integrate all complex constraints of a task set into constraints or equations. Graph-based algorithms with searching heuristics provide a good average performance for moderately utilized systems. The searching heuristic is responsible for aiding the graph-based algorithm to investigate the search-space and reduce the searching time, which for some problem formulation might take a few days or months to find a solution. As shown in the literature [81], tree-pruning techniques can be use to help the algorithm to reduce the search-space, which ultimately reduces the searching time by several orders of magnitude.

In the context of offline scheduling using search-tree pruning techniques, we classify the literature in three categories: (i) scheduling only; (ii) allocation-only; (iii) scheduling and allocation. Abdelzaher et al. proposed in [82] a scheduling only search-tree pruning technique that generates schedules for fully-preemptive task sets with constrained deadlines using the *Branch-and-Bound* (B&B) approach [83]. In their method, each edge between two nodes has a *cost* related to it. As the tree is traversed by the algorithm, the cost accumulates and it is used to prune the search space. Abdelzaher et al. show that their method provide optimal schedules for task sets with communication, precedence and exclusion constraints. In [84], Peng et al. proposed an allocation only search-tree pruning technique that uses B&B to optimally allocate tasks with communication and precedence constraints. Ahmad et al. proposed in [85] a solution to the same problem as in [84], but instead of using B&B, they used the A* algorithm proposed in [86] by Hart et al. Jonsson defined in [81] an scheduling and allocation search-tree pruning technique for non-preemptive jobs with precedence constraints. The author showed that by using depth-first search his method reduces by several orders of magnitude the searching time of the scheduling problem.

Korf proposed in [87] a depth-first method called *Iterative Deepening A** (IDA*). In his method, a heuristic function is used to estimate the cost of each node on tree. The cost of a node n is the sum of the accumulated cost from the root node until n plus the estimated cost to reach a leaf node, i.e., a solution node. Before starting the search, the algorithm sets an initial cost-bound, which is used to determine *how deep* the search should go before an expansion of the tree is required. In IDA*, a node can only be reached if its cost is lower than the cost-bound. If no node can be reached with the current value of the cost-bound, it is increased by the minimum required value to allow the algorithm to reach one more node. Every time the cost-bound is increased by the algorithm, it reiterates over the expanded search-tree, i.e., re-starts the search. Compared to A*, IDA* requires less memory and does not overestimates the solution cost.

Over the years, different authors proposed methods for a parallel version of IDA* [88, 85, 89, 90, 91]. The method proposed by Syed in [91, 92] combines the allocation strategy shown in [85] and response-time based symmetries. His method does allocation and scheduling for preemptive and non-preemptive tasks on heterogeneous processor and network architectures using a *parallelized* version of IDA*, which he calls as PIDA*. In his algorithm, each available processor (called worker in [91]) works on its own pile of nodes from the tree in a depth-first manner. The iterative process of searching the tree

consists of four steps: (i) pop the node that is at the top of the pile; (ii) create child nodes for the popped node; (iii) evaluate and sort child nodes; (iv) verify whether the best evaluated child node is a leaf (solution) node, if not, push it into the pile. This iterative process repeats until a leaf node is found by one of the workers.

II.5 Multi-Mode Systems

Nowadays safety-critical control systems can operate in a number of defined *modes*, e.g., in modern automotive systems the engine control system has start-up, cruise and driver control modes. Naturally, if a system has multiple operation modes, mode change protocols should be present in order to control how the system changes from one mode to another. In the context of RT systems, there are in the literature multiple protocols to handle mode changes [93, 94, 95, 96, 97, 98, 99].

As discussed by Burns in [100], the difference between multi-mode and multi-criticality systems is subtle. While the change from a low to high criticality level can be seen as a mode change in some cases, the abandonment of work normally present in such changes can only be seen in a particular form of mode change, e.g., when there is service degradation. In [100], Burns formalizes a *mode* according to six aspects: (i) Type; (ii) Trigger mechanism; (iii) Protocol; (iv) Attributes; (v) Definition; (vi) Analysis.

The first aspect, *Type*, defines whether a mode is part of an expected system behavior (e.g., changing from one functionality to another), if it is part of a rare scenario (e.g., *prepare to crash* functionality), or if it is part of a degradation service (e.g., system functionalities are reduced due to presence of faults). The *Trigger* mechanism defines whether the mode change occurs due to an event that happens or if it is a planned change. The former can be seen as an *input request* that causes the system to change from one mode to another, while the latter is a time coordinated change (e.g., a system changing to night mode after an specific time).

The *Protocol* defines how system activities (e.g., the execution of a job) should be handled when changing from one mode to another. An *Immediate* protocol defines that previous mode activities should be suspended/aborted immediately to allow current mode activities to execute. A *Bounded* protocol defines an interval of time for previous mode activities to finish after the mode change. After this interval, the system can initiate activities from the current mode. A *Phased* protocol allows *mode-overlap*, meaning that jobs from previous mode are allowed to finish their execution, while some jobs from current mode can start their execution.

The *Attribute* aspect of a mode defines its properties. For example, a mode can be *one-shot* if it can only be active once during runtime, while a *sink* mode means that further mode changes are no longer possible. In the same manner, an *initial* mode means that the system always starts in this mode. If the system systematically repeat an specific sequence of modes, it is said that those modes are part of a *cyclic* mode set. The *Definition* aspect defines what happens to tasks after a mode change, e.g., only run in one of the modes, has its parameters changed. From the schedulability point of view, the *Analysis* aspect of a mode defines the different code requirements tasks might have.

Sha et al. [93] investigated the impact of changing task parameters (e.g.,priorities)

after a mode change. The authors proposed a protocol with bounded delay for priority-driven preemptive schedulers, where deadlocks are avoided and it is guaranteed that a high priority job can be blocked for at most the duration of one critical section. In [94], Burns and Quiggle presented a solution for the general problem of dealing with mode changes during runtime. Tindell et al. proposed in [95] a protocol to guarantee the timing constraints of tasks after a mode change. They designed a protocol for fixed-priority preemptive scheduling systems, where tasks may have different WCET in the different modes. The authors provide a sufficient analysis to compute the worst case response time of tasks during a mode change. Pedro and Burns proposed in [96] a method to guarantee the schedulability of tasks that started their execution in the previous mode and have to finish it during the mode change period. In [97], Real and Crespo presented a survey of mode change protocols for uniprocessor, fixed-priority, preemptively scheduled real-time systems. The authors also proposed an asynchronous protocol that sets offsets to the first activation of tasks that were added to the system after a mode change. By adding specific offset values, Real and Crespo showed that their method increase system schedulability compared to previous methods. Emberson and Bate proposed in [98] a method to reduce the set of differences (e.g., additional task offset, migration) when changing from mode to another. The authors proposed three different methods. The first method investigates each mode in a sequential order, i.e., finds a solution for the first mode and uses it as a starting point for the next mode. The second method tries to find a single configuration that works for all modes, while the third method uses parallel communicating searches, i.e., each search tries to find a solution for its mode while trying to minimize the mode differences.

Vestal [101] proposed a model that assumes a system with different modes of execution. In his work, he investigates the schedulability of such systems where the WCET of tasks vary depending on the execution mode. In [99], Ekberg et al. proposed a task model to support complex arrival and synchronization patterns in systems with multiple modes. The authors proposed a mode switching protocol that generalizes previous graph-based task models. Burns and Davis presented in [102] a survey that covers state of the art research in the area of mixed criticality systems since the work done by Vestal [101]. In the survey, Burns and Davis focused on solutions for both single and multiprocessor systems. In [103], Baruah et al. considered a system with different modes and proposed a method to increase the schedulability of high criticality tasks over the ones with lower criticality. In [104], Santy et al. improved the work done in [103] and showed that is possible to eventually reactivate the execution of low criticality tasks that were suspended during a mode change. Burns and Baruah showed in [105] that after a mode change, low criticality tasks could be served again after stretching their periods, while Gettings et al. [106] proposed to skip the execution of some jobs after a mode change in order to respect the timing constraints of the system. In the realm of real-time networks, Kopetz et al. [107] proposed an adaptation to the Time-Triggered Protocol (TTP) to support mode changes. Heilmann et al. presented in [108] a method to facilitate mode changes in time-triggered networks such as TTEthernet.

II.6 Migration Techniques

Over the years, different techniques were proposed in the literature regarding process migration in the context of multi-core RT systems. Each of those techniques focused on a specific goal, which were to improve load/thermal/power balancing [109, 110, 111], or fault tolerance [112].

The term migration in the context of RT systems refers to allowing task instances to execute on different cores rather than binding all instances of a task to a given core. As briefly explained in Section II.2.2, depending on the adopted migration technique, task instances can migrate at any point in time (*full migration*) or only at the boundaries of their execution (*job-level migration*). The *migration policy* is responsible for deciding *which* task(s) should migrate, as well as *when* and *to where* they should be moved. The overhead associated with each migration operation is usually the main factor taken into consideration by the migration policy when performing its decision process. The *migration mechanism* dictates how task instances should be migrated from one core to another and it is the major contributor to the overhead associated to a migration operation. The choice of which mechanism to use directly depends on the underling system architecture as well as how the memory is organized.

In [113], Milojičić et al. presented a survey about a number of techniques (e.g., lazy state transfer, precopying, residual dependencies) available in literature to reduce the overhead caused by migration in distributed systems. Acquaviva et al. in [114] and Pittau et al in. [115] relied on techniques such as full/partial replication of tasks to cores in order to reduce the overhead caused by migration. In [116], Zhang et al. evaluated the impact of migration in distributed systems applying gang-scheduling. Brião et al. investigated in [117] the feasibility of migration in soft RT systems by checking how much impact migration can cause to tasks and how that could result in deadline misses. In [118], Choffnes et al. minimizes migration cost while ensuring fair-share scheduling in multi-core soft RT systems.

Almeida et al. presented in [109] a migration mechanism capable of performing load balancing during runtime. By relying on a tiny preemptive Real-Time Operating System (RTOS), their migration mechanism improves system performance while ensuring scalability. In [110], Ge et al. presented a framework for distributed thermal management in multi-core systems using proactive migration. A monitoring agent present in each core collects data regarding the local workload/temperature and based on that decides which tasks should migrate to a neighbor core. Zeng et al. proposed in [111] a migration method based on partitioned scheduling to reduce energy consumption in RT systems. Saraswat et al. proposed in [112] a greedy heuristic function that during runtime performs task migration in order to tolerate permanent faults in multi-core systems.

In the context of hard RT systems, Katre et al. proposed in [119] different migration policies for applications running on safety-critical embedded systems with multi-core architectures. By considering the number of cache lines that should be migrated and the underling migration mechanism, their greedy policies choose which of the possible migration candidates lead to the least overhead at any given time. Sarkar et al. pro-

posed in [120] a hardware-based push-assisted cache migration technique to retain locks on cache lines during migration. They showed that their technique is capable of delivering deterministic and efficient cache migration options to the scheduler. Megel et al. presented in [121] four migration techniques (*Total Copy*, *Prefetch Copy*, *Prefetch Postcopy*, *Mixed Copy*) that benefit from the static description of the temporal behavior of hard RT systems. By foreseeing when the task under migration is inactive, their techniques are capable of avoiding freeze time (i.e., the interval of time in which a task can not execute due to on-going migrations).

Munk et al. proposed in [122] a method to check at runtime the feasibility of a migration request in hard multi-core RT systems. Pourmohseni et al. presented in [123] a mapping reconfiguration methodology based on task migration for hard RT applications running on Network-on-Chip (NoC) based multi-core system. In order to achieve reconfiguration predictability, the proposed approach employs a design-time analysis to identify for each source and target mapping the most efficient migration latencies. In [124], Pourmohseni et al. proposed an online method to support migration on composable multi-core systems. Instead of relying on static mapping reconfiguration strategies, their method uses a predictable migration mechanism and a lightweight feasibility test to check whether or not, the migration overhead is acceptable for the hard RT application.

Chapter III

Challenges: Cause-Effect Chains in Safety-Critical Real-Time Systems

With the ongoing demand for more efficient safety-critical real-time (RT) systems, new solutions have to be proposed for the challenges that are emerging alongside this continuous demand. One example would be to propose solutions for ensuring the end-to-end (E2E) communication latency requirements of RT applications present in modern safety-critical RT systems. Since multiple safety-critical system functionalities might depend on those E2E latencies to work properly, it is pivotal in many modern automotive systems to ensure communication latency between RT applications to be within a specific timing interval.

With the recent shift by the automotive industry from single-core to multi-core systems, the analysis of whether or not the E2E communication latency requirements of modern automotive systems are respected became a non-trivial problem. The complexity comes mainly from the fact that RT application tasks can be allocated to different cores and execute in parallel making it hard to trace how data propagates during inter-task communication. Moreover, as tasks get tightly coupled to each other by means of numerous communication dependencies forming chained sequences of communicating tasks (cause-effect chain (CEC)), standard timing analysis tools can no longer cope with this continuous increase in system complexity. As shown in the industrial challenge proposed by BOSCH [11, 125, 126, 127] (See Section III.1), ensuring data consistency and temporal determinism along functional CECs is one of the main current challenges in the automotive industry.

Since 2015, different solutions for the challenge presented by BOSCH were proposed in the RT literature by the research community (See Section III.2). Up to this date, most of the authors that worked on the challenge focused on the interplay between CECs and the communication paradigm adopted by them in order to propose possible solutions to the challenge. Section III.1 presents the industrial challenge that partially motivated this dissertation. We analyze and discuss the characteristics of the proposed system model, while shedding light on the aspects of the challenge that we focus on and propose solutions in Chapter IV. Section III.2 presents the state of the art research in the field of timing analysis for multi-rate CECs in multi-core safety-critical automotive systems.

III.1 Motivation

The industrial challenge that partially motivated this dissertation was introduced to the RT research community by BOSCH during the International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS) [125] in 2015. Focused on applications developed in accordance to the AUTomotive Open System ARchitecture (AUTOSAR) standard, the initial goals of the challenge were:

1. Provide accurate analysis of the worst-case E2E latencies along multi-rate CECs
2. Interleaved worst-case execution time (WCET) and Worst-Case Response Time (WCRT) analysis for memory acesses
3. Optimize task to core mapping
4. Provide evaluation on multi-core platforms

In order to reduce the gap between industry and the RT research community, a real world automotive benchmark [11] was presented by BOSCH in the WATERS challenge. The benchmark described in [11] presents different aspects of an automotive application such as: (i) structure, (ii) task activation period, (iii) task communication, (iv) timing requirements, and (v) cost model. The first aspect, *structure*, defines that an automotive application should be structured in terms of *software components (SWCs)*. Each SWC can be further decomposed into several *runnables* — smallest executable unit within a SWC — that are grouped into tasks depending on their execution period.

The second aspect, *task activation period*, defines the triggering mechanism that is responsible for setting task's period. That is, tasks can be triggered in a time-synchronously manner (according to their period), triggered by asynchronous events, or according to the crankshaft rotation, i.e., angle-synchronous (See Section I.2). Task scheduling is based on fixed priorities and can be done by the operating system (OS) in a fully preemptive or cooperative manner.

The third aspect, *communication*, defines that *labels* — data storing mechanism — should be used for intra/inter-task communication. A label accesses can be *Read-Only*, *Write-Only*, or *Read-Write*. Access to labels could be done via two communication paradigms, explicit, or implicit (See Section I.2). The Logical Execution Time (LET) communication paradigm was later considered as part of the challenge in 2017 [127]. Below we formally define the three communication paradigms considered in WATERS challenge.

Definition 1. ***Explicit Communication.*** *Let J be a job of task τ and ShR a resource accessed by J. Under the explicit communication paradigm, the read and writes accesses of J to ShR can occur at anytime during J's execution.*

Definition 2. ***Implicit Communication.*** *Let J be a job of task τ and ShR a resource accessed by J. Under the implicit communication paradigm, the read access of J to ShR happens at the beginning of J's execution, while the write access to ShR happens at the end of J's execution.*

Definition 3. ***Logical Execution Time.*** *Let J be the i^{th} job of task τ with a well defined communication interval L_J and ShR be a resource accessed by J, where $i \in \mathbb{N}^+$. Under the logical execution time communication paradigm, the read access of J to ShR happens at the beginning of L_J, i.e., $read(J)$, while the write access happens at the end of L_J, i.e., $write(J)$. That is $L_J = [read(J), write(J)]$, where*

$$read(J) = (i-1)T_\tau + \phi_\tau, \qquad write(J) = (i)T_\tau + \phi_\tau \tag{III.1}$$

The fourth aspect, *timing requirements*, defines that all automotive systems are composed of hard-RT tasks, where synchronous tasks have deadlines equal to their periods, and angle-asynchronous tasks have deadlines equal to half of their period length. Additional timing requirements are present in form of *cause-effect chains*, e.g., an actuator must perform an action within a maximum E2E latency after the read of a sensor value (see sections III.1.1 and III.1.2).

The firth aspect, *cost model*, defines the execution costs that have to be added to the model given how tasks and labels are mapped into the target platform. In single-core systems, time to access labels are already included in the execution time, but scheduling effect like preemption are excluded. In multi-core systems, execution time also depends on where labels are placed, since the time to access them directly affect the execution time of tasks.

In Section III.1.3, we describe the full characteristics of an automotive application considering the five aspects described above, while in sections III.1.1 and III.1.2 we formally define a CEC and its E2E latencies respectively.

III.1.1 Cause-Effect Chains

In general, a cause-effect chain (CEC) represents an ordered sequence of communications carried out between a finite set of tasks. In this work we represent a cause-effect chain by E, where $E = \{\tau_1 \rightarrow \tau_2 \rightarrow \cdots \rightarrow \tau_{|E|}\}$ with $|E|$ being the number of tasks in E. Function $E(i)$ returns the i^{th} task in E, $i \in \{1, 2, \cdots, |E|\}$, where $\tau_i \neq \tau_{i+1}$. For a pair of consecutive tasks τ_i and τ_{i+1} in E, the '$\rightarrow$' operator indicates that τ_{i+1} acts as a consumer/reader task, while τ_i as a producer/writer task. We assume that E samples (acquires data) at every $read(J_1)$, J_1 being a job of task $E(1)$. We use z to represent the time interval between the occurrence of an external event (input) and its sampling by J_1. Likewise, we use z' represents the time interval between $write(J_{|E|})$ and the actuation (output). The two time intervals (z, z') described above correspond respectively to the *input* and *output* intervals defined by Günzel et al. in [128] and later described in Section III. $H(E)$ represent the hyperperiod of E, where $H(E) = LCM(T_1, T_2, \cdots, T_{|E|})$.

Definition 4. ***Job Chains.*** *Given a CEC E, a job chain $c^E = (J_1 \rightarrow \cdots \rightarrow J_{|E|})$ is a finite sequence of jobs representing one of the possible data propagation paths of E. In a job chain, the following requirements are respected:*

1. *J_i is a job of $E(i)$, $i \in \{1, 2, \cdots, |E|\}$.*
2. *The data written by J_i is read by J_{i+1}. That is, $write(J_i) \leq read(J_{i+1})$ for all $i \in \{1, 2, \cdots, |E| - 1\}$*

In this dissertation, we use c_i^E to represent the i^{th} job chain of E, $i \in \mathbb{N}^+$, while function $l(c^E)$ returns the time interval between $write(J_{|E|})$ and $read(J_1)$.

Figure III.1 shows the representation of a CEC E, where $E = \{\tau_1 \rightarrow \tau_2\}$. Task τ_1 is a periodic task with phase $\phi_{\tau_1} = 0$ and period $T_{\tau_1} = 3$, while Task τ_2 is a periodic task with phase $\phi_{\tau_2} = 0$ and period $T_{\tau_2} = 5$. Both tasks apply the LET communication paradigm. Note that in Figure III.1 each colored arrow represent a job chain of E according to the LET communication paradigm.

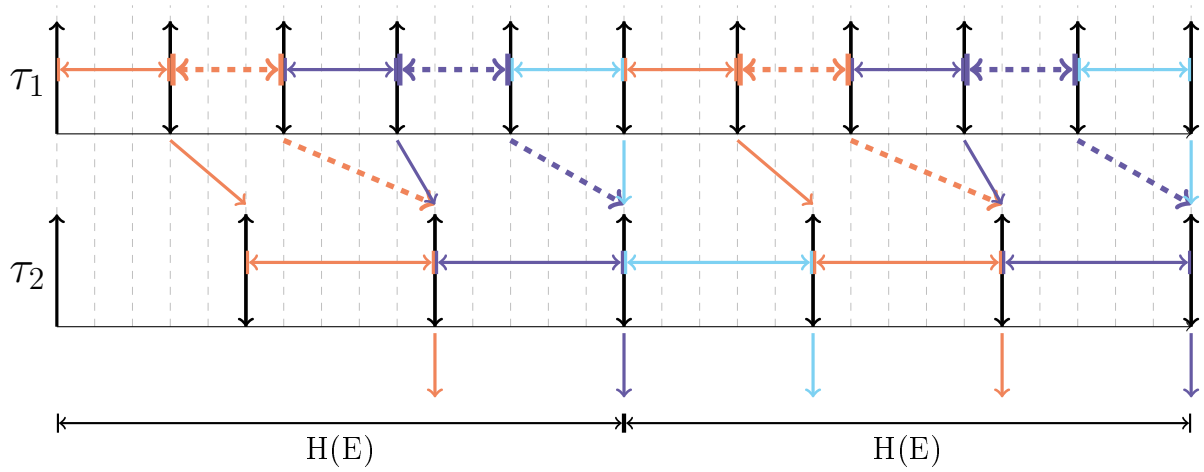


Figure III.1: *Data propagation of a cause-effect chain E with tasks applying the logical execution time paradigm*

III.1.2 End-to-End Latency

In the context of a cause-effect chain (CEC), end-to-end (E2E) latency represents the time it takes for data to travel along one of the possible propagation paths of the chain, from one end to the other. As briefly mentioned in Section I.3, when analyzing the latencies of multi-rate cause-effect chains, the most common latency metrics to be analyzed are the *reaction time* and *data age* latencies. Both metrics were initially introduced by Feiertag et al. in [10] (See Section III.2). Below we formally define the metrics reaction time and data age.

Definition 5. ***Reaction time*** *(defined as First-To-First latency in [10]). Measures system's reactivity to an asynchronous input, that is, how long does it take for an incoming input to traverse the entire CEC assuming the maximum sampling delay.*

Definition 6. ***Data age*** *(defined as Last-To-Last latency in [10]). Measures the freshness of system's output data, that is, for how long a given input influences the outputs produced by the system.*

The other metrics (*First-to-Last*, *Last-to-First*) defined by Feiertag et al. [10] and the later extensions done by Günzel et al. in [128] are discussed in Section III.2.

III.1.3 Application Characteristics - WATERS Automotive Benchmark

According to the benchmark in [11], 90% of the labels in an automotive application should have a data size of 1, 2, or 4 bytes. Table III.1 details the probability size of a label in an automotive application. Complex applications, such as the combustion engine control, might require up to 50000 labels depending on their implementation. The probability of a label being Read-Only is 40%, while 10% and 50% are the probabilities of a label being *Write-Only* and *Read-Write* respectively.

Size (byte)	Share
1	35%
2	49%
4	13%
5-8	0.8%
9-16	1.3%
17-32	0.5%
33-64	0.2%
>64	0.2%

Table III.1: *Label size probability in an automotive application*

Read-Write labels are used by runnables for intra/inter-task communications, where intra-task communication can be done in a *forward* or in a *backward* manner. When two communicating runnables are mapped to the *same* task and the writer (producer) runs *before* the reader (consumer), it is said that they have a intra-task forward communication pattern. Likewise, when the writer (producer) runs *after* the reader (consumer), it is said that they have a intra-task backward communication pattern. An inter-task communication pattern happens when two communicating runnables are mapped to *different* tasks. Table III.2 shows the characteristics of inter-task communication between tasks in an automotive application.

Period	1 ms	2 ms	5 ms	10 ms	20 ms	50 ms	100 ms	200 ms	1000 ms	Angle-Sync
1 ms				I	I		I			I
2 ms				I	I		I			
5 ms		I	IV	IV	II	II	I			
10 ms	II	II	II	VI	IV	II	IV	II	III	IV
20 ms	I	I	I	IV	VI	II	IV	I	II	IV
50 ms			II	II	II	III	I			
100 ms		I	I	V	IV	II	VI	II	III	IV
200 ms				I	I		I	I	I	
500 ms				III	II		III	I	IV	I
Angle-Sync	I	I	I	IV	IV	I	III	I	I	V

Table III.2: *Amount of inter-tasks communication between tasks depending on their period length*

Writer (producer) tasks are listed in the rows of Table III.2, while reader (consumer) tasks are listed in the columns. Note that an empty cell in Table III.2 indicates that there is no inter-task communication between tasks with those periods. Table III.3 contains the cell content code for Table III.2. The occurrence probability of an inter-task communication is 40%, while 25% and 35% are the occurrence probabilities for intra-task forward/backward communication respectively.

I	II	III	IV	V	VI
< 10	10-50	51-100	101-500	501-1000	>1000

Table III.3: *Cell content code for Table III.2*

In an automotive application, the total number of runnables usually varies between 1000 and 1500. Table III.4 presents the activation period probabilities for runnables. As explained in Section I.1, runnables with the same activation period are grouped into tasks. Therefore, Table III.4 also shows activation period probabilities for tasks.

Period	Probability
1 ms	3%
2 ms	2%
5 ms	2%
10 ms	25%
20 ms	25%
50 ms	3%
100 ms	20%
200 ms	1%
1000 ms	4%
Angle-Sync	15%

Table III.4: *Task activation period probability*

As specified in the benchmark [11], a Weibull distribution [129] should be used to distribute execution times to runnables as they are described in terms of *minimum time* (best-case), average, and *maximum time* (worst-case). Table III.5 describes the distribution of the average execution time of runnables according to their activation period. Although columns *Min* and *Max* show, respectively, the minimum and maximum values runnables can have, when grouped into a task, their average execution time should match the value in the column *Avg*. The benchmark [11] specifies that the WCET and best-case execution time (BCET) values for a runnable are obtained by multiplying the derived execution time from Table III.5 to a f value within the range shown in Table III.6. For example, the BCET value for a runnable belonging to a $1ms$ task is derived by multiplying a value from Table III.5 times a f value within the range $[f_{min}, f_{max}]$ in the column *Best*.

Period	Average Execution Time (us)		
	Min.	Max.	Avg
1 ms	0.34	30.11	5.00
2 ms	0.32	40.69	4.20
5 ms	0.36	83.38	11.04
10 ms	0.21	309.87	10.09
20 ms	0.25	291.42	8.74
50 ms	0.29	92.98	17.56
100 ms	0.21	420.43	10.53
200 ms	0.22	21.95	2.56
1000 ms	0.37	0.46	0.43
Angle-Sync	0.45	88.58	6.52

Table III.5: *Average execution time for runnables*

Period	Best		Worst	
	f_{min}	f_{max}	f_{min}	f_{max}
1 ms	0.19	0.92	1.30	29.11
2 ms	0.12	0.89	1.54	19.04
5 ms	0.17	0.94	1.13	18.44
10 ms	0.05	0.99	1.06	30.03
20 ms	0.11	0.98	1.06	15.61
50 ms	0.32	0.95	1.13	7.76
100 ms	0.09	0.99	1.02	8.88
200 ms	0.45	0.98	1.03	4.90
1000 ms	0.68	0.80	1.84	4.75
Angle-Sync	0.13	0.92	1.20	28.17

Table III.6: *Multiplier factor for determining the WCET and BCET values for runnables*

As explained in Section III.1, besides the typical execution timing requirements that runnables/tasks have, modern automotive control applications have additional timing requirements in form of CECs. Usually, an automotive control application has between 30 and 60 CECs, where most of them contain tasks with the same activation period. Table III.7 describes the probability of a CEC be constituted of tasks with the same of different activation period, i.e., the probability of having a multi-rate CEC.

Number of Activation Periods in the CEC	Probability
1	70%
2	20%
3	10%

Table III.7: *Average execution time for runnables*

Table III.8 specifies the number of runnables per activation period present in the CEC.

Number of Runnables	Probability
2	30%
3	40%
4	20%
5	10%

Table III.8: *Number of runnables per activation period present in the CEC*

As specified in the benchmark [11], the timing requirement for a CEC is the sum of the different activation periods present in the CEC. For example, the timing requirements for a CEC $E = \{\tau_1 \rightarrow \tau_2 \rightarrow \tau_3 \rightarrow \tau_4\}$, where $T_{\tau_1} = 1ms$, $T_{\tau_2} = 1ms$, $T_{\tau_3} = 10ms$, and $T_{\tau_4} = 5ms$ is $16ms$.

In 2017, an updated version of the WATERS challenge was proposed in [127]. The addition of LET as a communication paradigm, and the specification of two E2E latency metrics (*reaction latency* and *data age*) were the main additions of the challenge. As defined in Section III.1.2, the reaction time measures the *reactivity* of the system, while data age the *freshness* of data. The updated goals of the WATERS challenge are listed below:

1. Present ways in which the implicit and LET communication paradigms can be realized in multi-core systems
2. In terms of extra cycles, measure the overhead caused by memory accesses due to the proposed implementation of the implicit and LET communication paradigms
3. Propose methods to compute the reaction time and data age of multi-rate CECs containing tasks with harmonic and non-harmonic periods
4. Present label mapping methods to optimize memory access overhead
5. Measure the effect of contention on the E2E latencies of the CECs

The focus of this dissertation is on proposing solutions for items 1 and 3 of the extended version of the WATERS challenge. Note that although the focus of this dissertation is on specific items of the challenge, the contributions are not limited to that as presented in Section I.4. The following section presents the state of the art timing analysis methods and E2E latency optimization approaches for single/multi-rate CECs in safety-critical RT systems.

III.2 Related Work and State of the Art

Before 2007, most of the timing analysis research done in the context of automotive RT systems focused on the analysis of individual tasks. The verification of the timing requirements of each task through methods such as WCRT analysis was enough to guarantee the correct execution behavior of the system functionalities in which those tasks were part of. However, as automotive system functionalities started to become more complex during the last decades, it became necessary to design applications where multiple tasks could work cooperatively on the same data. This cooperative work between multiples tasks usually occurs through the propagation of their execution output to another task, i.e., the output of a task serves as input to another task forming *an ordered* sequence of tasks with complex data dependencies(CECs). As a consequence, the process of verifying the correct behavior of modern automotive system functionalities no longer solely depends on *when* each individual task executes. The verifying process also depends on *how long* does it takes for data to traverse the complete sequence of tasks that are part of the system functionality.

In order to evaluate the latency impact of this interplay of tasks in modern automotive systems, researchers focused their effort on analyzing the E2E latency of the CECs rather than on the response-time analysis (RTA) of individual tasks. During the last decades, multiple authors proposed different methodologies to provide upper bounds or exact latency values for CECs depending on the communication paradigm adopted by the tasks (See Section I.2.2). In the literature, Davare et al. [130] were the first to propose a method to bound the E2E latency of CECs applying the implicit communication paradigm. In [130], Davare et al. provided an optimization procedure to automate period assignment during design time of automotive applications in distributed systems. Assuming periodic tasks, Davare et al. showed that the E2E latency of a CEC $E = \{\tau_1, \tau_2, \cdots, \tau_n\}$ is upper bounded by:

$$\sum_{i=1}^{n} T_{\tau_i} + RT_{\tau_i} \tag{III.2}$$

where RT_{τ_i} represents the WCRT of task τ_i.

Feiertag et al. [10], were the first to further decompose the E2E latencies of a CEC into different latency metrics, which they described as:

- *First-to-First*
- *First-to-Last*
- *Last-to-First*
- *Last-to-Last*

In order to formally define the metrics above, Feiertag et al. [10] derived two time intervals and named them *input* and *output* interval. Figure III.2 illustrates the four latency metrics described by Feiertag et al. [10] as well as the additional input and output intervals.

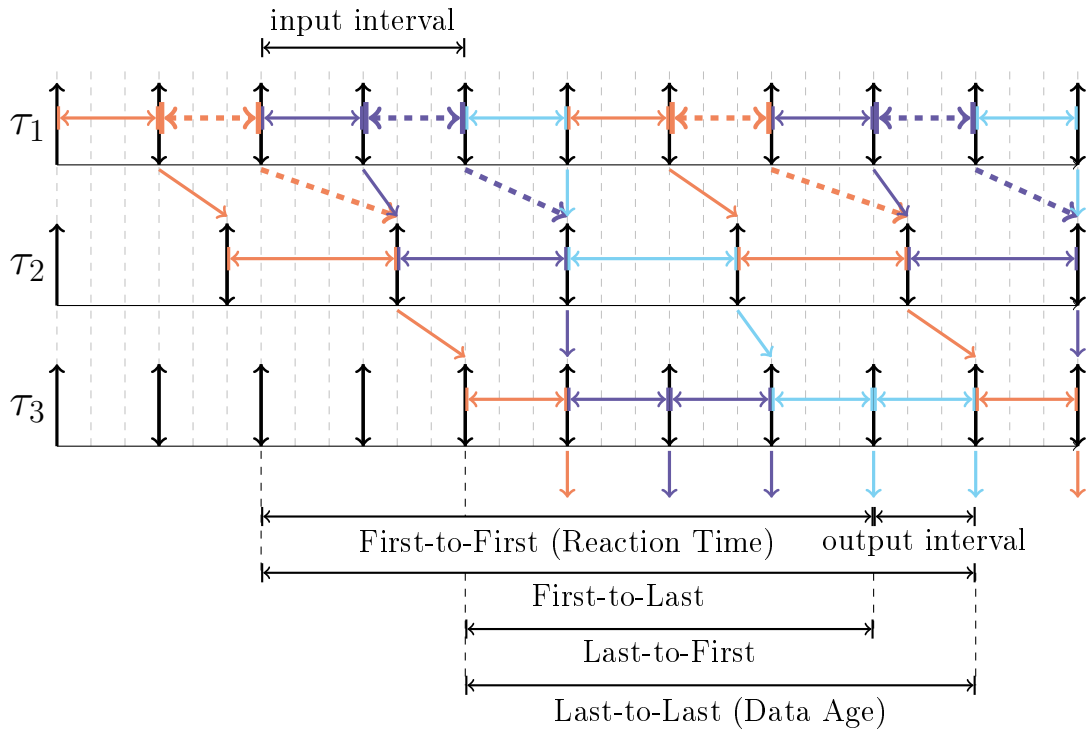


Figure III.2: *End-to-end latency metrics considering input/output intervals*

The *input interval* shown in Figure III.2 was described by Feiertag et al. [10] as the *sampling delay* that an asynchronous input might suffer when arriving in the system. This sampling delay is maximized when the incoming input *just miss* the read access of the first task in the CEC. That is, the incoming input has to wait until the next read access of the first task that does not have its output overwritten, i.e., propagates its output to the next task in the CEC. Such scenario is illustrated in Figure III.2 by the *input interval* between the *purple* and *blue* data propagation paths. When the asynchronous input *just misses* the *purple* data propagation path, it has to wait until the *blue* data propagation path to be recognized and propagated until the end of the CEC.

The *output interval* shown in Figure III.2 was described by Feiertag et al. [10] as the time interval between the outputs of the last task in the CEC that are based on the same asynchronous input value read by the first task in the CEC. That is, the time interval between the outputs of the CEC that are based on the same data propagation path, e.g., the time interval between the two outputs based on *blue* data propagation path in Figure III.2.

As defined by Feiertag et al. in [10], the *First-to-First* latency metric (also known as *reaction time*) represents the time interval between the *beginning* of the current *input interval* until the *beginning* of the next output interval. That is, the time interval taken by the CEC to *react* to an asynchrnous incoming input assuming the maximum input interval. The *First-to-Last* latency metric represents the time interval between the *beginning* of the current *input interval* until the *end* of the next output interval. That

is, for how long system's output is based on an asynchrnous incoming input assuming the maximum input interval. The *Last-to-First* latency metric represents the time interval between the *end* of the current *input interval* until the *beginning* of the next output interval. That is, once the incoming input is recognized by the CEC, how long does it take for the last task in the CEC to produce an output based on such input value. The *Last-to-Last* latency metric (also known as *data age*) represents the time interval between the *end* of the current *input interval* until the *end* of the next output interval. That is, once the incoming input is recognized by the CEC, for how long the outputs of the CEC will be based on such input value.

Since a CEC has multiple data propagation paths and each path has its corresponding reaction time and data age value, the worst-case scenario happens when those two metrics are maximized. The Maximum Reaction Time (MRT) latency metric represents the maximum reaction time value among all the possible data propagation paths of the CEC. That is, the maximum time interval required by the CEC to produce an output based on the arrival of an asynchronous input assuming the maximum sampling delay. In the same manner, the Maximum Data Age (MDA) latency metric represents the maximum data age value among all the possible propagation paths of the CEC. That is, the maximum time interval that outputs of the CEC are based on the same input data.

In 2019, the AUTOSAR Timing Extensions model [6] stated that,"*without over- and undersampling, age and reaction are the same*"([6], Section 7.2, p. 149). That is, in a single-rate CEC, the reaction time and data age values of a given job chain are the same. However, Günzel et al. proved in [128] that in fact, the reaction time and data age values are *always* equivalent, even for multi-rate CEC. Günzel et al. [128] showed that there was an inconsistency in the definitions made by Feiertag et al. [10] for the input and output intervals. The source of that inconsistency was the fact that Feiertag et al. [10] defined the input interval as a *passive* asynchronous arrival of data to the system, while the output interval was defined as an *active* periodic generation of data by the system. In [128], Günzel et al. showed that the output interval initially considered by Feiertag et al. [10] didn't consider all the possible output scenarios of a CEC. In fact, rather than considering only the scenario in which the system actuator is directly and periodically triggered by the write event of the last task in the CEC, Günzel et al. [128] considered that there might be an additional delay until the actuation takes place. That is, an additional time interval between the write event of the last task in the CEC and the actual system actuation might exist.

As shown by Günzel et al. in [128], the length of this extra time interval is maximized when the system actuation happens directly before the next write event of the last task in the CEC, i.e., right before the write event of the last task in the next data propagation path. Günzel et al. [128] proved that this actuation delay can be covered by extending the output interval previously defined by Feiertag et al. [10]. In order to cover the scenarios not considered by Feiertag et al. [10], Günzel et al. [128] proposed the addition of two new intervals (*reduced input*, *reduced output*) and the redefinition of the latency metrics. Figure III.3 shows the new configuration of the input and output intervals, as

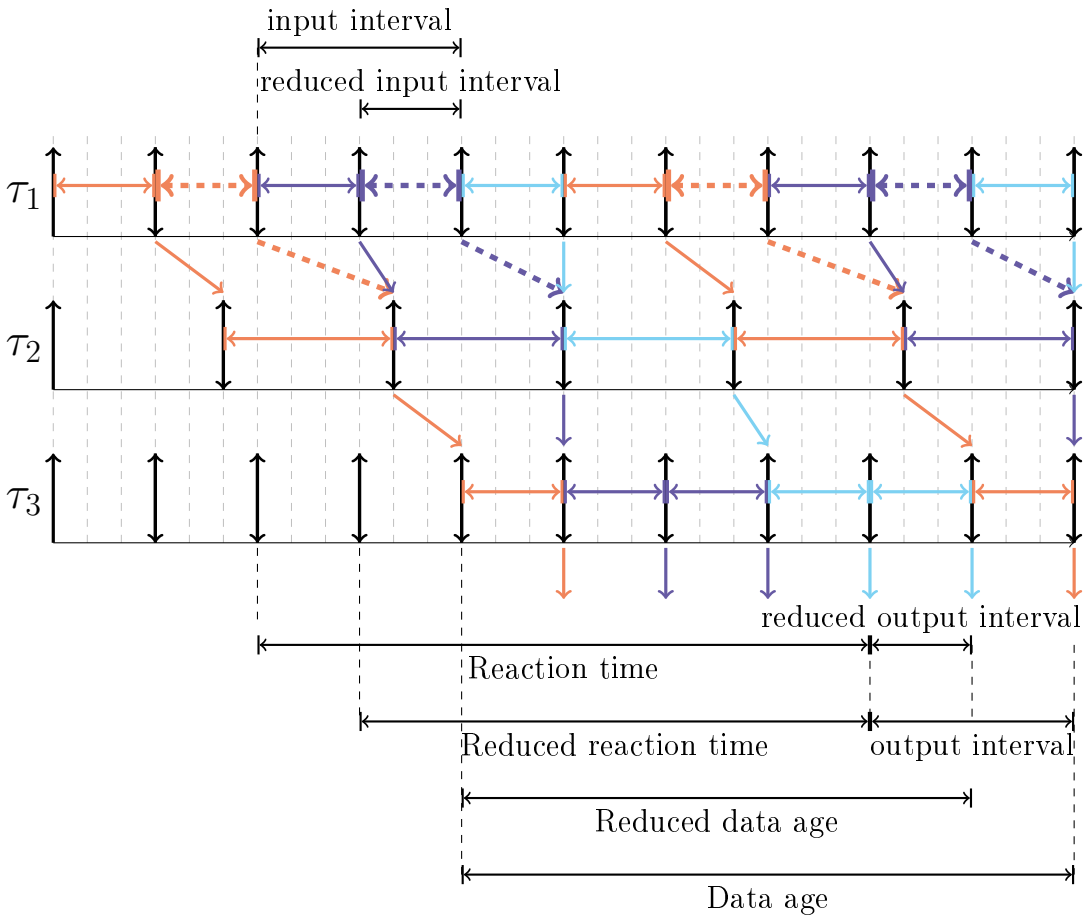


Figure III.3: *End-to-end latency metrics considering extended input/output intervals*

well as the E2E latency metrics according to the definitions made by Günzel et al. [128].

Following the new definitions proposed by Günzel et al. [128] for the input and output interval, the latency metric previously known as data age in [10] is now known as Reduced Data Age (RDA). Naturally, the Maximum Reduced Reaction Time (MRDA) of a CEC is the maximum RDA value among all the possible propagation paths the chain has. The new *data age* latency metric now considers the extended output interval, meaning that it now covers the scenario in which the system might suffer an actuation delay. The notion of the reaction time latency metric remains the same as in [10]. The newly added Reduced Reaction Time (RRT) latency metric was introduced by Günzel et al. [128] to the latency model as a way to keep the notions of input and output intervals balanced. Similarly to the MRDA latency metric, the Maximum Reduced Reaction Time (MRRT) of a CEC is the maximum RRT value among all the possible propagation paths the chain has. From now on we will follow the E2E latency model proposed by Günzel et al. [128].

Although Feiertag et al. formally defined in [10] four latency metrics for a CEC, they didn't propose methods to compute each metric. Becker et al. [131] were first to propose an analytical method to compute the MRDA latencies of a multi-rate CEC

applying the implicit communication paradigm. In [131], Becker et al. also proposed the use of a heuristic method to augment a task set by means of job-level dependencies such that the CECs present in the system can respect their MRDA constraints. By adding job-level dependencies between specific task instances, the method proposed by Becker et al. [131] restricts the possible propagation paths a CEC could have, which, in turn, bounds the MRDA latency values of the CEC. Becker et al. also showed in [131] that when correctly placed, job-level dependencies can lead to smaller MRDA latency values.

In [132], Becker et al. extended their previous work in [131] and presented analytical methods to compute the MRDA latency values of multi-rate CECs applying the other communication paradigms available in AUTOSAR. Becker et al. also proposed in [132] methods to verify the MRDA latency constraints of multi-rate CECs during different development stages. They showed that the accuracy of their timing analysis method increases as development stages progress and more information about the system such as exact schedule and WCRT values are available. Hamann et al. presented in [7] a method to model the implicit and LET communication paradigms present in AUTOSAR into the commercial timing analysis tools usually used by the automotive industry. By modeling as explicit communication the timing behavior of the implicit and LET paradigms, Hamann et al. [7] showed how more complex communication paradigms could be integrated in timing analysis tools that only support the explicit communication paradigm. In [7], Hamann et al. also evaluated the MRT latencies and communication overhead of their converted model in a full blown engine management system.

Kloda et al. [133] were the first to formally define an analytical method to compute the MRT latency of multi-rate CECs applying the implicit communication paradigm on a multiprocessor platform with a partitioned fixed-priority preemptive scheduler. In [133], Kloda et al. also showed how their method can be implemented and how easily it can be integrated on top of any tool that provides the WCRT of tasks. In [134], Martinez et al. presented a formal implementation of the implicit and LET communication paradigms. In [134], Martinez et al. also analyzed the impact introduced by those two communication paradigm with respect to their MRT and MRDA latencies, as well as their memory footprint. In [135], Tang et al. proposed the use of a new triggering mechanism for tasks that are part of a multi-rate CEC and apply the implicit communication paradigm. Instead of strictly using event-triggered (ET) or time-triggered (TT) activation for all tasks in a CEC, Tang et al. [135] introduced a third activation model named *event-triggered with data refreshing* to provide better MRT bounds. The new activation model proposed by Tang et al. [135] combines the benefits of ET and TT by limiting buffer size and allowing data refreshing. Results show that their method produce tighter bounds when compared to other CECs with TT tasks applying the implicit communication paradigm.

Extending the work previously done in [131] by Becker et al., Klaus et al. proposed in [136] an extension of the Real-Time Systems Compiler (RTSC) [137] so that it could enforce job-level dependencies on multi-core systems. In [136], Klaus et al. also showed that when job-level dependencies are not correctly placed, standard allocation and scheduling methods that focus on task-local deadlines are not suitable when trying

to optimize the MRDA latency of multi-rate CECs.

Schlatow et al. presented in [138] an analysis of the MRDA latency metric for periodic offset-synchronized tasks present in multi-rate CECs. In [138], Schlatow et al. proposed an Mixed Integer Linear Program (MILP)-based optimization method to minimize the MRDA latency of multi-rate CECs by assigning parameters such as priorities and offsets to the tasks that are part of those CECs. Schlatow et al. showed that their method can find feasible optimization solutions in under a minute, but the amount of time to find optimal solutions can be impractical in some cases. Biondi et al. presented in [139] a method to implement the LET communication paradigm on multi-core platforms. By using additional dedicated tasks, Biondi et al. [139] showed how to realize the logical behavior of the LET paradigm on a actual system. In [139], Biondi et al. also presented a comparison between two methods for optimizing the placement of communication labels in memory. The first optimization method utilizes MILP formulation, while the second models the optimization problem as a genetic algorithm. Experiments showed that their MILP formulation obtains optimal label placement in less than 2 hours, while the genetic algorithm can take up to 4 hours.

In [140], Biondi and Di Natale showed that the implementation of the LET paradigm on multi-core systems bring time and data-flow determinism to the system with almost negligible runtime overhead. Their proposed implementation was evaluated on a Infineon TriBoard v2 equipped with an Aurix TC275 microcontroller alongside the ERIKA Real-Time Operating System (RTOS) [141]. Results showed the potential of using the LET paradigm on multi-core systems for scheduling memory accesses and avoid excessive contention. Pazzaglia et al. proposed in [142] an efficient implementation of the LET paradigm on multi-core systems to enforce causality and determinism in safety-critical applications. The goal of their proposed method was to find an optimal mapping placement for runnables and labels given an automotive application. Their optimization metric was to minimize the WCRT of all tasks present in the system. In [142], the authors also provide a schedulability analysis test for partitioned tasks executing according to the LET paradigm.

In [143], Kloda et al. proposed to decouple the communication interval from the periods of tasks as an extension for the Timing Definition Language (TDL) [144], a successor of Giotto [8]. However, Kloda et al. [143] did not present a method to formally compute the additional offsets or how to actually decouple the communication intervals. Dürr et al. introduced in [145] the concepts of *immediate forward* (backward, respectively) job chains for the analysis of MRT and MRDA latency metrics. Below we formally define the two types of job chains as done in [145] by Dürr et al.

Definition 7. ***Immediate Forward Job Chain.*** *Let $c = \{J_1, J_2, \cdots, J_n\}$ be a job chain of a cause-effect chain $E = \{\tau_1, \tau_2, \cdots, \tau_n\}$. Job chain c is a immediate forward job chain if all $i \in \{2, \cdots, n\}$ the job J_i is the earliest job of task τ_i that has a read access no earlier than the write access of J_{i-1}. That is, job J_i is the earliest job of task τ_i that fulfills the properties of a job chain (See Definition 4).*

Definition 8. ***Immediate Backward Job Chain.*** *Let $c = \{J_1, J_2, \cdots, J_n\}$ be a job chain of a cause-effect chain $E = \{\tau_1, \tau_2, \cdots, \tau_n\}$. Job chain c is a immediate backward job chain if all $i \in \{n-1, \cdots, 1\}$ the job J_i is the latest job of task τ_i that has a write access no later than the read access of J_{i+1}. That is, job J_i is the latest job of task τ_i that fulfills the properties of a job chain (See Definition 4).*

Using their definitions of *immediate forward* (backward, respectively) job chains, Dürr et al. [145] showed that their bounds for the MRT and MRDA latency metrics were tighter than the one proposed by Davare et al. [130]. In [146], Ernst et al. extended the applicability of the LET communication paradigm from the scope of a single electronic control unit (ECU) to the scope of a full distributed system. Ernst et al. [146] proposed the concept of System Level LET (SYS-LET) in which time and data-flow determinism as well as composability of the LET paradigm are guaranteed in a distributed system. The SYS-LET model proposed by Ernst et al. [146] has relaxed synchronization requirements, it is schedule agnostic, and can be applied along with other communication paradigms. In [147], Günzel et al. presented a timing analysis of asynchronized distributed multi-rate CECs assuming the implicit communication paradigm.

Over the last decade, multiple techniques have been proposed in the literature to allow the computation and optimization of the E2E latencies of multi-rate CECs applying the LET communication paradigm. Kordon and Tang proposed in [148] a method to determine the MRDA based on a task dependency graph. They developed an algorithmic tool that models precisely the dependency between tasks allowing the tool to handle the complete graph at once rather than just specific chains. The proposed tool is also capable of extracting the critical paths of the graph.

In [149], Martinez et al. presented a phase-aware LET analysis to improve the E2E latencies of multi-rate CECs. Martinez et al. showed in [149] that by correctly phasing specific tasks applying the LET paradigm in a CEC, their method can lead to slightly better MRDA latency values. In [149], Martinez et al. also formally presented a set of equations to compute the MRDA latency metric of synchronous and asynchronous CECs. By identifying the *publishing* and *reading* points of the tasks that are part of the CEC, their method identifies precisely all the possible propagation paths a multi-rate CEC has within its hyperperiod. Martinez et al. [149] showed that by analyzing the latencies of all the propagation paths within the hyperperiod of the CEC, their method can compute the exact MRDA latency value of a given CEC.

In [150], Bradatsch et al. proposed a method to reduce the MRDA latency of multi-rate CECs applying the LET paradigm. In the literature, Bradatsch et al. [150] was the first to demonstrate how the communication intervals of tasks applying the LET paradigm could be safely modified. The method proposed by Bradatsch et al. [150] analyzes a schedulable tasks set according to the Rate-Monotonic (RM) scheduling policy and sets the communication intervals of tasks equal to their WCRT. By setting the end of a communication interval to be equal to task's WCRT instead of its period, the method proposed by Bradatsch et al. [150] reduces the overall output jitter of tasks which in turn leads to shorter E2E latency values.

In [151], Wang et al. presented a two-step optimization method to improve the MRDA of multi-rate CECs applying the LET communication paradigm. Improving the work

previously done in [149] by Martinez et al., Günzel et al. proposed in [152] an optimal phasing method to improve the MRT and MDA latencies of CECs applying the LET paradigm. Differently from what was proposed by Bradatsch et al. in [150], the method presented by Günzel et al. [152] does not consider that the length of the communication intervals of tasks applying the LET paradigm can be less than their period interval.

In this chapter, we identified the challenges related to the timing analysis of multi-rate CEC in safety-critical RT systems, more especially automotive applications with tasks applying the LET communication paradigm. We motivated our work based on the WATERS challenge [127] and presented different methods that were proposed in the literature to solve the main points of the challenge. In this chapter, we also formally defined a CEC and its E2E latency metrics, which we will further analyze in Chapter IV where we present our contributions.

In Chapter IV, we present in detail our contributions for the timing analysis of multi-rate CECs. More specifically, we present our methods to optimize the MRT and MDA latency values of multi-rate CECs applying the LET communication paradigm. We also present an analytical method to compute the E2E latencies of synchronous and asynchronous multi-rate CECs with optimized communication intervals. In Chapter IV, we also propose a method to reduce the system utilization of safety-critical systems running tasks applying the LET paradigm and with reconfigured communication intervals. We also investigate how to increase the feasibility of multi-rate CECs to meet their timing constraints in system with multiple execution modes. A framework to configure the communication intervals of LET tasks is presented in Chapter IV as well as the heuristic functions used to decide how the communication intervals should be configured.

Chapter IV

Timing Analysis of Multi-Rate Cause-Effect Chains

Many modern safety-critical systems have control functionalities that must react to external events with a predefined actuation within a specific interval of time. When failing to perform their predefined actuation within the defined interval, as a safety-critical system, their failure might endanger multiple human lives or lead to some catastrophic event. Therefore, for many safety-critical control systems it is pivotal to ensure that the time elapsed between the detection of an external event and its corresponding actuation response is within the time interval defined during design time.

With a focus on automotive systems, we discussed in chapters I and III how important it is and which are the challenges involved in guaranteeing system's timing requirements both at the task and functionality level. As explained in chapters I and III, modern automotive applications have system functionalities that are modeled as a chained sequence of communicating tasks. This ordered sequence of tasks that continuously communicate with other by means of shared resources are know as cause-effect chains (CECs). Since the tasks that are part of those CECs might have different activation periods and non-deterministic access time to shared resources, computing the required time for an input to traverse from one end of the CEC to the other is not trivial.

As discussed throughout Chapter III, inter-task communication paradigms directly affect the performance of control algorithms present in automotive applications based on the AUTomotive Open System ARchitecture (AUTOSAR) standard. For that reason, embedded software engineers from the automotive industry are focused on finding methods to guarantee and optimize the end-to-end (E2E) latencies of the CECs present in their systems. During the last decade, many methods considering the different communication paradigms present in AUTOSAR were proposed in the literature. As presented in Section III.2, there are many works that focused on providing E2E latency bounds for multi-rate CECs applying the implicit and(or) the Logical Execution Time (LET) communication paradigm [131, 132, 134, 139, 133, 147, 148]. Currently in the literature, there are only few works that focused on optimizing the Maximum Reaction Time (MRT) and Maximum Data Age (MDA) latency metrics of CECs considering the LET communication paradigm [150, 149, 151, 152].

Although some contributions have been made, there is still a research gap in the literature when it comes to the optimization of multi-rate CECs applying the LET paradigm. As discussed by Matic et al. in [3], by abstracting system semantics in the form of assuming that the length of tasks' communication interval is equal to their period length or deadline, the LET communication paradigm trades E2E latency for system composability. In fact, only the done by Bradatsch et al. [150] investigated this trade off and proposed methods to *safely* shrink the length of tasks' communication intervals (i.e. making them a subset of the original intervals). Although, the method proposed by Wang et al. [151] also applies an approach to shrink communication intervals in a second optimization step, their method does not achieve a safe shrinking in the sense of making them a subset of the original intervals, because in the first optimization step, the communication intervals are shifted.

It is worth mentioning that Kloda et al. [143] were the first to analyze the pessimistic E2E latencies trade offs of the LET paradigm as discussed by Matic et al. [3]. In [143], Kloda et al. discussed about the possible benefits of decoupling communication intervals from tasks' period length as an extension for the Timing Definition Language (TDL) [144], a successor of Giotto [8] which originated the LET paradigm. However, Kloda et al. did not present in [143] a formal method to decouple the communication intervals of the tasks applying the LET paradigm.

Whereas the optimization method proposed by Bradatsch et al. [150] can obtain tighter E2E latencies for all the CECs present in the system, their method follows a conservative approach that does not consider the possibility of changing tasks' parameters (e.g., phase, deadline) or the underlying schedule for further latency optimization. In [150], Bradatsch et al. restricted their method to the solely optimization of the Maximum Reduced Reaction Time (MRDA) latency metric. In fact, they didn't looked to the problem of optimizing the communication intervals of tasks applying the LET paradigm from the perspective of system utilization and systems with multiple execution modes.

As introduced in Section I.4, this dissertation focus on proposing methods and tools to optimize several parameters of safety-critical systems with multi-rate CECs applying the LET communication paradigm. In Section IV.1, we formally define our system model and the structure of the LET-based communication intervals adopted by our methods. In Section IV.2 we formally introduce our Schedule-Aware communication model, while in Section IV.3 we demonstrate how the MRT and MDA latency metrics can be computed for our model considering synchronous and asynchronous CECs. In Section IV.4, we show how the correct reconfiguration of communication intervals by means of precedence constraints can further optimize the MRT and MDA latencies of multi-rate CECs. In Section IV.5, we analyze the data-flow determinism enabled by the LET paradigm and demonstrate how our method to configure communication intervals can assist in minimizing system utilization. In Section IV.6, we demonstrate how to increase the feasibility of multi-rate CECs that apply the LET model to meet their E2E latency requirements in systems with multiple execution modes. In Section IV.7, we present our framework, while in Section IV.8, we introduce the heuristic functions used by our framework.

IV.1 System and Communication Models

In this section, we formally introduce the system model considered by the methods proposed in this dissertation. In order to ease the explanation of our system model, we split our model into two versions (*basic* and *extended*) according to the requirements of each contribution. We start by defining the *basic version* of our system model which is considered for the contributions proposed in sections IV.2, IV.3, IV.4, IV.5. For the contribution presented in Section IV.6, we extend our basic system model to its *extended version* which considers a multi-mode execution system where timing requirements of the CECs change from one execution mode to another. The details of our extended system model is detailed in the beginning of Section IV.6.1.

IV.1.1 Tasks and Jobs

In this dissertation, we consider a multi-core system composed of homogeneous cores. We use c_i to represent the i^{th} core present in our system, where $i \in \mathbb{N}^+$. We consider Γ as a task set containing periodic hard real-time (RT) tasks. A task $\tau \in \Gamma$ is a tuple $(C_\tau, T_\tau, D_\tau, \phi_\tau)$, where C_τ represents the worst-case execution time (WCET) of τ, T_τ is the period length, D_τ is the relative deadline, and ϕ_τ is the phase. We assume that tasks have constrained deadlines, i.e., the deadline is equal or less than the period length. Each task τ releases jobs $J_\tau(i)$ recurrently, where $i \in \mathbb{N}^+$. Job $J_\tau(i)$ has a release time at $\phi_\tau + (i-1)T_\tau$ and an absolute deadline D_τ units later. If the referenced task τ is clear, we omit the index τ from J for brevity. We use J_τ to denote a generic job of τ. A schedule $\mathcal{S}$ specifies the execution behavior of all jobs of τ according to a given scheduling policy. The *start time* of $J(i)$ according to $\mathcal{S}$ is $s^{\mathcal{S}}_{J(i)}$, while the *finishing time* is $f^{\mathcal{S}}_{J(i)}$. Note that if the referenced schedule $\mathcal{S}$ is clear, we omit the index $\mathcal{S}$ for brevity.

We assume that all jobs finish until their absolute deadline, which can be guaranteed by determining the Worst-Case Response Time (WCRT) $R_\tau := \sup_{\mathcal{S}} f^{\mathcal{S}}_{J_\tau(i)} - (\phi_\tau + (i-1)T_\tau)$ using time-demand analysis [18] and checking if $R_\tau \leq D_\tau$. The minimal start time is achieved if all jobs execute for 0 time units, and the maximal finishing time is achieved if all jobs execute for the WCET. Furthermore, if the execution time is fixed (to the WCET or to 0), then the schedule within interval $[\Phi(\Gamma) + H(\Gamma), \Phi(\Gamma) + 2H(\Gamma))$ repeats after $\Phi(\Gamma) + 2H(\Gamma)$ every $H(\Gamma)$ units of time, where $\Phi(\Gamma) = \max(\{\phi_\tau \mid \tau \in \Gamma\})$ is the maximal phase and $H(\Gamma) = LCM(\{T_\tau \mid \tau \in \Gamma\})$ is the hyperperiod [153, 158].

IV.1.2 Communication Model

We assume that communication between tasks happens through the use of shared resources and to be based on the LET communication paradigm. Each task τ performs its inter-task communications periodically according to a fixed and well defined communication interval L_τ (See Figure IV.1). We use $|L_\tau|$ to represent the length of interval L_τ. The inputs and outputs of τ are logically updated at the boundaries of L_τ. $read(\tau)$ and $write(\tau)$ are relative points in time representing the boundaries of L_τ. That is,

$$L_\tau = [read(\tau), write(\tau)]. \tag{IV.1}$$

Initially, we consider the traditional approach for LET in which L_τ spans over the period length of τ. That is, at release, task τ *logically* reads its input, while at the end of its period length it *logically* writes its outputs. We assume that each task τ is equipped with a read-phase $\phi_\tau^R \in \mathbb{R}$ and a write-phase $\phi_\tau^W \in \mathbb{R}$, which define the relative points in time where $read(\tau)$ and $write(\tau)$ take place. Therefore, we define the values of $read(\tau)$ and $write(\tau)$ as:

$$read(\tau) = \phi_\tau^R, \qquad write(\tau) = \phi_\tau^W, \tag{IV.2}$$

where $\phi_\tau^R = \phi_\tau$ and $\phi_\tau^W = \phi_\tau + T_\tau$, since L_τ spans over the period length of τ. By setting $read(\tau) = \phi_\tau^R$ and $write(\tau) = \phi_\tau^W$, we ensure that no job of τ starts before its input data is read or finishes after its output data is written. Moreover, the assumption that $|L_\tau| = T_\tau$ trivially ensures a safe communication interval, in the sense that for a feasible schedule, every job of τ always executes within its communication interval.

At the job level, each job J of τ has a communication interval L_J. The boundaries of L_J define when job J *logically* receives (read) input from a shared resource, as well as when it *logically* transmits (write) output to a shared resource. We define the communication interval L_J of job J as:

$$L_J = [read(J), write(J)], \tag{IV.3}$$

where $read(J)$ and $write(J)$ are absolute points in time. The logical read and write-events occur periodically according to T_τ. Therefore, for the i^{th} job $J(i)$ of task τ, we derive the logical read and write points of $J(i)$ as:

$$read(J(i)) = (i-1)T_\tau + read(\tau), \qquad write(J(i)) = (i-1)T_\tau + write(\tau). \tag{IV.4}$$

Figure IV.1 summarizes the notation of our communication model. As depicted in the traditional model, Figure IV.1 shows that the read and write accesses of each job to a shared resource only occurs at the boundaries of L_J.

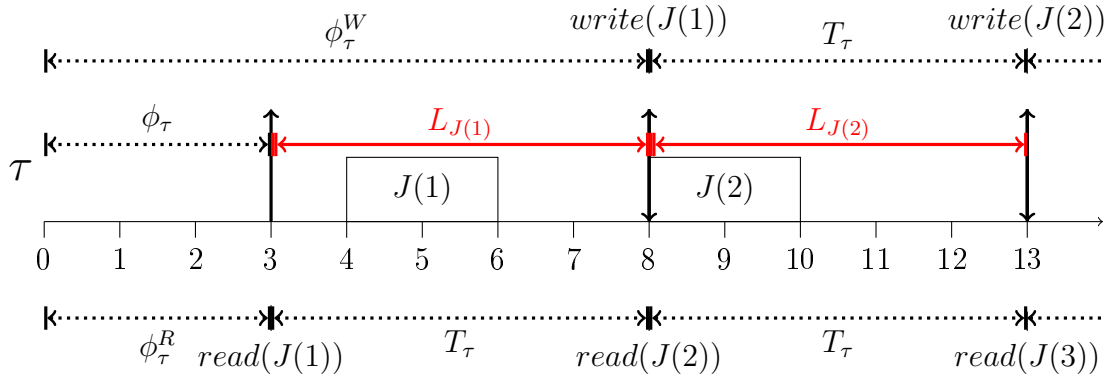


Figure IV.1: *Communication interval $L_{J(i)}$ for a given job $J(i)$ of task τ.*

Cause-Effect Chain (CEC)

We model a CEC as a finite ordered sequence of tasks communicating by means of shared resources. We represent a CEC as:

$$E = \{\tau_1 \rightarrow \tau_2 \rightarrow \cdots \rightarrow \tau_{|E|}\}, \tag{IV.5}$$

$|E|$ being the number of tasks in E. Function $E(i)$ returns the i^{th} task in E, $i \in \{1, 2, \cdots, |E|\}$. For a pair of tasks in E, the $\rightarrow$ operator indicates that τ_{i+1} acts as a consumer/reader task, while τ_i as a producer/writer task.

We assume that E samples (acquires data) at every $read(J_1)$, J_1 being a job of task $E(1)$. We use z to represent the extended input interval defined by Günzel et al. [128] when computing the MRT of a CEC. Likewise, we use z' to represent the extended output interval ([128]) when computing the MDA of a CEC.

Job Chain

Given a CEC E, a job chain (c^E) is a finite sequence of jobs representing one of the possible data propagation paths of E. We model a job chain as:

$$c^E = \{J_1 \rightarrow \cdots \rightarrow J_{|E|}\}. \tag{IV.6}$$

In a job chain, the following requirements must be respected:

- J_i is a job of $E(i)$, $i \in \{1, 2, \cdots, |E|\}$.
- The data written by J_i is read by J_{i+1}. That is, $write(J_i) \leq read_{L_{J_{i+1}}}$ for all $i \in \{1, 2, \cdots, |E| - 1\}$

We use c_i^E to represent the i^{th} job chain of E, $i \in \mathbb{N}^+$, while function $l(c^E)$ returns its length, i.e., the interval between $write(J_{|E|})$ and $read(J_1)$.

Table IV.1.2 summarizes the notation used throughout this dissertation.

Variable	Definition
$\tau \in \Gamma$	a task τ in task set Γ
$J_\tau(i), i \in \mathbb{N}^+$	i^{th} job of a task τ
$s^{\mathcal{S}}_{J(i)}$	start time of $J(i)$ according to schedule $\mathcal{S}$
$f^{\mathcal{S}}_{J(i)}$	finishing time of $J(i)$ according to schedule $\mathcal{S}$
L_J	communication interval of job J
$read(J)$	logical read-event of job J
$write(J)$	logical write-event of job J
$H(E)$	hyperperiod of CEC E
$E(i)$	the i^{th} task in E, $i \in \{1, 2, \cdots, \|E\|\}$
J_i	a job of $E(i)$, $i \in \{1, 2, \cdots, \|E\|\}$
c_i^E	i^{th} job chain of CEC E
$l(c^E)$	time interval between $write(J_{\|E\|})$ and $read(J_1)$

Table IV.1: *Notation Table*

IV.2 Safe Reconfiguration of Communication Intervals

As motivated in sections I.4 and III.1, designing safety-critical applications in embedded systems, such as in AUTOSAR, requires complex analysis for temporal properties, such as E2E latencies. During early design phases, designers abstract system semantics, e.g., scheduling algorithms, in order to reduce the complexity of verifying the temporal behavior of system functionalities. However, abstracting system semantics in this manner results in pessimistic E2E latencies as discussed by Matic et al. in [3].

The LET communication paradigm emerged as a solution which significantly reduces timing analysis complexity of multi-rate CECs. By having fixed inter-task communication points that are independent from the actual task execution, the LET communication paradigm brings timing and data-flow determinism to the analysis of multi-rate CECs. As a result, the LET communication paradigm helps abstracting from the actual system implementation (scheduling choices), and consequently reduces complexity of analysis, but at the cost of increased pessimism, i.e., larger E2E latency values.

In this section, we propose a method to reduce the pessimism present in the LET communication paradigm by taking scheduling choices into consideration. The method shortens and shifts communication intervals based on a chosen scheduling algorithm, i.e., we make them a subset of the original interval. The procedure to shorten and shift communication intervals consists in modifying the read-phase ϕ_τ^R and write-phase ϕ_τ^W values of each task $\tau \in \Gamma$ based on scheduling information. Therefore, our method is applied later in the design process (after scheduling choice). By analyzing a feasible schedule, our method derives new boundaries ($read(J)$ and $write(J)$) for the communication intervals.

As design phases progress and a schedule has to be determined, our method can be applied during later design phases to optimize the E2E latencies of multi-rate CECs applying the LET communication paradigm. Without losing the timing and data-flow determinism of LET, our method keeps tasks periodic and with well-defined communication points.

In Section IV.2.1, we show how by extracting information from a feasible schedule, our method reduces the pessimism present in the LET paradigm while maintaining its deterministic characteristics and tasks' periodicity. Instead of setting $|L_\tau| = T_\tau$, i.e., $L_\tau = [0, T_\tau]\ \forall \tau \in \Gamma$, our method derives new relative points in time for $read(\tau)$ and $write(\tau)$. By repositioning the boundaries of L_τ and therefore the boundaries of L_J for a job J, our method postpones its logical read-event and prepones its logical write-event.

IV.2.1 Defining Schedule-Aware Intervals

In order to derive new boundaries for L_τ and make it *schedule-aware* $\forall \tau \in \Gamma$, our method sets L_τ's length and position equal to a new time interval I_τ, where the length of I_τ is $C_\tau \leq |I_\tau| \leq T_\tau$. Note that the new interval I_τ is a sub-set of L_τ. As in L_τ, $read(I_\tau)$ and $write(I_\tau)$ delimit the boundaries of I_τ, i.e., $I_\tau = [read(I_\tau), write(I_\tau)]$, where $read(I_\tau) = \phi_\tau^R$ and $write(I_\tau) = \phi_\tau^W$.

By extracting information from the hyperperiod of a feasible schedule $\mathcal{S}$, our method defines the length and position of I_τ. As explained in Section IV.1.1, $\mathcal{S}$ specifies the start time s_J and the finishing time f_J for all $J \in \tau$. Below, we formally define the terms *relative start time* (S_J) and *relative finishing time* (F_J), which are necessary to derive the communication boundaries for I_τ. For the definitions below we assume that each job executes for its full WCET.

Definition 9. ***Relative Start Time (of a Job).*** *Let $J(i)$ be the i^{th} job of task τ in schedule $\mathcal{S}$. The relative start time ($S_{J(i)}$) of a job is the start time of $J(i)$ minus its release time.*

$$S_{J(i)} = s_{J(i)} - (\phi_\tau + (i-1)T_\tau) \tag{IV.7}$$

Definition 10. ***Relative Finishing Time (of a Job).*** *Let $J(i)$ be the i^{th} job of task τ in schedule $\mathcal{S}$. The relative finishing time ($F_{J(i)}$) of a job is the finishing time of $J(i)$ minus its release time.*

$$F_{J(i)} = f_{J(i)} - (\phi_\tau + (i-1)T_\tau) \tag{IV.8}$$

Note that according to the definitions above, depending on when each $J \in \tau$ executes between its release and deadline, the values for S_J and F_J can change for each $J \in \tau$, e.g., one job may execute early during its period, while another job may execute later. Therefore, in order to keep the timing and data-flow determinism of the LET paradigm when setting $L_\tau = I_\tau$, it is necessary to ensure that all $J \in \tau$ have a *common periodic communication interval* I_J, i.e., I_J respects S_J and F_J, for all $J \in \tau$.

Our method sets communication boundaries for I_τ by computing the *earliest relative start time* (ES_τ) and the *latest relative finishing time* (LF_τ) of a task τ based on $\mathcal{S}$. Below we formally define the terms ES_τ and LF_τ.

Definition 11. ***Earliest Relative Start Time (of a Task).*** *Let τ be a task in schedule $\mathcal{S}$. The earliest relative start time (ES_τ) of τ is the minimum relative start time among all jobs of τ in $\mathcal{S}$.*

$$ES_\tau = \min_{\forall J \in \tau} S_J \tag{IV.9}$$

Definition 12. ***Latest Relative Finishing Time (of a Task).*** *Let τ be a task in schedule $\mathcal{S}$. The latest relative finishing time (LF_τ) of τ is the maximum relative finishing time among all jobs of τ in $\mathcal{S}$.*

$$LF_\tau = \max_{\forall J \in \tau} F_J \tag{IV.10}$$

Example: In order to exemplify definitions 9 to 12, Figure IV.2 shows a schedule $\mathcal{S}$ for three tasks that are part of a CEC E, where $E = \{\tau_1 \rightarrow \tau_2 \rightarrow \tau_3\}$. In this example, we analyze task τ_2, which has three jobs during $\mathcal{S}$'s hyperperiod (HP). Following definitions 9 and 10, the first job of τ_2, $J(1)$, has $S_{J(1)} = 2$ and $F_{J(1)} = 3$, while $J(2)$ has $S_{J(2)} = 0$ and $F_{J(2)} = 1$. $J(3)$ has $S_{J(3)} = 1$ and $F_{J(3)} = 2$. Following definitions 11 and 12, $ES_{\tau_2} = 0$ and $LF_{\tau_2} = 3$ for task τ_2. Likewise, by using definitions 9 to 12, we can compute the ES_τ and LF_τ values for the other tasks in E, where $ES_{\tau_1} = 0$ and $LF_{\tau_1} = 1$ for task τ_1, while $ES_{\tau_3} = 1$ and $LF_{\tau_3} = 2$ for task τ_3. In Figure IV.2, we highlight a safe communication interval for each job of task τ_2 within the HP $H(E)$ of the CEC E.

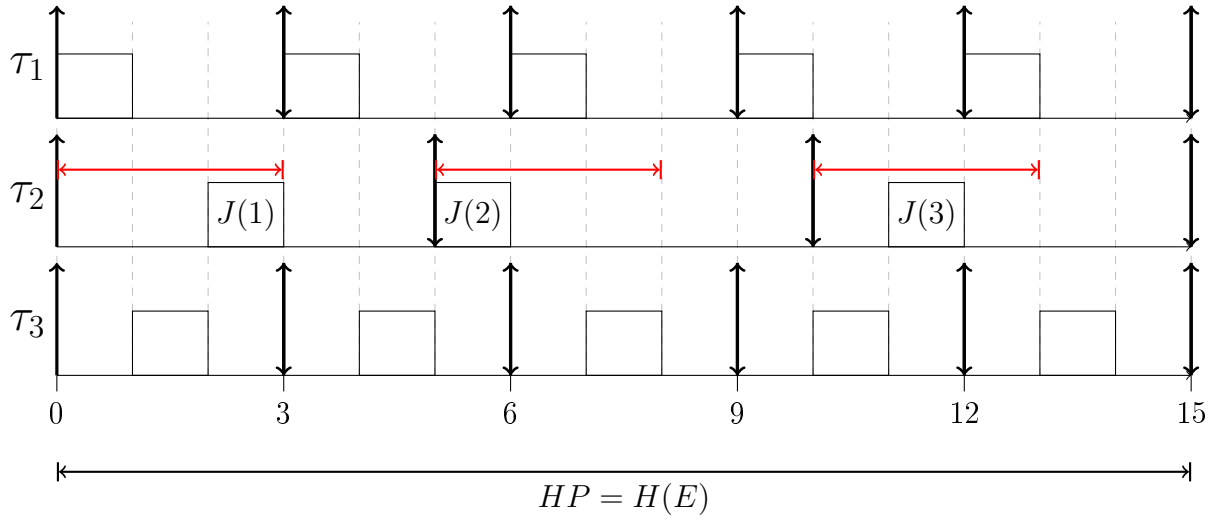


Figure IV.2: *Schedule $\mathcal{S}$ for cause-effect chain $E = \{\tau_1 \rightarrow \tau_2 \rightarrow \tau_3\}$*

As shown in Figure IV.2, by extracting scheduling information (ES_τ and LF_τ values) from a feasible schedule, our method can configure safe communication intervals I_τ for each $\tau \in \Gamma$. Our method reconfigures when τ starts its communication intervals by shifting them according to ES_τ. That is, it sets the read-phase of τ to be:

$$\phi_\tau^R = \phi_\tau = \phi'_\tau + ES_\tau, \tag{IV.11}$$

where ϕ'_τ is τ's initial phase. Likewise, our method reconfigures when τ ends its communication interval by shortening the length of I_τ according to LF_τ. That is, it sets the write-phase of τ to be:

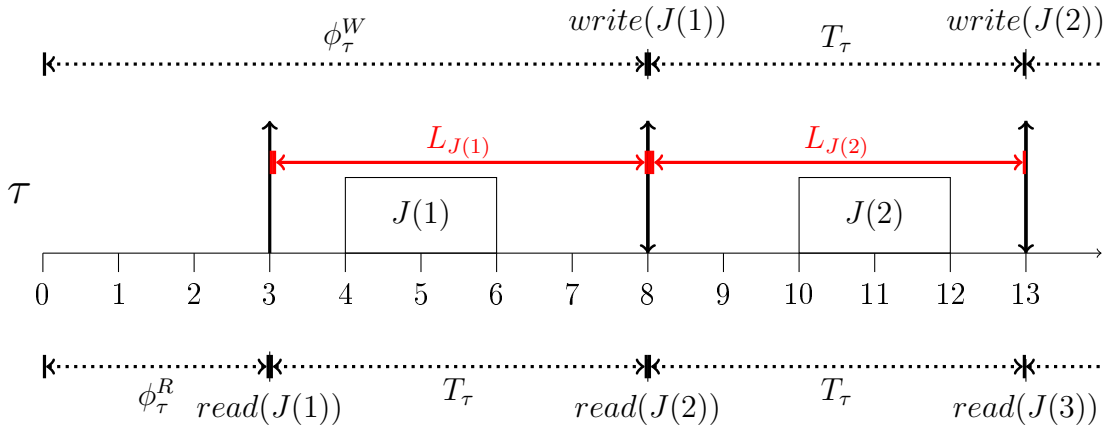
$$\phi_\tau^W = \phi'_\tau + LF_\tau \tag{IV.12}$$

Since the boundaries of the communication interval I_τ are relative points in time with respect to ϕ_τ, and our method shifts τ according to ES_τ as in Equation IV.11, then:

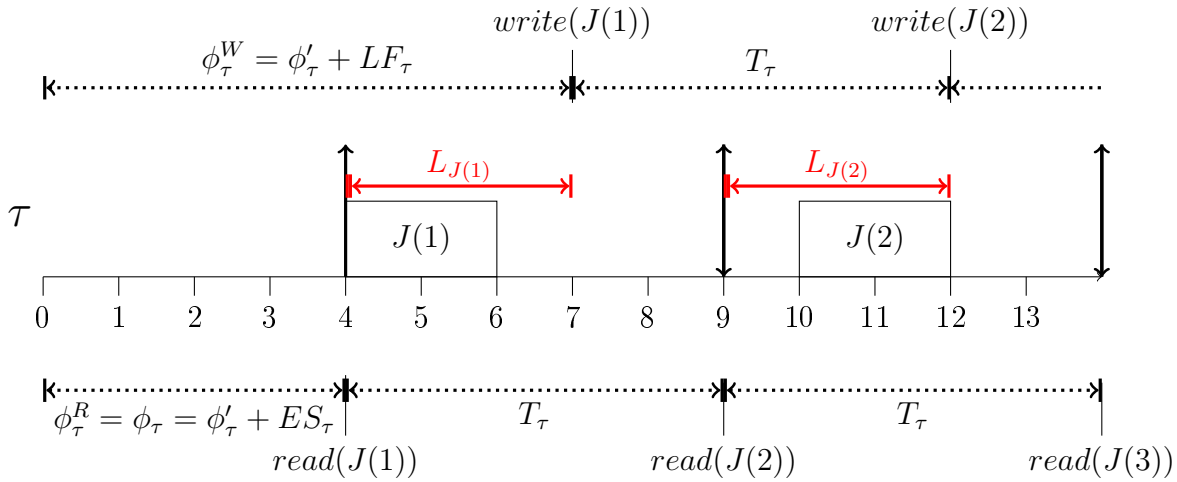
$$read(I_\tau) = 0, \qquad write(I_\tau) = LF_\tau - ES_\tau. \tag{IV.13}$$

Therefore, by setting $I_\tau = [read(I_\tau), write(I_\tau)]$, and $L_\tau = I_\tau$, we can say that our method safely shortens and shifts the communication intervals of τ in the sense that our method guarantees that all jobs of τ always execute within the new communication intervals L_τ. For instance, let us consider task τ_2 shown in Figure IV.2, instead of setting $L_{\tau_2} = [0, T_{\tau_2}]$, i.e., $[0, 5]$, our method sets $L_{\tau_2} = I_{\tau_2} = [0, 3]$. Note that for the example provided above, τ_2 is not shifted because $ES_{\tau_2} = 0$.

In Figure IV.3, we show the differences in configuration between communication intervals that are not schedule-aware and the ones that are for a task τ with $ES_\tau = 1$ and $LF_\tau = 4$.



(a) *Communication intervals tradionally configured as* $|L_\tau| = T_\tau$



(b) *Communication intervals reconfigured as* $|L_\tau| = LF_\tau - ES_\tau$

Figure IV.3: *Differences in configuration for the communication intervals of a task* τ

Figure IV.3b shows that, if the phase ϕ_τ of task τ is adjusted according its read-phase ϕ_τ^R, the resulting communication intervals are safe even when jobs do not execute for their full WCET.

Note that for task sets scheduled according to a fixed-priority scheduling policy (e.g., Rate-Monotonic (RM) , Deadline-Monotonic (DM)), applying a phase ES_τ to τ does not change its priority or affect the schedulability of task set Γ. That is, for any $J(i) \in \tau$, $i \in \mathbb{N}^+$, there is no $J(i)$ that executes before $(i-1)T_\tau + ES_\tau$ according to $\mathcal{S}$. All J of τ have to wait at least ES_τ time units after its release in order to execute. Therefore, as long as our method postpones the release of τ by ES_τ time units, $\forall \tau \in \Gamma$, the schedulability of the task set remains unaffected. For task sets scheduled according to the Earliest Deadline First (EDF) policy, our method adjusts task's deadlines to $D_\tau = LF_\tau - ES_\tau$, $\forall \tau \in \Gamma$, in order to enforce execution within the new communication intervals. Before applying changes to task's parameters and reconfiguring the communication intervals of all tasks in Γ scheduled according to EDF, our method uses feasibility tests based on Processor Demand Analysis (PDA) [156, 155, 154] to verify if the new communication intervals are feasible and if the tasks can safely execute within them.

In the section below, we show how to practically enforce the deterministic timing and data-flow behavior of the LET communication paradigm when reconfiguring the intervals according to our schedule-aware model.

IV.2.2 Practically Enforcing Deterministic Communication Points

The LET communication paradigm assumes that tasks' inputs and outputs are logically updated at the beginning and end of their communication intervals. It assumes that these *updates* incur zero computation time. However, up to date system platforms are not infinitely fast to realize such behavior. Therefore, the deterministic ordering and atomic execution of these updates must be explicitly enforced in order to preserve the desired logical behavior of the paradigm. At the implementation level, the LET communication paradigm is usually enforced by means of hardware/software mechanisms [157][139]. In the literature, different ways to implement the logical behavior of the LET model have been proposed [7][134][139][142][140].

One way to implement the logical behavior of the LET communication paradigm is by implementing auxiliary tasks, which are responsible for updating the inputs and outputs of the tasks [7][139]. For instance, the implementation of a task τ would consist of three tasks: (i) a reader (*copy-in*) task τ^R; (ii) an execution task τ^E; (iii) a writer (*copy-out*) task τ^W. Hereafter, we discuss one of the possibilities of implementing the logical behavior of the LET paradigm when shortening and shifting communication intervals. Note that other implementations respecting the logical behavior and the deterministic ordering of execution of the tasks could be used.

At the beginning of L_τ, the auxiliary reader task (τ^R) copies all the input data necessary for τ^E's execution to a local variable. At the end of L_τ, the auxiliary writer task (τ^W) copies τ^E's output data to the shared variable that will be accessed by the next task in the CEC that τ belongs to. Therefore, tasks τ^R and τ^W are responsible for updating the inputs and outputs of τ at the boundaries of L_τ.

Since the auxiliary reader (resp. writer) task has to be executed as close as possible to the start (resp. end) of L_τ, τ^R and τ^W have to be characterized by a very short WCET and a very high priority level [139]. Therefore, the correct positioning of τ^R and τ^W is important in order to achieve the expected logical behavior of τ [7].

We use the parameters of τ and its communication interval L_τ to set the parameters of τ^R, τ^E and τ^W. The auxiliary tasks τ^R and τ^W are periodic task with period T_{τ^R} (resp. T_{τ^W}) equal to T_τ. The correct positioning of τ^R and τ^W is done by means of an additional phase. Therefore, τ^R has a phase $\phi_{\tau^R} = \phi_{\tau^E} = \phi'_\tau + ES_\tau$, while τ^W has a phase $\phi_{\tau^W} = \phi'_\tau + LF_\tau$. We consider that the execution times of the copy-events done by τ^R and τ^W are very short and their overhead included in C_τ. The parameters $(C_{\tau^E}, T_{\tau^E}, D_{\tau^E})$ of τ^E are equal to the parameters of τ. We consider that auxiliary tasks with the same phase and period can be grouped into a single auxiliary task in order to reduce possible switching overhead.

Figure IV.4 shows how the auxiliary tasks τ^R and τ^W of a given task τ can be modeled. Note that in Figure IV.4 $\tau^E = \tau$.

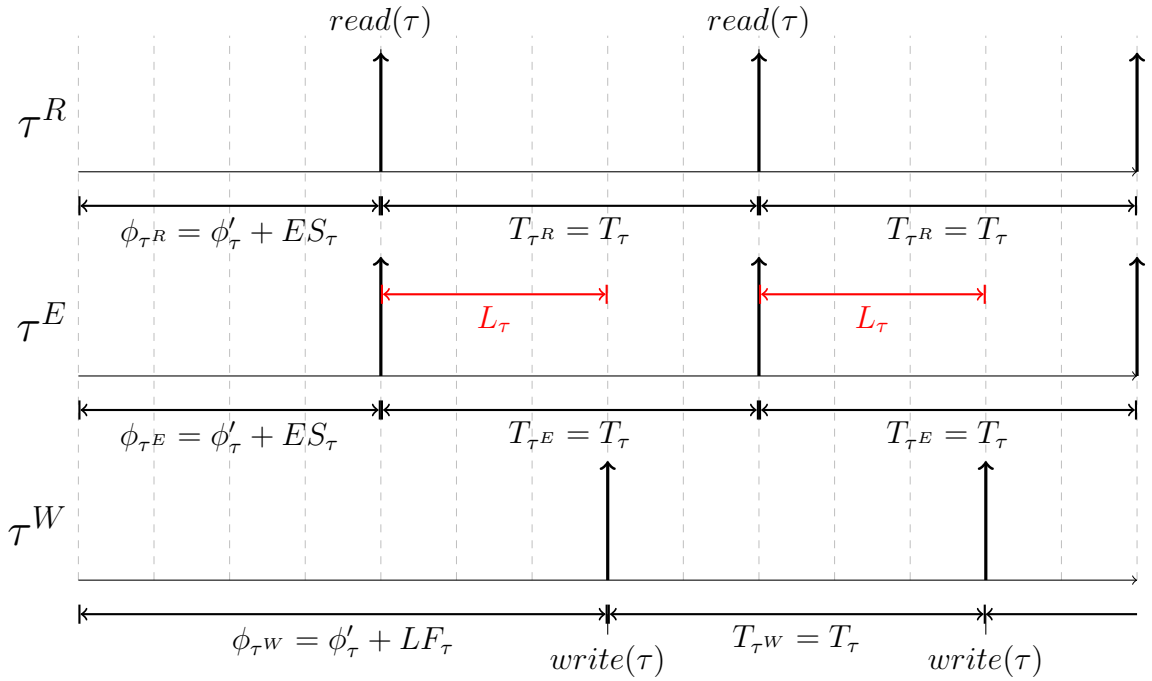


Figure IV.4: *Schedule $\mathcal{S}$ for cause-effect chain $E = \{\tau_1 \rightarrow \tau_2 \rightarrow \tau_3\}$*

The copying-events carried out by the auxiliary tasks (τ^R and τ^W) are facilitated via highest priority interrupts, which are added for each periodic auxiliary task. These interrupts are activated with the same period as the tasks they correspond to, allowing them to execute immediately. In order to obtain *freshest* data values, write output-events are favored over read input-events. Therefore, the write output-events are given the highest priority in the system, whereas read input-events are given the second highest priority.

In Section IV.3, we present an analytical framework to compute the MRT and MDA latency values of multi-rate CECs with tasks applying our schedule-aware LET paradigm.

IV.3 Timing Analysis of Asynchronous Cause-Effect Chains

Given that the logical read and write-events of a job J applying the LET paradigm happen at well defined points in time, it is possible to identify precisely the communication points when data propagates from one task to the other during runtime. Martinez et al. presented in [149] an analytical framework to compute the communication points between two tasks applying the LET paradigm assuming that $|L_\tau| = T_\tau$, $\forall \tau \in \Gamma$. Since our schedule-ware model presented in Section IV.2 shortens and shifts communication intervals, the assumption made by Martinez et al. in [149] does not hold anymore and their analysis is no longer applicable to our model. Although in [150] Bradatsch et al. presented a model where $|L_\tau| \neq T_\tau$, $\forall \tau \in \Gamma$, they did not present an analytical method to compute the MRDA latency value of multi-rate CECs.

In this section, we present an analytical framework to compute the communication points of tasks applying our schedule-aware model where $L_\tau = I_\tau$. Before presenting the core theorems that are part of our analytical framework, we first define below the terms *writing point* [1] and *reading point*, which are later used in theorems IV.1 and IV.2.

Definition 13. ***Writing Point.*** *Given a pair of tasks in a CEC E, where $\tau_i \rightarrow \tau_{i+1}$, $i \in \{1, 2, \cdots, |E| - 1\}$. Let a writing point ($W^n_{\tau_i,\tau_{i+1}}$) be the n^{th} point in time where the resource shared by tasks τ_i and τ_{i+1} is updated by τ_i. After $W^n_{\tau_i,\tau_{i+1}}$, no other logical write-event write(τ_i) of task τ_i will take place before the next logical read-event read(τ_{i+1}) of task τ_{i+1}.*

Definition 14. ***Reading Point.*** *Given a pair of tasks in a CEC E, where $\tau_i \rightarrow \tau_{i+1}$, $i \in \{1, 2, \cdots, |E| - 1\}$. Let a reading point ($R^n_{\tau_i,\tau_{i+1}}$) be the n^{th} point in time where the resource shared by tasks τ_i and τ_{i+1} is read by τ_{i+1}. The logical read-event read(τ_{i+1}) of task τ_{i+1} after the n^{th} writing point $W^n_{\tau_i,\tau_{i+1}}$ of task τ_i is the reading point $R^n_{\tau_i,\tau_{i+1}}$.*

It is important to highlight that depending on the period relation between τ_i and τ_{i+1} in a CEC E where $\tau_i \rightarrow \tau_{i+1}$, not all $write(J_i)$ events from jobs of τ_i are writing points according to Definition 13. That is, if τ_i has a shorter period length compared to τ_{i+1}, multiple $write(J_i)$ events from jobs of τ_i will take place before a $read(J_{i+1})$ event from a job of τ_{i+1}, but only the last $write(J_i)$ before the next $read(J_{i+1})$ is considered a writing point. However, if τ_i has a period length longer or equal compared to τ_{i+1}, every $write(J_i)$ event from jobs of τ_i will be considered a writing point according to Definition 13 (See Theorem IV.1).

Likewise, depending on the period relation between τ_i and τ_{i+1}, not all $read(J_{i+1})$ events from jobs of τ_{i+1} are reading points according to Definition 14. That is, if τ_i has a longer period length compared to τ_{i+1}, multiple $read(J_{i+1})$ events from jobs of τ_{i+1} will take place after a $write(J_i)$ event from a job of τ_i, but only the first $read(J_{i+1})$ after a $write(J_i)$ is considered a reading point. However, if τ_i has a period length shorter or equal compared to τ_{i+1}, every $read(J_{i+1})$ event from jobs of τ_{i+1} will be considered a reading point according to Definition 14 (See Theorem IV.2).

[1] Writing point is also known as *publishing point* in [149]

Bellow, we formally present the theorems from our analytical framework.

Theorem IV.1. *Let $\tau_i \rightarrow \tau_{i+1}$ be a pair of communicating tasks applying the LET communication paradigm with schedule-aware intervals in a CEC E, where $T_{\tau_i} \leq T_{\tau_{i+1}}$, $i \in \{1, 2, \cdots, |E| - 1\}$. Then the reading and writing points between τ_i and τ_{i+1} can be computed as:*

$$R^n_{\tau_i,\tau_{i+1}} = nT_{\tau_{i+1}} + \phi_{\tau_{i+1}} \tag{IV.14}$$

$$W^n_{\tau_i,\tau_{i+1}} = \left\lfloor \frac{R^n_{\tau_i,\tau_{i+1}} - \phi_{\tau_i} - write(\tau_i)}{T_{\tau_i}} \right\rfloor T_{\tau_i} + \phi_{\tau_i} + write(\tau_i) \tag{IV.15}$$

$$n \geq \begin{cases} 0, & \text{if } \phi_{\tau_{i+1}} \geq \phi_{\tau_i} + write(\tau_i) \\ \left\lceil \frac{\phi_{\tau_i} + write(\tau_i) - \phi_{\tau_{i+1}}}{T_{i+1}} \right\rceil, & otherwise \end{cases}$$

Proof. We prove this theorem in two steps.

Step 1 (Reading point): Since $T_{\tau_i} \leq T_{\tau_{i+1}}$, there is always one job of τ_i being released between two job releases of τ_{i+1}. That means, τ_i always updates the resource shared with τ_{i+1} before each logical read-event of τ_{i+1}. By definition (Section IV.1.2), the inputs of τ_{i+1} are logically updated at $read(\tau_{i+1})$, which occurs every $T_{\tau_{i+1}}$ time units after $\phi_{\tau_{i+1}}$. Therefore, a reading point between τ_i and τ_{i+1} occurs periodically according to $T_{\tau_{i+1}}$ such that

$$R^n_{\tau_i,\tau_{i+1}} = nT_{\tau_{i+1}} + \phi_{\tau_{i+1}}. \tag{IV.16}$$

Step 2 (Writing point): By definition (Section IV.1.2), the logical write-event of the *first* job of τ_i logically occurs at $\phi_{\tau_i} + write(\tau_i)$. That means, any reading point $R^n_{\tau_i,\tau_{i+1}}$ has to be $\geq$ than $\phi_{\tau_i} + write(\tau_i)$.

If $R^n_{\tau_i,\tau_{i+1}} = \phi_{\tau_i} + write(\tau_i)$, then

$$W^n_{\tau_i,\tau_{i+1}} = R^n_{\tau_i,\tau_{i+1}}. \tag{IV.17}$$

If $R^n_{\tau_i,\tau_{i+1}} > \phi_{\tau_i} + write(\tau_i)$, the writing point that immediately precedes $R^n_{\tau_i,\tau_{i+1}}$ depends on how many logical write-events of task τ_i happened within the spanned time interval $[\phi_{\tau_i} + write(\tau_i), R^n_{\tau_i,\tau_{i+1}}]$. Since logical write-events of τ_i occur periodically according to T_{τ_i}, the number of logical write-events within the considered interval is $\left\lfloor \frac{R^n_{\tau_i,\tau_{i+1}} - \phi_{\tau_i} - write(\tau_i)}{T_{\tau_i}} \right\rfloor$. Hence, the writing point that immediate precedes $R^n_{\tau_i,\tau_{i+1}}$ is equal to

$$W^n_{\tau_i,\tau_{i+1}} = \left\lfloor \frac{R^n_{\tau_i,\tau_{i+1}} - \phi_{\tau_i} - write(\tau_i)}{T_{\tau_i}} \right\rfloor T_{\tau_i} + \phi_{\tau_i} + write(\tau_i). \tag{IV.18}$$

□

Theorem IV.2. *Let $\tau_i \rightarrow \tau_{i+1}$ be a pair of communicating tasks applying the LET communication paradigm with schedule-aware intervals in a CEC E, where $T_{\tau_i} > T_{\tau_{i+1}}$ and $i \in \{1, 2, \cdots, |E|-1\}$. Then the writing and reading points between τ_i and τ_{i+1} can be computed as:*

$$W^n_{\tau_i,\tau_{i+1}} = nT_{\tau_i} + \phi_{\tau_i} + write(\tau_i) \tag{IV.19}$$

$$R^n_{\tau_i,\tau_{i+1}} = \left\lceil \frac{W^n_{\tau_i,\tau_{i+1}} - \phi_{\tau_{i+1}}}{T_{\tau_{i+1}}} \right\rceil T_{\tau_{i+1}} + \phi_{\tau_{i+1}} \tag{IV.20}$$

$$n \geq \begin{cases} 0, & \text{if } \phi_{\tau_{i+1}} \leq \phi_{\tau_i} + write(\tau_i) \\ \left\lfloor \frac{\phi_{\tau_{i+1}} - (\phi_{\tau_i} + write(\tau_i))}{T_i} \right\rfloor, & \text{otherwise} \end{cases}$$

Proof. We prove this theorem in two steps.

Step 1 (Writing point): Since $T_{\tau_i} > T_{\tau_{i+1}}$, there is always one job of τ_{i+1} being released between two job releases of τ_i. That means, τ_{i+1} always reads the resource shared with τ_i after each logical write-event of τ_i. By definition (Section IV.1.2), the outputs of τ_i are logically updated at $write(\tau_i)$, which occurs every T_{τ_i} after $\phi_{\tau_i} + write(\tau_i)$. Therefore, a writing point between τ_i and τ_{i+1} occurs periodically according to $T_{\tau_{i+1}}$ such that

$$W^n_{\tau_i,\tau_{i+1}} = nT_{\tau_i} + \phi_{\tau_i} + write(\tau_i). \tag{IV.21}$$

Step 2 (Reading point): By definition (Section IV.1.2), the logical read-event of the *first* job of τ_{i+1} logically occurs at $read(\tau_{i+1})$, i.e., $\phi_{\tau_{i+1}}$. That means, any reading point $R^n_{\tau_i,\tau_{i+1}}$ has to be $\geq$ than $\phi_{\tau_{i+1}}$.
By intuition, if $W^n_{\tau_i,\tau_{i+1}} \leq \phi_{\tau_{i+1}}$, then

$$R^n_{\tau_i,\tau_{i+1}} = \phi_{\tau_{i+1}}. \tag{IV.22}$$

If $W^n_{\tau_i,\tau_{i+1}} > \phi_{\tau_{i+1}}$, the reading point that immediately succeeds $W^n_{\tau_i,\tau_{i+1}}$ depends on how many logical read-events of task τ_{i+1} happened within the spanned time interval $[\phi_{\tau_{i+1}}, W^n_{\tau_i,\tau_{i+1}}]$. Since logical read-events of τ_{i+1} occur periodically according to $T_{\tau_{i+1}}$, the number of logical read-events within the considered interval is $\left\lceil \frac{W^n_{\tau_i,\tau_{i+1}} - \phi_{\tau_{i+1}}}{T_{\tau_{i+1}}} \right\rceil$. Hence, intuitively, the read point of $\tau_i \rightarrow \tau_{i+1}$ that immediately succeeds $W^n_{\tau_i,\tau_{i+1}}$ is equal to

$$R^n_{\tau_i,\tau_{i+1}} = \left\lceil \frac{W^n_{\tau_i,\tau_{i+1}} - \phi_{\tau_{i+1}}}{T_{\tau_{i+1}}} \right\rceil T_{\tau_{i+1}} + \phi_{\tau_{i+1}}. \tag{IV.23}$$

□

In Section IV.3.1, we show how to compute the E2E latencies of a given CEC using theorems IV.1 and IV.2.

IV.3.1 Computing End-to-End Latencies

In Section IV.2, we presented a method that extracts information from a feasible schedule and safely reconfigure the communication intervals of tasks applying the LET paradigm. Since our method shortens and shifts communication intervals, the analytical framework proposed by Martinez et al. in [149] was no longer applicable for the computation of the E2E latencies of multi-rate CECs applying our schedule-aware model. For that reason, we derived in Section IV.3 theorems IV.1 and IV.2 to identify the points in time when data propagates from one task to another in a CEC during runtime. In the following, we demonstrate how to identify which job chains have to be investigated during the E2E timing analysis of a multi-rate CEC applying our schedule-aware model. We also show how to compute the reaction time and data age latency metrics related to each of those job chains, as well as the MRT and MDA latency values of the CEC.

Identifying Job Chains

Due to the under/oversampling nature of multi-rate CECs, some job chains might have jobs in common. For instance, when multiple actuations (outputs) of a CEC E are based on the same sampled (input) data, multiple job chains have the same $J(i)$ as their J_1 in E. As explained in Section III.1.2 and in Definition 6, when analyzing the MDA of a multi-rate CEC, it is necessary to know for how long a sampled value affects the actuation of the CEC, i.e., know the time interval between two consecutive job chains with different $J(i)$ as their J_1. In order to distinguish job chains that have different $J(i)$ as their J_1, we define below the term *primary job chain*.

Definition 15. ***Primary Job Chain.*** *Given a set of job chains that have the same $J(i)$ as their J_1. We call primary job chain (pc^E), the job chain with the earliest write($J_{|E|}$) in the set. We represent the i^{th} primary job chain of E as:*

$$pc_i^E = \{J_1 \rightarrow J_2 \rightarrow \cdots \rightarrow J_{|E|}\}, \tag{IV.24}$$

where $i \in \mathbb{N}^+$. Given a primary job chain pc_i^E, we represent the next primary job chain of E after pc_i^E as pc_{i+1}^E. That is, pc_{i+1}^E is the first job chain after pc_i^E that has a different $J(i)$ as J_1. We use function $pc_i^E(k)$ to return the k^{th} job in the given primary job chain, where $k \in \{1, 2, \cdots, |E|\}$. Function $l(pc_i^E)$ return the time interval between write($pc_i^E(|E|)$) and read($pc_i^E(1)$). That is, $l(pc_i^E)$ returns the length of pc_i^E, i.e., the time interval between input data being read by J_1 and an output being produced by $J_{|E|}$.

In order to identify the primary job chains of a given CEC E, our method first identifies the job chains of E using algorithms 1 and 2 which are defined below. As shown by Leung et al. in [158], for a task set with periodic tasks and phases, a schedule $\mathcal{S}$ repeats itself every $LCM(T_{\tau_1}, \cdots, T_{\tau_{|E|}})$ units of time. Assuming that the task set consists of the tasks that part of CEC E, the schedule within interval $[\Phi(E) + H(E), \Phi(E) + 2H(E))$ repeats after $\Phi(E)+2H(E)$ every $H(E)$ units of time, where $\Phi(E) = \max(\phi_{\tau_1}, \cdots, \phi_{\tau_{|E|}})$ and $H(E) = LCM(T_{\tau_1}, \cdots, T_{\tau_{|E|}})$. Günzel et al. showed in [128] that for the sake of computing the MRT and MDA latency metrics of a given CEC E applying the LET

paradigm, it is enough to analyze the job chains within one of the repetition intervals after the *warm-up* period. As defined by Günzel et al. [128], the warm-up period covers the time interval required for an input to fully traverse a CEC E for the first time. Note that the job chains present in the spanned time interval $[\Phi(E) + H(E), \Phi(E) + 2H(E))$ are also present in the interval $[\Phi(E) + (n+1)H(E), \Phi(E) + (n+2)H(E))$, but shifted by $nH(E)$, where $n \geq 1$.

In order to identify which jobs are part of the job chains within one of the repetition intervals, our method applies Algorithm 1 to all jobs of $E(|E|)$ released within the repetition interval. The process of identifying jobs chains starts with the $J_{|E|}$ that has the *earliest* $read(J_{|E|})$ within the repetition interval. Our method starts the analysis with the last communication task pair in E, i.e., $\tau_{|E|-1} \rightarrow \tau_{|E|}$. By applying Algorithm 1 to a given $J_{|E|}$ of $E(|E|)$, our method obtains the writing point related to a $J_{|E|-1}$.

Algorithm 1 Compute writing points

Input: $read(J_i), \tau_{i-1}, \tau_i$

1: **if** $T_{\tau_{i-1}} \leq T_{\tau_i}$ **then**

2: According to Theorem IV.1

3: **else**

4: According to Theorem IV.2

5: **end if**

6: Find $m_{\max}$, the largest value of m such that $W^m_{\tau_{i-1},\tau_i} \leq read(J_i)$

Output: $W^m_{\tau_{i-1},\tau_i}$

Using the output of Algorithm 1 as an input to Algorithm 2, our method computes the writing and reading points of the previous communication task pair in E, i.e., $\tau_{|E|-2} \rightarrow \tau_{|E|-1}$. Note that the process of identifying a job chain starts from its tail ($J_{|E|}$) and stops when its head (J_1) is found. Our method identifies the remaining part of the job chain by applying Algorithm 2 recursively to all the other communication task pairs in E until it obtains the reading and writing point related to a job J_1 of $E(1)$.

Algorithm 2 Compute reading points

Input: $W^n_{\tau_{i-1},\tau_i}, \tau_{i-2}, \tau_{i-1}$

1: **if** $T_{\tau_{i-2}} \leq T_{\tau_{i-1}}$ **then**

2: According to Theorem IV.1

3: **else**

4: According to Theorem IV.2

5: **end if**

6: Find $m_{\max}$, the largest value of m such that $R^m_{\tau_{i-2},\tau_{i-1}} < W^n_{\tau_{i-1},\tau_i}$

7: $n = m_{\max}$

8: Compute $R^n_{\tau_{i-2},\tau_{i-1}}$ and $W^n_{\tau_{i-2},\tau_{i-1}}$

9: $i = i - 1$

Output: $R^n_{\tau_{i-1},\tau_i}$ and $W^n_{\tau_{i-1},\tau_i}$

Our method uses Equation IV.25 on *each* identified writing point ($W^n_{\tau_k,\tau_{k+1}}$), $k \in \{1, 2, \cdots, |E| - 1\}$, in order to determine the i^{th} job of each task τ_k that is part of the job chain under analysis.

$$i = 1 + \frac{W^n_{\tau_k,\tau_{k+1}} - write(\tau_k)}{T_{\tau_k}} \tag{IV.25}$$

Following Definition 15, our method selects, within the repetition interval, the primary job chains among the set of job chains identified using algorithms 1 and 2. Our method adds the selected primary job chains to a set ζ and use it to compute the MRT and MDA latency metrics of CEC E. Later in Section IV.5.1, we formally introduce an algorithm to identify jobs chain and filter out the one that can be classified as primary job chains.

In order to facilitate understanding and exemplify how algorithms 1 and 2 identify job chains, we reuse in Figure IV.5 the task set described in the example provided in Section IV.2. Note that here we consider that the communication intervals of tasks were already shortened and shifted according to our method, i.e., $L_{\tau_1} = [0, 1], L_{\tau_2} = [0, 3], L_{\tau_3} = [0, 1]$ and $\phi_{\tau_3} = 1$. Figure IV.5 shows and extended version of the schedule shown in Figure IV.2. As discussed in the beginning of this section, when identifying primary job chains to compute E2E latencies, it is necessary to consider the *warm-up* interval of the CEC [128]. Since the communication intervals of tasks were already shortened and shifted, we are also considering an asynchronous task set. That is, the schedule within $[\Phi(E)+H(E), \Phi(E)+2H(E))$ repeats every $H(E)$ after $\Phi(E)+2H(E)$.

Figure IV.5 shows that the first output fully processed by the CEC is produced by task τ_3 at time point 11. Since $\Phi(E) + H(E) = 16$, the first repetition interval spans from time point 16 until time point 31. The process to identify primary job chains starts with the first job from the last task in the CEC within the repetition interval under analysis. In this example, task τ_3 has 5 jobs $(J(6), J(7), J(8), J(9), J(10))$ within the repetition interval $[\Phi(E) + H(E), \Phi(E) + 2H(E))$. Note that in Figure IV.5 we represent jobs from task τ_3 just by their index number in order to improve their label readability given the page width. We highlight in Figure IV.5 the jobs that are part of job chains within the repetition interval.

Let us start with job $J(6)$ from task τ_3 that has $read(J_{\tau_3}(6)) = 16$. By applying Algorithm 1 to $J_{\tau_3}(6)$, the largest value of m from Equation IV.19 that satisfies $W^m_{\tau_2,\tau_3} \leq read(J_{\tau_3}(6))$ in Algorithm 1 is $m = 2$. That is, $W^2_{\tau_2,\tau_3} = 13$, which corresponds to $write(J_{\tau_2}(3))$ of task τ_2 according to Equation IV.25. Note that although $read(J_{\tau_3}(6))$ is a read-event of τ_3, the reading point related to $W^2_{\tau_2,\tau_3}$ is $R^2_{\tau_2,\tau_3} = 13$, which corresponds to $read(J_{\tau_3}(5))$ according to Theorem IV.2. That is, $read(J_{\tau_3}(5))$ is the first read-event of τ_3 after the $W^2_{\tau_2,\tau_3}$. By using the output of Algorithm 1 as input to Algorithm 2, the largest value of m from Equation IV.14 that satisfies $R^m_{\tau_1,\tau_2} < W^2_{\tau_2,\tau_3}$ is $m = 2$. That is, $R^2_{\tau_1,\tau_2} = 10$, which corresponds to $read(J_{\tau_2}(3))$ of task τ_2. By following Equation IV.15 from Theorem IV.1, the writing point that corresponds $m = 2$ and $R^2_{\tau_1,\tau_2} = 10$ is $W^2_{\tau_1,\tau_2} = 10$, which corresponds to $write(J_{\tau_1}(4))$ of task τ_1 according to Equation IV.25. In Figure IV.5, we highlighted in orange the jobs that are part of the first job chain within the repetition interval, i.e., $c^E_i = \{J_{\tau_1}(4), J_{\tau_2}(3), J_{\tau_3}(6)\}$. Note that we also highlighted in orange the job chain starting at time point 24 ($\{J_{\tau_1}(9), J_{\tau_2}(6), J_{\tau_3}(10)\}$) because it has the same J_{τ_1} as $\{J_{\tau_1}(4), J_{\tau_2}(3), J_{\tau_3}(6)\}$ but shifted by one $H(E)$.

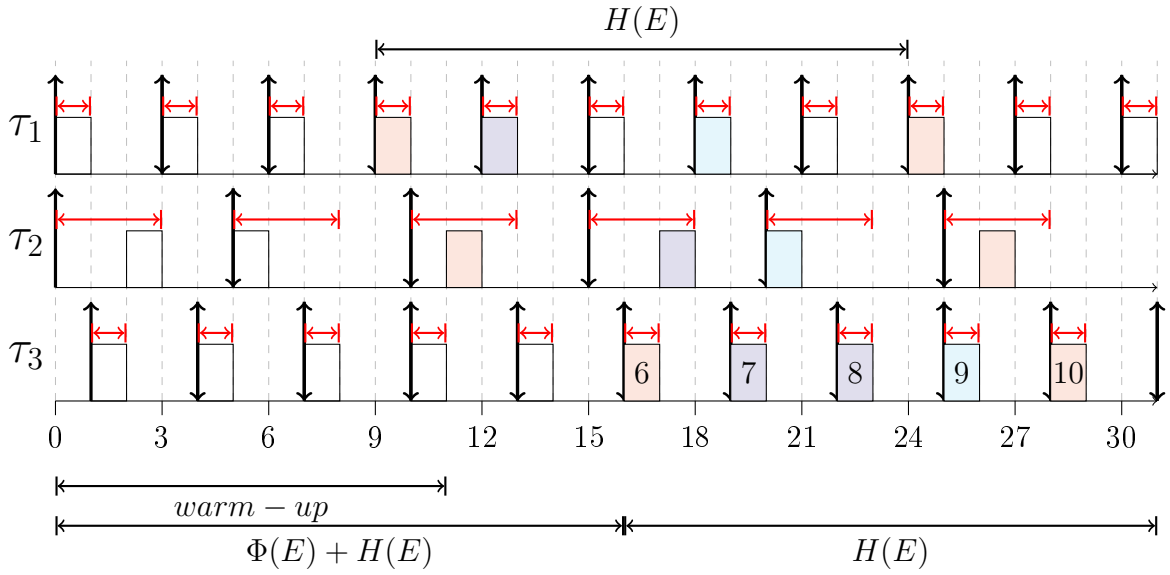


Figure IV.5: *Jobs chains of cause-effect chain E within a repetition interval*

By repeating the process of applying algorithms 1 and 2 to each other job of task τ_3 withing the repetition interval, we obtain the following job chains:

- $c^E_{i+1} = \{J_{\tau_1}(5), J_{\tau_2}(4), J_{\tau_3}(7)\}$
- $c^E_{i+2} = \{J_{\tau_1}(5), J_{\tau_2}(4), J_{\tau_3}(8)\}$
- $c^E_{i+3} = \{J_{\tau_1}(7), J_{\tau_2}(5), J_{\tau_3}(9)\}$
- $c^E_{i+4} = \{J_{\tau_1}(9), J_{\tau_2}(6), J_{\tau_3}(10)\}$.

Once our method identifies all job chains within a repetition interval, it follows Definition 15 and adds to a set ζ only the jobs chains that can be classified as a primary job chain. Therefore, for the example shown above, ζ consists of:

$$\zeta = \left\{ \begin{array}{c} pc^E_i = \{J_{\tau_1}(4), J_{\tau_2}(3), J_{\tau_3}(6)\}, \\ pc^E_{i+1} = \{J_{\tau_1}(5), J_{\tau_2}(4), J_{\tau_3}(7)\}, \\ pc^E_{i+2} = \{J_{\tau_1}(7), J_{\tau_2}(5), J_{\tau_3}(9)\}, \\ pc^E_{i+3} = \{J_{\tau_1}(9), J_{\tau_2}(6), J_{\tau_3}(10)\} \end{array} \right\}.$$

Note that job chain c^E_{i+2} is not part of ζ because it does not comply with Definition 15. The compliance is not met because job chain c^E_{i+1} is also based on job $J_{\tau_1}(5)$ and it produces an output before job chain c^E_{i+2}, which declassifies c^E_{i+2} as a primary job chain according to Definition 15. Naturally, pc^E_{i+4} is similar to pc^E_{i+1} but shifted by one $H(E)$.

As explained early in the section, all the job chains present in the repetition interval $[\Phi(E) + H(E), \Phi(E) + 2H(E))$ are also present in the next repetition intervals $[\Phi(E) + (n+1)H(E), \Phi(E) + (n+2)H(E))$ but shifted by $nH(E)$, where $n \geq 1$.

Below, we demonstrate how to use primary job chains to compute the MRT and MDA latency metrics of a CEC, as well as the reaction time and data age of each one of them.

Computing the Maximum Reaction Time and Data Age Latencies

As discussed in the paragraphs above, in order to compute the MRT and MDA latency metrics of a given CEC E, our method first needs to initialize the set ζ with all the primary job chains of E within one of the repetition intervals after the warm-up period. Below, we demonstrate how to compute the E2E latencies of a CEC E given the set of primary job chains within a repetition interval.

We follow the definitions for *reaction time* and *data-age* as described by Günzel et al. in [128] and illustrated in Figure III.3. As mentioned in Section IV.1.2, we consider an additional delay z between the occurrence of an external event (input) and its sampling by J_1. In the same manner, we consider that z' represents an additional delay between $write(J_{|E|})$ and the actuation (output). Therefore, in order to compute the MRT and MDA latency metrics according to the definition proposed by Günzel et al. [128], we append z to the beginning of a primary job chain pc^E and z' to its end. As a result,

$$pc^E = \{z, J_1, \cdots, J_{|E|}, z'\}. \tag{IV.26}$$

Maximum Reaction Time

For a given CEC E and its set of primary job chains ζ, our method computes the reaction time for all $pc_i^E \in \zeta$ considering the maximum delay z that an incoming input could suffer. In the worst case, for a primary job chain pc_i^E, an input arrives right *after* the logical read-event of the first job of pc_i^E. As a result, the incoming input has to wait until the logical read-event of the next primary job chain (pc_{i+1}^E) in order to be recognized and propagated through E, Therefore,

$$z = read(pc_{i+1}^E(1)) - read(pc_i^E(1)). \tag{IV.27}$$

We consider, as Günzel et al. [128], that actuation takes place at the logical write-event of $pc_{i+1}^E(|E|)$, i.e., $write(pc_{i+1}^E(|E|))$. Therefore,

$$z' = 0. \tag{IV.28}$$

Equation IV.29 shows how to compute the reaction time of a given primary job chain pc_i^E assuming the maximum input delay.

$$RT(pc_i^E) = z + l(pc_{i+1}^E) + z', \tag{IV.29}$$

Therefore, the MRT of a CEC E can be computed as:

$$MRT(E) = \max_{\forall pc_i^E \in \zeta} RT(pc_i^E). \tag{IV.30}$$

Figure IV.6 shows the individual reaction time values for the primary job chains from our example CEC $E = \{\tau_1 \rightarrow \tau_2 \rightarrow \tau_3\}$, where $MRT(E) = 14$. For instance, the reaction time of pc_1^E can be computed as: $RT(pc_1^E) = (12 - 9) + (20 - 12) + 0 = 11$.

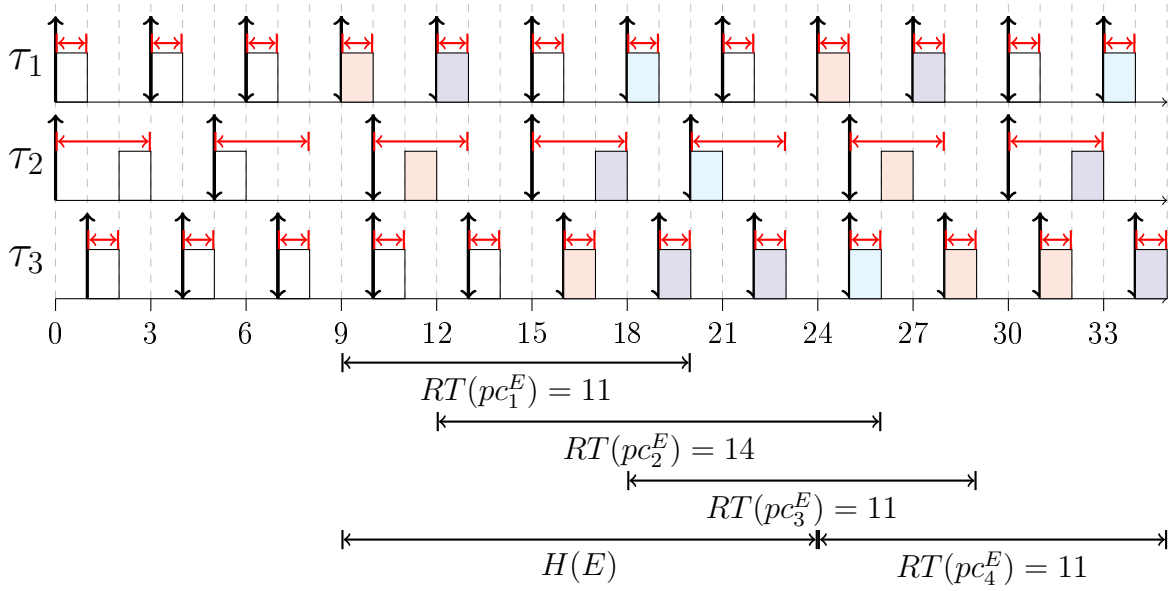


Figure IV.6: *Reaction time for the primary job chains of cause-effect chain E*

Maximum Data Age

For a given CEC E and its set of primary job chains ζ, our method computes the data age for all $pc_i^E \in \zeta$ considering the maximum delay z' that an actuation output could suffer. In the worst case, for a primary job chain pc_i^E, the last actuation based on a given input happens right *before* the logical write-event of the last job of the next primary job chain pc_{i+1}^E, i.e., $write(pc_{i+1}^{E,S}(|E|))$. Therefore,

$$z' = write(pc_{i+1}^E(|E|)) - write(pc_i^E(|E|)). \tag{IV.31}$$

We consider, as Günzel et al. [128], that the input data on which the outputs are based is read during the logical read-event of $pc_i^E(1)$, i.e., $read(pc_i^{E,S}(1))$. Therefore,

$$z = 0. \tag{IV.32}$$

Equation IV.33 shows how to compute the data age of a given primary job chain pc_i^E assuming the maximum output delay.

$$DA(pc_i^E) = z + l(pc_i^E) + z'. \tag{IV.33}$$

Therefore, the MDA of a CEC E can be computed as:

$$MDA(E) = \max_{\forall pc_i^E \in \zeta} DA(pc_i^E). \tag{IV.34}$$

Figure IV.7 shows the individual data age values for the primary job chains from our example CEC $E = \{\tau_1 \rightarrow \tau_2 \rightarrow \tau_3\}$, where $MDA(E) = 14$. For instance, the data age of pc_1^E can be computed as: $DA(pc_1^E) = 0 + (17 - 9) + (20 - 17) = 11$.

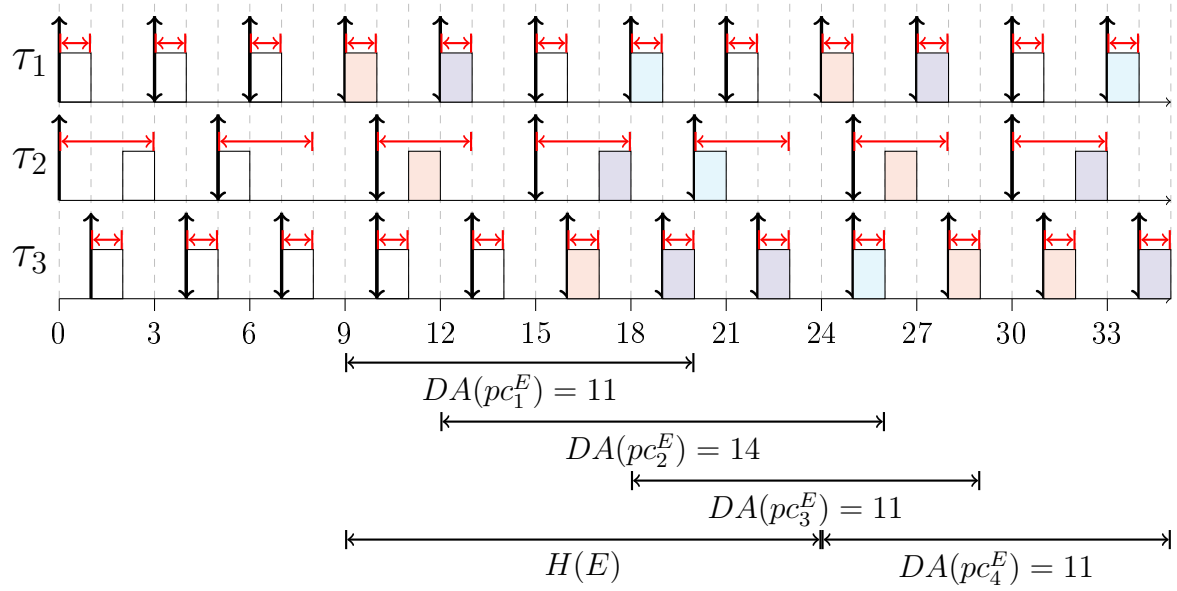


Figure IV.7: *Data age for the primary job chains of cause-effect chain E*

It is worth mentioning that with the contributions presented in sections IV.2 and IV.3.1, we addressed point 1 from the Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS) challenge [125] and points 1, 3 from the extended version of the challenge [7]. Moreover, as shown in Chapter V, our contribution from Section IV.2 greatly improves the E2E latency metrics of multi-rate CECs applying the LET communication paradigm.

IV.4 Schedule Manipulation to Improve End-to-End Latencies

Throughout this dissertation we have been discussing about the LET communication paradigm and its benefits (deterministic timing behavior, fixed communication intervals) to the timing analysis of multi-rate CECs in multi-core systems. In Section IV.2, we showed that the significant pessimism in form of longer E2E latencies present in the LET paradigm can be reduced if the communication intervals of tasks are well configured. In this section, we take one step further to optimize the E2E latencies of multi-rate CECs applying the LET communication paradigm.

As discussed in the beginning of this chapter, before our contribution presented in Section IV.2, only the work done by Bradatsch et al. in [150] proposed a method to safely reconfigure the communication intervals of tasks applying the LET paradigm. Although our schedule-aware model greatly improves the E2E latencies of multi-rate CECs applying the LET paradigm (See Chapter V), it is still a conservative approach that leaves potential for optimization to methods that actively modifies the schedule to further safely reconfigure tasks' communication intervals.

In this section, we propose a method to reduce the E2E latencies of multi- rate CECs applying the LET paradigm by actively influencing the schedule through the establishment of precedence constraints. Specifically, we show how precedence constraints, i.e., restrictions that delay the start of a job's execution, can be combined with our schedule-aware model presented in Section IV.2 to further safely reconfigure tasks' communication intervals. By interval reconfiguration, we mean finding the read and write offset combination that simultaneously best reduces the E2E latencies of the CECs present in the system. Furthermore, we show that rather than shortening the communication intervals of all tasks present in the task set, a more elaborated reconfiguration strategy focusing on the communication intervals of specific tasks can lead to more optimized E2E latencies.

We consider that when a schedulable task set is available, it is possible to change the order in which some jobs execute by establishing precedence constraints between specific jobs. By changing the order in which jobs execute, our method can influence the earliest start time (ES_τ) and latest finishing time (LF_τ) values of multiple tasks in the task set at the same time. Below, we formally define the term precedence constraint.

Definition 16. ***Precedence Constraint*** *($\prec$). Let $J(i)$ be the i^{th} job of a given task and $J(k)$ be the k^{th} job of another task in task set Γ. A precedence constraint between $J(i)$ and $J(k)$ establishes that job $J(k)$ can not start its execution until the execution of job $J(i)$ is fully completed despite its priority or release time. The execution of $J(k)$ can only start once all its precedence constraints have been resolved.*

$$J(i) \prec J(k) \tag{IV.35}$$

Figure IV.8 shows how precedence constraints influence the earliest start time and latest finishing time of the tasks previously used in the example provided in Section IV.2.1. By introducing a single precedence constraints between specific jobs of τ_1 and τ_2, new communication intervals can be obtained for all tasks present in the CEC. Specifically, adding $J_{\tau_2}(1) \prec J_{\tau_1}(1)$ and $J_{\tau_2}(1) \prec J_{\tau_1}(1)$ results in:

- $ES_{\tau_1} = 0$ and $LF_{\tau_1} = 2$ for τ_1,
- $ES_{\tau_2} = 0$ and $LF_{\tau_2} = 1$ for τ_2,
- $ES_{\tau_3} = 1$ and $LF_{\tau_3} = 3$ for τ_3.

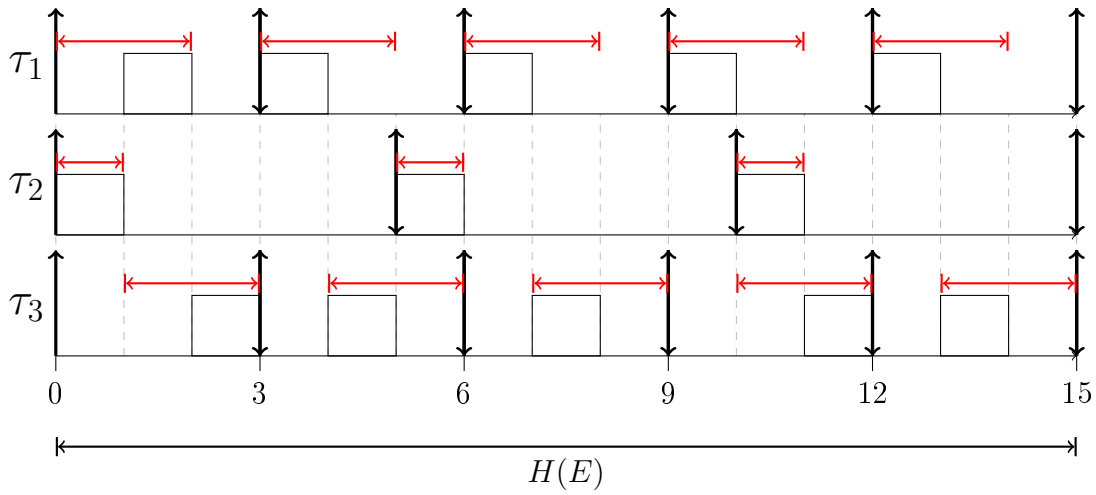


Figure IV.8: *Reconfiguring communication intervals by means of precedence constraints*

Note that by adding those two precedence constraints the communication intervals of the three tasks changes from:

- $[0, 1]$ to $[0, 2]$ for L_{τ_1},
- $[0, 3]$ to $[0, 1]$ for L_{τ_2},
- $[1, 2]$ to $[1, 3]$ for L_{τ_3}.

In the context of multi-rate CECs, precedence constraints were previously used in the literature by Becker et al. [131] to control data propagation through multi-rate CECs applying communication paradigms different than LET, e.g., the implicit communication paradigm. As a contribution of this dissertation, in this section, we use precedence constraints to modify the boundaries of communication intervals based on the LET paradigm and our schedule-aware model (See Section IV.2). Note that when communication intervals have their length equal to task' period, the establishment of precedence constraints to modify the execution order of jobs does not affect the configuration of the communication intervals because they are schedule agnostic. Therefore, adding precedence constraints to tasks applying the LET paradigm is only enabled because of how our schedule-aware model defines communication intervals.

Moreover, when adding precedence constraints between specific jobs in a multi-rate CEC applying our schedule-aware model, our method can further manipulate the configuration (length and position) of communication intervals (L_τ) of several tasks at once. As a result, new primary job chains are obtained and this ultimately affects which jobs in the CEC affect or not the E2E latencies. By evaluating different configurations for the communication intervals, our method can select the configuration that best reduces E2E latency metrics such as MRT and MDA.

In order to demonstrate that interval reconfiguration by means of precedence constraints can lead to more optimized E2E latency metrics when compared to methods that focus solely on reducing intervals, let us consider the following example depicted in figures IV.9 and IV.10. Note that we highlighted the primary job chains of CEC E in both figures.

Example: Consider a CEC E, where $E = \{\tau_1 \rightarrow \tau_2 \rightarrow \tau_3\}$, $\tau_1(1, 7, 7, 0)$, $\tau_2(1, 3, 3, 0)$, $\tau_3(1, 7, 7, 0)$. Figure IV.9 shows a schedule $\mathcal{S}$ for the tasks present in E where all the communication intervals are reduced. Note that in Figure IV.9, $MRT(E)$ (respectively $MDA(E)$) is equal to 17 time units.

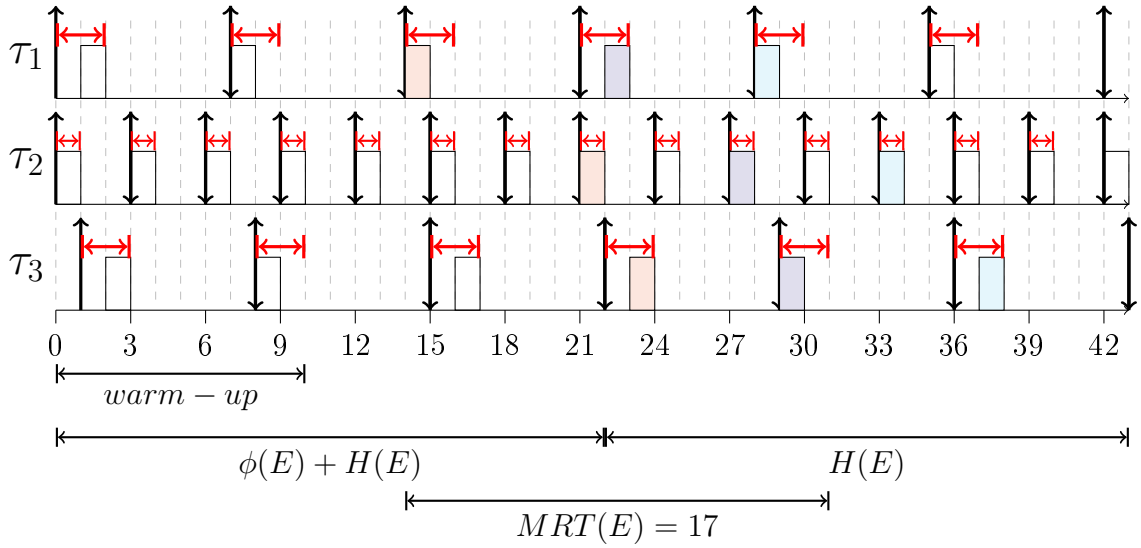


Figure IV.9: *Shrinking Models*

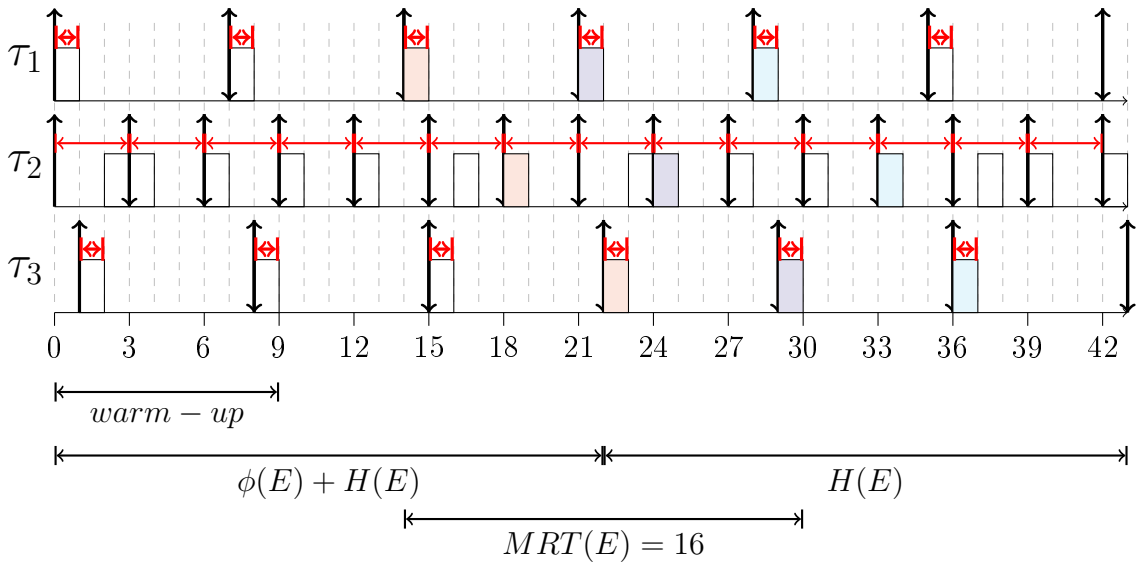


Figure IV.10: *Combination of precedence constraints and our schedule-aware model*

Figure IV.10 shows that compared to models that focus solely on reducing the communication intervals of all tasks (e.g., [150]), the combination of precedence constraints and our schedule-aware model to reconfigure the communication intervals of specific tasks results in E2E latencies that are better optimized compared to previous methods available in the literature. In Figure IV.10, we show that by establishing two precedence constraints (see below), our method obtained interval configurations that reduced $MRT(E)$ from 17 time units to 16 time units. The first established precedence is between the first jobs of τ_2 and τ_3, i.e., $J_{\tau_3}(1) \prec J_{\tau_2}(1)$. The second precedence is between the third job of τ_3 and the sixth job of τ_2, i.e., $J_{\tau_3}(3) \prec J_{\tau_2}(6)$. Note that once a precedence constraint is established, it repeats in every recurrent hyperperiod.

As long as the task set remains schedulable, the establishment of n precedence constraints can be made to the task set, where $n \in \mathbb{N}$. Therefore, it is possible to define different configurations for L_τ depending on how the execution order of jobs is arranged according to the established precedence constraints. There are multiple options to find the configuration of L_τ which best safely reconfigures the communication intervals of specific tasks to reduce the overall E2E latencies of the CECs present in the system. Since precedence constraints modifies the execution order of jobs, it directly affects the scheduler and the scheduling policy used to select jobs to execute. Therefore, additional measures have to be implemented in order to achieve the desired job execution order by precedence constraints. Instead of proposing a new method to modify the scheduler and enforce constraints at the job-level during runtime, we rely on methods already available in literature such as the works done by Becker et al. [131] and Klaus et al. [159]. Below, we present a method to find the set precedence constraints that best fits our goal of reducing MRT and MDA latency metrics.

IV.4.1 Tree Search

We model our problem of reconfiguring communication intervals by means of of precedence constraints as a level order tree search. Note that Integer Linear Program (ILP) or Satisfiability Modulo Theories (SMT) solvers are also possible options, but for convenience we used the framework proposed in [160] to generate our tree. Figure IV.11 shows the structure of our search tree and how we model its nodes and edges.

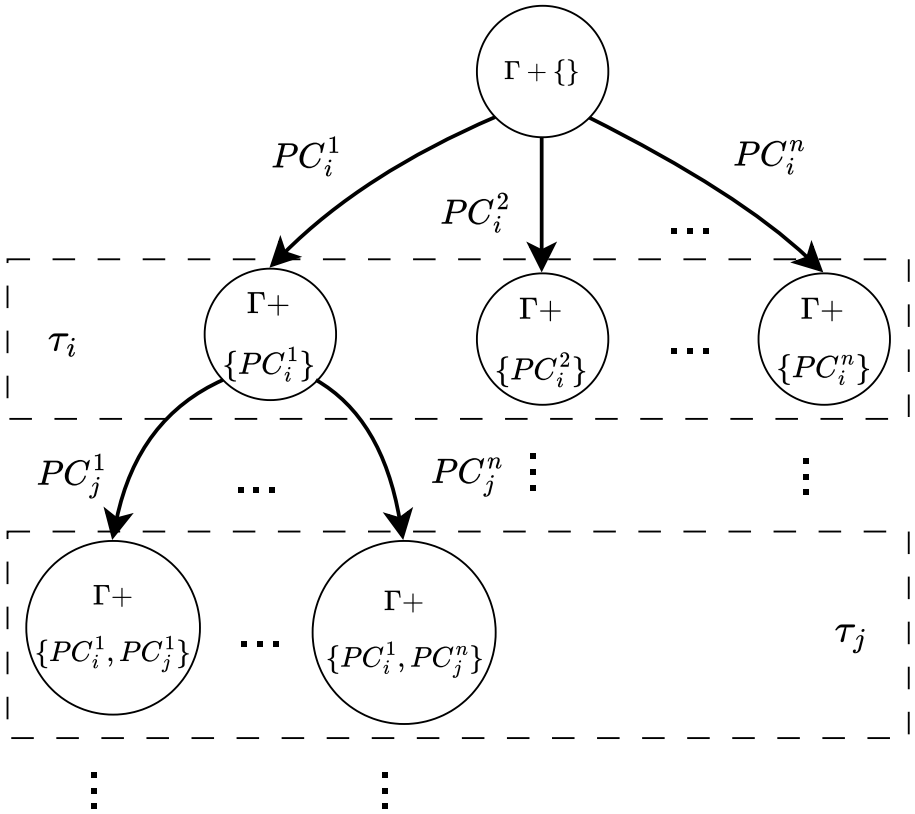


Figure IV.11: *Structure of our search tree*

The root node represents a schedulable task set Γ according to a given scheduling policy. Note that the set of precedence constraints is empty at the root node. An edge (PC_i^n) represents a single precedence constraint between two jobs from different tasks in Γ, where PC_i^n is the n^{th} possible precedence to a job of task τ_i on the current level of the search tree. Each node represents task set Γ plus the set of precedence constraints established at each edge on the path between the root node and the current node. At each level of the tree, jobs of a given task in Γ establish precedence relations with jobs from other tasks in Γ (see Section IV.8). Note that multiple jobs of a task can establish multiple precedence relations. Therefore, a task could be on more than one level of the tree. A node is considered a solution node when the set of precedence constraints established so far result in communication intervals that lead to reduced E2E latencies to all CECs present in the system.

For each node in the tree, our method computes the communication intervals of each task based on schedule-aware model described in Section IV.2 and recomputes the E2E latencies of all CECs following the equations presented in Section IV.3.1. If the set of precedence constraints present in a node resulted in larger E2E latencies for one or more chains, our method assigns penalty points based on the number of chains that were affected. Likewise, if the set of precedence constraints resulted in shortened E2E latencies, our method assigns bonus points to the node. After sorting the nodes based on their points and compliance with our heuristic function (see Section IV.8), our method selects which precedence constraint should be established to the task set under consideration. If all precedence constraints worsened latency values, our method backtracks to the node at the previous level of the tree and selects a new node to investigate. Note that the generation of a node containing no changes to the schedule is also important for the cases when none of the candidates jobs generated a feasible schedule after the establishment of precedence constraints.

In Section IV.8, we describe how our method generates nodes and how it decides which path to follow in the tree. In Chapter V, we evaluate the combination of precedence constraints and our schedule-aware model.

IV.5 Improving System Utilization

In previous sections of this chapter, our contributions focused on two aspects of multi-rate CECs applying the LET paradigm: (i) provide methods to safely reconfigure the communication intervals of tasks, (ii) improve their E2E latency metrics (MRT, MDA). In this section, we shift our focus from E2E latency and investigate the benefits of an overlooked feature enabled by the LET paradigm, its data-flow determinism.

As explained by Gemlau et al. in [157], in a producer-consumer relation between two tasks in a CEC applying the LET paradigm, a job of the consumer task will always read (*consume*) from the same job of the producer task during any execution run of the CEC. Moreover, Biondi et al. observed in [140] that depending on the period of the tasks present in the CEC, not all jobs of a task need to update their shared resource. As a result, not all jobs of the tasks present in the CEC actively contribute to data propagation, i.e., those jobs do not affect the E2E latency metrics of the CEC and their execution only wastes processing resources.

Although it may seem simple to recognize for a CEC E all the data propagation paths ending within a repetition interval, in a real-world example where several tasks with different periods are part of the same CEC, this turns out to be a non-trivial task. In the section below, we show how to identify jobs that propagate data through complex CECs and how to decrease system utilization after identifying those jobs.

IV.5.1 Reducing System Utilization by Skipping Specific Task Instances

Given a set of CECs and the tasks composing them, our method determines for each CEC E which jobs are responsible for data propagation within one repetition interval using the concept of *primary job-chains* (See Definition 15) introduced in Section IV.3.1.

The process of identifying for a CEC E the primary jobs chains that are ending within one of the repetition intervals of E, e.g., $[\Phi(E)+H(E), \Phi(E)+2H(E))$, starts with a job J of $\tau_{|E|}$ within the interval under analysis. Starting from $\tau_{|E|}$, our method recursively computes all job chains ending within the interval using algorithms 1 and 2, which were previously introduced in Section IV.3. Once our method identifies a job chain, it is added to set ζ replacing any previously added chain that does not respect Definition 15. We summarize in Algorithm 3 our method to identify all job chains within one repetition interval and filter out the ones that can be classified as primary job chains.

Algorithm 3 Identifying Primary Job Chains

taskVector ▷ Contains tasks present in E

$\zeta = \emptyset$ ▷ Set containing primary job chains

headInstances $= \emptyset$ ▷ Contains J_1's of the primary job chains

jobVector ▷ Contains jobs of $\tau_{|E|}$ within a given $H(E)$

```
1:  procedure identifyPrimaryChains(jobVector, taskVector)
2:      for J ∈ jobVector do
3:          chain = getChain(J, taskVector)   ▷ Identify job chain using algorithms 1, 2
4:          J_1 = getChainHead(chain)                          ▷ Get J_1 of chain
5:          if J_1 ∉ headInstances then
6:              insert(chain, ζ)                                ▷ Add chain to ζ
7:              insert(J_1, headInstances)                      ▷ Add J_1 to headInstances
8:          else
9:              storedChain = getStoredChain(J_1,ζ)
10:             if write(L_{J_|E|}) of chain < write(L_{J_|E|}) of storedChain then
11:                 replace(storedChain,chain,ζ)        ▷ Replace storedChain by chain in ζ
12:             end if
13:         end if
14:     end for
15: end procedure
```

In order to demonstrate how system utilization can be reduced after identifying primary job chains, let us consider the following example.

Example: Consider a CEC E with three tasks, where $E = \{\tau_1 \rightarrow \tau_2 \rightarrow \tau_3\}$, $\tau_1(1, 5, 5, 0)$, $\tau_2(1, 3, 3, 0)$, $\tau_3(1, 5, 5, 0)$. Figure IV.12 shows a schedule $\mathcal{S}$ for the tasks present in E. Let us analyze task τ_3, which has three jobs in $\mathcal{S}$. Following definitions 9 and 10, job $J(i)$ has $S_{J(i)} = 2$ and $F_{J(i)} = 3$. Job $J(i+1)$ has $S_{J(i+1)} = 2$ and $F_{J(i+1)} = 3$, while $J(i+2)$ has $S_{J(i+2)} = 1$ and $F_{J(i+2)} = 2$. Following definitions 11 and 12, τ_3 has $ES_{\tau_3} = 1$ and $LF_{\tau_3} = 3$. By doing the same to the other tasks in $\mathcal{S}$, their ES_τ and LF_τ values are: $ES_{\tau_1} = 0$ and $LF_{\tau_1} = 2$ for τ_1; $ES_{\tau_2} = 0$ and $LF_{\tau_2} = 1$ for τ_2.

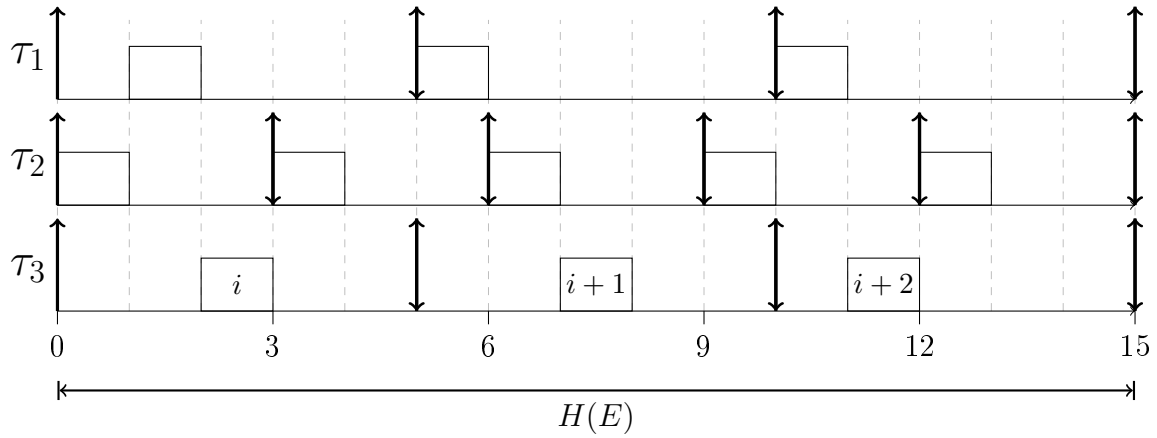


Figure IV.12: *Schedule $\mathcal{S}$ for cause-effect chain $E = \{\tau_1 \rightarrow, \tau_2 \rightarrow \tau_3\}$*

Using Algorithm 3, we show in Figure IV.13 the primary job chains of CEC E. Note that in Figure IV.13 tasks' communication intervals were already shortened and shifted according to our schedule-aware model (See Section IV.2), i.e., $L_{\tau_1} = [0, 2]$, $L_{\tau_2} = [0, 1]$, $L_{\tau_3} = [0, 2]$ and $\phi_{\tau_3} = 1$. We highlighted the primary job chains of E in Figure IV.13.

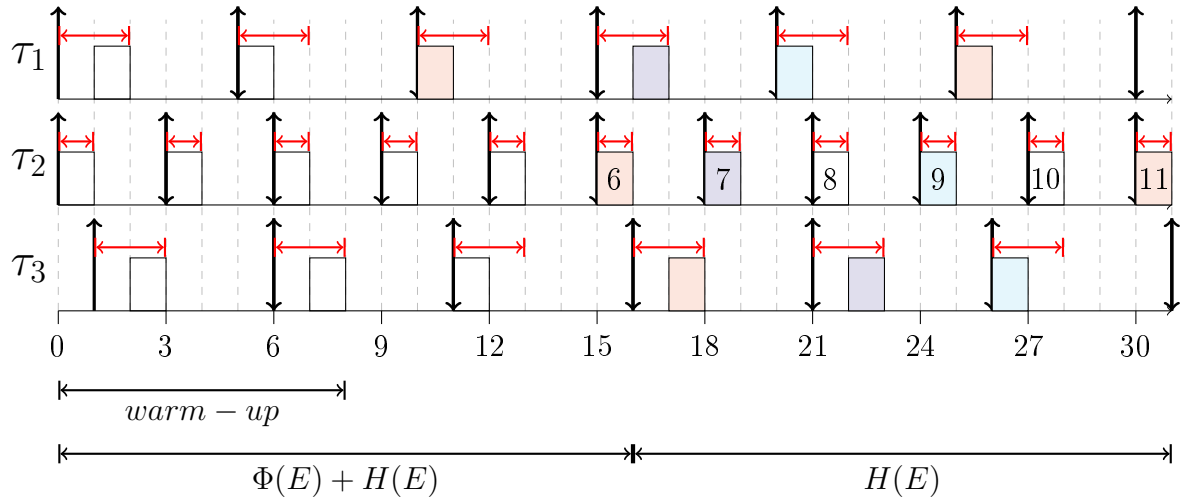


Figure IV.13: *Primary job chains of cause-effect chain E*

Note that in Figure IV.13 there are two job primary job chains colored with the same orange color. As explained in Section IV.3, job chains of tasks applying the LET communication paradigm repeat every $H(E)$. As a result, the job chain starting at the third job of τ_1 in Figure IV.13 is the same as the one starting at the last job of τ_1 in Figure IV.13 but one $H(E)$ apart. We show in Figure IV.13 that the jobs of task τ_2 that are not part of a primary job chain $(J(8), J(10))$ have their outputs overwritten by other jobs during data propagation. As a result, the execution of those jobs, e.g., $J(8), J(10)$, is only wasting processing resources.

In Figure IV.13, we left uncolored the jobs of task τ_2 that have their outputs overwritten and therefore are not part of primary job chains. By skipping in each $H(E)$ the execution of jobs that are not part of primary job chains, system utilization can be reduced without affecting the worst-case E2E latencies of CECs and the applications they belong to. In the literature, Maggio et al. showed in [161] that control applications with timing requirements can, in fact, skip the execution of some jobs without affecting the overall quality of the controlled application. Therefore, for control applications with CECs applying the LET communication paradigm, it is safe to skip in each $H(E)$ the execution of jobs that are not part of primary job chains given the data-flow determinism enabled by the LET paradigm. For example, by skipping the execution of the uncolored jobs of τ_2 in Figure IV.13, the system utilization reduces from 0.73 to 0.6.

Note that in order to not disturb the sampling (resp. actuation) behavior of the application, our method does not skip jobs from tasks located at the boundaries of the CEC, e.g., tasks τ_1 and τ_3 in Figure IV.13. Also, if a task belongs to more than one CEC, only jobs that are not needed in all those CECs are skipped by our method.

In order to further optimize system utilization as we did in Section IV.4 for E2E latencies, our method needs to manipulate the configuration of communication intervals (length and position) since they determine which jobs are part of or not part of any primary job chain.

As shown in sections IV.2 and IV.4, the configuration of communication intervals (length and position) determine which jobs are part of or not part of any primary job chain. Therefore, in order to further optimize system utilization, our method needs a technique to freely manipulate the configuration of communication intervals. In this dissertation, we use our approach presented in Section IV.4 to freely manipulate the configuration of communication intervals by means of precedence constraints between specific task instances. We model the problem of establishing precedence constraints as a search tree and search for the set of constraints that maximize the number of jobs that can be skipped. In Section IV.8, we describe how our method generates nodes for the search tree and how it decides which path to follow.

In the following section, IV.5.2, we demonstrate a possible solution on how to resolve complex precedence constraints before runtime and how to keep task releases periodic even when skipping the execution of jobs. In Chapter V, we evaluate the performance of our method in reducing the utilization of systems with multi-rate CECs applying the LET communication paradigm.

IV.5.2 Translating Sporadic Jobs Into Periodic Tasks While Resolving Precedence Constraints

As shown in Section IV.5.1, due to the data-flow determinism present in the LET communication paradigm, system utilization can be decreased by skipping the execution of jobs that are not part of primary job chains. However, skipping a job's execution in a periodic task results in a sporadic release of jobs for that same task. For instance, let us consider in Figure IV.14 the task set from the example provided in Section IV.5.1 and the following set of precedence constraints, $\{J(1)_{\tau_3} \prec J(1)_{\tau_2}, J(2)_{\tau_3} \prec J(3)_{\tau_2}\}$, where job's index represents the i^{th} instance of job J_τ within $H(E)$. In Figure IV.14, we colored the jobs that are part of primary job chains.

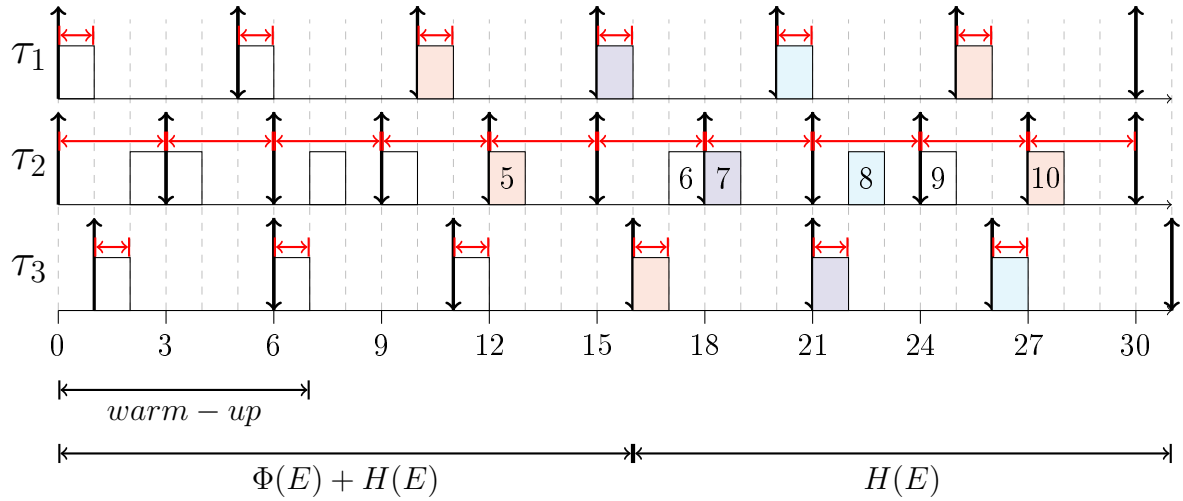


Figure IV.14: *Primary job chains of cause-effect chain E witch precedence constraints*

Although τ_2 executes five jobs within $H(E)$, the execution of $J(6)$ and $J(9)$ can be skipped to decrease system utilization as they are not part of primary job chains. In order to skip the execution of $J(6)$ and $J(9)$, τ_2 should not release those two jobs when they were supposed to. By doing so, τ_2 no longer behaves as a periodic task, but as a sporadic task.

In order to obtain a sporadic release pattern but using a period task, we build our method on top of the work done by Dobrin et al. in [162]. Their method translates offline schedules into fixed-priority schedule schemes while coping with complex timings requirements. The goal of our method is to translate jobs that were not selected to be skipped by our algorithms into periodic tasks. Note that in order to ensure that the execution order of jobs remains intact after our process of translating jobs into tasks, we follow the assumptions made by Dobrin et al. in [162] and only consider tasks with *unique* fixed-priority. That is, each task $\tau \in \Gamma$ has a given priority level P_τ, where $P_\tau < P_{\tau'}$ means that τ has lower priority than τ'.

For example, Figure IV.15 shows in detail the sporadic release pattern of task τ_2 from Figure IV.14. Figure IV.16 shows that the same execution pattern showed in Figure IV.15 can be obtained by translating τ_2's jobs into three periodic tasks. Note that $\tau_{2,3}$'s execution is delayed due to τ_3 as shown in Figure IV.14.

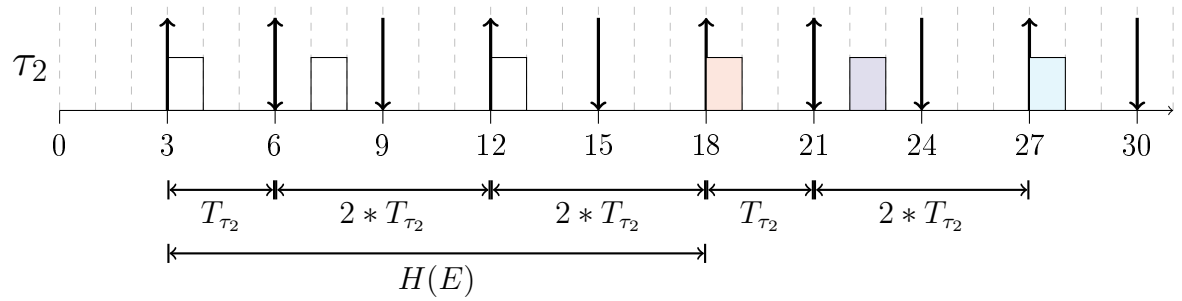


Figure IV.15: *Sporadic release of jobs by τ_2*

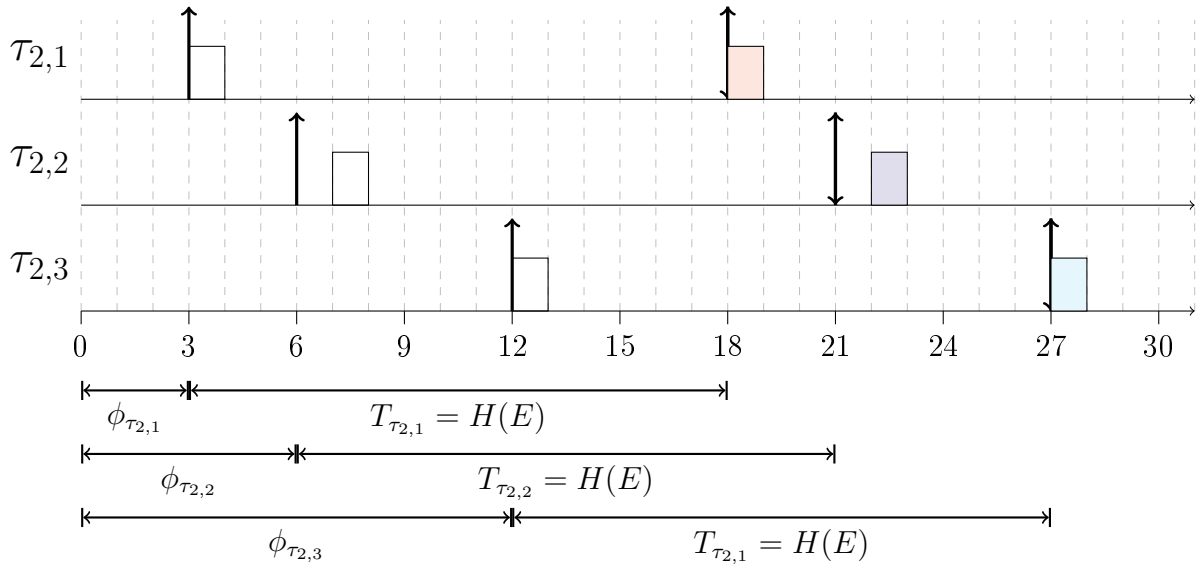


Figure IV.16: *Sporadic jobs of τ_2 translated into periodic tasks*

Given a task $\tau \in \Gamma$, where τ is part of a CEC E and has jobs that are *not* part of primary job chains, our method translates the sporadic execution of τ into periodic execution by translating τ into n new tasks, where n is the number of jobs that τ has as part of primary job chains. For each job J of task τ that is part of a primary job chain in any given CEC, our method adds J to job set ζ and checks if there are *offset assignment conflicts* and/or *priority assignment conflicts*. By resolving offset assignment conflicts, our method solves the sporadic releases of jobs when skipping jobs, while resolving priority assignment conflicts resolves precedence constraints before runtime. In the following paragraphs, we explain the two types of conflicts and how to resolve them.

Definition 17. ***Offset Assignment Conflict.*** *Let an offset assignment conflict be when two consecutive jobs of a task τ in ζ, require different offset values. That is, given two consecutive jobs of τ in ζ, $J(i)$ and $J(k)$, where $i < k$, an offset assignment conflict exists if $k - i > 1$.*

For any two consecutive jobs of τ that have an offset assignment conflict, our method splits τ into n instances, where n is the number of jobs τ has in ζ. Therefore, each J of τ in ζ becomes a task τ_J that has $C_{\tau_J} = C_\tau$, $P_{\tau_J} = P_\tau$, $L_{\tau_J} = L_\tau$. The phase of τ_J is: $\phi_{\tau_J} = \phi'_\tau + (i-1)T_\tau - \lfloor \frac{S_{J(i)}}{H(E)} \rfloor H(E)$, where i is the i^{th} instance of J and ϕ'_τ is τ's initial phase. We define the period of τ_J as $T_{\tau_J} = H(E)$. As a result, each τ_J has only a single job within $H(E)$. During runtime, phase ϕ_{τ_J} ensures that τ_J will start its execution at the same point in time as job J would have started in the original schedule. Therefore, τ_J acts like an execution replacement for J. For example, the first job of $\tau_{2,3}$ in Figure IV.16 replaces the third job of τ_2 in Figure IV.15.

Definition 18. ***Priority Assignment Conflict.*** *Let a priority assignment conflict be when jobs of the same task τ in ζ require different priority levels to ensure a given execution order. That is, a priority assignment conflict exists for every J in ζ that has a precedence constraint that modifies its priority level P_τ.*

For each conflict $J' \prec J$, where J is a job of τ and J' is a job of τ', our method splits τ into n new tasks, where n is the number of jobs τ has in ζ. Job J becomes a task τ_J with $\Psi < P_{\tau_J} < P_{\tau'}$, where $\Psi = \max(P_\tau | \tau \in \psi)$ and ψ is the set of tasks with priorities lower than $P_{\tau'}$. The other jobs of τ become tasks with the same priority as τ. Note that our method sets the remaining parameters of the new tasks in the same manner as when resolving offset assignment conflicts.

Figure IV.17 shows an example of the offset (resp. priority) assignment conflicts present in Figure IV.14 as a result of skipping jobs (resp. establishing precedence constraints).

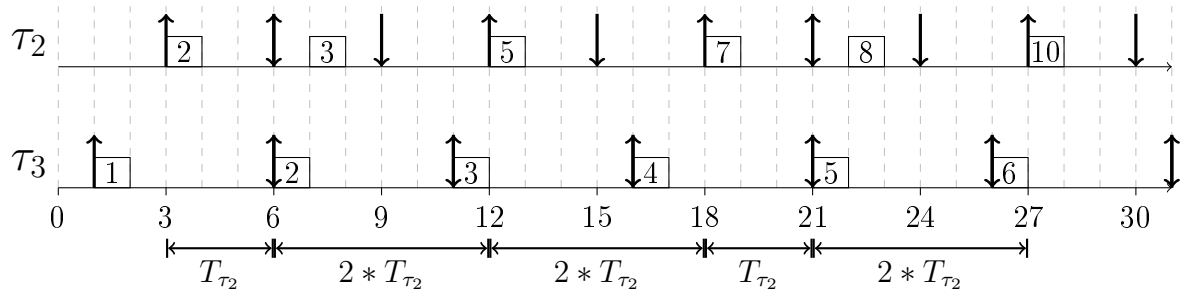


Figure IV.17: *Offset and priority assigment conflits*

In Figure IV.17, an offset assignment conflict between $J(3)_{\tau_2}$ and $J(5)_{\tau_2}$ exists because $J(4)_{\tau_2}$ is not part of a primary job chain (See Figure IV.14). A priority assignment conflict between $J(2)_{\tau_2}$ and $J(3)_{\tau_2}$ exists due to the precedence constraint added between $J(2)_{\tau_3}$ and $J(3)_{\tau_2}$, i.e., $J(2)_{\tau_3} \prec J(3)_{\tau_2}$.

By resolving the offset (resp. priority) assignment conflicts of τ_2 before runtime, our method obtains in Figure IV.18 the same execution pattern as in Figure IV.14, but with reduced system utilization and without the need of precedence constraints between τ_2 and τ_3. Note that available solutions to enforce the execution behavior of precedence constraints during runtime can also be used such as the ones proposed by Becker et al. in [131] and Klaus et al. in [159].

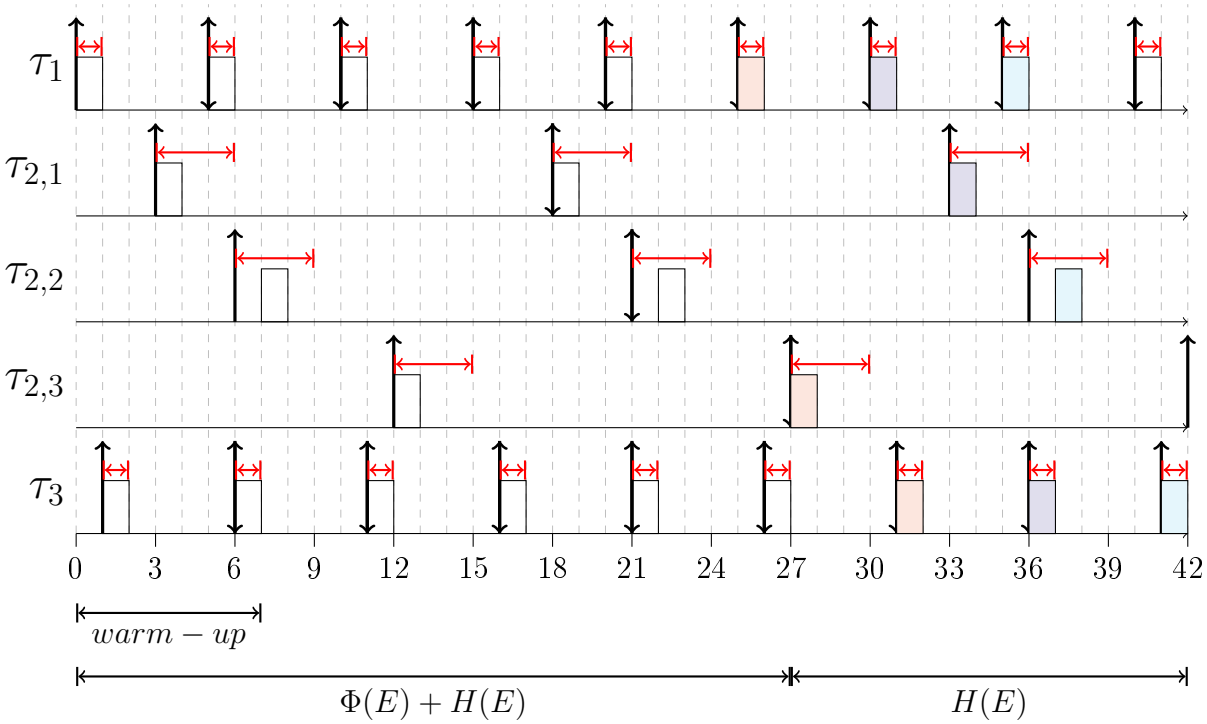


Figure IV.18: *Job chains after translating jobs into tasks*

Figure IV.18 shows what the schedule previously showed in Figure IV.14 looks like after resolving the offset (resp. priority) assignment conflicts. Note that in Figure IV.18, our method translated τ_2 into three periodic tasks ($\tau_{2,1}, \tau_{2,2}, \tau_{2,3}$). Although the parameters of τ_2 changed, the E2E latencies between τ_1 and τ_3 are the same as in Figure IV.14.

IV.6 Improving End-to-End Latency Feasibility in Multi-Execution Mode Systems

So far in this dissertation we have only considered systems with a single execution mode, but that is not always the case in the automotive industry. Safety-critical applications running on multi-core systems often need to operate under multiple execution modes during runtime. A transition from one mode to another may cause the task set to change, e.g., new tasks join the system, which might increase the E2E latencies of certain CECs and cause them to violate their latency requirements. In fact, there is

currently a gap in the literature, since, for the best of our knowledge, there is no other work that investigates how much impact a change in the execution mode of a system can cause to the E2E latency metrics of multi-rate CECs.

Modern safety-critical applications demand not only a high degree of reliability and strict timing guarantees for the control applications they run, but also increasingly demand greater flexibility and adaptability to accommodate dynamic system changes. In the automotive domain, such dynamic changes may arise from varying road conditions, the activation of new driver assistance features, or fault recovery mechanisms. For instance, a vehicle driving on a highway may employ distinct control strategies and timing configurations compared to urban driving, where tighter response times for obstacle avoidance and emergency braking may be required. Similarly, when faults are detected, the vehicle transitions to a degraded mode of operation, where certain functions may be disabled or reconfigured, necessitating corresponding adjustments to the underlying real-time scheduling and communication strategies.

In order to support such dynamic behavior, mode change services are usually employed in such safety-critical applications [97]. Those services assume that a distinct mode of operation for each anticipated scenario is defined during design time and uses an agreement protocol to enable timely switching between the defined modes at runtime. As a contribution of this dissertation, we do not aim to design a new mode change protocol; rather, we concentrate on the core challenge of ensuring that the timing requirements of control applications with multi-rate CECs are satisfied across all execution modes.

Usually, the LET communication paradigm is both schedule- and mode-agnostic, producing a single static configuration across all modes based solely on task periods. As the task set might change from one execution mode to another and E2E latency constraints might become stricter, both the original LET paradigm and approaches that exploit scheduling knowledge to shorten LET-based communication intervals (e.g., Bradatsch et al. in [150]) may fail to ensure stricter E2E latencies for certain chains. This limitation arises because, in different execution modes, additional tasks and/or chains may be introduced into the system.

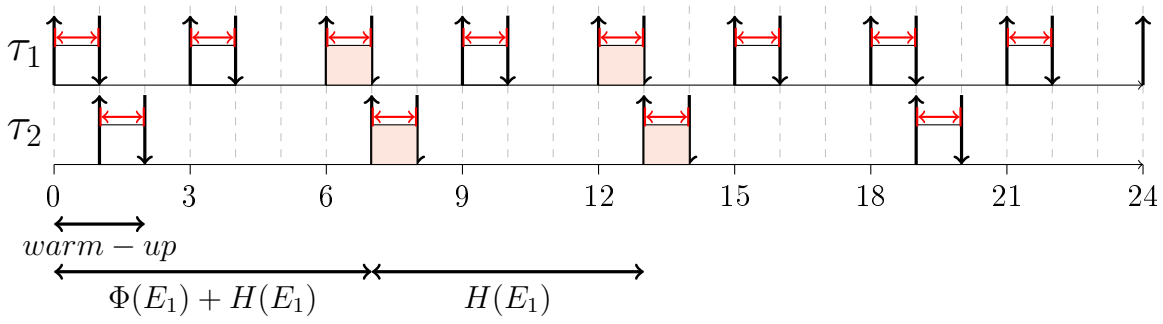
In this section, we propose a method for enhancing the feasibility of multi-rate CECs that apply the LET communication paradigm to meet E2E latency requirements in multi-core systems with multiple execution modes. Specifically, our method selectively migrates task instances to enable the reconfiguration of LET-based communication intervals, thereby increasing the feasibility of meeting E2E latency constraints as execution modes change. Our method validates the E2E latency constraints of CECs by identifying communication intervals that require further reconfiguration relative to the preceding mode. By migrating selected task instances, our method enables new data propagation paths that remain within the specified timing bounds of the current execution mode. Although our method reconfigures communication intervals, it maintains the periodicity of tasks and preserves the well-defined inter-task communication points that are characteristic of the LET paradigm. Our method only reconfigures communication intervals when selected task instances can be feasibly migrated and guaranteed to finish execution within a predetermined deadline.

IV.6.1 Migrating Task Instances to Meet Time Constraints

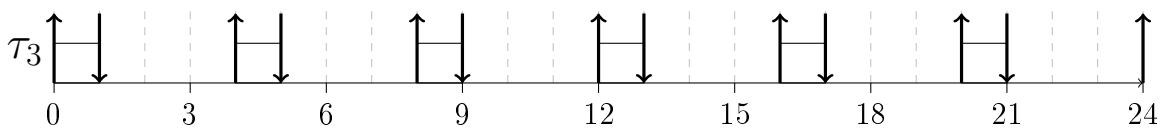
For this contribution, we consider the *extended version* of our system model introduced in Section IV.1. Specifically, we add assumptions regarding the execution modes and migration techniques available in the system. We assume that multiple execution modes are available in our multi-core system, where M represents the finite set of modes the system has, e.g., $M = \{m_1, m_2, \cdots, m_n\}$ $(n \in \mathbb{N}^+)$. Each mode m_i consists of a task set γ_i, where $\gamma_i \subseteq \Gamma$. We consider that CECs have E2E latency constraints, which may vary among different modes. We assume that changes to an execution mode take place at the end of the system's HP and that changes made to task's parameters due to interval reconfiguration are not carried to the new mode. Regarding task and job migration, we assume that our multi-core system applies job-level migration, meaning that jobs can only migrate to another core at the boundaries of their execution, i.e., no migration is allowed when resuming the execution of a job. We also consider task replication as our migration technique, that is, during runtime only the state of the task must migrate from one core to another. Therefore, migration has a negligible impact on CPU scheduling. We assume that tasks are scheduled according to the EDF policy.

In order to demonstrate how a change in the execution mode can affect the feasibility of CECs meeting their E2E timing requirements, we present the following example.

Example: Let us assume a system comprised of two cores (c_1, c_2) and two execution modes (m_1, m_2). In mode m_1, γ_1 consists of three task $\{\tau_1, \tau_2, \tau_3\}$, and one CEC E_1, where $E_1 = \{\tau_1 \rightarrow \tau_2\}$, $\tau_1(1, 3, 3, 0)$, $\tau_2(1, 6, 6, 0)$, $\tau_3(1, 4, 4, 0)$. We assume that tasks are scheduled according to EDF and allocated to cores as shown in Figure IV.19. Note that in Figure IV.19, the tasks present in γ_1 had their communication intervals and parameters reconfigured according to our schedule-aware model (See Section IV.2). Following equations IV.30 and IV.34, $MRT(E_1) = MDA(E_1) = 8$.



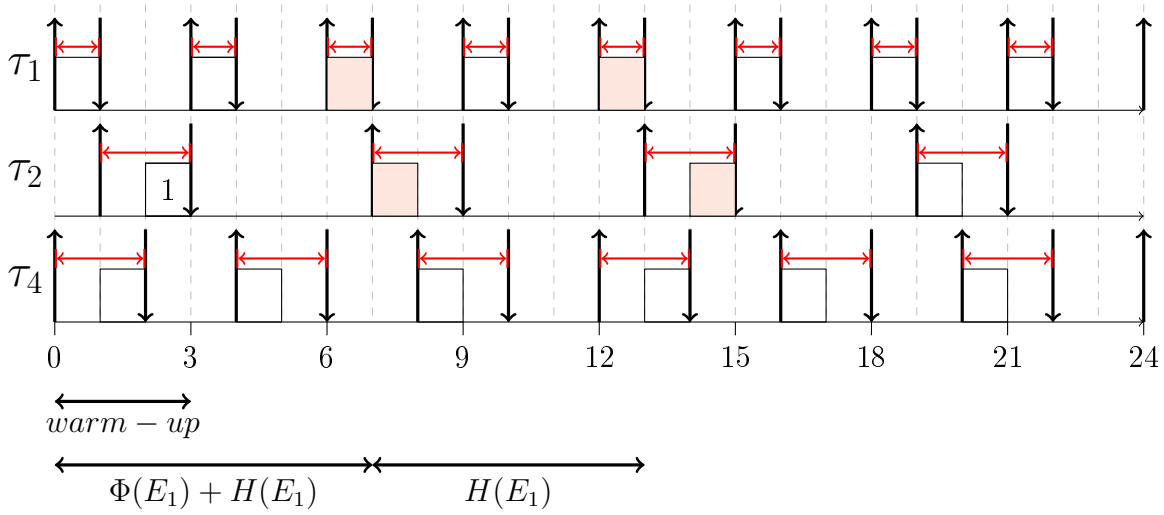
Task allocation and schedule in core c_1



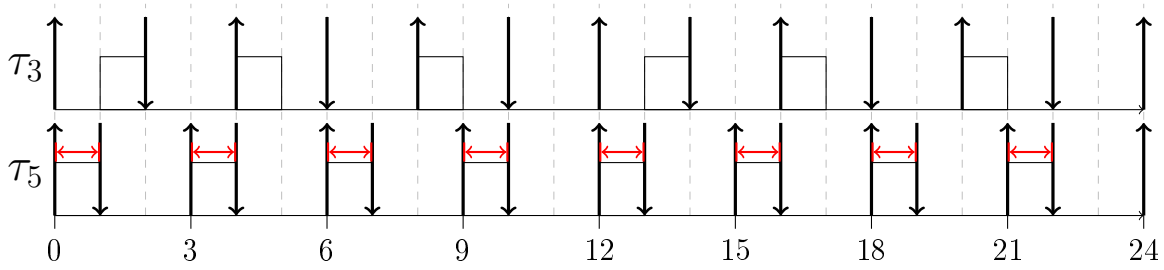
Task allocation and schedule in core c_2

Figure IV.19: *System setup in mode m_1*

Now, let us assume that the system changes its execution mode from m_1 to m_2. In mode m_2, CEC $E_2 = \{\tau_4 \rightarrow \tau_5\}$ joins the system, where $\tau_4(1, 4, 4, 0)$, $\tau_5(1, 3, 3, 0)$. Once in m_2, the system reschedule all tasks in γ_2 considering their original parameters. Figure IV.20 shows the allocation of tasks to cores and the communication intervals obtained based on the schedule of γ_2. Figure IV.20 shows that the addition of τ_4 to c_1 causes additional interference to the execution of job $J(1)$ from task τ_2, which in turn causes the communication interval of τ_2 to have a different configuration in m_2 when compared to the one in m_1. As a result, the MRT and MDA values of E_1 increase from 8 to 9 time units. In Figure IV.20, we colored the primary job chains of E_1.



Task allocation and schedule in core c_1

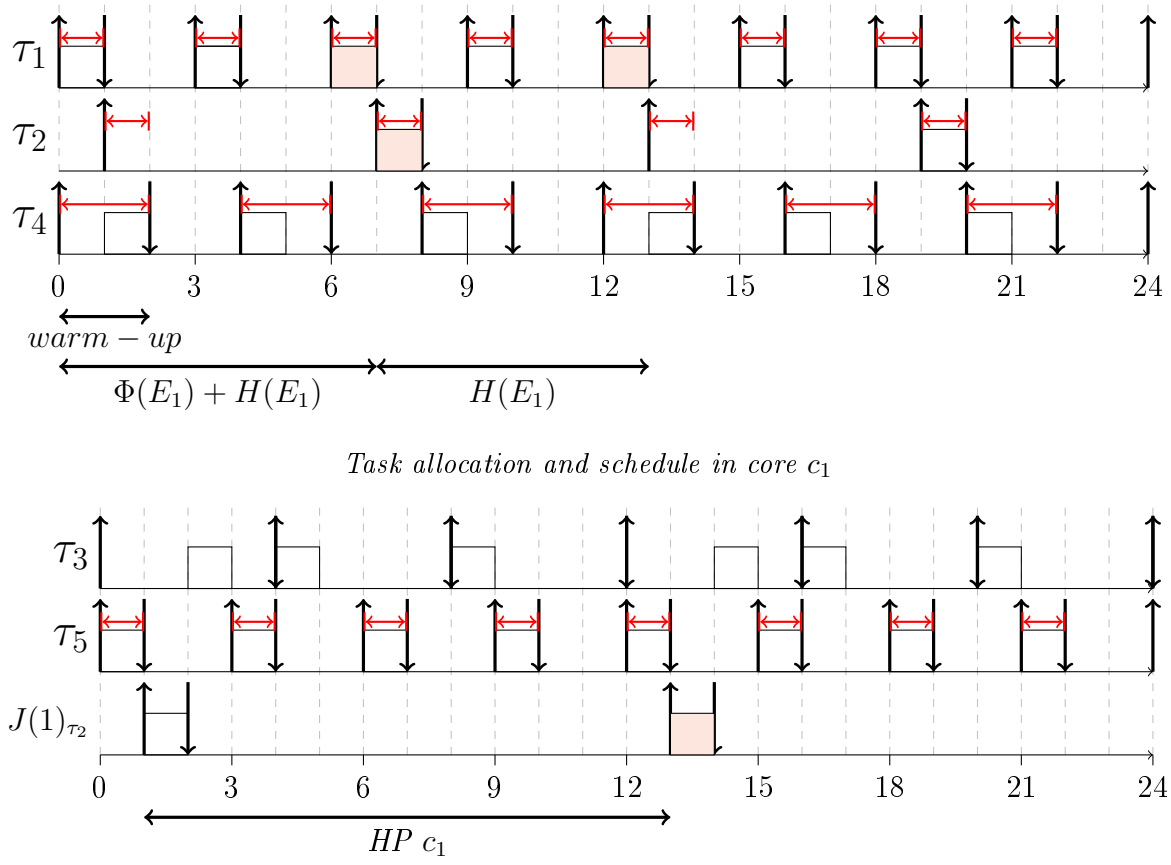


Task allocation and schedule in core c_2

Figure IV.20: *System setup in mode m_2*

Figure IV.20 shows that without further reconfiguration, E_1 can no longer respect its latency constraints when assuming that E_1 should maintain in m_2 the latency values it had in m_1. Note that the application alone of any LET schedule-based approach (e.g., [150]) is not capable of retaining the required E2E latencies for E_1 in m_2. Therefore, we need a method to further reconfigure the intervals of tasks belonging to E_1, i.e., we need a method to alleviate the additional interference caused to tasks present in E_1.

Figure IV.21 shows that by feasibly migrating job $J(1)_{\tau_2}$ to core c_2 at time point 1 and with an absolute deadline of 2, we can reconfigure the communication intervals of τ_2 such that the values of $MRT(E_1)$ and $MDA(E_1)$ go back to 8 time units.



Task allocation and schedule in core c_1

Task allocation and schedule in core c_2

Figure IV.21: *System setup after migration of $J(1)_{\tau_2}$*

Note that migrating $J(1)_{\tau_2}$ to core c_2 does not cause interference to τ_5, which, in turn, does not affect the E2E latency metrics of E_2. For this example, the migration of $J(1)_{\tau_2}$ only causes interference to the execution of τ_3, which is not part of a CEC. As such, τ_3 does not need to keep a reduced deadline and execute within a given communication interval like tasks that are part of a CEC. By restoring the deadline of τ_3 to its original value during the feasibility test to migrate $J(1)_{\tau 2}$ to core c_2 (See Section IV.6.3), the leeway in c_2 increases and allows the feasible migration of $J(1)_{\tau_2}$ and all its future iterations. Note that if τ_3 was part of a CEC, as long as τ_3 has a communication interval with enough leeway to accommodate the additional interference of $J(1)_{\tau_2}$, the migration would still be allowed.

In the section below, we describe how our method determines which jobs should migrate and which target execution windows they must have in order to enhance the feasibility of CECs meeting their timing requirements after a change in execution mode.

IV.6.2 Migration Strategy

As shown in the example above, the migration of specific task instances can help with increasing the feasibility of CECs meeting their E2E timing requirements after an execution mode change. In this section, we describe the procedure used by our method to select which task instances should undergo migration.

In case of an execution mode change, e.g., from m_i to m_j, our method: (i) reschedules tasks considering their original parameters, (ii) recomputes communication intervals, (iii) checks whether or not tasks that are part of CECs maintained in mode m_j the communication interval configuration they had in mode m_i. Since our method configures communication intervals for each $\tau \in \Gamma$ according to our schedule-aware model (See Section IV.2), τ can only retain its communication interval configuration if τ's earliest relative start time (ES_τ) and latest relative finishing time (LF_τ) were not affected in mode m_j. Naturally, the values of ES_τ and LF_τ can only remain the same if task τ does not suffer any additional interference from tasks that joined the system after changing from m_i to m_j. Note that in case of additional interference, the non-reconfiguration of τ's communication intervals would result in a violation of LET's properties, since not all instances of τ would execute within their defined communication interval.

For each CEC E in mode m_j, our method computes the MRT and MDA values of E. If the E2E latency metrics of E (MRT and MDA) are not respected in m_j, our method starts to do interval reconfiguration by means of migration. The strategy used by our method to select which jobs of E should migrate depends on two factors. First, our method checks if E was present or not in the previous mode (m_i). Second, our method checks if the E2E latency requirements of E were relaxed, tightened or remained the same. If E was present in the previous mode (m_i) and its E2E latency requirements were relaxed or remained the same, our method focuses on restoring the interval configuration obtained in m_i for each τ that is part of E (*Migration Scenario I*). If the E2E latency requirements of E were tightened, or E was *not* present in the previous mode (m_i), our method focuses on obtaining interval configurations that are as tight as C_τ for each τ that is part of E (*Migration Scenario II*).

Migration Scenario I

For each τ that is part of a CEC E that falls in the conditions of *Migration Scenario I*, our method analyzes all jobs of τ within the core's HP. By identifying which jobs of τ suffered additional interference in the current mode (m_j) compared to the previous mode (m_i), our method identifies which jobs of τ caused the values of ES_τ and LF_τ to change. We consider that the job set Ω contains all the jobs of τ that suffer additional interference in m_j and are selected to migrate. Note that if $ES_\tau^{m_j} \neq ES_\tau^{m_i}$, our method adds all jobs of τ to Ω and sets $\phi_\tau^{m_j} = \phi_\tau^{m_i}$. By migrating all jobs in Ω and ensuring their execution within $[ES_\tau^{m_i}, LF_\tau^{m_i}]$ relative to their release time (See Section IV.6.3), our method can restore the interval configuration τ had in m_i. Since the E2E latency of E depends on the communication interval configuration of each task τ in E, our method recomputes the latency metrics of E for each reconfigured interval. Note that in the worst case, all tasks in E need to have their intervals configured with the same configuration as in m_i.

Migration Scenario II

For each τ that is part of a CEC that falls in the conditions of *Migration Scenario II*, our method attempts to obtain communication intervals for τ that are as tight as C_τ. In order to do that, our method adds all jobs of τ that are within the core's HP and are suffering the highest amount of interference in the current mode to the job set Ω. Our method adds the remaining jobs of τ to the job set Ω^*. Let ω (resp. ω^*) represents the highest amount of interference suffered by the jobs in Ω (resp. Ω^*). For each job J_τ in Ω, our method checks by means of a feasibility test (See Section IV.6.3) if J_τ can completely execute within the interval $[ES_\tau, LF_\tau - \xi]$ relative to the release of J_τ, where $\xi = \omega - \omega^*$. If the feasibility test $\forall J_\tau \in \Omega$ is successful, our method starts an iterative process. In the first step, our method removes the set of jobs suffering the highest amount of interference in Ω^* and adds them to Ω. In the second step, our method recomputes the value of ξ. In the third step, our method rechecks the feasibility of migrating all J_τ in Ω. This iterative process continues until our method can no longer migrate all J_τ in Ω, or until it obtains communication intervals for τ that are tight as C_τ. Once the iterative process ends, our method migrates the latest job set Ω in which all jobs could be feasibly migrated. Last, our method recomputes the E2E latencies of E after adjusting the communication intervals of τ. If the latency requirements of E are still not respected, our method repeats the procedure for the next task in E.

Migration Process

The migration process to reconfigure communication intervals starts with the last task in the chain. The reasoning for this decision is that the more we reduce the output jitter in the end of E, the more we reduce its overall E2E latency. In order to ensure that each J_τ of τ in Ω can completely execute within the interval defined by the selected migration scenario, our method performs a feasibility test for each J_τ in Ω (See Section IV.6.3). Note that upon a successful migration of J_τ, all its future iterations during the next core's HP will also be migrated in the same manner.

We designate as *candidate cores* the cores that can possibly allocate J_τ. Our method sorts the candidate cores according to their utilization, i.e., the lower the utilization of a core, the greater the chance of being able to allocate J_τ. When migrating J_τ to a core c_i, our method must fulfill three requirements:

1. The schedule on c_i remains feasible
2. J_τ can completely execute within the interval defined by the migration scenario
3. The timing requirements of the CECs with tasks running on c_i are not affected by the migration of J_τ

In order to ensure that those three requirements are fulfilled, our method performs the feasibility test proposed by Alkoudsi et al. in [154] (See Section IV.6.3) before the migration of J_τ to a candidate core. Note that our method performs the feasibility test for each J_τ of τ in Ω. For the cases in which our feasibility test fails, our method tries to migrate J_τ to the next candidate core in the queue. If none of the cores can accept J_τ, our method moves on to the next CEC that has to be reconfigured.

IV.6.3 Migration Feasibility

In this section, we describe the feasibility test proposed by Alkoudsi et al. [154] and show how we use it to ensure that the three requirements listed in Section IV.6.2 are fulfilled when migrating J_τ to a core c_i.

The migration feasibility analysis for a specific job and its subsequent periodic occurrences can be formally reduced to the problem of schedulability analysis for a system extended with a new, asynchronous periodic task.

Reduction to Periodic Task Admission

The core of the migration method involves moving a specific job $J(k)_{\tau_i}$ of a task τ_i from its original core to a candidate core. This migration is not a one-time event; it includes all future instances of this job that occur at the same relative phase within each subsequent HP of the core c where τ_i is allocated. We formally define this set of migrated jobs and prove its equivalence to a single periodic task.

Definition 19 (Migrated Job Set)**.** *Let $J(k)_{\tau_i}$ be the first job of task τ_i selected for migration, where $\tau_i \in E$ and $H(c)$ is the hyperperiod of core c in which τ_i is allocated. The resulting set of all migrated jobs, J_{mig}, is defined as:*

$$J_{mig} = \{J(k + n \cdot (H(c)/T_{\tau_i}))_{\tau_i} \mid n \in \mathbb{N}_0 = \{0, 1, 2, \dots\}\}$$

Theorem IV.3. *Checking the feasibility of migrating the job set J_{mig} to a candidate core c with an existing task set γ_c is equivalent to checking the schedulability of a new, combined task set $\gamma_{c+m} = \gamma_c \cup \{\tau_{mig}\}$, where τ_{mig} is a single asynchronous periodic task.*

Proof. By definition, a periodic task τ_{mig} is a tuple (ϕ_{mig}, C_{mig}, T_{mig}, D_{mig}). Therefore, we must show that the set of jobs J_{mig} is identical to the set of jobs generated by such a task.

Let $J(n)_{mig}$ be the n^{th} job in the set J_{mig} (where $n = 0$ corresponds to $J(k)_{\tau_i}$). Since the release time of a job $J(j)_\tau$ is defined as $r(J(j)_\tau) = \phi_\tau + (j-1)T_\tau$, the release time of $J(n)_{mig}$ is:

$$\begin{aligned} r(J(n)_{mig}) &= r\left(J(k + n \cdot (H(c)/T_{\tau_i}))_{\tau_i}\right) \\ &= \phi_{\tau_i} + \left(k + n \cdot \frac{H(c)}{T_{\tau_i}} - 1\right) T_{\tau_i} \\ &= (\phi_{\tau_i} + (k-1)T_{\tau_i}) + n \cdot H(c) \end{aligned}$$

Now, consider a new periodic task τ_{mig} with the following parameters:

- **Period $T_{\tau_{mig}}$:** $T_{\tau_{mig}} = H(c)$. This is the hyperperiod of core c in which τ_i is allocated.
- **Phase $\phi_{\tau_{mig}}$:** $\phi_{\tau_{mig}} = \phi_{\tau_i} + (k-1)T_{\tau_i}$. This is the release time of the first migrated job $J^k_{\tau_i}$.
- **WCET: $C_{\tau_{mig}}$:** $C_{\tau_{mig}} = C_{\tau_i}$. This is the worst-case execution time of any job in J_{mig} since all these jobs are instances of τ_i.
- **Deadline: $D_{\tau_{mig}}$:** $D_{\tau_{mig}} = D_{\tau_i}$. This is the relative deadline of any job in J_{mig} since all these jobs are instances of τ_i.

The release time of the j^{th} job of τ_{mig} (for $j \in \mathbb{N}^+ = \{1, 2, \dots\}$) is:

$$\begin{aligned} r(J(j)_{\tau_{mig}}) &= \phi_{mig} + (j-1)T_{mig} \\ &= (\phi_{\tau_i} + (k-1)T_{\tau_i}) + (j-1)H(c) \end{aligned}$$

By substituting $n = j - 1$, we see that the set of release times $\{r(J(n)_{mig}) \mid n \in \mathbb{N}_0\}$ is identical to the set $\{r(J(j)_{\tau_{mig}}) \mid j \in \mathbb{N}^+\}$.

Since the set of jobs generated by τ_{mig} and the set of jobs J_{mig} have identical release times, worst-case execution times, and absolute deadlines, checking the schedulability of J_{mig} within γ_c is equivalent to checking the schedulability of the combined periodic task set $\gamma_{c+m} = \gamma_c \cup \{\tau_{mig}\}$. □

Note that in order to fulfill requirement 2 from Section IV.6.2, our method sets the relative deadline of the migrated task to the length of the interval defined by the selected migration scenario. In order to fulfill requirement 3 from Section IV.6.2, during the feasibility test, our method considers that all tasks running on the candidate core and that are part of a CEC have a relative deadline equal to the length of their communication interval. For tasks that not part of a CEC and are on the candidate core, our method considers during the feasibility test that those tasks have their original relative deadline value. Note that in case of a feasible migration, the relative deadline of tasks that are not part of a CEC are restored back to their original value and might only be modified again after the next execution mode change that triggers a reconfiguration of communication intervals.

Feasibility Analysis

We sequentially examine each CEC and iteratively assess the feasibility of the jobs migrated from that CEC. Once a job is determined to be admissible on the candidate core, it is permanently integrated into that core's task set. As a result, the job is retained in all subsequent feasibility analyses for any future migrations targeting the same candidate core.

We now analyze the feasibility of the combined task set $\gamma_{c+m} = \gamma_c \cup \gamma_m$ using the test proposed by Alkoudsi et al. [154], where γ_c is the set of tasks already assigned to the candidate core c and γ_m is the set of all newly-defined periodic tasks representing the migrations.

Note that the addition of γ_m to γ_c may expand the HP of the task set on the candidate core c, as $\mathrm{HP}(\gamma_{c+m}) = \mathrm{LCM}(\mathrm{HP}(\gamma_m), \mathrm{HP}(\gamma_c))$ may be larger than $\mathrm{HP}(\gamma_c)$.

Due to the optimality of the EDF scheduling policy in preemptive single-core systems, the feasibility analysis reduces to EDF-schedulability analysis. The standard tool for this purpose is the PDA [156, 155]. The concept for this analysis verifies that in all time intervals, the requested processor demand by tasks does not exceed the amount of available time within the interval.

It follows from the work done by Baruah et al. in [156] that EDF-schedulability of the periodic task set γ_{c+m} can be verified using PDA by checking that

$$\sum_{\tau_i \in \gamma_{c+m}} \mathrm{dbf}(\tau_i, t_1, t_2) \;\leq\; t_2 - t_1 \tag{IV.36}$$

holds for all $t_1 < t_2$, where dbf is the demand bound function for an asynchronous periodic task τ_i and is given by

$$\mathrm{dbf}(\tau_i, t_1, t_2) = C_{\tau_i} \cdot \max\left(0, \left\lfloor \frac{t_2 - \phi_{\tau_i} - D_{\tau_i}}{T_{\tau_i}} \right\rfloor \;\; - \max\left(0, \left\lceil \frac{t_1 - \phi_{\tau_i}}{T_{\tau_i}} \right\rceil \right) + 1 \right)$$

As shown by Baruah et al. in [156], it suffices to restrict t_1 and t_2 to task release times and absolute deadlines, respectively. Furthermore, Alkoudsi et al. demonstrated in [154] that the ranges of t_1 and t_2 can be constrained as follows:

$$\phi_{\max}^{c+m} \leq t_1 < \phi_{\max}^{c+m} + \mathrm{HP}(\gamma_{c+m}), \tag{IV.37}$$

$$\phi_{\max}^{c+m} \;=\; \max_{\tau_i \in \gamma_{c+m}} (\phi_{\tau_i}), \tag{IV.38}$$

$$t_2 < t_1 + \mathrm{B}(\gamma_{c+m}), \tag{IV.39}$$

$$\mathrm{B}(\gamma_{c+m}) \;=\; \frac{\sum_{\tau_i \in \gamma_{c+m}} (T_{\tau_i} - D_{\tau_i}) U_{\tau_i}}{1 - \mathrm{U}(\gamma_{c+m})}, \tag{IV.40}$$

$$\mathrm{U}(\gamma_{c+m}) \;=\; \sum_{\tau_i \in \gamma_{c+m}} U_{\tau_i}. \tag{IV.41}$$

Here, $\mathrm{HP}(\gamma_{c+m})$ denotes the hyperperiod of the combined task set, and $\mathrm{B}(\gamma_{c+m})$ is a utilization-based bound ensuring that if the task set is unschedulable, a deadline miss will occur before this bound [156, 154]. In the special case where $\mathrm{U}(\gamma_{c+m}) = 1$, Equation IV.40 becomes undefined. In this case, Equation IV.39 is replaced by:

$$t_2 < t_1 + \mathrm{HP}(\gamma_{c+m}).$$

In Chapter V, we evaluate the performance of our method in enhancing the feasibility of multi-rate CECs that apply the LET communication paradigm to meet E2E latency requirements after an execution mode change.

IV.7 Interval Configuration Framework

For this dissertation, we developed a framework in C++ to do the safe reconfiguration of communication intervals of tasks applying the LET communication paradigm and present in multi-rate CECs. Throughout this dissertation, we demonstrated multiple benefits that the safe reconfiguration of communication intervals can bring to a safety-critical system without losing the key properties of the LET paradigm such as time and data-flow determinism. In this section, we present the structure of our framework, its main functions, and input (resp. output) files.

We build our framework on top of the offline scheduler framework presented by Syed in [91, 92]. The main goal of his work [91, 92] was to provide a modular framework that facilitates the creation of different types of schedulers by providing code-reuse, modularity, flexibility, and template-based thread-safe library of search algorithms (e.g., PIDA*). Figure IV.22 illustrate the basic structure of our framework.

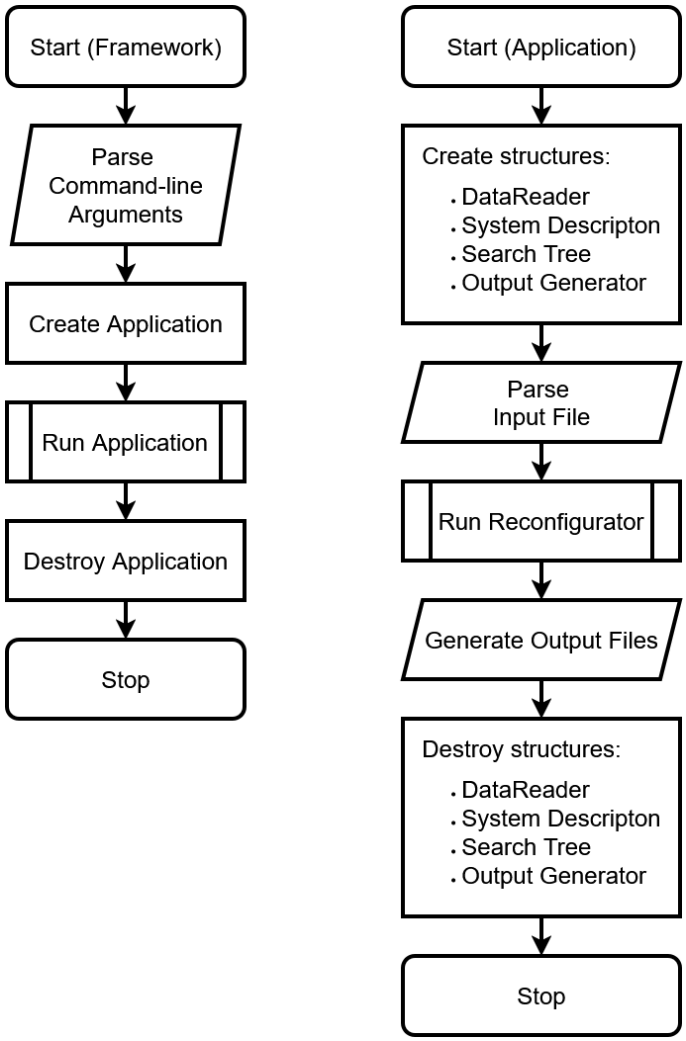


Figure IV.22: *Framework Core Flow Chart*

As shown in Figure IV.22, our framework is mainly divided in two parts. In the first part, our framework reads the XML based input files and creates binary structures to store the read data. A standard input file contains information regarding the system platform such as: number and type of available cores, number of software components (SWCs) and the CECs present in them, as well as the properties (phase, WCET,period) of the tasks that are part of those SWCs. In Appendix B.1, we show a sample input file based on the example shown in Section IV.2.1. In the second part, our framework runs the reconfiguration strategies presented from Section IV.2 until Section IV.6 according to the requests provided by the user via command-line arguments. Our framework provides four optimization options to the user:

1. E2E latency optimization *without* precedence constraints
   (Contribution from Section IV.2)

2. E2E latency optimization *with* precedence constraints
   (Contribution from Section IV.4)

3. System utilization optimization
   (Contribution from Section IV.5)

4. Increase E2E latency feasibility in multi-execution mode system
   (Contribution from Section IV.6)

As shown in Figure IV.22, after parsing the command-line arguments that will define which optimization methods will be applied by our framework, it creates an application object to store all the necessary data for the reconfiguration of the communication intervals. Figure IV.23 shows as a UML diagram the relationship between different classes present inside the framework.

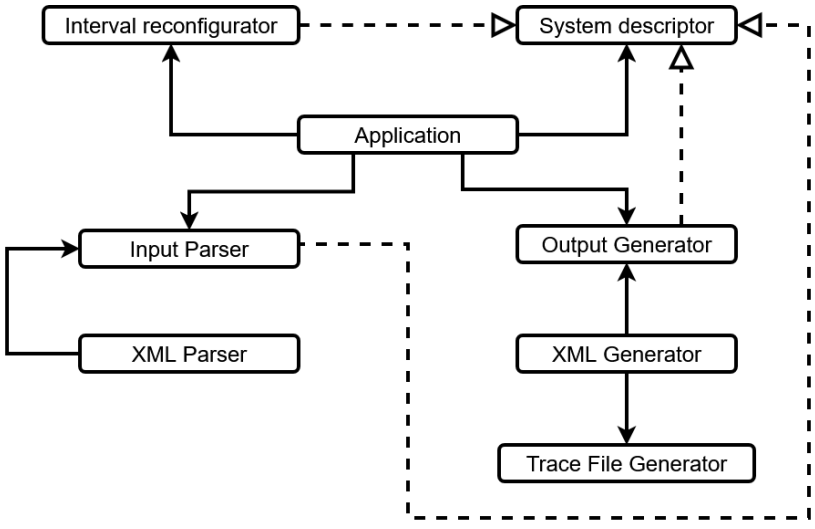


Figure IV.23: *UML diagram of the relationships between classes in the framework*

The created application object validates the read input data and controls the flow of information through the framework. The function of the *Input Parser* (realized by the Data Reader structure in Figure IV.22) is to generate all the necessary input data structures. The Data Reader class is then responsible for parsing all the information contained in the XML input files. Likewise, the *Output Generator* is responsible for writing the new task's parameters (phase, communication interval and precedence constraints) in an XML based output file. In Appendix B.2, we show a sample output file based on the example shown in Section IV.4. All the data structures responsible for the reconfiguration of the communication intervals and establishment of precedence constraints are embedded into the *System Description* class. By integrating all the system information into a single data package, our framework eases the process of handling data by multiple different classes. The actual process of reconfiguring communication intervals according to a given optimization option is done by the *Reconfigurator* class. It manipulates all the data structures present in the *System Description* data structure and uses it to construct a safe communication interval configuration for each task present in the system according to the given optimization parameter. Once optimized, all of the data is stored back in the *System Description* data structure, from which it is written to an output XML file by the *Output Generator*. Figure IV.24 shows, as a flow chart, the structure of the *Interval Reconfigurator*.

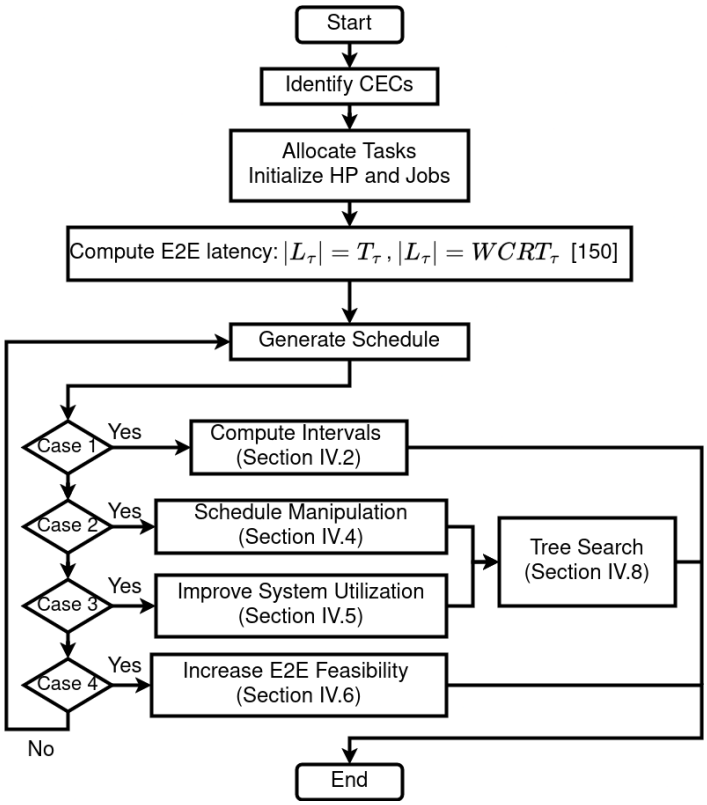


Figure IV.24: *Representation of the interval reconfigurator as a flow chart*

IV.8 Search and Heuristic Functions

In this section, we present the heuristic functions used to generate nodes and decide which path to follow when traversing our search tree proposed in Section IV.4 and used in sections IV.4 and IV.5. Note that in both cases, the tree structure (See Figure IV.11) remains the same. As explained in Section IV.7, our interval reconfiguration framework is flexible and can be adjusted to achieve different goals such as reducing E2E metrics, system utilization, and(or) improving E2E latency feasibility in systems with multiple execution modes.

Below, we present the underlying algorithm responsible for generating the nodes for both methods. Naturally, besides the goal, the main differences when comparing the search trees used in sections IV.4 and IV.5 are: (i) the order in which jobs are analyzed; (ii) which nodes are selected to be created based on precedence constraints; (iii) how nodes are evaluated. Algorithm 4 shows the pseudo-code of the main function responsible for creating and sorting the nodes in our framework.

Algorithm 4 Creation of nodes

```
   jobVector                          ▷ Ordered vector containing jobs of the tasks in cecVector
 1: procedure createAndSortChildNodes(node, childNodes)
 2:     if TIMEOUT then
 3:         exit
 4:     else
 5:         nodes = {}                                          ▷ Initialize empty vector
 6:         job_i = jobVector[node.nextJob]                     ▷ Get next job to analyze
 7:         γ = getCandidates(job_i)                ▷ Set of possible jobs to precede job_i
 8:         for candidate ∈ γ do
 9:             setConstraint(node, job_i, candidate)           ▷ Set: candidate ≺ job_i
10:             if isSchedulable(Γ) then                        ▷ Checks Γ's schedulability
11:                 childNode = createNode(node)                ▷ Create a new node
12:                 computeCommIntervals(childNode)             ▷ Recompute L_τ, ∀τ ∈ Γ
13:                 heuristic(childNode)                        ▷ Evaluate childNode
14:                 insert(nodes, childNode)                    ▷ Add childNode to nodes
15:             end if
16:         end for
17:         insert(nodes, node)   ▷ Child node without additional precedence constraint
18:         sortNodes(nodes)              ▷ Sort nodes according to their evaluation metric
19:     end if
20:     return      ▷ The head of vector nodes will be the next node to be investigated
21: end procedure
```

As briefly explained in Section IV.4, in our search tree every node represents a feasible schedule of Γ after the establishment of a precedence constraint between two jobs. Starting from the root node, Algorithm 4 selects the job in the head of *jobVector* (job_i) to receive a precedence constraint from selected jobs of other tasks $\in \Gamma$ (line 6). In order to be part of *jobVector*, a job has to fulfill some requirements that might change for each optimization method (e.g., reduced E2E latency, improved system utilization). If no additional requirement is present, all jobs within system's HP are eligible to receive a precedence constraint and be part of *jobVector*. The order in which they are added to *jobVector* might also change depending on the optimization method. In Section IV.8.1, we detail the extra requirements for our method that safely reconfigure communication intervals of specific tasks to improve E2E latencies.

We call as *candidate*, a job that can serve as precedence constraint for job job_i, i.e., $candidate \prec job_i$. A job can be considered a candidate job if it executes in between the release and deadline of job_i. Algorithm 4 iterates over all other tasks in the task set checking which of them have jobs that can be classified as candidate for job job_i (line 7). The number of jobs investigated as possible candidates per task depends on the system's HP. Each job that can serve as candidate to job_i is added by our algorithm to the candidate set γ.

Algorithm 4 iterates over the list of possible candidates and checks if a feasible schedule can be obtained after establishing a precedence between job job_i and one of the candidates (lines 8 to 10). For every established precedence constraint that keeps the task set schedulable, Algorithm 4 generates a new node (*child node*) to be later investigated (line 11). Note, that the establishment of precedence constraints prevents the use traditional schedulability tests, which requires Algorithm 4 to simulate system's execution through an entire HP. For every generated child node, Algorithm 4 recomputes the communication intervals $\forall \tau \in \Gamma$ (line 12).

In line 13, Algorithm 4 assigns an evaluation metric to the child node depending on the heuristic function considered. In Section IV.8.1, we explain in detail the heuristic function used for our method that safely reconfigure communication intervals of specific tasks to further optimize E2E latency metrics (See Section IV.4). In Section IV.8.2, we explain the heuristic function used for out method that reduces system utilization (See Section IV.5).

Depending on how the search tree is being traversed, i.e., which precedence constraints have been assigned on previous nodes, it is beneficial for the heuristic function to consider the possibility where no precedence constraint is assigned on the current node (line 17). The generation of a child node containing no changes to the schedule is also important for the cases when none of the candidates jobs generated a feasible schedule after the establishment of a precedence constraint. In line 18, Algorithm 4 sorts the created child nodes according to the evaluation metric assigned by the heuristic function on line 13. The child node with the best evaluation metric is selected as the next node to be investigated (line 20). Note that since our framework is based on a version of the PIDA* algorithm [160], depending on how many processors are available in the machine running our framework, multiple child nodes can be evaluated in parallel.

IV.8.1 Heuristic Function to Optimize End-to-End Latency Metrics

For our method presented in Section IV.4, the heuristic function's goal is to find the minimum number of precedence constraints that have to be established in order to optimize the most number of CECs simultaneously. Therefore, when establishing and evaluating precedence constraints for a task τ, our method considers three aspects: (i) τ's position in the chain; (ii) the period relation between τ, its predecessor and successor in the chain; (iii) which jobs of τ define the boundaries of its communication intervals.

We say a job $J(i)$ defines the interval boundaries of τ if $s^S_{J(i)} - (\phi_\tau + (i-1)T_\tau) = ES_\tau$ or $f^S_{J(i)} - (\phi_\tau + (i-1)T_\tau) = LF_\tau$. Our framework adds each job J of τ that fulfills (iii) into *jobVector* from Algorithm 4. The order in which jobs from tasks are added by our framework into *jobVector* depends on the length of the CEC those tasks belong to. For instance, jobs from tasks that are part of CECs with short lengths are added first to *jobVector* by our framework.

As explained in the beginning of Section IV.8, our algorithm creates a set γ containing jobs from other tasks which can be scheduled within the release and deadline of J. Each job in γ is a *candidate* to establish a precedence with J. That is, for each J fulfilling (iii) and for each *candidate* in γ, our method generates a child node if a feasible schedule is obtained after the establishment of the precedence constraint, i.e., adding *candidate* $\prec J$ to Γ results in a schedulable task set.

In order to perform node evaluation for each established precedence constraint, our heuristic considers that a CEC consists of three segments: *head*, *body* and *tail*. Depending on which segment of the chain a task τ belongs to, our heuristic uses different strategies to evaluate the effectiveness of a given precedence constraint. The number of tasks in the chain defines which and how many tasks are part of each segment.

We say a task is part of the *head* segment if it is located among first third of tasks present in the chain. If a task is on the second third, it belongs to the *body* segment, otherwise, it is part of the *tail* segment.[2] Below, we detail how our heuristic expects the communication intervals of τ to be configured depending on which segment the chain task τ is part of. For a CEC consisting of two tasks, we consider it as a standard producer/consumer relation between two tasks.

Tail Segment: For a task τ in the *tail* segment, our heuristic favors precedence constraints that: (i) result in intervals that are short, (ii) minimize the maximum time interval between the read-event of τ and the write-event of its predecessor in the chain. The rationale is, by having short intervals in the tail segment, our heuristic reduces the output delay of τ, which contributes to reduce the overall E2E latencies of the chain. Likewise, minimizing the maximum time interval between the read-event of τ and the write-event of its predecessor reduces the data propagation delay.

[2] If the number of tasks in the chain is a multiple of three, each segment has the same length. If not, the head and tail segments have their length equal to the quotient, while the body segment has its length equal to the quotient plus the remainder.

Head Segment: For a task τ in the head segment, if τ is the first task in the chain, our method favors precedence constraints that result in intervals that start close to the beginning of τ's period interval. The rationale is, by having intervals positioned close to the beginning of τ's period interval, we reduce the amount of phasing that might have to be added for all the subsequent tasks in the chain. If τ is not the first task in the chain, our method favors the reconfiguration of τ as done in the tail segment.

Body Segment: For a task τ positioned in the body segment, our method reconfigures the intervals of τ depending on the period relation between τ, its predecessor and its successor in the chain. More specifically, if task τ is part of an under/oversampling relation with its predecessor and part of an under/oversampling relation with its successor, then our method favors the non reconfiguration of τ's intervals. The rationale is that, since the successor or predecessor of τ cannot keep up with its sampling rate, reducing the communication intervals of τ significantly will not necessarily reduce data propagation delay. In this case, it is more beneficial to keep τ's communication interval longer and allow precedence constraints to shape the intervals of its successor or predecessor, so that they align better with the intervals of τ. In this case, our method generates a node that contains the same set of precedence constraints as the parent node and assign more bonus points to it. Note that our method also generates nodes based on possible precedence constraints to τ. However, those nodes do not receive bonus points which make them less likely to be chosen as the next node to be investigated. For all the other scenarios, our method favors the reconfiguration of τ as done in the tail segment.

IV.8.2 Heuristic Function to Improve System Utilization

For our method presented in Section IV.5, the heuristic function's goal is to find the minimum number of precedence constraints that have to be established in order to maximize the number of jobs that can have their execution skipped. Since tasks can be part of more than one CEC, our heuristic function favors precedence constraints that maximize the number of jobs belonging to multiple *primary job chains* on different CECs. By favoring precedence constraints in this manner, our method maximizes the number of jobs that can have their execution skipped, since a lower number of jobs are required to keep the E2E latencies of the CECs unaffected.

As explained in the beginning of Section IV.8, our algorithm creates a set γ containing jobs from other tasks which can be scheduled within the release and deadline of the job under analysis. Each job in γ serves as a *candidate* to establish a precedence contraint relation with the job under analysis. That is, for each *candidate* in γ, our method generates a child node if a feasible schedule is obtained after establishing the precedence constraint.

Moreover, our heuristics takes into consideration the fact that the establishment of precedence constraints to decrease system utilization might unintentionally lead to an increase or decrease of the E2E latencies of some CECs, but not to a significant decrease in system utilization. Therefore, when evaluating a node created after the establishment of a precedence constraint, our heuristic assigns higher evaluation values to nodes

containing precedence constraints that best reduce system utilization independent of the E2E latencies. However, bonus points are assigned to a node by the heuristic function if the E2E latencies of some CECs reduced. Likewise, the heuristic function removes points if the E2E latencies of some CECs increased. From the evaluation point of view of our heuristic function, the penalty for worsening E2E latency metrics of a given CEC is higher than the bonus for reducing the latencies.

Chapter V

Evaluation

This chapter presents a detailed evaluation of the optimization methods presented throughout Chapter IV. In order to perform a thorough evaluation of our proposed methods, we conducted two sets of experiments for each method. In Section V.0.1, we present the results for the first set of experiments, in which we consider an industrial case study from the automotive industry, more precisely, we consider the benchmark proposed as part of the Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS) challenge (See Section III.1.3). In Section V.0.2, we present the results for the second set of experiments, in which we consider synthetic task sets. Throughout this chapter, we show the benefits of our methods, particularly the advantages of performing the safe reconfiguration of communication intervals.

All the experiments were conducted on top of a octa-core processor i7-9700 with 16GB of RAM. As explained in Section III.2, before this dissertation, only the work done by Bradatsch et al. in [150] could safely reconfigure the communication intervals of tasks applying the Logical Execution Time (LET) paradigm. Therefore, we mainly compare the performance of our methods with the approach presented by Bradatsch et al. [150] and the LET paradigm. We reference as *WCRT-LET* the method proposed by Bradatsch et al. [150] since it reconfigures communication intervals based on the Worst-Case Response Time (WCRT) of tasks.

We conducted the two set of experiments in our framework presented in Section IV.7. We configured our framework to consider a system comprising four cores, to allocate tasks according to the worst-fit policy, and to always schedule tasks according to the Deadline-Monotonic (DM) scheduling policy, except for the experiments considering execution mode changes, in which the framework was configured to consider the Earliest Deadline First (EDF) scheduling policy. For the experiments that require the search capabilities of our framework, we configured a timeout of 5 minutes.

The remainder of this chapter is organized as follows. In Section V.0.1, we present in detail how we generated task sets based on the automotive benchmark presented in the WATERS challenge. In Section V.0.2, we present in detail how we generated synthetic task sets as well as which parameters of the automotive benchmark were modified for this second set of experiments. In order to ease the reader's readability, we split the evaluation into subsections for each method proposed in Chapter IV.

V.0.1 Automotive Benchmark

For each evaluated contribution, we generated and tested 500 schedulable task sets based on the parameters of the real-world automotive benchmarks proposed as part of the WATERS challenge (See Section III.1.3). We derive task's parameters and configurations for the cause-effect chains (CECs) in accordance to the benchmark. Since all the evaluation is done in our framework via simulation, we do not set size to labels as in Table III.1. We assign periods to tasks following the definitions of Table III.4. The range of possible periods is: [1, 2, 5, 10, 20, 50, 100, 200, 1000]ms. Note that the sum of the probabilities for possible periods according to Table III.4 is 85%. The remaining 15% is for angle-asynchronous tasks. Since, we do not consider angle-asynchronous tasks, we divided all probability values by 0.85.

For inter-task communications, we follow the distribution of Table III.2 to define which tasks could have producer(resp. consumer) relation based on their activation period. For each task, we generated a worst-case execution time (WCET) following the distribution of tables III.5, and III.6. As stated in the automotive benchmark [11], in typical engine control applications there are between 30 and 60 CECs. In our experiments, on average, there are 38 CECs per task set. The number of periods per CEC is randomly chosen between the interval [1,3] following the distribution of Table III.7. For each period that composes the CEC, we follow distribution from Table III.8. In each of our task sets, each CEC is composed of 2 to 14 tasks.

We assume that all tasks are part of at least one CEC and they communicate with each other according to the LET paradigm. Therefore, we consider that only homogeneous CECs are present in the system. The total utilization of cores is $\approx 70\%$ (per core), on average, except for our method that considers multiple execution modes, since we tested its performance under different system loads (See Section IV.6).

Safe Reconfiguration of Communication Intervals

In this section, we present the evaluation of our method presented in Section IV.2. For this set of experiments, our framework reconfigured tasks' communication intervals solely based on information extracted from the simulated schedule, e.g., earliest start time and latest finish time. We evaluated the performance of our method according to different latency metrics and parameters. First, we evaluated the improvement of our method with respect to the Maximum Reaction Time (MRT) and Maximum Data Age (MDA) latency metrics (See figures V.1 and V.2). Then, we investigated in detail the end-to-end (E2E) latency improvement according to the properties of the CECs, e.g., number of activation periods or tasks (See figures V.3 and V.4 respectively). We also performed a thorough evaluation of a randomly selected task set to verify the exact amount of improvement our method achieves compared to state of the art (See Figure V.5).

Figures V.1 and V.2 show the results for the MRT and MDA latency metrics respectively. The box plots show the 25th percentile, average, 75th percentile and the maximum & minimum-case values. Note that in Figure V.1, we normalized the results with respect to the MRT obtained by the LET paradigm, while in Figure V.2 we normalized the results with respect to the MDA metric.

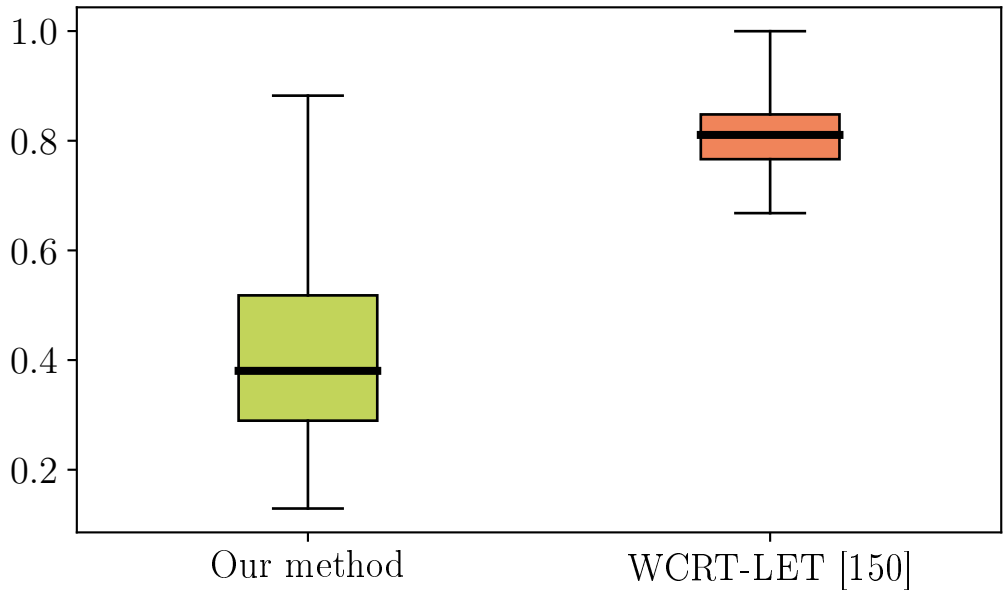


Figure V.1: *Normalized maximum reaction time w.r.t LET from the automotive benchmark*

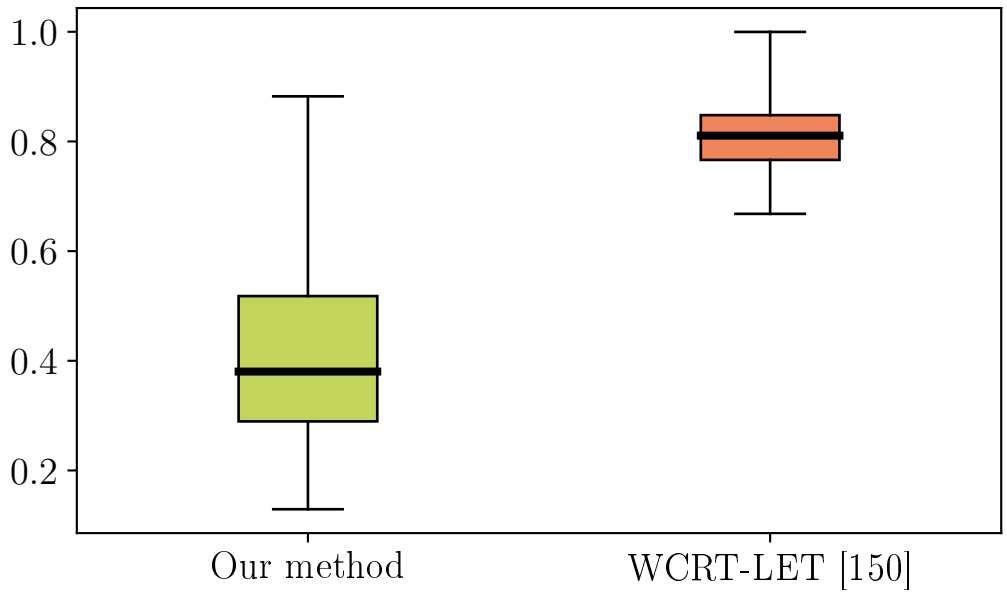


Figure V.2: *Normalized maximum data age w.r.t LET from the automotive benchmark*

Figure V.1 shows that our model managed to obtain MRT values that are on average, $\approx 63\%$ lower than the values obtained by LET. Similarly, the WCRT-LET model managed to improve MRT values by $\approx 20\%$. Figure V.2 shows that the latency improvement for the MDA latency metric, was the same as for the MRT latency shown in Figure V.1.

Figures V.3 and V.4 show the improvements achieved by our method when analyzing the individual E2E latencies of each available job chain. Note that in both figures, we normalized the results with respect to the LET paradigm.

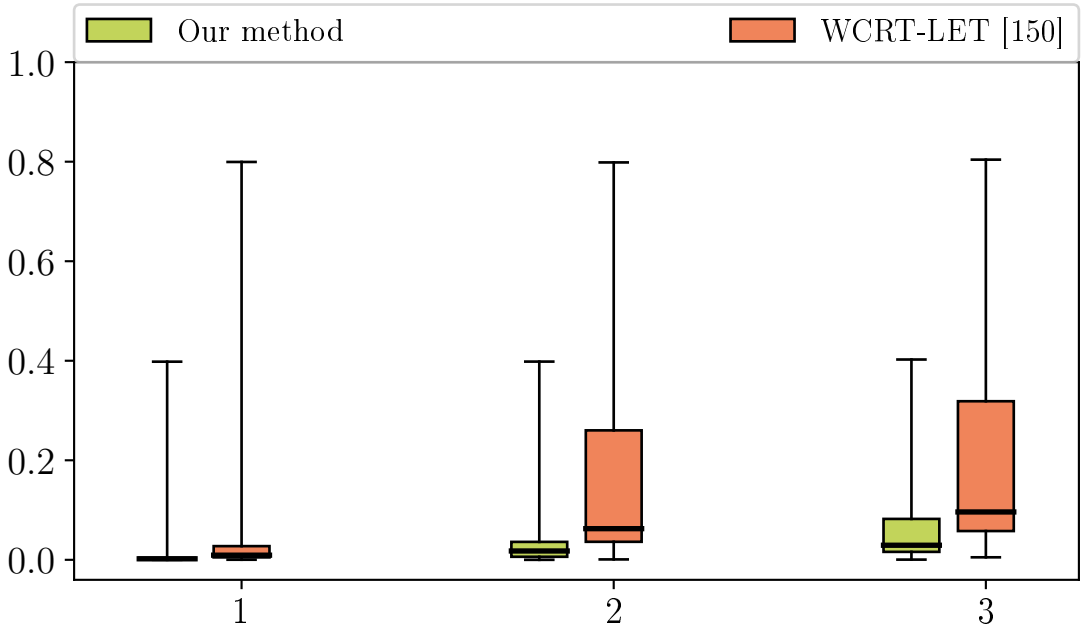


Figure V.3: *Normalized end-to-end latency w.r.t LET per number of periods in the chain*

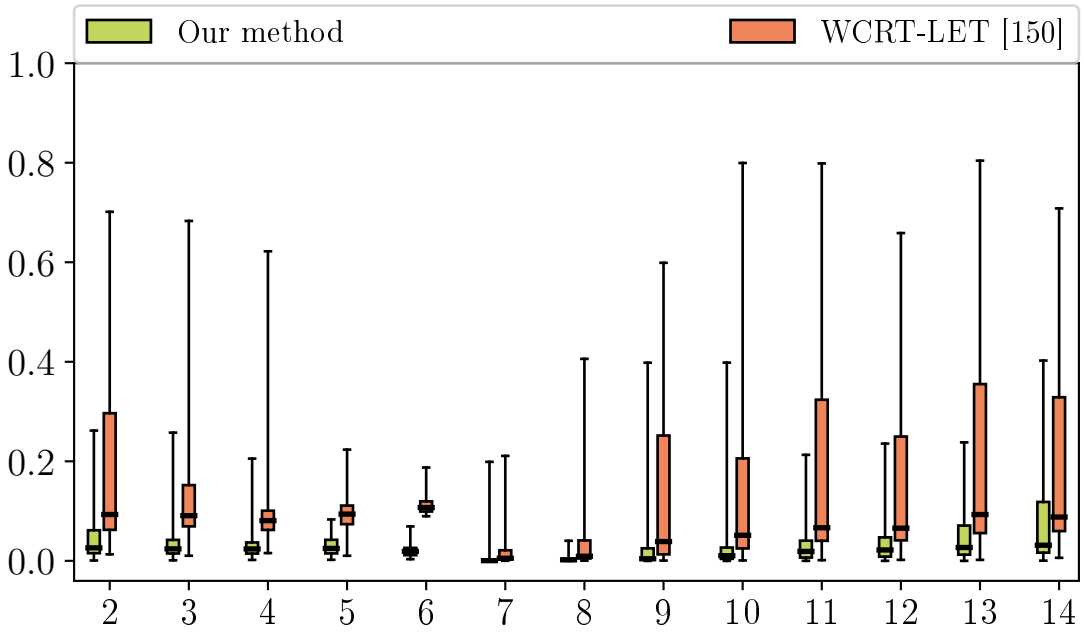


Figure V.4: *Normalized end-to-end latency w.r.t LET per number of tasks in the chain*

Table III.7 shows that 70% of the CECs present in the automotive benchmarks are single-rate. Given the fact that our method from Section IV.2 shortens and shifts tasks' communication intervals, it makes possible for incoming inputs to propagate through the CEC within one repetition interval rather than in multiple as with $|L_\tau| = T_\tau$. As a result, in some cases, our method resulted in a latency improvement exceeding 80% of that achieved using the LET paradigm (See figures V.1 and V.2).

In Figure V.5, we analyze a single task set to further show how much improvement our method achieves over the LET paradigm with respect to the MDA latency metric. We randomly selected to analyze one task set from the 500 schedulable available task sets. The selected task set has 32 CECs and a total of 131 tasks distributed on the 4 cores. On average, the total utilization of cores is $\approx 81\%$ (per core). Note that in Figure V.5, we sorted the CECs in ascending order according to their ID label. For example, the CEC with ID=14 is a single-rate CEC consisting of 4 tasks and a 20ms activation period. Figure V.5 shows that for the CEC with ID=14, our method improved the MDA latency value by $\approx 78\%$, while the WCRT-LET model improved it by $\approx 12.5\%$.

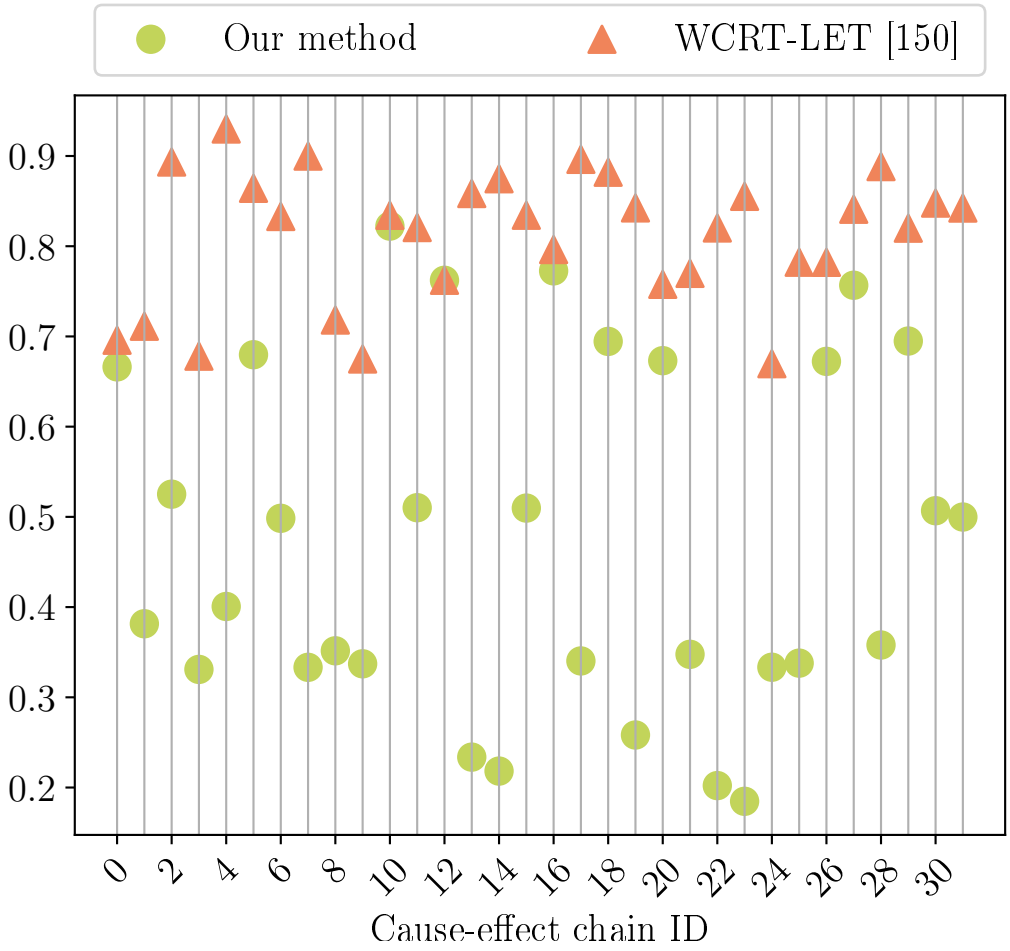


Figure V.5: *Comparison of the normalized maximum data age values of the CECs present in a randomly selected task set from the automotive benchmark*

Figure V.6 further shows the improvements of our method for another randomly selected task set. There are 50 CECs and 241 tasks in the task set under analysis.

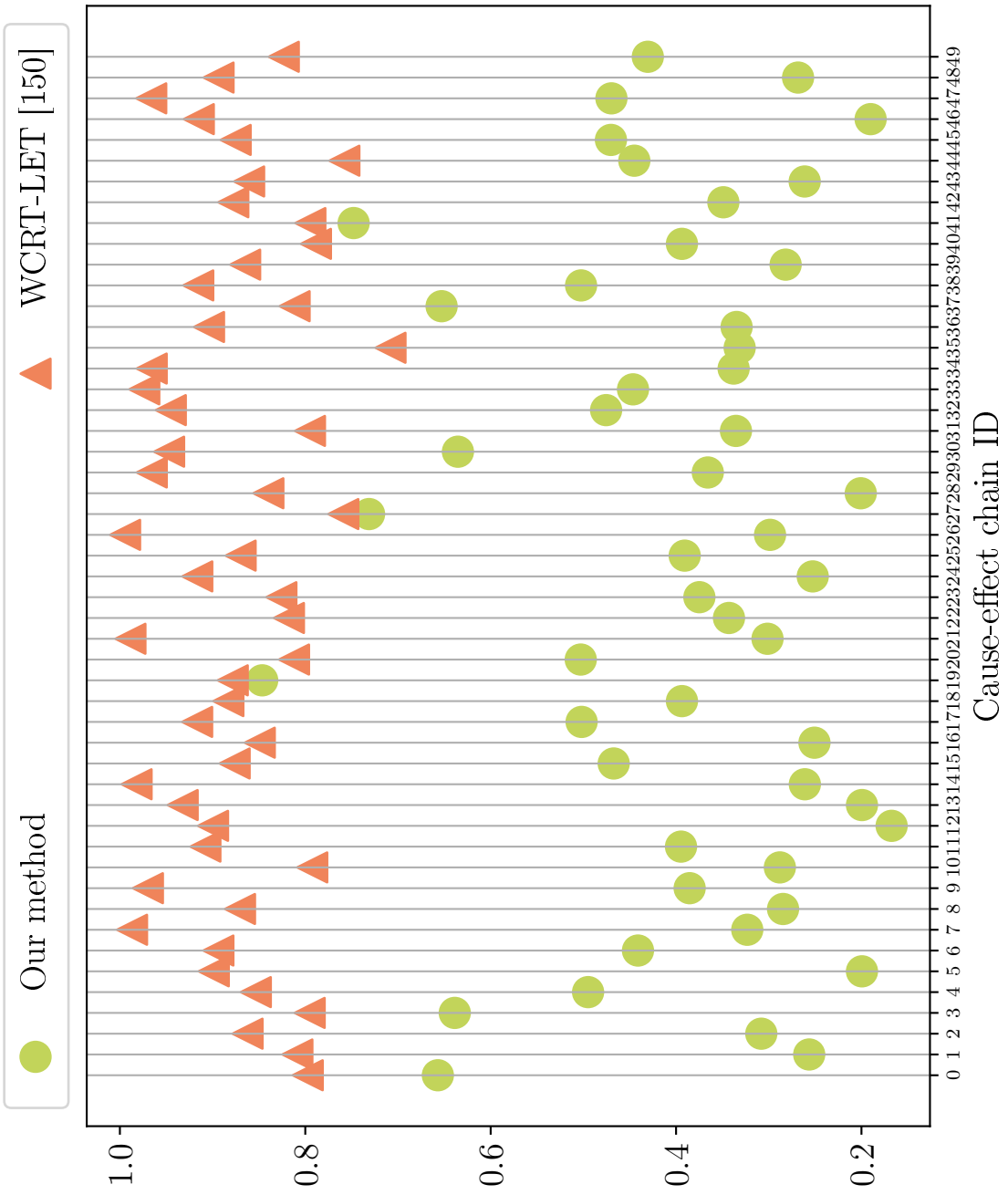


Figure V.6: *Detailed comparison of the normalized maximum data age values of the CECs present in a randomly selected task set from the automotive benchmark*

Schedule Manipulation to Improve End-to-End Latencies

In this section, we present the evaluation of our method presented in Section IV.4. For this set of experiments, our framework reconfigured tasks' communication intervals by manipulating the underlying schedule through the establishment of precedence constraints. The framework derived boundaries for the communication intervals based on the earliest start time and latest finish time of tasks after the establishment of precedence constraints outputted by its search functions (See Section IV.8). We evaluated the performance of our method according to the MRT and MDA latency metrics (See figures V.7 and V.8). For this set of experiments, we also investigated the performance gain of our method, which adds precedence constraints, compared to our schedule-aware model proposed in Section IV.2.

Figures V.7 and V.8 show, respectively, the results for MRT and MDA latency metrics of the CECs that our method was able to further optimize. Note that in both figures we normalized the results with respect to the latency obtained by the LET paradigm.

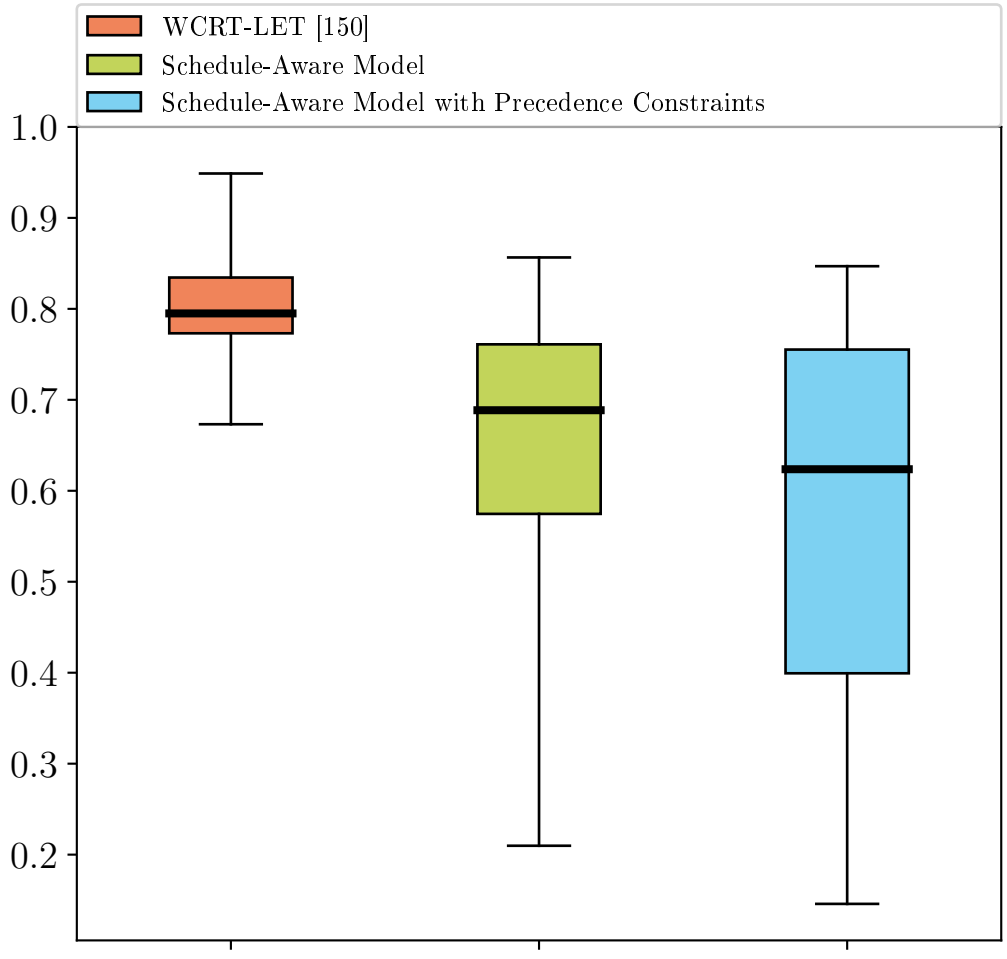


Figure V.7: *Normalized maximum reaction time w.r.t LET from the automotive benchmark*

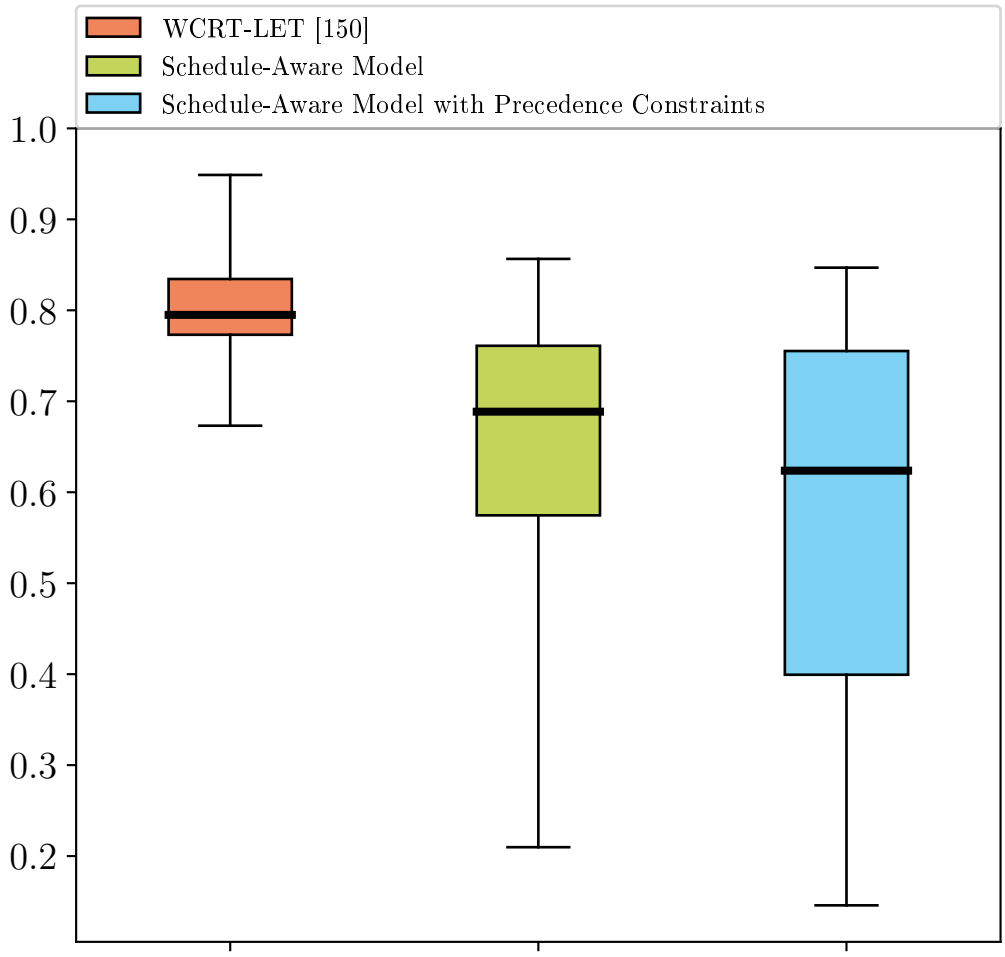


Figure V.8: *Normalized maximum data age w.r.t LET from the automotive benchmark*

Figures V.7 and V.8 show that by establishing precedence constraints, our method can obtain more optimized MRT and MDA latency metrics for some CECs when compared to other methods. The improvement over our method proposed in Section IV.2 can be up to $\approx 16\%$ for some CECs depending on the task set. For the results shown in figures V.7 and V.8, our method established an average of 14 precedence constraints per task set, being 6 the minimum and 78 the maximum. It is worthy mentioning that we noticed during the experiments that for some CECs, although the communication intervals have a different configuration, our method obtains the same MRT and MDA latency values as our schedule-aware model.

Improving System Utilization

In this section, we present the results of our method to decrease the utilization of systems with multi-rate CECs applying the LET communication paradigm (See Section IV.5). For this set of experiments, our framework reconfigured the communication intervals of tasks in a way that the number of task instances that can have their execution skipped is maximized. Note that although we configured our framework to maximize the number of task instances that can have their execution skipped, only solutions that don't increase the E2E latencies of the CECs present in the task sets are considered as a valid plan for interval reconfiguration.

Figure V.9 shows that for task sets based on the automotive benchmark, our method reduced system utilization by $\approx 28\%$, on average, compared to when no task instances have their execution skipped.

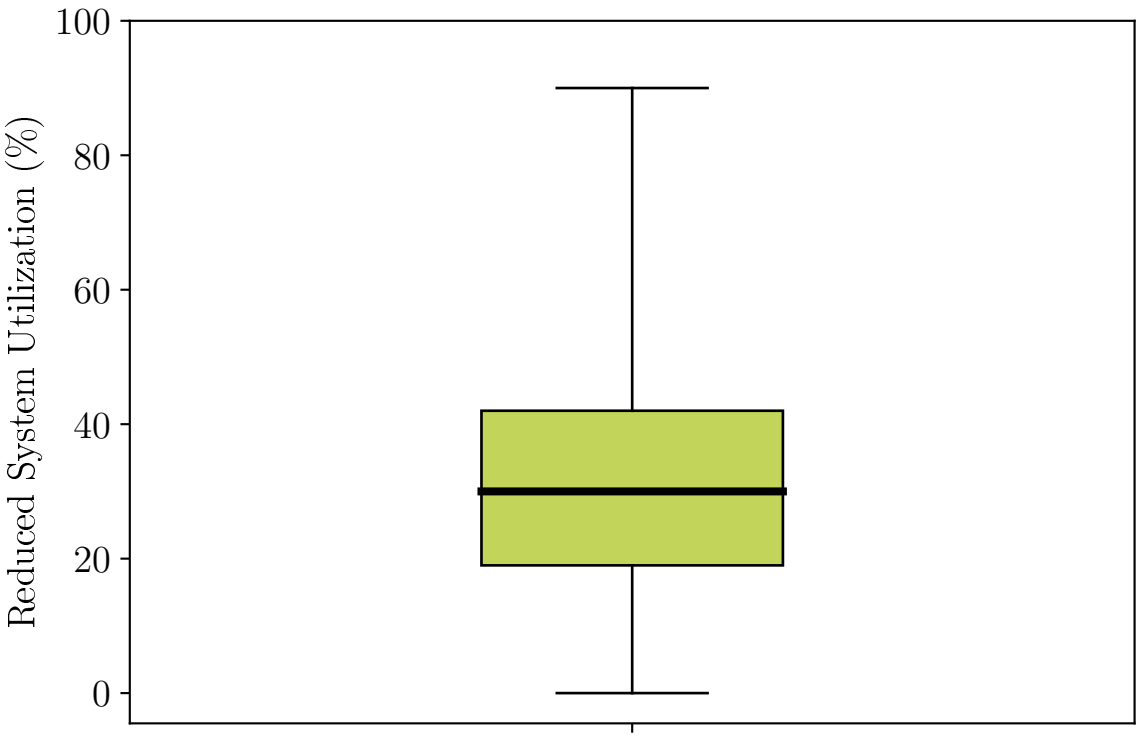


Figure V.9: *Reduction in system utilization of task sets from the automotive benchmark*

Improving End-to-End Latency Feasibility in Multi-Execution Mode Systems

In this section, we present the evaluation of our method presented in Section IV.6. For this first set of experiments, we configured our framework to simulate a system comprised of two cores and four execution modes, being 25% the probability of a CEC be part of one, two, three or four modes. Task are scheduled by our framework according to the EDF scheduling policy. Moreover, we assume that each task is part of at least one CEC and that only the E2E latency requirements of the CECs can change after switching from one execution mode to another. That is, task parameters such as WCET and period activation remain the same after a mode change.

We randomly selected the E2E latency requirements for the CECs present in each mode from a predefined interval Δ, where the upper bound of Δ was the latency obtained by the LET paradigm and the lower bound was the latency obtained by our method proposed in Section IV.2. We evaluated the performance of our method under different loads. The range of possible system utilization values is: [30, 40, 50, 60, 70, 80, 90]. Figure V.10 shows the measured performance of our method. Note that in the Figure V.10, the x-axis represents the average system utilization given the number of cores available.

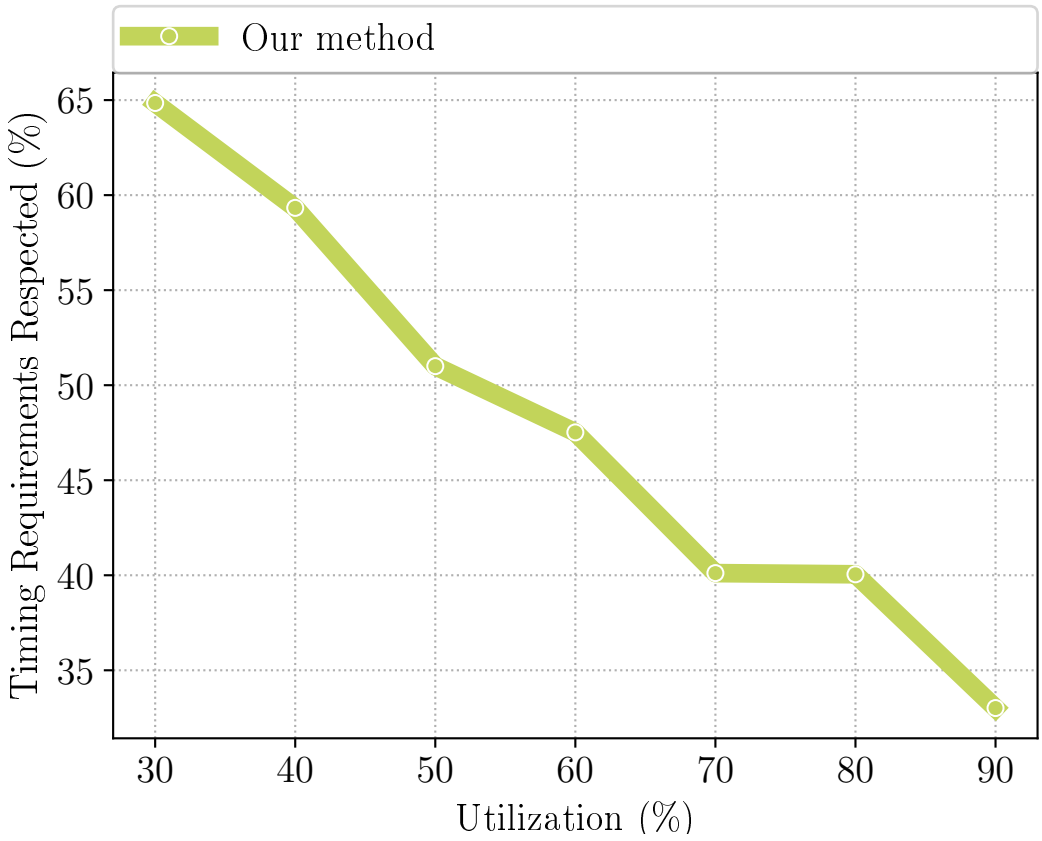


Figure V.10: *End-to-end latency feasibility in a system with two cores and tasks from the automotive benchmark*

Figure V.10 shows that the performance of our method degrades as the system utilization increases. However, this is an expected behavior since, as system utilization increases, the leeway to migrate jobs from one core to another without affecting the E2E latencies of other CECs decreases. That is, in a system setup in which all tasks are part of at least one CEC, the performance of our method is limited to the amount of available leeway that allows job migration without affecting tasks' earliest start time and latest finishing time.

In order to evaluate the scalability of our method on systems with more cores available, we performed two more sets of experiments. In both of them, we followed the same configuration steps as in the previous experiment. The only change was that we evaluated our method on a system comprised of three and four cores. Figure V.11 shows that, as the number of available cores increases, the performance of our method remains almost constant for most of the time.

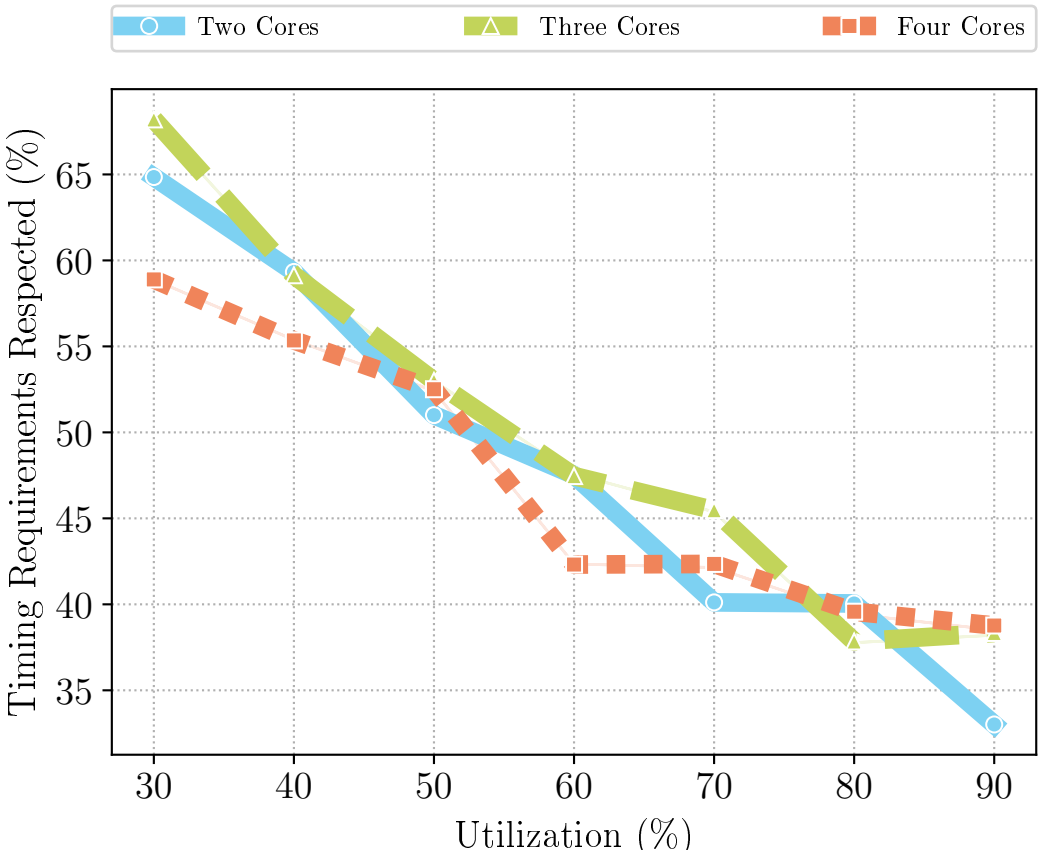


Figure V.11: *Performance comparison of our method on systems with a different number cores and tasks from the automotive benchmark*

V.0.2 Synthetic Task Sets

For this second set of experiments, we randomly generated another 500 schedulable task sets for each evaluated contribution, where we derived some task's parameters based on the automotive benchmark, while others were modified. For instance, we chose task periods and inter-task communication in the same manner as done for the first set of experiments (See Section V.0.1). For this second set of experiments, we made 3 main changes on how we derive some task's parameters and configurations for the CECs. Below we list the changes made in comparison with the first set of experiments:

- tasks can have higher WCET values
- increased the number of possible periods per CEC from 3 to 5
- reduced the probability of single-rate CECs from 70% to 7%

Note that the 63% difference after reducing the probability of single-rate CECs was split equally (63%/4=15.75%) to the other probability rates. The new probabilities for possible number of periods per chain are $\{1\colon 7\%, 2\colon 35.75\%, 3\colon 25.75\%, 4\colon 15.75\%, 5\colon 15.75\%\}$. As a result of increasing the number of possible periods in a CEC, we also increased the maximum number of tasks that could compose a CEC from 14 to 22. Except for our method that considers multiple execution modes (See Section IV.6), for this second set of experiments we chose randomly the amount of CECs per task set from interval $[10, 20]$, with an average of 13 CECs per task set. For our method that considers multiple execution modes, we chose the amount of CECs active per execution mode from the same interval as in the first set of experiments (See Section V.0.1).

The total utilization of cores is $\approx 80\%$ (per core), on average, except for our method that considers multiple execution modes, since we tested its performance under different system loads.

Safe Reconfiguration of Communication Intervals

In this section, we present the evaluation of our method presented in Section IV.2. For this first part of the experiments based on synthetic task sets, we allowed our framework to reconfigure communication intervals solely based on information extracted from the simulated schedule. We evaluated the optimization performance of our method according to the MRT and MDA latency metrics (See figures V.12 and V.13). We also investigated the E2E latency improvement of our method according to the number of activation periods or tasks present in the CECs (See figures V.14 and V.15 respectively). In order to show in detail the improvements of our method when the probability of multi-rate CECs is higher, we performed a thorough evaluation of randomly selected task sets (See figures V.16 and V.17).

Figures V.12 and V.13 summarize the results obtained by our method for the MRT and MDA latency metrics respectively. Note that in both figures we normalized the results with respect to the latency values obtained by the LET paradigm.

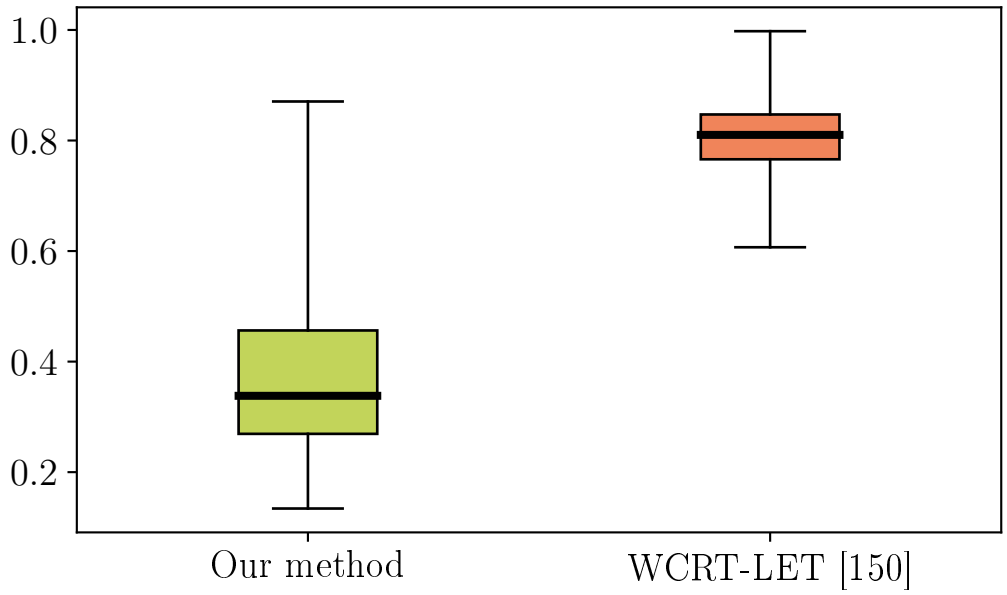


Figure V.12: *Normalized maximum reaction time w.r.t LET from synthetic tasks sets*

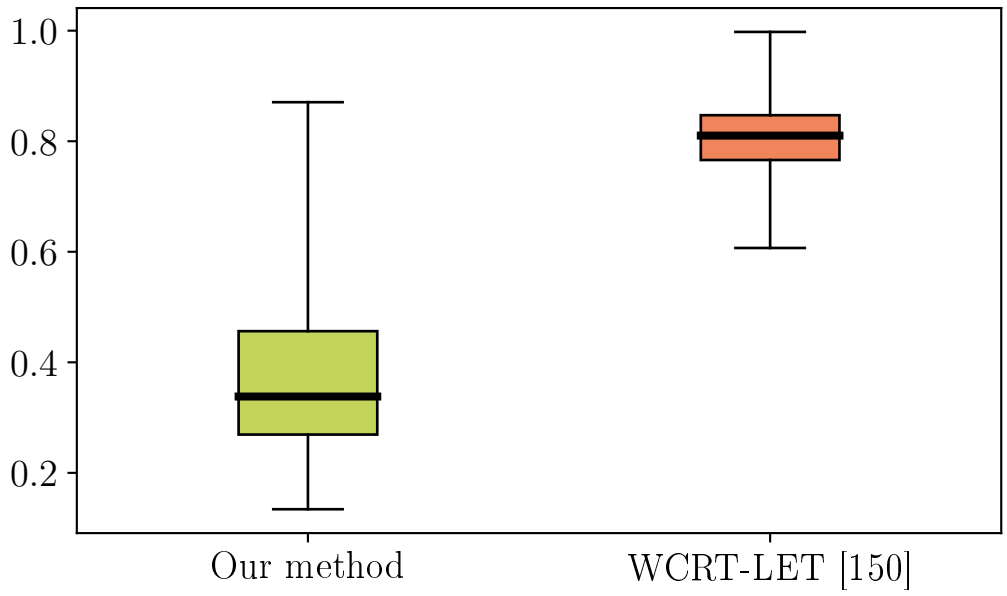


Figure V.13: *Normalized maximum data age w.r.t LET from synthetic tasks sets*

Figure V.12 shows that our model managed to obtain MRT values that are on average, $\approx 67\%$ lower than the values obtained by LET. Similarly, the WCRT-LET model managed to improve MRT values by $\approx 20\%$. Note that our method achieved the same optimization for both latency metrics (MRT and MDA) as shown in figures V.12 and V.13.

Figures V.14 and V.15 show the improvements achieved by our method when analyzing the individual E2E latencies obtained by each job chain in all task sets. Note that in both figures, we normalized the results with respect to the LET paradigm.

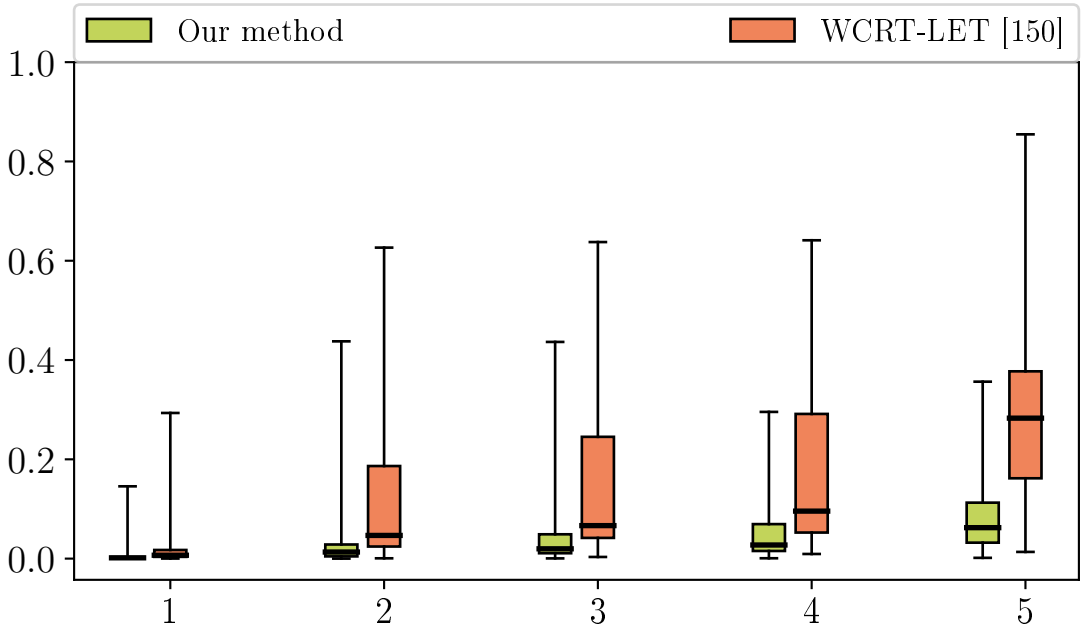


Figure V.14: *Normalized end-to-end latency w.r.t LET per number of periods in the chain*

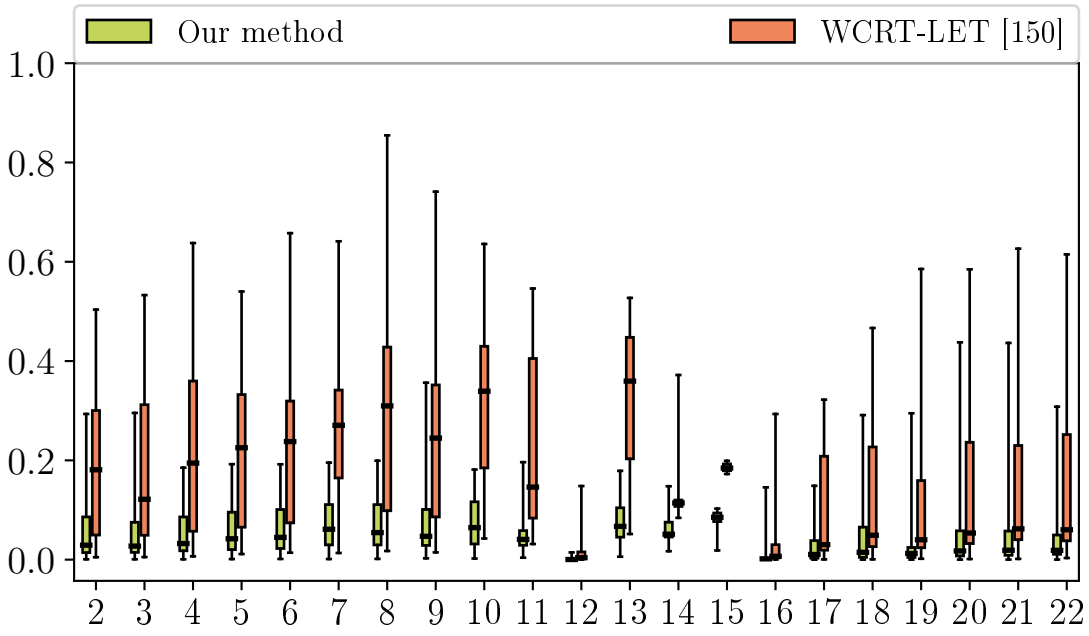


Figure V.15: *Normalized end-to-end latency w.r.t LET per number of tasks in the chain*

In order to further analyze how much improvement our method achieves over the LET paradigm and the WCRT-LET model, we show in Figure V.16 the MDA latency values of each CEC present in a randomly selected synthetic task set. As shown in Figure V.16, the randomly selected task set has 12 CECs and a total of 108 tasks distributed on the 4 available cores. On average, the total utilization of cores is $\approx 74\%$ (per core). Note that in Figure V.16, we sorted the CECs in ascending order according to their ID label. For example, the CEC with ID=11 is a multi-rate CEC consisting of 14 tasks with one of the following activation periods [1, 10, 20, 100, 1000]ms. Figure V.16 shows that for the CEC with ID=11, our method improved the MDA latency value by $\approx 76\%$, while the WCRT-LET model improved it by $\approx 23\%$.

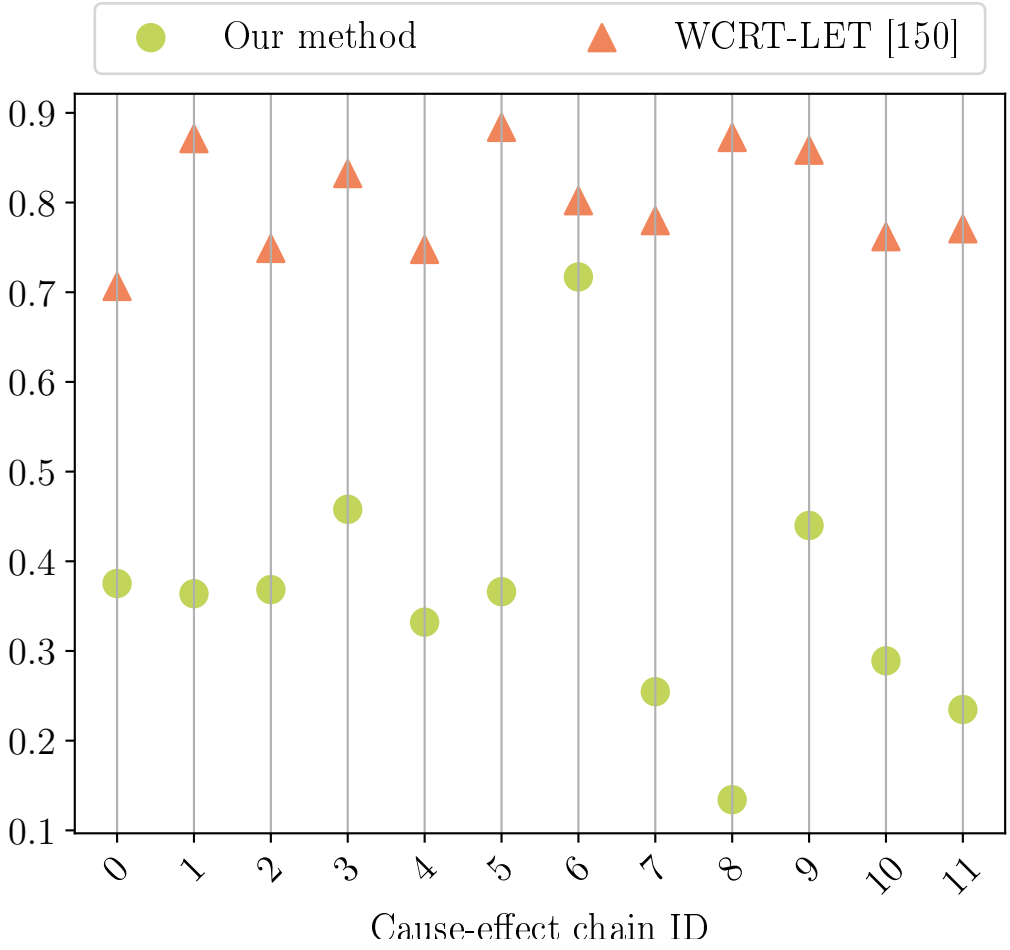


Figure V.16: *Detailed comparison of the normalized maximum data age values of the CECs present in a randomly selected synthetic task set*

Figure V.17 further shows the improvements of our method for another randomly selected task set, which in this case has a total of 10 CECs and 89 tasks.

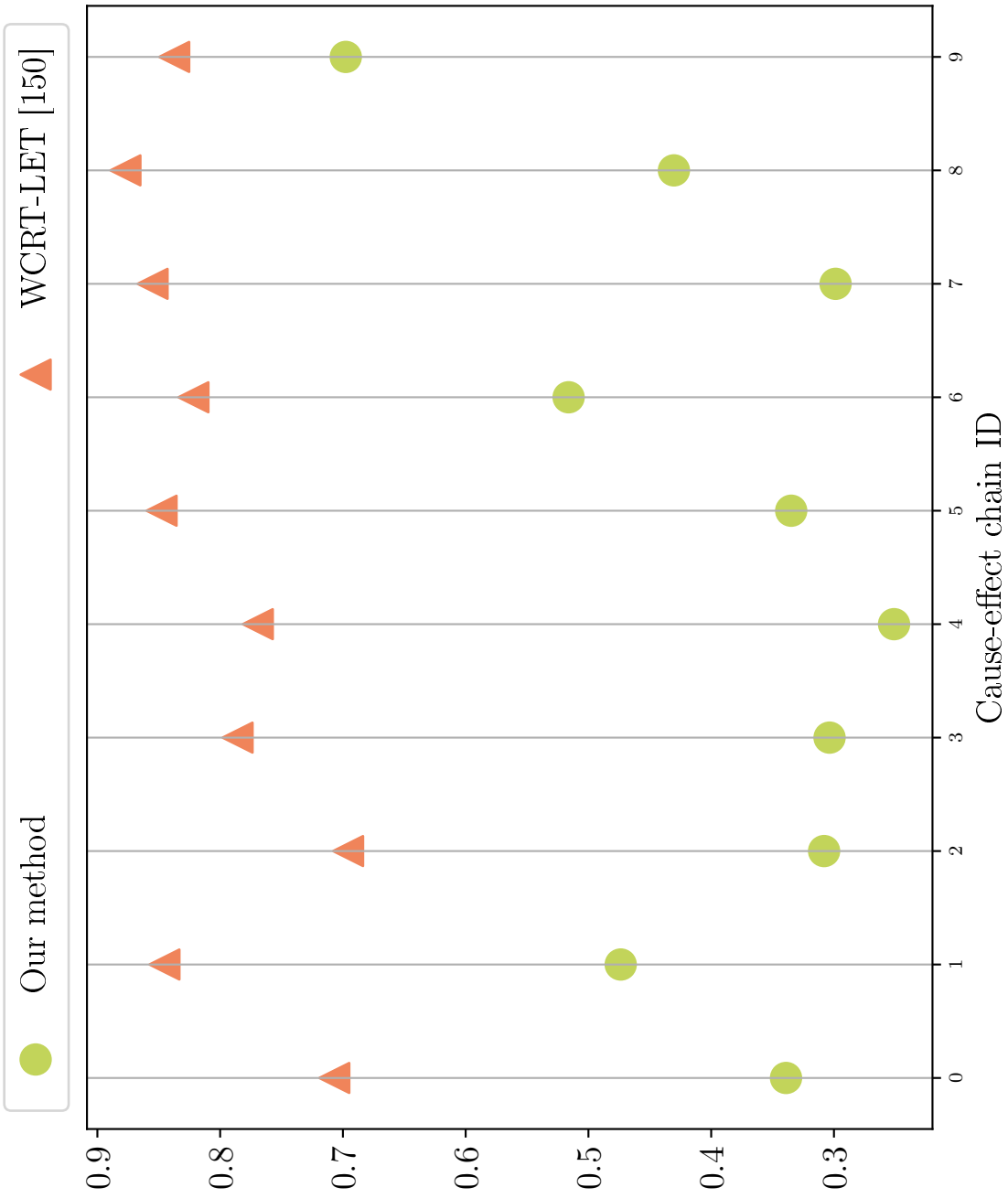


Figure V.17: *Detailed comparison of the normalized maximum data age values of the CECs present in a randomly selected synthetic task set*

Schedule Manipulation to Improve End-to-End Latencies

In this section, we present the evaluation of our method presented in Section IV.4. For this set of the experiments based on synthetic task sets, we allowed our framework to reconfigure communication intervals based on information extracted from a scheduled manipulated by means of precedence constraints. We evaluated the performance of our method according to the MRT and MDA latency metrics (See figures V.18 and V.19). Note that in both figures we normalized the results with respect to the latency values obtained by the LET paradigm.

Figures V.18 and V.19 show that for synthetic task sets, the improvement of our proposed method can be up to $\approx 14\%$ for some CECs. For the results shown in both figures, a minimum of 4 and a maximum of 62 precedence constraints were established by our method.

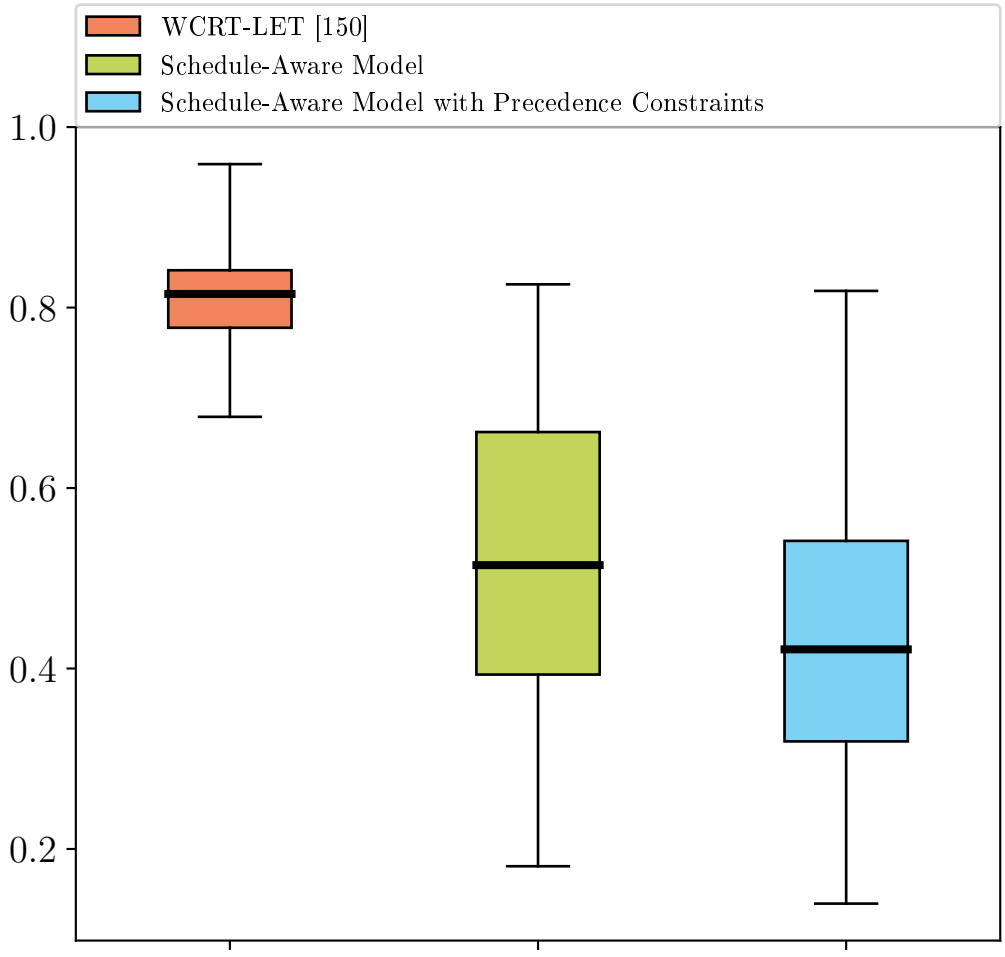


Figure V.18: *Normalized maximum reaction time w.r.t LET from synthetic task sets*

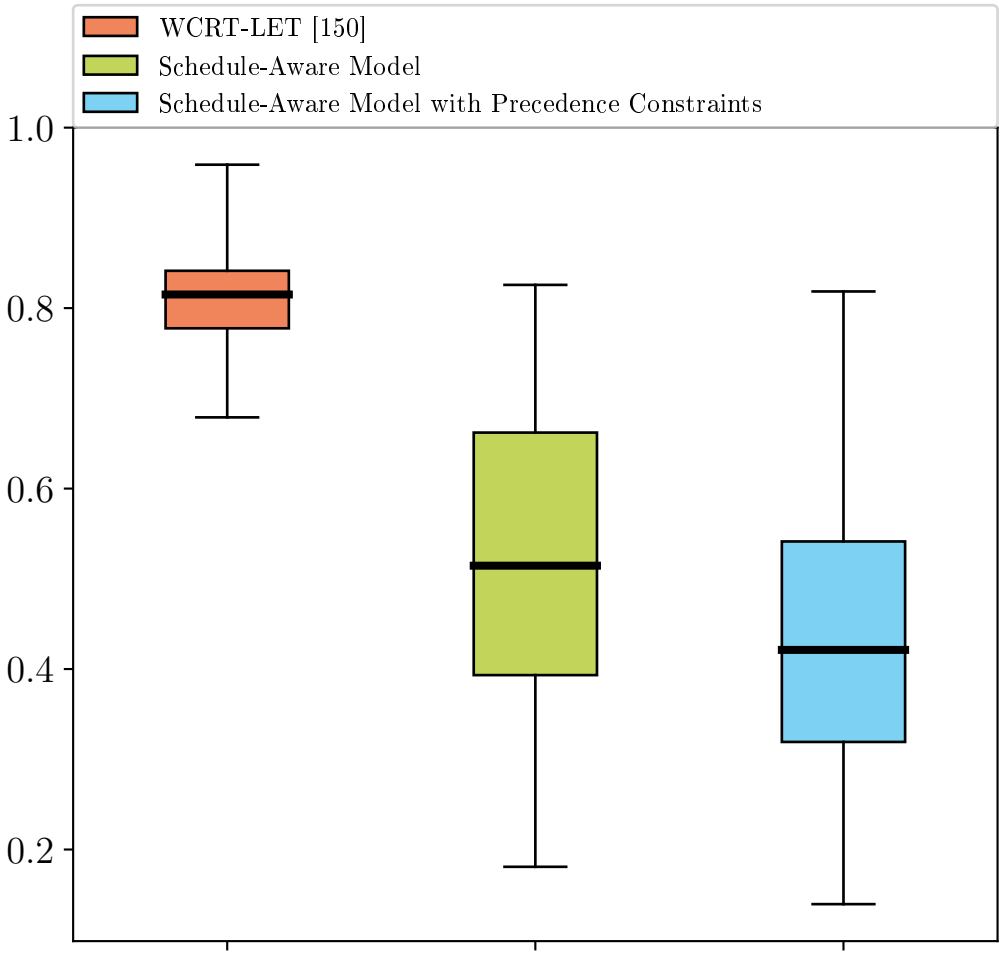


Figure V.19: *Normalized maximum data age w.r.t LET from synthetic task sets*

Improving System Utilization

In this section, we evaluate the performance of our method proposed in Section IV.5 to decrease the utilization of systems with multi-rate CECs applying the LET communication paradigm. For this set of experiments considering synthetic task sets, we configured our framework in the same manner as done for the automotive benchmark (See Section V.0.1).

Figure V.20 shows that for synthetic task sets, our method was capable of reducing system utilization by $\approx$ 55%, on average, compared to when no task instances have their execution skipped. We believe that the additional improvement of almost 30% compared to the results obtained for the automotive benchmark is due to the fact that there are more multiple-rate CECs in our second experimental setup. By decreasing the probability of single-rate CECs from 70% to 7% and increasing the probability of multi-rate CECs, more communicating tasks in the chains suffer with under/oversampling effects. As a result, there are more task instances that can have their execution skipped since their outputs are not propagated and don't affect the E2E latencies of the CECs.

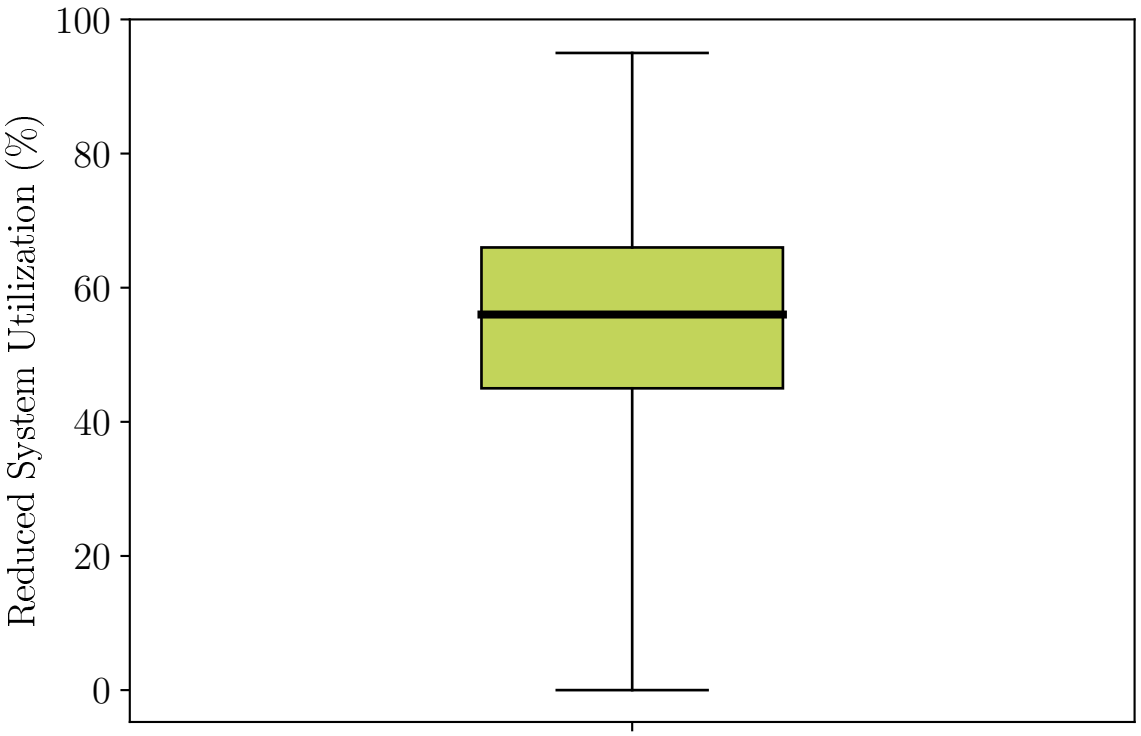


Figure V.20: *Reduction in system utilization from synthetic task sets*

Improving End-to-End Latency Feasibility in Multi-Execution Mode Systems

In this section, we evaluate the performance of our method proposed in Section IV.6. For this set of experiments considering synthetic task sets, we configured our framework and derived some parameters/requirements for tasks/ CECs in the same manner as done for the first set of experiments when we considered the automotive benchmark.

The main difference of this experimental setup is that we did not enforce all tasks present in the task sets to be part of a CEC. That is, we considered the possibility of tasks entering and leaving the system without necessarily being part of a CEC as execution modes change. We evaluated the performance of our method under different loads. The range of possible system utilization values is: [30, 40, 50, 60, 70, 80, 90].

Figure V.21 shows that our method benefits from the fact that not all tasks present in the system are part of a CEC. The reasoning behind this improvement comes from the fact that during the migration process, our feasibility test does not need to consider stricter relative deadlines for the tasks that are not part of a CEC on the candidate core. As a result, it increases the probability of a successful migration, since the tasks that are not part of a CEC can have their execution delayed as long as they finish before their deadline.

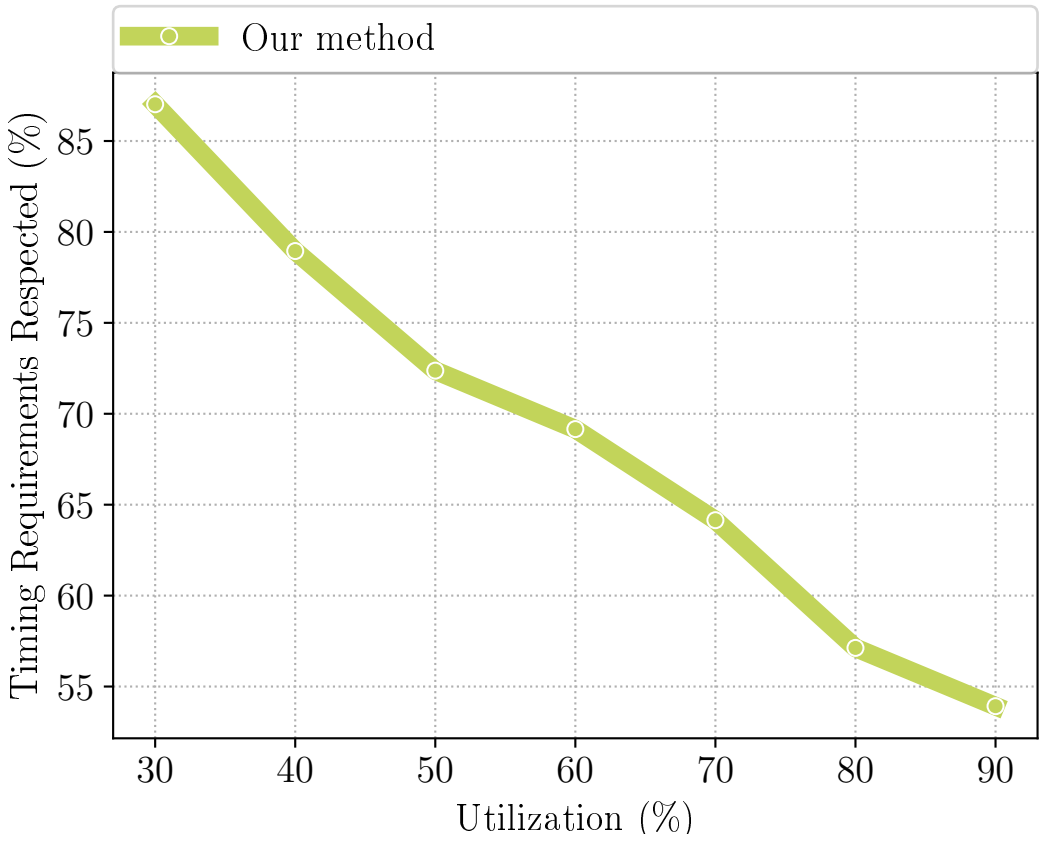


Figure V.21: *End-to-end latency feasibility in a system with two cores and tasks from synthetic task sets*

Figure V.21 shows that when tasks that are not part of CECs are present in the system, our method is able to provide better results even during high system utilization loads. In Figure V.22, we show the performance of our method when increasing the number of cores available in the system from two to three, and then from three to four. Figure V.22 shows, once more, that as the number of cores increases, the performance of our method remains almost constant for most of the time.

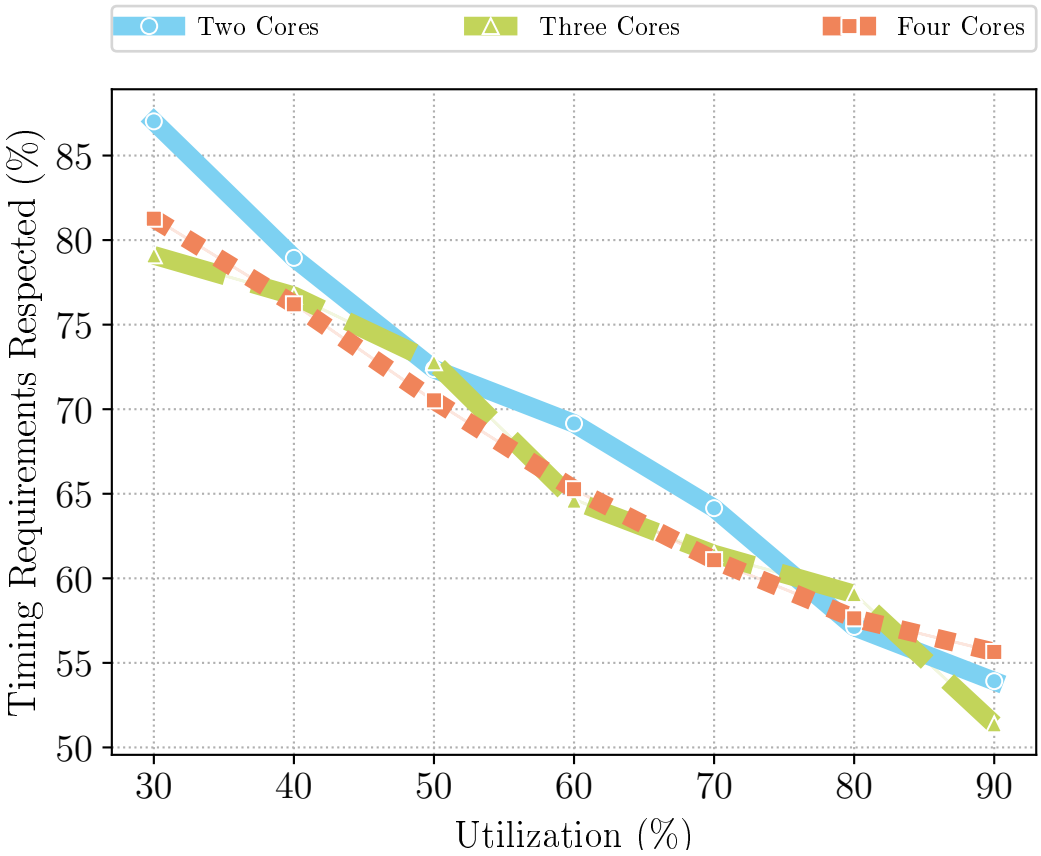


Figure V.22: *Performance comparison of our method on systems with a different number cores and tasks from synthetic task sets*

Chapter VI

Conclusions and Future Work

"For those who come after."

- Gustave, *Clair Obscure: Expedition 33*

Throughout this dissertation, we discussed about the challenges involved in the computation of the end-to-end (E2E) latencies of multi-rate cause-effect chains (CECs) in safety-critical applications present in multi-core real-time (RT) systems. Specifically, we investigated in detail the benefits and disadvantages of the Logical Execution Time (LET) communication paradigm that became part of the AUTomotive Open System ARchitecture (AUTOSAR) standard. In this chapter, we conclude our dissertation. In Section VI.1, we provide a summary of the main contributions of this dissertation, which we presented in Chapter IV. In Section VI.2, we share our view of potential research directions that could lead to further contributions to the literature.

VI.1 Summary of Contributions

In this dissertation, we showed that although the LET paradigm facilitates the timing analysis of multi-rate CECs during early design stages, it also introduces unnecessary pessimism in the form of longer E2E latencies. Throughout Chapter IV, we presented different methods that, without losing the properties of the LET paradigm (timing and data-flow determinism), reduce the long E2E latencies, decrease the waste of processing resources and increase the E2E latency feasibility in multi-execution mode systems.

Safe Reconfiguration of Communication Intervals

In Section IV.2, we demonstrated that by correctly configuring tasks' communication intervals, the E2E latency metrics of multi-rate CECs can be significantly improved while maintaining all the benefits of the LET paradigm. By extracting information from a feasible schedule, our method derives new safe boundaries for tasks' communication intervals. We also demonstrated that by postponing read-events and preponing write-events, our method allows data to propagate through different communication paths, which ultimately result in significantly shorter values for E2E latency metrics

such as Maximum Reaction Time (MRT) and Maximum Data Age (MDA) (See sections V.0.1 and V.0.2).

Timing Analysis of Asynchronous Cause-Effect Chains

Before this dissertation, the analytical methods available in the literature to compute the E2E latencies of multi-rate CECs with tasks applying the LET paradigm didn't consider the possibility of tasks with reconfigured intervals. Given that our method proposed in Section IV.2 to improve the E2E latency metrics of multi-rate CECs reconfigures the communication intervals of tasks applying the LET paradigm, we derived and presented in Section IV.3 a new set of analytical equations to allow the computation of the E2E latencies present in asynchronous CECs.

Schedule Manipulation to Improve End-to-End Latencies

In Section IV.4, we demonstrated how minor manipulations on the schedule by means of precedence constraints can further reduce the E2E latencies of CECs containing tasks with reconfigured communication intervals. We showed that by manipulating when specific task instances start and end their execution, our method presented in Section IV.2 has more flexibility to define new boundaries for tasks' communication intervals. We modeled the problem of finding the set of precedence constraints that best fits the designer's requirements as a search tree and proposed in Section IV.8.1 a searching heuristic function to guide the search. Experiments using automotive benchmarks and synthetic task sets (see sections V.0.1 and V.0.2) showed that the E2E latency metrics of some CECs can be further improved by establishing precedence constraints to safely reconfigure the communication intervals of specific tasks.

Improving System Utilization

Throughout this dissertation, we discussed about the benefits of the timing and dataflow determinsim present in the LET communication paradigm. In Section IV.5, we demonstrated that due to those benefits, it is safe to reduce system utilization by reconfiguring communication intervals and skipping the execution of task instances that don't propagate data values throughout the CEC, i.e., don't affect the E2E latencies of the multi-rate CECs. Since the configuration of tasks' communication intervals defines how data propagates through the CEC, which in turn defines which task instances can be skipped, we modeled the problem of reconfiguring communication intervals as a search tree and proposed in Section IV.8.2 a heuristic function to guide the search. The experiments presented in sections V.0.1 and V.0.2 show that our proposed method can reduce system utilization significantly, especially when the task set is mainly composed of multi-rate CECs.

Improving End-to-End Latency Feasibility in Multi-Execution Mode Systems

As safety-critical control applications often need to operate under multiple system modes during runtime, mode transitions may cause new tasks to join the system which might

increase the E2E latency of certain CECs present in the current execution mode. Since for a safety-critical control application it is important to ensure that the E2E latency requirements are always respected, we demonstrated in Section IV.6 how to increase the feasibility of meeting E2E latency requirements of multi-rate CECs in multi-execution mode systems. We showed that by means of process migration, our method selectively migrates task instances in order to validate the timing constraints of CECs that had their latencies affected after an execution mode change. Moreover, we showed that for every CEC that violates its E2E latency requirements after an execution mode change, our method identifies which tasks of that CEC must have their communication intervals reconfigured by means of process migration. The experiments presented in sections V.0.1 and V.0.2 show that by migrating specific task instances and ensuring their execution within a given interval, our method can improve the E2E feasibility of multi-rate CECs even in scenarios with high system utilization.

VI.2 Future Work

While this dissertation makes significant contributions and advances the state of the art regarding the topic of timing analysis of multi-rate CECs in safety-critical RT systems, we believe that the scientific and technical knowledge in this topic can be further developed. There are still open problems and assumptions that have to be loosened in order to better match the challenges faced by the industry when developing safety-critical RT systems with multi-rate CECs.

In this dissertation, we focused on the LET communication paradigm due to its major benefits in simplifying the timing analysis of complex and heavily interdependent control systems. However, as discussed in the first chapter of this dissertation (See Section I.2.2), in the AUTOSAR standard used by the automotive industry, inter-task communication it is not limited to the LET paradigm. In fact, there are control applications in the automotive industry in which CECs are composed by tasks with different communication paradigms. Therefore, one possible research direction would be to investigate how our reconfiguration method (See Section IV.2) works in CECs with heterogeneous communication paradigms. Specifically, how to properly and safely reconfigure tasks' communication intervals, taking into account the uncertain access times to shared resources from tasks applying paradigms different from LET. Moreover, new equations will need to be derived in order to compute the E2E of the heterogeneous CECs.

Another possible research direction is to investigate how the timing and data-flow determinism of LET along side our method to skip the execution of task instances can improve fault tolerance in safety-critical systems without affecting the E2E latencies of the CECs present in the system. More precisely, we could derive a method that, given the occurrence of a fault during the execution of a job, would identify which job from another task can have its execution skipped in order to allow the re-execution of the faulty job. That is, instead of skipping the execution of a task instance to reduce system utilization, the proposed method would skip the execution of a given job to allow the re-execution of a faulty job. By skipping the execution of jobs that don't affect the E2E latencies or increase the MRT (resp. MDA) latency metrics, the proposed method would be able to increase the tolerance of the system to faults.

In the topic of multi-execution mode systems, we believe that further investigation could be done assuming that tasks' parameters change after an execution mode change. This assumption would impact which task instances should be selected to migrate. Moreover, from our point of view, it would be interesting to investigate the performance of our method during the transitional interval of time between one execution mode and the other.

While there are still many open and challenging problems to be solved in the topic of timing analysis of multi-rate CECs in safety-critical RT systems, we believe that the discussions and contributions presented throughout this dissertation provide valuable insights on how to advance state of the art and explore new research directions bringing new contributions to the literature.

Appendix A

Appendix for Chapter II

A.1 Rate-Monotonic Schedule

Figure A.1 shows the schedule obtained after scheduling the task set shown in Table A.1 according to the Rate-Monotonic (RM) policy.

Task (τ_i)	worst-case execution time (WCET)	period (T_i)
τ_1	1	4
τ_2	1	14
τ_3	4	7

Table A.1: *Sample task set for Appendix A.1*

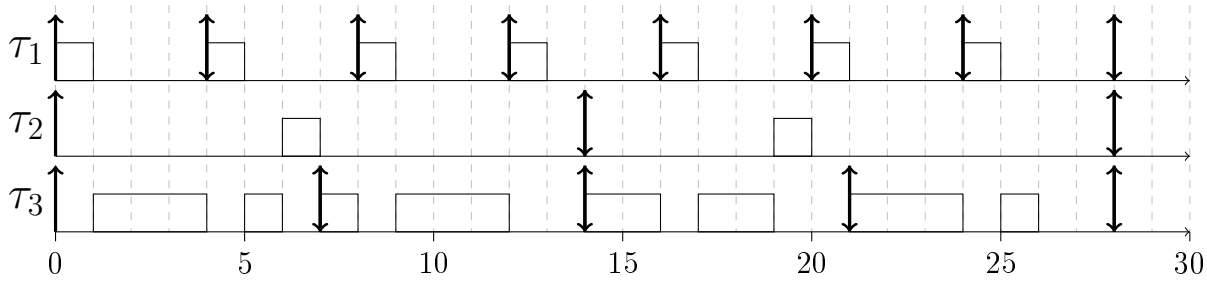


Figure A.1: *Schedule of the task set in Table A.1 according to RM*

A.2 Deadline Monotonic Schedule

Figure A.2 shows the schedule obtained after scheduling the task set shown in Table A.2 according to the Deadline-Monotonic (DM) policy.

Task (τ_i)	WCET	period (T_i)	deadline (D_i)
τ_1	1	4	2
τ_2	1	14	4
τ_3	3	7	6

Table A.2: *Sample task set for Appendix A.2*

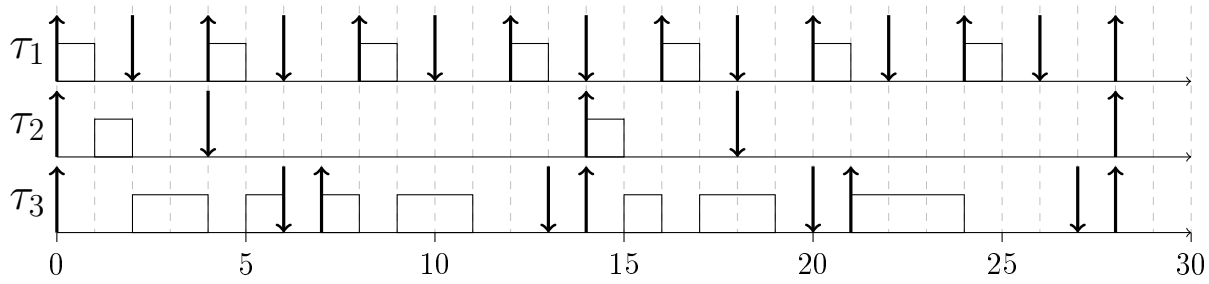


Figure A.2: *Schedule of the task set in Table A.2 according to DM*

A.3 Earliest Deadline First Schedule

Figure A.3 shows the schedule obtained after scheduling the task set shown in Table A.3 according to the Earliest Deadline First (EDF) policy.

Task (τ_i)	WCET	period (T_i)
τ_1	1	4
τ_2	1	14
τ_3	4	7

Table A.3: *Sample task set for Appendix A.3*

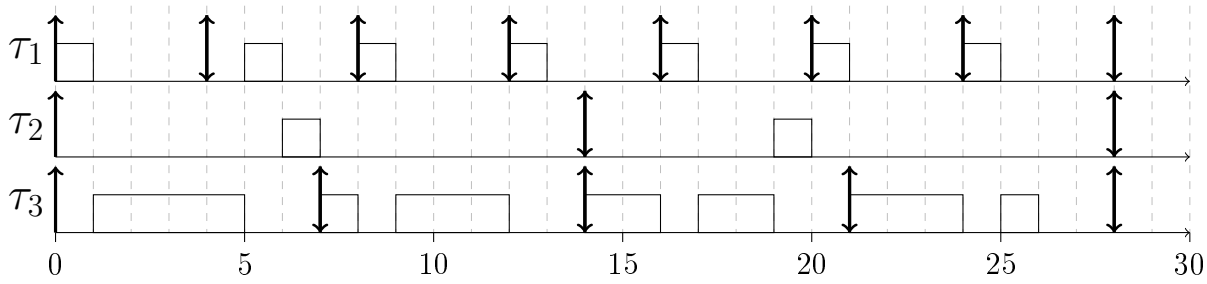


Figure A.3: *Schedule of the task set in Table A.3 according to EDF*

Appendix B

Appendix for Chapter IV

B.1 Interval Configuration Framework Input File

Listing B.1: *Sample XML Input File*

```
1 <?xml version='1.0' encoding='utf-8'?>
2 <SoftwareApplication nCore="1">
3   <CauseEffectChain numberOfPeriods = "2" numberOfTasks = "3">
4     <Communication>
5       <Connection from = "t0" to = "t1"></Connection>
6       <Connection from = "t1" to = "t2"></Connection>
7     </Communication>
8     <Taskset>
9       <Task name = "t0" period = "3" offset = "0" wcet = "1000"
            bestMinWCET = "1000" bestMaxWCET = "1000" worstMinWCET = "
            1000" worstMaxWCET = "1000"></Task>
10      <Task name = "t1" period = "5" offset = "0" wcet = "1000"
            bestMinWCET = "1000" bestMaxWCET = "1000" worstMinWCET = "
            1000" worstMaxWCET = "1000"></Task>
11      <Task name = "t2" period = "3" offset = "0" wcet = "1000"
            bestMinWCET = "1000" bestMaxWCET = "1000" worstMinWCET = "
            1000" worstMaxWCET = "1000"></Task>
12    </Taskset>
13  </CauseEffectChain>
14 </SoftwareApplication>
```

B.2 Interval Configuration Framework Output File

Listing B.2: *Sample XML Output File*

```
<?xml version="1.0" encoding="UTF-8"?>
<System>
 <SoftwareApplication numberOfPeriods ="2" numberOfTasks ="3" ID ="0">
  <Communication>
   <Connection from ="t0" to ="t1"></Connection>
   <Connection from ="t1" to ="t2"></Connection>
  </Communication>
  <Allocation>
   <Core id="0">
    <Task name ="t0" period ="7000" wcet ="1000.0"></Task>
    <Task name ="t1" period ="3000" wcet ="1000.0"></Task>
    <Task name ="t2" period ="7000" wcet ="1000.0"></Task>
   </Core>
  </Allocation>
  <Taskset previousMemoryConsuption ="0">
   <Task name ="t0" period ="7000" offset ="0.0" wcet ="1000.0" esp ="
       0.0" lwp ="3000.0" core ="0">
    <Job name ="t0_0" dependency =""></Job>
    <Job name ="t0_1" dependency =""></Job>
    <Job name ="t0_2" dependency =""></Job>
    <Job name ="t0_3" dependency =""></Job>
    <Job name ="t0_4" dependency =""></Job>
    <Job name ="t0_5" dependency =""></Job>
    <Job name ="t0_6" dependency =""></Job>
    <Job name ="t0_7" dependency =""></Job>
    <Job name ="t0_8" dependency =""></Job>
   </Task>
   <Task name ="t1" period ="3000" offset ="0.0" wcet ="1000.0" esp ="
       0.0" lwp ="2000.0" core ="0">
    <Job name ="t1_0" dependency ="2_0,"></Job>
    <Job name ="t1_1" dependency =""></Job>
    <Job name ="t1_2" dependency =""></Job>
    <Job name ="t1_3" dependency =""></Job>
    <Job name ="t1_4" dependency =""></Job>
    <Job name ="t1_5" dependency ="2_2"></Job>
    <Job name ="t1_6" dependency =""></Job>
    <Job name ="t1_7" dependency ="2_3"></Job>
    <Job name ="t1_8" dependency =""></Job>
    <Job name ="t1_9" dependency =""></Job>
```

```
38      <Job name ="t1_10" dependency =""></Job>
39      <Job name ="t1_11" dependency =""></Job>
40      <Job name ="t1_12" dependency ="2_5,"></Job>
41      <Job name ="t1_13" dependency =""></Job>
42      <Job name ="t1_14" dependency ="2_6"></Job>
43      <Job name ="t1_15" dependency =""></Job>
44      <Job name ="t1_16" dependency =""></Job>
45      <Job name ="t1_17" dependency =""></Job>
46      <Job name ="t1_18" dependency =""></Job>
47      <Job name ="t1_19" dependency ="2_8,"></Job>
48      <Job name ="t1_20" dependency =""></Job>
49    </Task>
50    <Task name ="t2" period ="7000" offset ="1000.0" wcet ="1000.0" esp =
          "0.0" lwp ="1000.0" core ="0">
51      <Job name ="t2_0" dependency =""></Job>
52      <Job name ="t2_1" dependency =""></Job>
53      <Job name ="t2_2" dependency =""></Job>
54      <Job name ="t2_3" dependency =""></Job>
55      <Job name ="t2_4" dependency =""></Job>
56      <Job name ="t2_5" dependency =""></Job>
57      <Job name ="t2_6" dependency =""></Job>
58      <Job name ="t2_7" dependency =""></Job>
59      <Job name ="t2_8" dependency =""></Job>
60    </Task>
61  </Taskset>
62  <Schedule core ="0" previousUtilization ="0.62" currentUtilization ="
        0.62">
63  </Schedule>
64  <CauseEffectChain outputNode ="t2" migration = "false">
65    <standardLET path ="t2,t1,t0," nPeriods ="2" nTasks ="3"
          WCreactionLatency ="28000" WCdataAge ="28000"></standardLET>
66    <wcrtLET path ="t2,t1,t0," nPeriods ="2" nTasks ="3"
          WCreactionLatency ="17000" WCdataAge ="17000"></wcrtLET>
67    <scheduleAwareLET path ="t2,t1,t0," nPeriods ="2" nTasks ="3"
          WCreactionLatency ="16000" WCdataAge ="16000"></scheduleAwareLET>
68  </CauseEffectChain>
69 </SoftwareApplication>
70</System>
```

Bibliography

[1] P. Martí, R. Villa, J. M. Fuertes, and G. Fohler, "On real-time control tasks schedulability," in *2001 European control conference (ECC)*. IEEE, 2001, pp. 2227–2232.

[2] https://www.autosar.org/, [Accessed 17-09-2025].

[3] S. Matic and T. A. Henzinger, "Trading end-to-end latency for composability," in *26th IEEE International Real-Time Systems Symposium (RTSS'05)*. IEEE, 2005, pp. 12–pp.

[4] A. S. Tanenbaum and M. Van Steen, *Distributed systems*. CreateSpace Independent Publishing Platform, 2017.

[5] Tindell, Hansson, and Wellings, "Analysing real-time communications: controller area network (can)," in *1994 Proceedings Real-Time Systems Symposium*. IEEE, 1994, pp. 259–263.

[6] https://www.autosar.org/fileadmin/standards/R19-11/CP/AUTOSAR_TR_TimingAnalysis.pdf, [Accessed 22-09-2025].

[7] A. Hamann, D. Dasari, S. Kramer, M. Pressler, and F. Wurst, "Communication centric design in complex automotive embedded systems," in *29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2017, pp. 10–1.

[8] T. A. Henzinger, B. Horowitz, and C. M. Kirsch, "Giotto: A time-triggered language for embedded programming," in *International Workshop on Embedded Software*. Springer, 2001, pp. 166–184.

[9] M. Verucchi, M. Theile, M. Caccamo, and M. Bertogna, "Latency-aware generation of single-rate dags from multi-rate task sets," in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2020, pp. 226–238.

[10] N. Feiertag, K. Richter, J. Nordlander, and J. Jonsson, "A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics," in *IEEE Real-Time Systems Symposium: 30/11/2009-03/12/2009*. IEEE Communications Society, 2009.

[11] S. Kramer, D. Ziegenbein, and A. Hamann, "Real world automotive benchmarks for free," in *6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*, vol. 130, 2015.

[12] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46–61, 1973.

[13] A. K.-L. Mok, "Fundamental design problems of distributed systems for the hard-real-time environment," Ph.D. dissertation, Massachusetts Institute of Technology, 1983.

[14] J. Martin, *Programming real-time computer systems*. Englewood Cliffs, NJ, Prentice-Hall,, 1965.

[15] G. K. Manacher, "Production and stabilization of real-time task schedules," *Journal of the ACM (JACM)*, vol. 14, no. 3, pp. 439–465, 1967.

[16] B. W. Lampson, "A scheduling philosophy for multiprocessing systems," *Communications of the ACM*, vol. 11, no. 5, pp. 347–360, 1968.

[17] M. Joseph and P. Pandya, "Finding response times in a real-time system," *The Computer Journal*, vol. 29, no. 5, pp. 390–395, 1986.

[18] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings, "Applying new scheduling theory to static priority pre-emptive scheduling," *Software engineering journal*, vol. 8, no. 5, pp. 284–292, 1993.

[19] E. Bini, G. Buttazzo, and G. Buttazzo, "A hyperbolic bound for the rate monotonic algorithm," in *Proceedings 13th Euromicro Conference on Real-Time Systems*. IEEE, 2001, pp. 59–66.

[20] E. Bini, G. C. Buttazzo, G. Buttazzo *et al.*, "Rate monotonic scheduling: The hyperbolic bound," *IEEE Transactions on Computers*, vol. 52, pp. 933–942, 2003.

[21] J. Y.-T. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," *Performance evaluation*, vol. 2, no. 4, pp. 237–250, 1982.

[22] G. C. Buttazzo, "Rate monotonic vs. edf: Judgment day," *Real-Time Systems*, vol. 29, no. 1, pp. 5–26, 2005.

[23] S. K. Baruah, L. E. Rosier, and R. R. Howell, "Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor," *Real-time systems*, vol. 2, no. 4, pp. 301–324, 1990.

[24] C. L. Liu, "Scheduling algorithms for multiprocessors in a hard real-time environment," *JPL Space Programs Summary, 1969*, 1969.

[25] S. K. Dhall and C. L. Liu, "On a real-time scheduling problem," *Operations research*, vol. 26, no. 1, pp. 127–140, 1978.

[26] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, "Proportionate progress: A notion of fairness in resource allocation," in *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, 1993, pp. 345–354.

[27] J. H. Anderson and A. Srinivasan, "Early-release fair scheduling," in *Proceedings 12th Euromicro Conference on Real-Time Systems. Euromicro RTS 2000.* IEEE, 2000, pp. 35–43.

[28] S. K. Baruah, J. E. Gehrke, and C. G. Plaxton, "Fast scheduling of periodic tasks on multiple resources," in *Proceedings of 9th International Parallel Processing Symposium.* IEEE, 1995, pp. 280–288.

[29] B. Andersson, S. Baruah, and J. Jonsson, "Static-priority scheduling on multiprocessors," in *Proceedings 22nd IEEE Real-Time Systems Symposium (RTSS 2001)(Cat. No. 01PR1420).* IEEE, 2001, pp. 193–202.

[30] J. Carpenter, S. H. Funk, P. Holman, A. Srinivasan, J. H. Anderson, and S. K. Baruah, "A categorization of real-time multiprocessor scheduling problems and algorithms." Department of Computer Science, University of North Carolina at Chapel Hill, Tech. Rep., 2004.

[31] M. Bertogna, M. Cirinei, and G. Lipari, "New schedulability tests for real-time task sets scheduled by deadline monotonic on multiprocessors," in *International Conference on Principles of Distributed Systems.* Springer, 2005, pp. 306–321.

[32] B. Andersson, "Global static-priority preemptive multiprocessor scheduling with utilization bound 38%," in *International Conference on Principles of Distributed Systems.* Springer, 2008, pp. 73–88.

[33] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM computing surveys (CSUR)*, vol. 43, no. 4, pp. 1–44, 2011.

[34] A. Srinivasan and S. Baruah, "Deadline-based scheduling of periodic task systems on multiprocessors," *Information processing letters*, vol. 84, no. 2, pp. 93–98, 2002.

[35] J. Goossens, S. Funk, and S. Baruah, "Priority-driven scheduling of periodic task systems on multiprocessors," *Real-time systems*, vol. 25, no. 2, pp. 187–205, 2003.

[36] T. P. Baker, "Multiprocessor edf and deadline monotonic schedulability analysis," in *RTSS 2003. 24th IEEE Real-Time Systems Symposium, 2003.* IEEE, 2003, pp. 120–129.

[37] ——, "An analysis of edf schedulability on a multiprocessor," *IEEE transactions on parallel and distributed systems*, vol. 16, no. 8, pp. 760–768, 2005.

[38] B. Andersson and E. Tovar, "Multiprocessor scheduling with few preemptions," in *12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06).* IEEE, 2006, pp. 322–334.

[39] B. Andersson and K. Bletsas, "Sporadic multiprocessor scheduling with few preemptions," in *2008 Euromicro Conference on Real-Time Systems.* IEEE, 2008, pp. 243–252.

[40] K. Bletsas and B. Andersson, "Notional processors: an approach for multiprocessor scheduling," in *2009 15th IEEE Real-Time and Embedded Technology and Applications Symposium.* IEEE, 2009, pp. 3–12.

[41] S. Kato and N. Yamasaki, "Real-time scheduling with task splitting on multiprocessors," in *13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2007).* IEEE, 2007, pp. 441–450.

[42] ——, "Portioned edf-based scheduling on multiprocessors," in *Proceedings of the 8th ACM international conference on Embedded software*, 2008, pp. 139–148.

[43] ——, "Semi-partitioned fixed-priority scheduling on multiprocessors," in *2009 15th IEEE Real-Time and Embedded Technology and Applications Symposium.* IEEE, 2009, pp. 23–32.

[44] S. Kato, N. Yamasaki, and Y. Ishikawa, "Semi-partitioned scheduling of sporadic task systems on multiprocessors," in *2009 21st Euromicro Conference on Real-Time Systems.* IEEE, 2009, pp. 249–258.

[45] K. Lakshmanan, R. Rajkumar, and J. Lehoczky, "Partitioned fixed-priority preemptive scheduling for multi-core processors," in *2009 21st Euromicro Conference on Real-Time Systems.* IEEE, 2009, pp. 239–248.

[46] I. Shin, A. Easwaran, and I. Lee, "Hierarchical scheduling framework for virtual clustering of multiprocessors," in *2008 Euromicro Conference on Real-Time Systems.* IEEE, 2008, pp. 181–190.

[47] H. Leontyev and J. H. Anderson, "A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees," *Real-Time Systems*, vol. 43, no. 1, pp. 60–92, 2009.

[48] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization," *IEEE Transactions on computers*, vol. 39, no. 9, pp. 1175–1185, 2002.

[49] T. P. Baker, "A stack-based resource allocation policy for realtime processes," in *[1990] Proceedings 11th Real-Time Systems Symposium.* IEEE, 1990, pp. 191–200.

[50] J. H. Anderson, S. Ramamurthy, and K. Jeffay, "Real-time computing with lock-free shared objects," *ACM Transactions on Computer Systems (TOCS)*, vol. 15, no. 2, pp. 134–165, 1997.

[51] H. Kopetz and J. Reisinger, "The non-blocking write protocol nbw: A solution to a real-time synchronization problem," in *1993 Proceedings Real-Time Systems Symposium.* IEEE, 1993, pp. 131–137.

[52] R. Rajkumar, L. Sha, and J. P. Lehoczky, "Real-time synchronization protocols for multiprocessors." in *RTSS*, vol. 88, 1988, pp. 259–269.

[53] P. Gai, G. Lipari, and M. Di Natale, "Minimizing memory utilization of real-time task sets in single and multi-processor systems-on-a-chip," in *Proceedings 22nd ieee real-time systems symposium (rtss 2001)(Cat. No. 01PR1420).* IEEE, 2001, pp. 73–83.

[54] U. C. Devi, H. Leontyev, and J. H. Anderson, "Efficient synchronization under global edf scheduling on multiprocessors," in *18th Euromicro Conference on Real-Time Systems (ECRTS'06).* IEEE, 2006, pp. 10–pp.

[55] A. Block, H. Leontyev, B. B. Brandenburg, and J. H. Anderson, "A flexible real-time locking protocol for multiprocessors," in *13th IEEE international conference on embedded and real-time computing systems and applications (RTCSA 2007).* IEEE, 2007, pp. 47–56.

[56] J. Hennig, H. von Hasseln, H. Mohammad, S. Resmerita, S. Lukesch, and A. Naderlinger, "Towards parallelizing legacy embedded control software using the let programming paradigm. in 2016 ieee real-time and embedded technology and applications symposium (rtas)," *IEEE Computer Soc*, vol. 51, 2016.

[57] S. Resmerita, A. Naderlinger, M. Huber, K. Butts, and W. Pree, "Applying real-time programming to legacy embedded control software," in *2015 IEEE 18th International Symposium on Real-Time Distributed Computing.* IEEE, 2015, pp. 1–8.

[58] S. Resmerita, A. Naderlinger, and S. Lukesch, "Efficient realization of logical execution times in legacy embedded software," in *Proceedings of the 15th ACM-IEEE International Conference on Formal Methods and Models for System Design*, 2017, pp. 36–45.

[59] C. Sofronis, S. Tripakis, and P. Caspi, "A memory-optimal buffering protocol for preservation of synchronous semantics under preemptive scheduling," in *Proceedings of the 6th ACM & IEEE International conference on Embedded software*, 2006, pp. 21–33.

[60] E. Yip, E. Lalo, G. Lüttgen, and A. Sailer, "Lightweight semantics-preserving communication for real-time automotive software," in *2019 IEEE 13th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*. IEEE, 2019, pp. 372–379.

[61] G. Fohler, "How different are offline and online scheduling," *Gerhard Fohler, RTSOPS*, 2011.

[62] A. R. Wasicek, "Security in time-triggered systems," Ph.D. dissertation, Technische Universität Wien, 2011.

[63] K. Krüger, G. Fohler, and M. Volp, "Improving security for time-triggered real-time systems against timing inference based attacks by schedule obfuscation," in *Work-in-Progress Proceedings ECRTS'17*, 2017.

[64] K. Krüger, G. Fohler, M. Völp, and P. Esteves-Verissimo, "Improving security for time-triggered real-time systems with task replication," in *2018 IEEE 24th international conference on embedded and real-time computing systems and applications (RTCSA)*. IEEE, 2018, pp. 232–233.

[65] K. Krüger, M. Volp, and G. Fohler, "Vulnerability analysis and mitigation of directed timing inference based attacks on time-triggered systems," *Leibniz International Proceedings in Informatics*, vol. 106, 2018.

[66] K. Krüger, N. Vreman, R. Pates, M. Maggio, M. Völp, and G. Fohler, "Randomization as mitigation of directed timing inference based attacks on time-triggered real-time systems with task replication," *Leibniz Transactions on Embedded Systems*, vol. 7, no. 1, pp. 01–1, 2021.

[67] G. Fohler, "Joint scheduling of distributed complex periodic and hard aperiodic tasks in statically scheduled systems," in *Proceedings 16th IEEE Real-Time Systems Symposium*. IEEE, 1995, pp. 152–161.

[68] K. Sandstrom, C. Eriksson, and G. Fohler, "Handling interrupts with static scheduling in an automotive vehicle control system," in *Proceedings Fifth International Conference on Real-Time Computing Systems and Applications (Cat. No. 98EX236)*. IEEE, 1998, pp. 158–165.

[69] G. Fohler, T. Lennvall, and G. Buttazzo, "Improved handling of soft aperiodic tasks in offline scheduled real-time systems using total bandwidth server," in *ETFA 2001. 8th International Conference on Emerging Technologies and Factory Automation. Proceedings (Cat. No. 01TH8597)*. IEEE, 2001, pp. 151–157.

[70] Spuri and Buttazzo, "Efficient aperiodic service under earliest deadline scheduling," in *1994 Proceedings Real-Time Systems Symposium*. IEEE, 1994, pp. 2–11.

[71] M. Spuri and G. Buttazzo, "Scheduling aperiodic tasks in dynamic priority systems," *Real-Time Systems*, vol. 10, no. 2, pp. 179–210, 1996.

[72] D. Isovic and G. Fohler, "Handling mixed sets of tasks in combined offline and online scheduled real-time systems," *Real-time systems*, vol. 43, no. 3, pp. 296–325, 2009.

[73] ——, "Efficient scheduling of sporadic, aperiodic, and periodic tasks with complex constraints," in *Proceedings 21st IEEE Real-Time Systems Symposium.* IEEE, 2000, pp. 207–216.

[74] K. W. Tindell, A. Burns, and A. J. Wellings, "Allocating hard real-time tasks: an np-hard problem made easy," *Real-Time Systems*, vol. 4, no. 2, pp. 145–165, 1992.

[75] D. Tamas-Selicean, P. Pop, and W. Steiner, "Synthesis of communication schedules for ttethernet-based mixed-criticality systems," in *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, 2012, pp. 473–482.

[76] M. T. B. Waez, J. Dingel, and K. Rudie, "Timed automata for the development of real-time systems," *Research Report 2011–579*, 2011.

[77] L. Zhang, D. Goswami, R. Schneider, and S. Chakraborty, "Task-and network-level schedule co-synthesis of ethernet-based time-triggered systems," in *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC).* IEEE, 2014, pp. 119–124.

[78] S. S. Craciunas and R. S. Oliver, "Combined task-and network-level scheduling for distributed time-triggered systems," *Real-Time Systems*, vol. 52, no. 2, pp. 161–200, 2016.

[79] D.-T. Peng and K. G. Shin, "Optimal scheduling of cooperative tasks in a distributed system using an enumerative method," *IEEE Transactions on Software Engineering*, vol. 19, no. 3, pp. 253–267, 1993.

[80] P. Bratley, M. Florian, and P. Robillard, "Scheduling with earliest start and due date constraints on multiple machines," *Naval Research Logistics Quarterly*, vol. 22, no. 1, pp. 165–173, 1975.

[81] J. Jonsson, "Effective complexity reduction for optimal scheduling of distributed real-time applications," in *Proceedings. 19th IEEE International Conference on Distributed Computing Systems (Cat. No. 99CB37003).* IEEE, 1999, pp. 360–369.

[82] T. F. Abdelzaher and K. G. Shin, "Combined task and message scheduling in distributed real-time systems," *IEEE Transactions on parallel and distributed systems*, vol. 10, no. 11, pp. 1179–1191, 2002.

[83] A. H. Land and A. G. Doig, "An automatic method for solving discrete programming problems," in *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art.* Springer, 2009, pp. 105–132.

[84] D.-T. Peng and K. G. Shin, "Static allocation of periodic tasks with precedence constraints in distributed real-time systems," in *Proceedings. The 9th International Conference on Distributed Computing Systems.* IEEE Computer Society, 1989, pp. 190–191.

[85] I. Ahmad and Y.-K. Kwok, "Optimal and near-optimal allocation of precedence-constrained tasks to parallel processors: defying the high complexity using effective search techniques," in *Proceedings. 1998 International Conference on Parallel Processing (Cat. No. 98EX205).* IEEE, 1998, pp. 424–431.

[86] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[87] R. E. Korf, "Depth-first iterative-deepening: An optimal admissible tree search," *Artificial intelligence*, vol. 27, no. 1, pp. 97–109, 1985.

[88] V. N. Rao, V. Kumar, and K. Ramesh, *A parallel implementation of iterative-deepening-a.* Artificial Intelligence Laboratory, University of Texas at Austin, 1987.

[89] U. Hönig and W. Schiffmann, "Fast optimal task graph scheduling by means of an optimized parallel a*-algorithm." in *PDPTA*, 2004, pp. 842–848.

[90] S. Kurkovsky, "Experimenting with ida* search algorithm in heterogeneous pervasive environments," *Artificial Intelligence Review*, vol. 29, no. 3, p. 277, 2008.

[91] A. A. J. Syed, "Model-based design and adaptive scheduling of distributed real-time systems," Ph.D. dissertation, Technische Universität Kaiserslautern, 2018.

[92] A. Syed, "Generic Framework for Real-time Offline Scheduling," Master's thesis, TUKL, Kaiserslautern, Germany, 2014.

[93] L. Sha, R. Rajkumar, J. Lehoczky, and K. Ramamritham, "Mode change protocols for priority-driven preemptive scheduling," *Real-Time Systems*, vol. 1, no. 3, pp. 243–264, 1989.

[94] A. Burns and T. Quiggle, "Effective use of abort in programming mode changes," *ACM SIGAda Ada Letters*, vol. 10, no. 6, pp. 61–67, 1990.

[95] K. Tindell, A. Burns, and A. J. Wellings, "Mode changes in priority pre-emptively scheduled systems." in *RTSS*, vol. 92, 1992, pp. 100–109.

[96] P. Pedro and A. Burns, "Schedulability analysis for mode changes in flexible real-time systems," in *Proceeding. 10th EUROMICRO Workshop on Real-Time Systems (Cat. No. 98EX168).* IEEE, 1998, pp. 172–179.

[97] J. Real and A. Crespo, "Mode change protocols for real-time systems: A survey and a new proposal," *Real-time systems*, vol. 26, no. 2, pp. 161–197, 2004.

[98] P. Emberson and I. Bate, "Minimising task migration and priority changes in mode transitions," in *13th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS'07)*. IEEE, 2007, pp. 158–167.

[99] P. Ekberg, M. Stigge, N. Guan, and W. Yi, "State-based mode switching with applications to mixed criticality systems," *Proc. WMC, RTSS*, pp. 61–66, 2013.

[100] A. Burns, "System mode changes-general and criticality-based," in *Proc. of 2nd Workshop on Mixed Criticality Systems (WMC)*, 2014, pp. 3–8.

[101] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *28th IEEE international real-time systems symposium (RTSS 2007)*. IEEE, 2007, pp. 239–243.

[102] A. Burns and R. I. Davis, "A survey of research into mixed criticality systems," *ACM Computing Surveys (CSUR)*, vol. 50, no. 6, pp. 1–37, 2017.

[103] S. K. Baruah, A. Burns, and R. I. Davis, "Response-time analysis for mixed criticality systems," in *2011 IEEE 32nd Real-Time Systems Symposium*. IEEE, 2011, pp. 34–43.

[104] F. Santy, G. Raravi, G. Nelissen, V. Nelis, P. Kumar, J. Goossens, and E. Tovar, "Two protocols to reduce the criticality level of multiprocessor mixed-criticality systems," in *Proceedings of the 21st International Conference on Real-Time Networks and Systems*, 2013, pp. 183–192.

[105] A. Burns and S. Baruah, "Towards a more practical model for mixed criticality systems," in *Workshop on Mixed-Criticality Systems (colocated with RTSS)*, 2013.

[106] O. Gettings, S. Quinton, and R. I. Davis, "Mixed criticality systems with weakly-hard constraints," in *Proceedings of the 23rd International Conference on Real Time and Networks Systems*, 2015, pp. 237–246.

[107] H. Kopetz, R. Nossal, R. Hexel, A. Krüger, D. Millinger, R. Pallierer, C. Temple, and M. Krug, "Mode handling in the time-triggered architecture," *Control Engineering Practice*, vol. 6, no. 1, pp. 61–66, 1998.

[108] F. Heilmann, A. Syed, and G. Fohler, "Mode-changes in cots time-triggered network hardware without online reconfiguration," *ACM SIGBED Review*, vol. 13, no. 4, pp. 55–60, 2016.

[109] G. M. Almeida, S. Varyani, R. Busseuil, G. Sassatelli, P. Benoit, L. Torres, E. A. Carara, and F. G. Moraes, "Evaluating the impact of task migration in multiprocessor systems-on-chip," in *Proceedings of the 23rd symposium on Integrated circuits and system design*, 2010, pp. 73–78.

[110] Y. Ge, P. Malani, and Q. Qiu, "Distributed task migration for thermal management in many-core systems," in *Proceedings of the 47th Design Automation Conference*, 2010, pp. 579–584.

[111] G. Zeng, Y. Matsubara, H. Tomiyama, and H. Takada, "Task migration for energy saving in real-time multiprocessor systems," in *2014 IEEE Intl Conf on High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC, CSS, ICESS)*. IEEE, 2014, pp. 685–692.

[112] P. K. Saraswat, P. Pop, and J. Madsen, "Task migration for fault-tolerance in mixed-criticality embedded systems," *ACM SIGBED Review*, vol. 6, no. 3, pp. 1–5, 2009.

[113] D. S. Milojičić, F. Douglis, Y. Paindaveine, R. Wheeler, and S. Zhou, "Process migration," *ACM Computing Surveys (CSUR)*, vol. 32, no. 3, pp. 241–299, 2000.

[114] A. Acquaviva, A. Alimonda, S. Carta, and M. Pittau, "Assessing task migration impact on embedded soft real-time streaming multimedia applications," *EURASIP journal on embedded systems*, vol. 2008, no. 1, p. 518904, 2007.

[115] M. Pittau, A. Alimonda, S. Carta, and A. Acquaviva, "Impact of task migration on streaming multimedia for embedded multiprocessors: A quantitative evaluation," in *2007 IEEE/ACM/IFIP Workshop on Embedded Systems for Real-Time Multimedia*. IEEE, 2007, pp. 59–64.

[116] Y. Zhang, H. Franke, J. E. Moreira, and A. Sivasubramaniam, "The impact of migration on parallel job scheduling for distributed systems," in *European Conference on Parallel Processing*. Springer, 2000, pp. 242–251.

[117] E. W. Brião, D. Barcelos, F. Wronski, and F. R. Wagner, "Impact of task migration in noc-based mpsocs for soft real-time applications," in *2007 IFIP International Conference on Very Large Scale Integration*. IEEE, 2007, pp. 296–299.

[118] D. Choffnes, M. Astley, and M. J. Ward, "Migration policies for multi-core fair-share scheduling," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 1, pp. 92–93, 2008.

[119] K. M. Katre, H. Ramaprasad, A. Sarkar, and F. Mueller, "Policies for migration of real-time tasks in embedded multi-core systems," *Work-In-Progress, Proceedings Real-time systems symposium (RTSS)*, p. 17, 2009.

[120] A. Sarkar, F. Mueller, and H. Ramaprasad, "Predictable task migration for locked caches in multi-core systems," in *Proceedings of the 2011 SIGPLAN/SIGBED conference on Languages, compilers and tools for embedded systems*, 2011, pp. 131–140.

[121] T. Megel, M. Jan, V. David, C. Fraboul, and I. INP-ENSEEIHT, "Evaluation of task migration mechanisms for hard real-time distributed systems." in *RTNS*, 2011, pp. 159–168.

[122] P. Munk, B. Saballus, J. Richling, and H.-U. Heiss, "Position paper: Real-time task migration on many-core processors," in *ARCS 2015-The 28th International Conference on Architecture of Computing Systems. Proceedings.* VDE, 2015, pp. 1–4.

[123] B. Pourmohseni, S. Wildermann, M. Glaß, and J. Teich, "Hard real-time application mapping reconfiguration for noc-based many-core systems," *Real-Time Systems*, vol. 55, no. 2, pp. 433–469, 2019.

[124] B. Pourmohseni, F. Smirnov, S. Wildermann, and J. Teich, "Real-time task migration for dynamic resource management in many-core systems," in *Workshop on Next Generation Real-Time Embedded Systems (NG-RES 2020).* Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2020, pp. 5–1.

[125] https://www.ecrts.org/forum/download/RealWorldAutomotiveBenchmarksForFree-ECRTS-WATERS2015.pdf, [Accessed 12-12-2025].

[126] A. Hamann, D. Ziegenbein, S. Kramer, and M. Lukasiewycz, "Demonstration of the fmtv 2016 timing verification challenge," in *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2016, p. 1.

[127] A. Hamann, D. Dasari, S. Kramer, M. Pressler, F. Wurst, and D. Ziegenbein, "Waters industrial challenge 2017," in *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2017.

[128] M. Günzel, H. Teper, K.-H. Chen, G. von der Brüggen, and J.-J. Chen, "On the equivalence of maximum reaction time and maximum data age for cause-effect chains," in *35th Euromicro Conference on Real-Time Systems (ECRTS 2023).* Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2023.

[129] W. Weibull, "A statistical theory of strength of materials," *IVB-Handl.*, 1939.

[130] A. Davare, Q. Zhu, M. Di Natale, C. Pinello, S. Kanajan, and A. Sangiovanni-Vincentelli, "Period optimization for hard real-time distributed automotive systems," in *Proceedings of the 44th annual Design Automation Conference*, 2007, pp. 278–283.

[131] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte, "Synthesizing job-level dependencies for automotive multi-rate effect chains," in *2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA).* IEEE, 2016.

[132] ——, "End-to-end timing analysis of cause-effect chains in automotive embedded systems," *Journal of Systems Architecture*, vol. 80, pp. 104–113, 2017.

[133] T. Kloda, A. Bertout, and Y. Sorel, "Latency analysis for data chains of real-time periodic tasks," in *2018 IEEE 23rd international conference on emerging technologies and factory automation (ETFA)*, vol. 1. IEEE, 2018, pp. 360–367.

[134] J. Martinez, I. Sanudo, P. Burgio, and M. Bertogna, "End-to-end latency characterization of implicit and let communication models," in *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS), Dubrovnik, Croatia*, 2017.

[135] Y. Tang, N. Guan, X. Jiang, Z. Dong, and W. Yi, "Reaction time analysis of event-triggered processing chains with data refreshing," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[136] T. Klaus, F. Franzmann, M. Becker, and P. Ulbrich, "Data propagation delay constraints in multi-rate systems: Deadlines vs. job-level dependencies," in *Proceedings of the 26th International Conference on Real-Time Networks and Systems*, 2018, pp. 93–103.

[137] F. Scheler and W. Schröder-Preikschat, "The rtsc: Leveraging the migration from event-triggered to time-triggered systems," in *2010 13th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*. IEEE, 2010, pp. 34–41.

[138] J. Schlatow, M. Mostl, S. Tobuschat, T. Ishigooka, and R. Ernst, "Data-age analysis and optimisation for cause-effect chains in automotive control systems," in *2018 IEEE 13th international symposium on industrial embedded systems (SIES)*. IEEE, 2018, pp. 1–9.

[139] A. Biondi, P. Pazzaglia, A. Balsini, and M. Di Natale, "Logical execution time implementation and memory optimization issues in autosar applications for multicores," in *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2017.

[140] A. Biondi and M. Di Natale, "Achieving predictable multicore execution of automotive applications using the let paradigm," in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2018, pp. 240–250.

[141] http://www.erika.tuxfamily.org/drupal/, [Accessed 23-02-2026].

[142] P. Pazzaglia, A. Biondi, and M. Di Natale, "Optimizing the functional deployment on multicore platforms with logical execution time," in *2019 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2019, pp. 207–219.

[143] T. Kloda, B. d'Ausbourg, and L. Santinelli, " ," in *12th International Workshop Quantitative Aspects of Programming Languages and Systems-at ETAPS 2014*, 2014.

[144] W. Pree, J. Templ, P. Hintenaus, A. Naderlinger, and J. Pletzer, "Tdl-steps beyond giotto: A case for automated software construction." *Int. J. Softw. Informatics*, vol. 5, no. 1-2, pp. 335–354, 2011.

[145] M. Dürr, G. V. D. Brüggen, K.-H. Chen, and J.-J. Chen, "End-to-end timing analysis of sporadic cause-effect chains in distributed systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 5s, pp. 1–24, 2019.

[146] R. Ernst, L. Ahrendts, and K.-B. Gemlau, "System level let: Mastering cause-effect chains in distributed systems," in *IECON 2018-44th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2018, pp. 4084–4089.

[147] M. Günzel, K.-H. Chen, N. Ueter, G. von der Brüggen, M. Dürr, and J.-J. Chen, "Timing analysis of asynchronized distributed cause-effect chains," in *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2021, pp. 40–52.

[148] A. Kordon and N. Tang, "Evaluation of the age latency of a real-time communicating system using the let paradigm," in *ECRTS 2020*, vol. 165. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2020.

[149] J. Martinez, I. Sañudo, and M. Bertogna, "Analytical characterization of end-to-end communication delays with logical execution time," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2244–2254, 2018.

[150] C. Bradatsch, F. Kluge, and T. Ungerer, "Data age diminution in the logical execution time model," in *International conference on Architecture of computing systems*. Springer, 2016, pp. 173–184.

[151] S. Wang, D. Li, A. H. Sifat, S.-Y. Huang, X. Deng, C. Jung, R. Williams, and H. Zeng, "Optimizing logical execution time model for both determinism and low latency," in *2024 IEEE 30th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2024.

[152] M. Günzel and M. Becker, "Optimal task phasing for end-to-end latency in harmonic and semi-harmonic automotive systems," in *2025 IEEE 31th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2025.

[153] M. Günzel, K.-H. Chen, N. Ueter, G. v. d. Brüggen, M. Dürr, and J.-J. Chen, "Compositional timing analysis of asynchronized distributed cause-effect chains," *ACM Transactions on Embedded Computing Systems*, vol. 22, no. 4, pp. 1–34, 2023.

[154] M. I. Alkoudsi, S. Baruah, P. Ekberg, G. Fohler, J. Goossens, and M. Moslehi, "Extending periodic task sets with sporadic tasks: Computational complexity and exact feasibility tests," in *Proceedings of the 33rd International Conference on Real-Time Networks and Systems (RTNS 2025)*, Pisa, Italy, 2025.

[155] R. Pellizzoni and G. Lipari, "Feasibility analysis of real-time periodic tasks with offsets," *Real-Time Systems*, vol. 30, no. 1, pp. 105–128, 2005.

[156] S. Baruah, R. Howell, and L. Rosier, "Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor," *Real-Time Systems: The International Journal of Time-Critical Computing*, vol. 2, pp. 301–324, 1990.

[157] K.-B. Gemlau, L. Köhler, R. Ernst, and S. Quinton, "System-level logical execution time: Augmenting the logical execution time paradigm for distributed real-time automotive software," *ACM Transactions on Cyber-Physical Systems*, vol. 5, no. 2, pp. 1–27, 2021.

[158] J. Leung and M. Merrill, "A note on preemptive scheduling of periodic, real-time tasks," *Information processing letters*, vol. 11, no. 3, pp. 115–118, 1980.

[159] T. Klaus, M. Becker, W. Schröder-Preikschat, and P. Ulbrich, "Constrained data-age with job-level dependencies: How to reconcile tight bounds and overheads," in *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2021, pp. 66–79.

[160] A. Syed and G. Fohler, "Efficient offline scheduling of task-sets with complex constraints on large distributed time-triggered systems," *Real-Time Systems*, vol. 55, pp. 209–247, 2019.

[161] M. Maggio, A. Hamann, E. Mayer-John, and D. Ziegenbein, "Control-System Stability Under Consecutive Deadline Misses Constraints," in *32nd Euromicro Conference on Real-Time Systems (ECRTS 2020)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 165. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

[162] R. Dobrin, G. Fohler, and P. Puschner, "Translating off-line schedules into task attributes for fixed priority scheduling," in *Proceedings 22nd IEEE Real-Time Systems Symposium (RTSS 2001)(Cat. No. 01PR1420)*. IEEE, 2001, pp. 225–234.

[163] L. Maia, M. Günzel, and G. Fohler, "Safe reconfiguration of LET communication intervals to reduce end-to-end latencies," in *Proceedings of the 33rd International Conference on Real-Time Networks and Systems (RTNS 2025)*, 2025.

[164] L. Maia and G. Fohler, "Reducing end-to-end latencies of multi-rate cause-effect chains in safety critical embedded systems," in *12th European Congress on Embedded Real Time Software and Systems (ERTS 2024)*, 2024.

[165] L. Maia, M. I. Alkoudsi, and G. Fohler, "Ensuring end-to-end latencies of multi-rate cause-effect chains in multi-mode systems through migration," in *13th European Congress of Embedded Real Time Systems (ERTS 2026)*, 2026.

[166] L. Maia and G. Fohler, "Decreasing utilization of systems with multi-rate cause-effect chains while reducing end-to-end latencies," in *2025 28th International Symposium on Real-Time Distributed Computing (ISORC 2025)*. IEEE, 2025, pp. 1–10.

Glossary

Adaptive Cruise Control

It is a type of advanced driver-assistance system for automotive vehicles. The system automatically adjusts the vehicle's speed to maintain a safe distance from the vehicles ahead. 4

Anti-lock Breaking System

Safety anti-skid braking system used on aircraft and on land vehicles, such as cars, motorcycles, trucks, and buses. 2

Automotive Open System Architecture

Standard in the automotive domin to develop control software. 2–6, 28, 36, 45, 47, 51, 56, 125, 127

Best-Case Execution Time

Execution time of a task in the best-case scenario, e.g., no contention, blocking time or intereference. xix, 16, 40, 41

Cause-Effect Chain

Chained propagation of data from one task to another. xix, 8, 9, 11–13, 35–38, 41–53, 55, 56, 58, 60–62, 64–77, 79, 80, 82, 84–86, 88–95, 99–101, 104, 107–114, 117–119, 121–123, 125–128

Commercially Available Off-the-shelf

Hardware or software products, which are adapted aftermarket to the needs of the purchasing organization, rather than the commissioning of custom-made, or bespoke, solutions. 21

Controller Area Network

Vehicle bus standard designed to enable efficient communication primarily between electronic control units (ECUs). 3

Deadline-Monotonic

Task-level fixed-priority scheduling algorithm. The shorter the deadline of a task, the higher its priority. xvii, 20, 23, 25, 103, 130

Deadline-Monotonic with Priority Migration

Semipartitioned scheduling algorithm for multi-core systems. 25

Dynamic Buffering Protocol

Communication protocol. 28

Earliest Deadline First

Job-level fixed-priority scheduling algorithm. The closest the deadline of a task, the higher its priority. xvii, 20, 21, 23–27, 29, 86, 93, 103, 112, 131

Earliest Deadline First with window-constrained migration

Semipartitioned scheduling algorithm for multi-core systems. 25

EDF with task splitting and k processors in a group

Semipartitioned scheduling algorithm for multi-core systems. 24, 25

Electronic Control Unit

Electronic device responsible for controlling a specific funcionallity of the vehicle. 1–4, 8, 49

End-to-End

Time interval between source and destination. 1–4, 7–13, 35–37, 42, 43, 46, 49–52, 56, 61, 64, 65, 67, 69, 72, 74–77, 80, 84–90, 93, 95, 97–101, 104, 106, 111–114, 116, 121, 125–127

Event-Triggered

Task activation paradigm in which the release of a task is triggered by an event. 15–17, 47

Finite-State Machine

Represents a mathematical model of computation where an abstract machine that can be in exactly one of a finite number of states at any given time. 5

Hyperperiod

Interval of time in which a given schedule start to repeat itself. It has a length equal to the LCM of all tasks present in the system. 18, 28, 58, 86, 88, 90–92, 98

Integer Linear Program

Integer Linear Program is a mathematical optimization or feasibility program in which some decision variables are not discrete. 76

Least Common Multiple

Defines the least common multiple of a number. 18

Logical Execution Time

Time and data-flow deterministic inter-task communication model where tasks only communicate with each other at the boundaries of their communication interval. 2, 7–14, 28, 36, 38, 42, 47–54, 56, 57, 60–65, 72, 74, 77, 80, 81, 85, 87, 89, 93, 94, 103–107, 109, 111, 112, 114–117, 119, 121, 125–127, 157–159, 162–164

Maximum Data Age

Maximum interval of time that an input can affect the output of a cause-effect chain (CEC). 45, 50–52, 55, 61, 62, 65, 67, 69, 71, 74, 75, 77, 89, 104, 105, 107, 109, 110, 114, 115, 117, 119, 126, 127

Maximum Reaction Time

Maximum time taken by a cause-effect chain (CEC) to produce an output based on the asynchronous arrival of an input. 45, 47–52, 55, 61, 65, 67, 69, 70, 74, 75, 77, 87, 89, 104, 105, 109, 110, 114, 115, 119, 126, 127

Maximum Reduced Reaction Time

Maximal interval of time that an input can affect the output of a cause-effect chain (CEC) assuming the reduced output interval. 46–49, 52

Maximum Reduced Reaction Time

Maximum time interval taken by a cause-effect chain (CEC) to produce an output based on the reduced input interval. 46

Mixed Integer Linear Program

Mixed Integer Linear Program is a mathematical optimization or feasibility program in which some decision variables are not discrete. 48

Network-on-Chip

Network-based communications subsystem on an integrated circuit, normally used to connect different modules in a system on a chip (SoC). 34

Operating System

Software at the system level that manages computer hardware and software resources, and provides common services for computer programs. 3–5, 15, 16, 18, 22, 28, 36

Point-To-Point

Communication protocol. 28

Priority Ceiling Protocol

Method for eliminating unbounded priority inversion. 26, 27

Priority Inheritance Protocol

Method for eliminating unbounded priority inversion. 26

Processor Demand Analysis

Sufficient schedulability tests for task sets scheduled according to the EDF scheduling policy. 93

Rate-Monotonic

Task-level fixed-priority scheduling algorithm. The shorter the period of a task, the higher its priority. xvii, 19–23, 49, 129

Read or Write

Read or write operation. 6, 7, 28

Real-Time

Real-Time. 4, 8, 15–18, 22, 29, 31, 33–37, 42, 43, 50, 53, 125, 127, 128

Real-Time Operating System

An operating system for real-time applications. 33, 48

Real-Time System

Real-Time Systems are hardware or software systems in which the correctness of their results is subject to predefined timeliness constraints. 1

Reduced Data Age

Interval of time that an input can affect the output of a cause-effect chain (CEC) assuming the reduced output interval. 46

Reduced Reaction Time

Time taken by a cause-effect chain (CEC) to produce an output based on the reduced input interval. 46

Response-Time Analysis

A timing analysis method for verifying the response time of a task in a real-time system. 19, 43

Satisfiability Modulo Theories

satisfiability modulo theories consistis of the problem of determining whether a mathematical formula is satisfiable or not. 76

Software Component

General structure of a automotive application according to the AUTOSAR standard. 3, 36, 95

Stack Resource Policy

Method for eliminating unbounded priority inversion. 26, 27

Static Buffering Protocol

Communication protocol. 28

System Level LET

An inplmentation of the LET communication paradigm at the system level, allowing time and data-flow dterminism among multiple electronic control units in a a distributed system. 49

Telematic Control Unit

embedded system that wirelessly connects the vehicle to cloud services or other vehicles. 2

Time-Triggered

Task activation paradigm in which the release of a task is triggered according to time. 15–17, 29, 47

Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems

International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems. 36, 42, 50, 72, 103, 104

Worst-Case Execution Time

Execution time of a task in the worst-case scenario, e.g., maximum contention, blocking time, intereference. xix, 16, 25, 32, 36, 40, 41, 53, 57, 59, 89, 95, 104, 112, 129–131

Worst-Case Response Time

Describe the worst-case response time of a task among all of its jobs. 36, 43, 47–49, 53, 103

Summary

The continuous demand for more powerful, safe and efficient safety-critical real-time systems in different sectors of the automotive and avionic industry have been affecting how new control applications are modeled and developed by system designers in the past decades. As a consequence, traditional timing analysis methods commonly used for the verification of individual real-time tasks started to be insufficient due to the continuous increasing in system complexity and inter-task communication dependencies. The shift from single-core to multi-core architecture added an extra complexity layer to the problem of verifying timing constraints, since a multi-core architecture allows tasks to execute in parallel and issue memory requests concurrently, which can lead to unpredictable contention at the bus and/or memory level. Since in the worst-case scenario, contention can result in the deadline miss of a task and the consequent failure of the system, it is pivotal for safety-critical real-time systems to mitigate or bound the amount of contention suffered by tasks during runtime.

As multiple tasks from different applications have to constantly communicate with each other to achieve safety-critical system functionalities in modern automotive and avionic systems, it became necessary to shift the focus of existing timing analysis methods. Rather than focusing on the time required for the execution of an individual task, it became necessary to measure and verify the amount of time taken to propagate data through an ordered sequence of communicating tasks known in the literature as cause-effect chains. For those chained tasks, inter-task communication occurs by means of shared resources, with the output produced by one task serving as input for the next one. Since access time to the shared resources depends on when task execute, the problem of verifying this end-to-end latency in data propagation became a non-trivial problem, specially when tasks with different activation periods are part of the same cause-effect chain (multi-rate cause-effect chain).

During the last decade, multiple authors proposed different methods considering specific communication paradigms. The Logical Execution Time (LET) paradigm, originally proposed as part of a time triggered programming language, gained attention by the automotive industry due to its timing and data-flow deterministic properties. By defining inter-task communication points, the LET paradigm allows system designers to trace how data propagates through the multi-rate cause-effect chains, which in turn allows them to compute precisely different end-to-end latency metrics of the cause-effect

chains. However, by abstracting system semantics (scheduling choices) to ease timing analysis, the end-to-end latencies obtained by the LET paradigm are much more pessimistic, i.e., longer than other available communication paradigms such as the implicit communication paradigm.

Throughout this dissertation, we study the challenges and propose solutions to the problem of performing end-to-end timing analysis in safety-critical real-time systems with multi-rate cause-effect chains and multi-core architecture. We propose multiple contributions in this dissertation, each one of them focusing on a different aspect of the cause-effect chain such as: **(i)** latency metrics, **(ii)** system utilization, **(iii)** end-to-end latency feasibility in systems with multiple execution modes. All the main contributions proposed in this dissertation focus on safety-critical real-time systems applying the LET communication paradigm. Specifically, all the contributions are based on the idea of safely reconfiguring the communication intervals of tasks applying the LET paradigm.

Below, we summarize the main contributions of this dissertation:

1. we propose a method to reduce end-to-end latencies of multi-rate cause-effect chains applying the LET paradigm by considering knowledge of the schedule in later design phases

2. we propose a method to further reconfigure the communication intervals of tasks applying the LET paradigm by establishing precedence constraints between specific task instances using a heuristic function

3. we propose a method to decrease system utilization by skipping the execution of task instances that don't affect end-to-end latencies of the cause-effect chains present in the system

4. we propose a method to increase the feasibility of multi-rate cause-effect chains to meet their end-to-end latency constraints in multi-execution mode systems

5. we present an evaluation of all our proposed methods using an automotive benchmark and synthetic task sets shows in order to show their benefits under different end-to-end latency metrics such as reaction time and data age

Chapter I

In this chapter, we present a brief introduction of the research area explored in this dissertation. In addition, in this introductory chapter, we discuss the main problems related to the timing analysis and end-to-end latencies of multi-rate cause-effect chains in safety-critical real-time systems. We conclude this first chapter by presenting our main contributions and a brief overview of the remaining chapters of the dissertation.

Chapter II

In Chapter II, we present the concepts and scheduling theory of single/multi-core real-time systems as well as the task models most used in the literature. We also present the concepts and properties of shared resources, offline scheduling approaches, as well as load balancing techniques such as task migration.

Chapter III

In Chapter III, we do literature review in the broaden area of timing analysis of multi-core safety-critical real-time system. We discuss about the current state of the art and position our work with respect to it. In Chapter III, we also present the challenge that inspired most of the methods proposed in this dissertation.

Chapter IV

In Chapter IV, we present the main contributions of this dissertation. In Section IV.2, we present of our Schedule-Aware communication model. By extracting information from a feasible schedule, our method safely derives new boundaries for the communication intervals of tasks applying the LET paradigm, which in turn allows data to propagate through different propagation paths resulting in reduced end-to-end latency values. Since our Schedule-Aware model reconfigures the communication intervals of tasks, the methods available in the literature to compute the end-to-end lantencies of multi-rate cause-effect chains with tasks applying the LET model are no longer applicable. Therefore, in Section IV.3, we propose a new set of equations and derive a new analytical method to compute the end-to-end latencies of multi-rate cause-effect chains applying our Schedule-Aware model. In Section IV.4, we show that the end-to-end latencies of multi-rate cause-effect chains can be further reduced by adding precedence constraints between specific task instances in order to manipulate task's communication intervals. In Section IV.5, we further analyze the deterministic proprieties of the LET paradigm. More precisely, we investigate its data-flow determinism property and show how system utilization can be reduced by skipping the execution of tasks instances that don't affect the end-to-end latencies of the chains present in the system. In Section IV.6, we expand our analysis to systems with multiple execution modes. We propose a method to increase the feasibility of multi-rate cause-effect chains that apply the LET model to meet their end-to-end latency requirements after a change in the execution mode. By analyzing the extra interference caused by tasks and(or) cause-effect chains entering the system after the mode change, our method migrates specific task instances in order to be able to reconfigure tasks' communication intervals such that the end-to-end latency requirements are respected. Moreover, in Section IV.7, we propose a framework to assist designers on how to properly configure the communication intervals of LET tasks. In Section IV.8, we propose heuristic functions for our framework.

Chapter V

In Chapter V, we evaluate our work using an automotive benchmark as well as synthetic task sets. We show the benefits of our method under different end-to-end latency metrics and system load depending on the method being evaluated.

Chapter VI

In Chapter VI, we conclude our work and present possible research directions to continue our work.

Zusammenfassung

Zeitliche Analyse von Multi-Rate Cause-Effect Chains in sicherheitskritischen Echtzeitsystemen

Die stetige Nachfrage nach leistungsfähigeren, sichereren und effizienteren sicherheitskritischen Echtzeitsystemen in verschiedenen Bereichen der Automobil- und Luftfahrtindustrie hat in den letzten Jahrzehnten Einfluss darauf genommen, wie neue Steuerungsanwendungen von Systementwicklern modelliert und entwickelt werden. Infolgedessen reichten die traditionellen Methoden der Timing-Analyse, die üblicherweise zur Verifikation einzelner Echtzeit Tasks eingesetzt wurden, aufgrund der stetig zunehmenden Systemkomplexität und der Kommunikationsabhängigkeiten zwischen den Tasks nicht mehr aus. Der Übergang von Single-Core- zu Multi-Core-Architekturen fügte dem Problem der Verifizierung von Timing-Beschränkungen eine zusätzliche Komplexitätsebene hinzu, da eine Multi-Core-Architektur die parallele Ausführung von Tasks und die gleichzeitige Abgabe von Speicheranforderungen ermöglicht, was zu unvorhersehbaren Konflikten auf Bus- und/oder Speicherebene führen kann. Da Konflikte im schlimmsten Fall dazu führen können, dass eine Task ihre deadline verpasst und das System dadurch ausfällt, ist es für sicherheitskritische Echtzeitsysteme von entscheidender Bedeutung, das Ausmaß der Konflikte, denen Tasks während der Laufzeit ausgesetzt sind, zu mindern oder zu begrenzen.

Da in modernen Automobil- und Avioniksystemen mehrere Tasks aus verschiedenen Anwendungen ständig miteinander kommunizieren müssen, um sicherheitskritische Systemfunktionen zu gewährleisten, wurde es notwendig, den Schwerpunkt bestehender Timing-Analyse-Methoden zu verlagern. Anstatt sich auf die für die Ausführung einzelner Tasks benötigte Zeit zu konzentrieren, wurde es notwendig, die Zeit zu messen und zu verifizieren, die für die Weitergabe von Daten durch diese geordnete Abfolge kommunizierender Tasks benötigt wird, die in der Literatur als cause-effect-chain bezeichnet wird. Bei diesen verketteten Tasks erfolgt die Kommunikation zwischen den Tasks über gemeinsam genutzte Ressourcen, wobei die von einer Task erzeugte Ausgabe als Eingabe für die nächste dient. Da die Zugriffszeit auf die gemeinsam genutzten Ressourcen davon abhängt, wann die Tasks ausgeführt werden, wurde die Verifizierung dieser End-zu-End-Latenz bei der Datenübertragung zu einem nicht trivialen Problem, insbesondere wenn

Tasks mit unterschiedlichen Activation periods Teil derselben cause-effect-chain sind (multi-rate-cause-effect-chain).

Im Laufe des letzten Jahrzehnts haben zahlreiche Autoren verschiedene Methoden vorgeschlagen, die spezifische Kommunikationsparadigmen berücksichtigen. Das Logical Execution Time (LET)-Paradigma, das ursprünglich als Teil einer zeitgesteuerten Programmiersprache vorgeschlagen wurde, fand aufgrund seiner deterministischen Eigenschaften hinsichtlich Zeitsteuerung und Datenfluss Beachtung in der Automobilindustrie. Durch die Definition von Kommunikationspunkten zwischen Tasks ermöglicht das LET-Paradigma Systementwicklern, die Ausbreitung von Daten durch die multi-rate-cause-effect-chain nachzuverfolgen, was ihnen wiederum erlaubt, verschiedene End-zu-End-Latenzmetriken der cause-effect-chain präzise zu berechnen. Durch die Abstraktion der Systemsemantik (Scheduling-Entscheidungen) zur Vereinfachung der Timing-Analyse fallen die durch das LET-Paradigma ermittelten End-zu-End-Latenzen jedoch deutlich pessimistischer aus, i.e., sie sind größer als bei anderen verfügbaren Kommunikationsparadigmen wie dem impliziten Kommunikationsparadigma.

In dieser Dissertation untersuchen wir die Herausforderungen und schlagen Lösungen für die Durchführung einer durchgängigen Zeitanalyse in sicherheitskritischen Echtzeitsystemen mit cause-effect-chains mit unterschiedlichen Activation periods und Multi-Core-Architektur vor. Wir schlagen in dieser Dissertation mehrere Beiträge vor, von denen sich jeder auf einen anderen Aspekt der cause-effect-chain konzentriert, wie zum Beispiel: **(i)** End-zu-End-Latenzenmetriken, **(ii)** Systemauslastung, **(iii)** Durchführbarkeit in Systemen mit mehreren Ausführungsmodi. Alle in dieser Dissertation vorgeschlagenen Hauptbeiträge konzentrieren sich auf sicherheitskritische Echtzeitsysteme, die das LET-Paradigma anwenden. Insbesondere basieren alle Beiträge auf der Idee, die Kommunikationsintervalle von Tasks unter Anwendung des LET-Paradigmas sicher neu zu konfigurieren.

Im Folgenden fassen wir die Hauptbeiträge dieser Dissertation zusammen:

1. Wir schlagen eine Methode vor, um die End-zu-End-Latenzen von cause-effect-chain mit mehreren Activation periods unter Anwendung des LET-Paradigmas zu reduzieren, indem wir das Wissen über den Zeitplan in späteren Entwurfsphasen berücksichtigen

2. Wir schlagen eine Methode vor, um die Kommunikationsintervalle von Tasks unter Anwendung des LET-Paradigmas weiter neu zu konfigurieren, indem wir mithilfe einer heuristischen Funktion Vorrangbedingungen zwischen bestimmten Task instances festlegen

3. Wir schlagen eine Methode vor, um die Systemauslastung zu verringern, indem die Ausführung von Task instances übersprungen wird, die keinen Einfluss auf die End-zu-End-Latenzen der im System vorhandenen cause-effect-chain haben

4. Wir stellen eine Methode vor, um die Feasibility von cause-effect-chain mit mehreren Activation periods zu erhöhen, damit diese ihre End-zu-End-Latenzbeschränkungen in Systemen mit Mehrfachausführung einhalten können

5. Wir präsentieren eine Evaluierung aller von uns vorgeschlagenen Methoden unter Verwendung eines Benchmark aus der Automobilindustrie und synthetischer Task sets, um deren Vorteile unter verschiedenen End-zu-End-Latenz metriken wie Reaktionszeit und Datenalter aufzuzeigen

Kapitel I

In diesem Kapitel geben wir eine kurze Einführung in das in dieser Dissertation untersuchte Forschungsgebiet. Darüber hinaus erörtern wir in diesem Einführungskapitel die zentralen Probleme im Zusammenhang mit der Zeitanalyse und den End-zu-End-Latenzen von multi-rate-cause-effect-chain in sicherheitskritischen Echtzeitsystemen. Wir schließen dieses erste Kapitel mit einer Darstellung unserer wichtigsten Beiträge und einem kurzen Überblick über die übrigen Kapitel der Dissertation ab.

Kapitel II

In Kapitel II stellen wir die Konzepte und die Scheduling-Theorie von Echtzeitsystemen mit einem oder mehreren Cores sowie die in der Literatur am häufigsten verwendeten Task modelle vor. Außerdem behandeln wir die Konzepte und Eigenschaften gemeinsam genutzter Ressourcen, Offline-Planungsansätze sowie Techniken zum Lastausgleich wie beispielsweise die Task migration.

Kapitel III

In Kapitel III führen wir eine Literaturrecherche im weiteren Bereich der Zeitanalyse von sicherheitskritischen Echtzeitsystemen mit mehreren Kernen durch. Wir erörtern den aktuellen Stand der Technik und ordnen unsere Arbeit in diesem Zusammenhang ein. In Kapitel III stellen wir zudem das Problem vor, die den größten Teil der in dieser Dissertation vorgeschlagenen Methode inspiriert hat.

Kapitel IV

In Kapitel IV stellen wir die wichtigsten Ergebnisse dieser Dissertation vor. In Abschnitt IV.2 stellen wir unser Schedule-Aware Kommunikationsmodell vor. Durch die Extraktion von Informationen aus einem realisierbaren Schedule leitet unsere Methode sicher neue Grenzen für die Kommunikationsintervalle von Tasks ab, die das LET-Paradigma anwenden, was wiederum ermöglicht, dass Daten über verschiedene Übertragungspfade weitergeleitet werden, was zu reduzierten End-zu-End-Latenzwerten führt. Da unser Schedule-Aware Modell die Kommunikationsintervalle von Tasks neu konfiguriert, sind die in der Literatur verfügbaren Methoden zur Berechnung der End-zu-End-Latenzen von multi-rate-cause-effect-chain mit Tasks, die das LET-Modell anwenden,

nicht mehr anwendbar. Daher schlagen wir in Abschnitt IV.3 neue Berechnungsmethoden vor und leiten eine neue analytische Methode zur Berechnung der End-zu-End-Latenzen von multi-rate-cause-effect-chain unter Anwendung unseres Schedule-Aware-Modells ab. In Abschnitt IV.4 zeigen wir, dass die End-zu-End-Latenzen von multi-rate-cause-effect-chain weiter reduziert werden können, indem Präzedenzbedingungen zwischen bestimmten Task instances hinzugefügt werden, um die Kommunikationsintervalle der Tasks zu manipulieren. In Abschnitt IV.5 analysieren wir die deterministischen Eigenschaften des LET-Paradigmas weiter. Genauer gesagt untersuchen wir die Eigenschaft des Datenfluss-Determinismus und zeigen, wie die Systemauslastung reduziert werden kann, indem die Ausführung von Task instances übersprungen wird, die keinen Einfluss auf die End-zu-End-Latenzen der im System vorhandenen Chains haben. In Abschnitt IV.6 erweitern wir unsere Analyse auf Systeme mit mehreren Ausführungsmodi. Wir schlagen eine Methode vor, um die Feasibility von multi-rate-cause-effect-chains zu erhöhen, die das LET-Modell anwenden, um ihre End-zu-End-Latenzanforderungen nach einer Änderung des Ausführungsmodus zu erfüllen. Durch die Analyse der zusätzlichen Interferenzen, die durch Tasks und/oder cause-effect-chain verursacht werden, die nach dem Moduswechsel in das System eintreten, migriert unsere Methode bestimmte Task instances, um die Kommunikationsintervalle der Tasks so neu zu konfigurieren, dass die End-zu-End-Latenzanforderungen eingehalten werden. Darüber hinaus schlagen wir in Abschnitt IV.7 ein Framework vor, das Designern dabei hilft, die Kommunikationsintervalle von LET-Tasks richtig zu konfigurieren. In Abschnitt IV.8 schlagen wir eine heuristische Funktion für unser Framework vor.

Kapitel V

In Kapitel V bewerten wir unsere Arbeit anhand der von BOSCH [11] vorgeschlagenen Benchmark für den Automobilbereich sowie anhand synthetischer Task sets. Wir zeigen die Vorteile unserer Methode mit verschiedenen End-zu-End-Latenzmetriken und bei unterschiedlicher Systemauslastung auf.

Kapitel VI

In Kapitel VI fassen wir die Ergebnisse unserer Arbeit zusammen und schlagen mögliche Forschungsrichtungen für künftige Arbeiten vor.

LUIZ GONZAGA NUNES MAIA NETO

M.Sc. Embedded Systems - Computer Engineer

@ maianeto@rptu.com Kaiserslautern, Germany

WORKING EXPERIENCE

Researcher

Technische Universität Kaiserslautern - Chair of Real-Time Systems

April 2020 – Ongoing Germany, Kaiserslautern

- Ongoing research and projects in the area of end-to-end latencies of multi-rate cause-effect chains in safety critical systems.
- Ongoing research and projects in the area of real-time networks and offline schedulers for time-triggered systems.
- Responsible for supervising students during their master's thesis on topics related to my research.
- Responsible for teaching lectures in the RTS1/RTS2/OS courses.

Research Assistant

Technische Universität Kaiserslautern - Chair of Real-Time Systems

May 2017 – February 2020 Germany, Kaiserslautern

- Complete development of an offline scheduler for time-triggered network on chip as part of a project with AIRBUS.
- Implementation of software to compute the worst-case response-time of virtual links in AFDX (Avionics Full-Duplex Switched Ethernet) networks using the Trajectory Approach method.
- Implementation of software to compute an upper bound for the backlog on each output buffer of AFDX networks with multiple priority traffic.
- Development of software to clock synchronize time stampers in a distributed system using LPC1837 boards as end systems.

Engineering Internship

EMBRAER

September 2015 – August 2016 Brazil, Sao Jose dos Campos

International company specialized in the manufacture of commercial, executive and military aircraft.

- Improvement of the tool that simulates the autopilot of the military aircraft KC-390.
- Assistance in development of the software that analyses the behavior of the KC-390 aircraft during the use of the maneuver Steady Heading Sideslip with Full Pedal.
- Development of a web environment for a real-time management and monitoring of the results of the flight test campaign of the military aircraft KC-390.

Engineering Internship

BOREALIS - Montana Space Grant Consortium

May 2014 – August 2014 USA, Bozeman

Aerospace research center funded by NASA and responsible for the development of systems that support the space study and exploration.

- Development of software to handle data transmission between the ground station and the space using long range RF modems and high-potency antennas.
- Development of software to transmit photos and videos captured by the HAB (High Altitude Balloon) from the space to the ground station.

ACADEMIC PUBLICATIONS

- **Luiz Maia, Ibrahim Alkoudsi, Gerhard Fohler. "Ensuring End-To-End Latencies of Multi-Rate Cause-Effect Chains in Multi-Mode Systems Through Migration"**
  13th European Congress on Embedded Real-Time Systems, ERTS26
- **Luiz Maia, Mario Gunzel, Gerhard Fohler. "Safe Reconfiguration of LET Communication Intervals to Reduce End-to-End Latencies**
  33rd International Conference on Real-Time Networks and Systems", RTNS25
- **Luiz Maia and Gerhard Fohler. "Decreasing Utilization of Systems with Multi-Rate Cause-Effect Chains While Reducing End-to-End Latencies"**
  28th IEEE International Symposium On Real-Time Distributed Computing, ISORC25
- **Luiz Maia and Gerhard Fohler. "Reducing End-to-End Latencies of Multi-Rate Cause-Effect Chains in Safety Critical Embedded Systems"**
  12th European Congress on Embedded Real-Time Software and Systems, ERTS24

EDUCATION

M.Sc. Electrical and Computer Engineering - Embedded Systems

Technische Universität Kaiserslautern - TUKL

April 2017 – March 2020

Thesis title: Offline Scheduler for Time-Triggered Network-On-Chip (Grade: 1.0 - maximum grade)

B.Sc. Computer Engineering

Universidade Federal de Itajuba - UNIFEI

March 2010 – August 2016

Special Merit Award: Best graduate student in the Computer Engineering Department in 2016.

Exchange Student

Montana State University - MSU

June 2013 – August 2014

Exchange student sponsored by the Brazilian government program "Ciencias Sem Fronteiras". Academic achievement: GPA 3.75 out of 4. Member of the honor list, fall semester 2013.

TEACHING ACHIEVEMENTS

- **1st Place - Best exercise lecturer**
  Summer Semmester 2025
- **3rd Place - Best exercise lecturer**
  Summer Semmester 2024
- **1st Place - Best exercise lecturer**
  Summer Semmester 2023
- **2nd Place - Best exercise lecturer**
  Summer Semmester 2022

SKILLS

Hard-working | Dedicated | Eye for detail

PROGRAMMING LANGUAGES

C | C++ | Embedded C | Python | Java | JavaScript | PHP

LANGUAGES

Portuguese
Native language

English
TOEFL iBT: 102/120 (October 2016)

German
Telc Deutsch A2: 57.5/60 (September 2025)

PRIZES

- **3rd Place - Programming Contest - Regional Stage**
  Contest organized by BCS (Brazilian Computer Society) in 2011
- **6th Place - Programming Contest - State Stage**
  Contest organized by Algar Telecom in 2012

SUPERVISED STUDENTS

Project
Vinayaka Kirani Srinatha
January 2025 – Ongoing

Improving the performance of our LET-Scheduler

Project
Aaryaa Shekhar
June 2024 – October 2024

Improving the timestamping capabilities of an TTEthernet network for end-to-end latency analysis

Master's Thesis
Katherine Sirois
April 2023 – February 2024

Investigating the Worst-Case Latencies of Rate-Constrained Traffic on a Time-Triggered Ethernet Network

Researhc Assitant
Priya
May 2023 – August 2024

Implementation of a C code for computing end-to-end time of cause-effect chains applying the implicit communication model