Mälardalen University Press Dissertations
No. 123

# FAULT-TOLERANCE STRATEGIES AND PROBABILISTIC GUARANTEES FOR REAL-TIME SYSTEMS

**Hüseyin Aysan**

**2012**


School of Innovation, Design and Engineering

ISBN 978-91-7485-076-5
ISSN 1651-4238
Printed by Mälardalen University, Västerås, Sweden

Mälardalen University Press Dissertations
No. 123

# FAULT-TOLERANCE STRATEGIES AND PROBABILISTIC GUARANTEES FOR REAL-TIME SYSTEMS

Hüseyin Aysan

Akademisk avhandling

som för avläggande av teknologie doktorsexamen i datavetenskap vid Akademin för innovation, design och teknik kommer att offentligen försvaras tisdagen den 19 juni 2012, 13.15 i Gamma, Mälardalens högskola, Västerås.

Fakultetsopponent: Professor Petru Eles, Linköping University, Department of Computer and Information Science


Akademin för innovation, design och teknik

Abstract

Ubiquitous deployment of embedded systems is having a substantial impact on our society, since they interact with our lives in many critical real-time applications. Typically, embedded systems used in safety or mission critical applications (e.g., aerospace, avionics, automotive or nuclear domains) work in harsh environments where they are exposed to frequent transient faults such as power supply jitter, network noise and radiation. They are also susceptible to errors originating from design and production faults. Hence, they have the design objective to maintain the properties of timeliness and functional correctness even under error occurrences.

Fault-tolerance plays a crucial role towards achieving dependability, and the fundamental requirement for the design of effective and efficient fault-tolerance mechanisms is a realistic and applicable model of potential faults and their manifestations. An important factor to be considered in this context is the random nature of faults and errors, which, if addressed in the timing analysis by assuming a rigid worst-case occurrence scenario, may lead to inaccurate results. It is also important that the power, weight, space and cost constraints of embedded systems are addressed by efficiently using the available resources for fault-tolerance.

This thesis presents a framework for designing predictably dependable embedded real-time systems by jointly addressing the timeliness and the reliability properties. It proposes a spectrum of fault-tolerance strategies particularly targeting embedded real-time systems. Efficient resource usage is attained by considering the diverse criticality levels of the systems' building blocks. The fault-tolerance strategies are complemented with the proposed probabilistic schedulability analysis techniques, which are based on a comprehensive stochastic fault and error model.

ISBN 978-91-7485-076-5
ISSN 1651-4238

To my *great*-uncle Himmet Atayol,
who has been a *great* inspiration.

# Abstract

Ubiquitous deployment of embedded systems is having a substantial impact on our society, since they interact with our lives in many critical real-time applications. Typically, embedded systems used in safety or mission critical applications (e.g., aerospace, avionics, automotive or nuclear domains) work in harsh environments where they are exposed to frequent transient faults such as power supply jitter, network noise and radiation. They are also susceptible to errors originating from design and production faults. Hence, they have the design objective to maintain the properties of timeliness and functional correctness even under error occurrences.

Fault-tolerance plays a crucial role towards achieving dependability, and the fundamental requirement for the design of effective and efficient fault-tolerance mechanisms is a realistic and applicable model of potential faults and their manifestations. An important factor to be considered in this context is the random nature of faults and errors, which, if addressed in the timing analysis by assuming a rigid worst-case occurrence scenario, may lead to inaccurate results. It is also important that the power, weight, space and cost constraints of embedded systems are addressed by efficiently using the available resources for fault-tolerance.

This thesis presents a framework for designing predictably dependable embedded real-time systems by jointly addressing the timeliness and the reliability properties. It proposes a spectrum of fault-tolerance strategies particularly targeting embedded real-time systems. Efficient resource usage is attained by considering the diverse criticality levels of the systems' building blocks. The fault-tolerance strategies are complemented with the proposed probabilistic schedulability analysis techniques, which are based on a comprehensive stochastic fault and error model.

# Acknowledgements

As this will be the most read part of the entire thesis, I would have liked to spend a lot of time individually thanking everyone who made this thesis possible. But, of course, I have just finished writing the other parts that are more crucial to get a PhD, with not so many hours left before the deadline for printing. What I have definitely *not* mastered during all these years as a PhD student is to meet the *deadlines* in a slightly more relaxed way. So please don't get mad if your name is missing here. I will get you a beer.

I would like to begin with expressing my sincere gratitude to my supervisors Sasikumar Punnekkat, Radu Dobrin and Hans Hansson for the guidance and support, all the encouragements, and all the work they put in.

I greatly appreciate all the constructive suggestions and advices Iain Bate has given during the past three years. I wish to thank Julián Proenza who also contributed to this work and provided feedback in extreme detail! I am also grateful to Abhilash Thekkilakattil, Barbara Gallina, Björn Lisper, Fredrik Ekstrand, Heinz Schmidt, Kateryna Mishchenko, Mikael Sjödin and Rolf Johansson for their cooperation, proof reading and feedback on this work.

Many thanks to Lars Asplund who lured me to stay in Sweden using his robots.

Very special thanks to Fredrik Ekstrand for being a great friend and being so fun, and funny!

I really enjoyed the long lunch sessions with the lunch gang, Aneta Vulgarakis, Bob, Juraj Feljan, Leo Hatvani, Séverine Sentilles and Svetlana Girs. Their company and my laziness prevented me from cooking lunches most of the time. I would like to thank my officemates, Abhilash Thekkilakattil, Adnan Causevic, Aida Causevic, Andreas Johnsen,

Jiale Zhou, Mikael Åsberg and Moris Behnam, for the good times we have had, despite the temperature wars, torturing me and my plants with extreme heat! I would like to thank Andreas Gustavsson, Antonio Cicchetti, Conny Collander and Ingrid Runnérus for their company during the tough training hours, Maria Lindén for the introduction to ice-skating, Dag Nyström and Séverine Sentilles, for revealing the best routes during mushroom hunting trips, Andreas Gustavsson, Juraj Feljan, Dag Nyström, Svetlana Girs and Christer Norström for the ski lessons. I would also like to thank many more people at the department, Andreas Hjertström, Baran Çürüklü, Barbara Gallina, Batu Akan, Carl Ahlberg, Cristina Seceleanu, Damir Isovic, Daniel Sundmark, Eduard Paul Enoiu, Farhang Nemati, Federico Ciccozzi, Frank Lüders, Giacomo Spampinato, Guillermo Rodriguez-Navas, Gunnar Widfors, Harriet Ekwall, Hongyu Pei Breivold, Ivica Crnkovic, Jagadish Suryadevara, Jan Carlson, Johan Fredriksson, Josip Maras, Jörgen Lidholm, Jukka Mäki-Turja, Kivanç Doğanay, Luka Lednicki, Malin Rosqvist, Markus Bohlin, Mohammad Ashjaei, Monica Wasell, Martin Ekström, Mehrdad Saadatmand, Mikael Ekström, Nikola Petrovic, Paul Pettersson, Peter Wallin, Rafia Inam, Raluca Marinescu, Saad Mubeen, Sara Afshar, Sara Dersten, Stig Larsson, Susanne Fronnå, Thomas Nolte, Tiberiu Seceleanu, Yue Lu and others for all the fun during coffee breaks, lunches, parties, conferences and PROGRESS trips!

Finally and most importantly, many thanks to Lena, Atena, Ilhan, Ümran, my parents and all my friends who all contributed a lot to my life during this period in many ways!

Hüseyin Aysan
Västerås, June, 2012

# List of Publications

The following is a list of publications that form the basis of the thesis (in reverse chronological order):

- **A Strategy for Achieving Reliability and Timing Guarantees using Temporal and Spatial Redundancy** Hüseyin Aysan, Radu Dobrin, Sasikumar Punnekkat, Iain Bate, *In submission.*

- **Schedulability Guarantees for Dependable Distributed Real-Time Systems under Error Bursts** Hüseyin Aysan, Radu Dobrin, Sasikumar Punnekkat, *To appear in Advances in Intelligent Control Systems and Computer Science, Springer, 2012.*

- **Probabilistic Scheduling Guarantees in Distributed Real-Time Systems under Error Bursts**, Hüseyin Aysan, Radu Dobrin, Sasikumar Punnekkat, Julián Proenza, *17th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, Poland, September, 2012.

- **On Voting Strategies for Loosely Synchronized Dependable Real-Time Systems**, Hüseyin Aysan, Radu Dobrin, Sasikumar Punnekkat, Iain Bate, *7th IEEE International Symposium on Industrial Embedded Systems, SIES*, Germany, June, 2012.

- **Probabilistic Schedulability Guarantees for Dependable Real-Time Systems under Error Bursts**, Hüseyin Aysan, Radu Dobrin, Sasikumar Punnekkat, Rolf Johansson, *8th IEEE International Conference on Embedded Software and Systems, ICESS*, China, November, 2011.

- **Task-Level Probabilistic Scheduling Guarantees for Dependable Real-Time Systems - A Designer Centric Approach**, Hüseyin Aysan, Radu Dobrin, Sasikumar Punnekkat, *2nd IEEE International Workshop on Object / component / service-oriented Real-Time Networked Ultra-dependable Systems, WORNUS*, U.S.A, March, 2011.

- **Efficient Fault Tolerant Scheduling on Controller Area Network (CAN)**, Hüseyin Aysan, Abhilash Thekkilakattil, Radu Dobrin, Sasikumar Punnekkat, *15th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, Spain, September, 2010.

- **New Strategies for Ensuring Time and Value Correctness in Dependable Real-Time Systems**, Hüseyin Aysan, Licentiate Thesis, Mälardalen University, Sweden, May, 2009.

- **Maximizing the Fault Tolerance Capability of Fixed Priority Schedules**, Radu Dobrin, Hüseyin Aysan, Sasikumar Punnekkat, *14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, Taiwan, August 2008.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Embedded systems are deployed ubiquitously in many critical applications that interact with our lives. Typically, the main requirement for those embedded systems used in safety or mission critical applications is to provide continuity of correct and timely service even under the effects of faults. For instance, in the automotive domain, the systems are often subjected to high degrees of Electromagnetic Interference (EMI) from the operational environment, which can potentially cause errors. The common causes of such interferences include cellular phones and other radio equipments inside the vehicle, electrical devices like switches and relays, radio transmissions from external sources, lightning in the environment, etc. Electromagnetic Compatibility (EMC) has been seriously considered by the automotive industry for more than 40 years, and several legislations and directives are in effect to tackle the EMI problem [75]. However, even today it is not possible to completely eliminate the effects of EMI since exact characterization of all such interferences defy comprehension. These systems are also susceptible to internal faults originating from the technological advances. For example, nano-level shrinking of electronic devices are making them highly susceptible to transient errors, or increased clock frequencies also increase the chance of a transient pulse getting latched thus affecting the logic parts as well [65].

Fault-tolerance plays a crucial role towards achieving dependability, and the fundamental requirement for the design of effective and efficient fault-tolerance mechanisms is a realistic and applicable model of poten-

tial faults, their manifestations and consequences. Moreover, systems' resources for providing fault-tolerance are often limited in the embedded systems domain, due to the underlying constraints such as space, weight and cost. Hence, the design process of developing fault-tolerant embedded systems involves the selection of appropriate fault-tolerance strategies to be used in critical parts of the system with the help of design and analysis tools.

## 1.1 Problem Statement

### 1.1.1 Fault and Error Modeling

The behaviour of errors which are caused by transient and intermittent faults can be very complex. The main reason behind this complexity is the random behaviour of these errors, which depend on several factors, such as the type and the severity of the fault to which the system is exposed, the resistance of the hardware to the fault, and the effectiveness of the fault detection and fault-tolerance mechanisms.

Majority of the earlier research efforts were based on a simplified error model assumption that only singleton errors can occur in the systems, and that they are separated at least by a known minimum inter-arrival time. However, error bursts of varying lengths are not uncommon and they may have an adverse effect on systems' timeliness. Hence the versatility and applicability of the existing models are limited, in the sense that they are incapable of representing complex error scenarios, thus potentially leading to inaccurate analysis results.

### 1.1.2 Fault-Tolerance Strategies

Design of safety and mission critical applications most often incorporates fault-tolerance in the form of spatial and temporal redundancy. Each approach has advantages and disadvantages over the other, in terms of cost, performance and error coverage, and the decision of which approach to choose mainly depends on the application and the environment in which it is deployed.

The conventional fault-tolerance strategies need to be extended and elaborated to include specific design elements that are crucial in the embedded real-time systems domain. Examples of these extensions include the detection and masking of errors in the time domain, and addressing

the mixed criticality of the various parts of the embedded systems to provide *efficient resource usage*.

### 1.1.3 Fault-tolerant Schedulability Analysis

Real-time scheduling research is typically based on worst-case assumptions, and providing timeliness guarantees under these assumptions. In cases where it is not possible to derive an absolute worst-case property, the research strives to find the possible bounds in order to keep the worst-case guarantees valid, although at a cost of somewhat pessimistic results. Similarly, the fault-tolerant scheduling research most often assumes worst-case fault rates (which hold for a certain probabilities), and develops schedulability analysis techniques based on these assumptions. However, such a bound may not exist for fault and error arrival rates, as these events are typically random by their nature. As a result, in many cases, the existing analysis techniques do not permit tuning the assumptions at a later stage, in order to adapt to the changing environments and system properties, thus limiting their applicability.

## 1.2 Publications

This following are the publications that the thesis is primarily based on.

- **Licentiate Thesis** The sections that present the background and the fault-tolerance strategies in this thesis include some of the work presented in the licentiate thesis [13, 30, 14].

- **Paper A** *On Voting Strategies for Loosely Synchronized Dependable Real-Time Systems*, Hüseyin Aysan, Radu Dobrin, Sasikumar Punnekkat, Iain Bate, 7th IEEE International Symposium on Industrial Embedded Systems, SIES, Germany, June, 2012.

  This paper presents our proposed majority voter (VTV) and and its evaluation.

- **Paper B** *Maximizing the Fault-Tolerance Capability of Fixed Priority Schedules*, Radu Dobrin, Hüseyin Aysan, Sasikumar Punnekkat, 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA, Taiwan, August, 2008.

This paper proposes a fault-tolerant scheduling technique, for task scheduling assuming a mixed criticality task set.

- **Paper C** *Efficient Fault-Tolerant Scheduling on Controller Area Network (CAN)*, Hüseyin Aysan, Abhilash Thekkilakattil, Radu Dobrin, Sasikumar Punnekkat, 15th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA, Spain, September, 2010.

  This paper proposes a fault-tolerant scheduling technique, for message scheduling on CAN assuming a message set with multiple criticality levels where criticality of messages are translated into fault-tolerance requirements.

- **Paper D** *Task-Level Probabilistic Scheduling Guarantees for Dependable Real-Time Systems - A Designer Centric Approach*, Hüseyin Aysan, Radu Dobrin, Sasikumar Punnekkat, 2nd IEEE International Workshop on Object/component/service-oriented Real-Time Networked Ultra-dependable Systems, WORNUS, U.S.A, March, 2011.

  This paper proposes a method which allows the system designer to specify task-level reliability requirements and provides a scheduling analysis to test if these requirements are met for each task in a mixed criticality task set.

- **Paper E** *Probabilistic Schedulability Guarantees for Dependable Real-Time Systems under Error Bursts*, Hüseyin Aysan, Radu Dobrin, Sasikumar Punnekkat, Rolf Johansson, 8th IEEE International Conference on Embedded Software and Systems, ICESS, China, November, 2011.

  This paper proposes a probabilistic schedulability analysis for task scheduling assuming a random burst error model.

- **Paper F** *Probabilistic Scheduling Guarantees in Distributed Real-Time Systems under Error Bursts*, Hüseyin Aysan, Radu Dobrin, Sasikumar Punnekkat, Julián Proenza, 17th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA, Poland, September, 2012.

  This paper proposes a probabilistic schedulability analysis for message scheduling assuming a random burst error model.

- **Paper G** *A Strategy for Achieving Reliability and Timing Guarantees using Temporal and Spatial Redundancy*, Hüseyin Aysan, Radu Dobrin, Sasikumar Punnekkat, Iain Bate, *In submission.*

  This paper proposes a cascading redundancy strategy combining the spatial redundancy and the temporal redundancy approaches and a corresponding reliability analysis for the proposed strategy.

**Contributors to these papers:** Hüseyin Aysan is the main contributor and the main author of the papers, with the exception of Paper B which is based on the original idea by Radu Dobrin. All these works have been performed under the supervision of Radu Dobrin and Sasikumar Punnekkat. The works in Paper A and Paper G have been carried out in close cooperation with Iain Bate. Julián Proenza contributed to Paper F with his extensive knowledge on CAN. Rolf Johansson contributed to Paper E with his expertise in safety-critical systems, and Abhilash Thekkilakattil performed the simulation studies in Paper C.

## 1.3 Thesis Contributions

The major contributions of this thesis are as follows:

- A majority voting strategy which performs voting in both the time and the value domains, applicable to loosely synchronized dependable real-time systems and its evaluation. This contribution was published in *Paper A* and contributes to the material in Chapter 3.

- A method for providing fault-tolerance using the temporal redundancy approach for scheduling task sets with mixed criticality levels and its evaluation. This contribution was published in *Paper B* and is presented in Chapter 3.

- A method for providing fault-tolerance using the temporal redundancy approach for scheduling message sets with mixed criticality levels and its evaluation. The error model assumes singleton errors, however the amount of fault-tolerance can be tuned by the designer for each message in the message set in terms of maximum

allowed re-transmission attempts. This contribution was published in *Paper C* and contributes to the material in Chapter 3.

- A probabilistic real-time analysis technique for fixed-priority task scheduling, where the designer can specify separate reliability requirements for each individual task and get analysis results for the overall schedulability, therefore enabling task-level tuning of resource allocations for fault-tolerance. Fault-tolerance level for each task can be analyzed individually by the designer and instead of specifying the number of maximum allowed re-executions, the designer can specify desired reliability levels for each task. This contribution was published in *Paper D* and contributes to the material in Chapter 4.

- A stochastic fault and error model capable of modeling errors with various characteristics, such as single errors vs error bursts, and multiple factors affecting the fault and error occurrence rates. This error model was published in *Papers E* and *Paper F* and contributes to the material in Chapter 4.

- A probabilistic real-time analysis technique for fixed-priority task scheduling, assuming the stochastic fault and error model. The error model assumes both singleton and burst errors. The maximum number of error bursts per task instance can be more than one, however error bursts are treated as continues blocks of errors. This contribution was published in *Paper E* and contributes to the material in Chapter 4.

- A probabilistic real-time analysis technique for message scheduling, assuming the stochastic fault and error model. The error model assumes both singleton and burst errors, with the possibility of specifying both the burst rate as well as the error rate within a burst. The maximum number of error burst for any message instance is limited to one. This contribution has been proposed in *Paper F* and contributes to the material in Chapter 4.

- A framework to combine the spatial and temporal redundancy approaches to bring out their synergetic effects. The spatial redundancy mechanism provides fault-tolerance by masking errors in both the time and the value domains with the usage of the real-time voting strategy VTV, presented in *Paper C*. It also works as

an error detector in certain scenarios where the total number of errors exceeds the masking capability of the spatial redundancy stage. In such scenarios, the temporal redundancy stage comes into operation to perform error recovery. The framework includes a joint response time and reliability analysis for both an ideal voter, whose *False Negative Rate (FNS)* and *False Positive Rate (FPS)* are zero, and real voters in case information regarding the real-world performance is available for the voter. This contribution has been proposed in *Paper G* and contributes to the material in Chapter 5.

## 1.4 Thesis Outline

The rest of the thesis starts with **Chapter 2** which introduces the basic terminology, concepts and theory regarding dependability, error modeling, real-time systems, real-time communication, real-time scheduling and real-time analysis.

Following the introduction and the background chapters, the conducted research is presented in three chapters:

- **Chapter 3** presents three fault-tolerance strategies in detail, one with spatial redundancy and two with temporal redundancy addressing mixed criticality levels of tasks and messages.
- **Chapter 4** presents three probabilistic schedulability analysis techniques. The first analysis technique addresses the scheduling of mixed criticality task sets and the other two analysis techniques consider the burst error model for task and message scheduling respectively.
- **Chapter 5** presents a cascading redundancy strategy combining the spatial redundancy and the temporal redundancy approaches and a corresponding reliability analysis for the proposed strategy.

Finally, **Chapter 6** concludes the thesis, and discusses the future work.

# Chapter 2

# Background

This chapter presents the theoretical background for this thesis, and begins with introducing two fundamental system properties, viz., dependability and timeliness, which are crucial properties for the majority of the embedded real-time systems. The chapter continues with the description of the commonly used error modeling approaches that form the basis for various fault-tolerance strategies. Then, the real-time scheduling policies, relevant to the thesis, used for task and message scheduling are presented, followed by the description of the commonly used real-time analysis techniques. Controller Area Network (CAN), which is a communication protocol commonly used in the embedded real-time systems domain, is presented in detail since it will be used throughout the thesis.

## 2.1 Dependability

This section presents the widely used and accepted basic concepts of dependability and the related terminology, proposed by Laprie et al. [10, 57]. *Dependability* of a system is the ability to provide services that can be justifiably trusted by its users. A systematic representation of the dependability concept can be done as shown in Figure 2.1 where the main components are the *threats* to dependability, *attributes* of dependability and the *means* to achieve dependability. The threats, attributes and the means mainly focused in this thesis are marked in this figure.

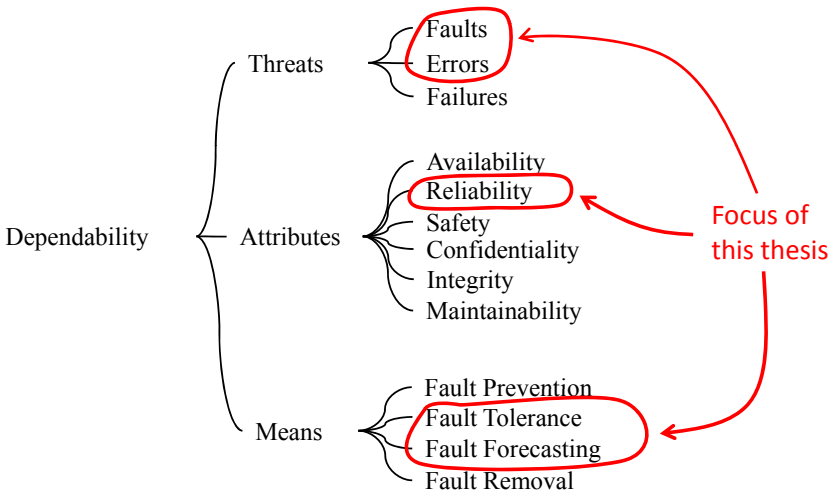


Figure 2.1: The dependability tree [10]

### 2.1.1 Failures, Errors and Faults

A system *failure* is the deviation of its delivered service from the specified service, therefore threatening the confidence of the system to deliver a service that can be trusted. The characteristics of systems with respect to the *controllability* and the *severity* of their failure behaviour can be described in terms of *failure modes*. A major challenge in the design and development of systems is the assurance that the system failures occur only in the specified modes, i.e. controllability of the potential failures [10]. Examples of failure modes, presented in [10, 63, 76], include the following:

**Fail-safe:** A system whose failures do not result in severe consequences is a fail-safe system.

**Fail-stop:** A system that omits producing any outputs upon a failure and continues to stay in this mode until restarted is a fail-stop system. Typically, fail-stop systems signal the failures to its users with a warning signal.

**Crash failure:** A system that fails in a crash failure mode omits producing any outputs upon a failure and continues to stay in this mode until restarted. Differently from fail-stop mode, the failure may remain undetected.

An *error* is a system state that may lead to a system failure through propagations, i.e., state transformations. Errors can be classified based on several aspects, such as, domain, persistence, consistency, homogeneity, impact and criticality [16, 80, 99, 53] as shown in Figure 2.2. The

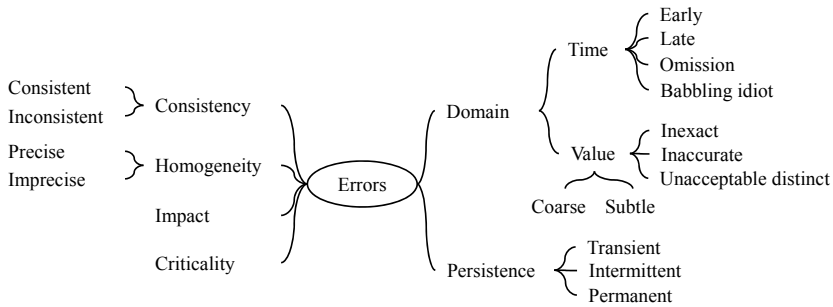


Figure 2.2: Error classification

domain and consistency properties may determine the types of error handling mechanisms to be used. The other properties describe with what frequency the errors may occur, and once they occur, the probability of causing a system failure as well as the severity of the consequences of such failures. Hence, they may determine the appropriate locations for these mechanisms and the amount of resources that should be reserved for adequate handling of the expected errors. Modeling error scenarios, error transformations and error propagations, is a crucial step in the design of dependable real-time systems, in order to effectively and efficiently introduce mechanisms to prevent system failures.

A *fault* is the adjudged cause of an error. During an operation of a system, faults that may be transformed into errors are classified into two categories with respect to system boundaries, as *internal faults*, e.g., hardware faults, and *external faults*, e.g., electromagnetic interferences (EMI) [10, 57, 93, 78]. Faults can also be classified into three categories with respect to the level of persistence, as being *permanent*, e.g., a permanent damage in the hardware, *intermittent*, e.g., software design

faults that cause difficult to reproduce errors (*Heisenbugs*) [52, 43], or *transient*, e.g., EMI from mobile phones [93, 78]. The error rates caused by transient and intermittent faults are much higher than that caused by permanent faults [90, 79, 53, 48].

The relationship between faults, errors and failures is shown in Figure 2.3 where each arrow represents a relation of cause and effect [10]. A fault causes an error to happen in case it gets activated and an error causes a failure when it is propagated to the service interface of the system. This relation of cause and effect acts like a chain, i.e., a failure in one system causes a fault in another system that contains the failed system, or it causes an external fault for the systems that interact with it.


Figure 2.3: The chain of dependability threats [10]

### 2.1.2 Reliability, Availability and Safety

Reliability is the ability to continue delivering correct service, i.e., perform failure-free operation, for a specified period of time. Availability is the probability of being operational and able to deliver correct service at a given time [10]. These two concepts are often mixed with each other, however, they are different properties. A system that fails very frequently has a low reliability, but can still have very high availability provided that the recoveries are performed very quickly. Similarly, if a system breaks down very rarely, but the repair action takes a long time, its reliability is high, while its availability is low. Safety is the absence of catastrophic consequences of system failures on the user of the system or the environment in which the system is operating.

Though being different concepts, these attributes of dependability are closely connected. For example, if system reliability is improved, then its availability is improved as well (although the opposite case is not always true). In this thesis, we propose strategies for improving reliability of real-time systems which also has a positive effect on the safety and the availability for the stated reason.

### 2.1.3 Fault-Tolerance

Fault-tolerance is the set of measures and techniques that are used to enable continuity of correct service delivered by a system even in case of errors. Two essential steps for providing fault-tolerance are *error detection* and *error recovery*. Other optional measures include *fault diagnosis* and *fault isolation*, in order to prevent the faults from causing more errors.

There exist various types of error detection strategies targeting different types of errors. Examples are *timing checks* for timing errors, *reasonableness checks* for coarse value errors and *replica comparison* for subtle value errors. Each detection approach has a different resource requirement and error coverage, where resource requirement generally grows as the coverage increases. Apparently, this may not be a linear relation since the error coverage may consist of various error types which cannot be compared with each other.

Error recovery is the action to transform the system state into an error-free state. There are three main approaches to perform error recovery:

1. Backward error recovery is an approach which involves taking the system back to a state that was saved before the error has been detected. The saved state is called a *checkpoint*.

2. Forward error recovery is an approach which involves switching the system state to a state known to be error-free.

3. Compensation through redundancy is another error recovery approach which uses error-free replicas to compensate the error state. The redundancy can be achieved in the spatial domain by replicating the computing nodes, or in the temporal domain, by execution of recovery blocks [46], re-execution of the same actions or execution of alternate actions.

This thesis focuses on the third type of error recovery approach, viz., compensation through redundancy.

### 2.1.4 Fault-Forecasting

Fault-forecasting is the prediction of systems' ability to satisfy the dependability attributes. Along with fault-tolerance, it is one of the major

complementary techniques to attain dependability. The commonly used other techniques to attain dependability are *fault prevention* and *fault removal* [10] which are outside the scope of this thesis.

Fault-forecasting can either be done *qualitatively*, by identifying in which failure modes the system or parts of the system can fail, or *qualitatively*, by deriving the probabilities that the system meets its dependability requirements. This thesis predominantly focuses on the qualitative fault-forecasting techniques by the usage of probabilistic schedulability analysis techniques for real-time systems.

## 2.2 Modeling Faults and Errors

Errors occur with different probabilities depending on the operational environments and the characteristics of the systems, such as the internal design faults, production faults or the resistance to external faults. Hence, in order to efficiently allocate resources to fault-tolerance mechanisms or fault-forecasting techniques, error scenarios should be considered individually for each system. Depending on the purpose of the approach, whether it is a fault-tolerance mechanism, fault-forecasting technique or a combination of both, different error modeling techniques have been proposed, some of them being briefly described in the following subsections.

### 2.2.1 Category-based Fault and Error Models

Category-based fault and error modeling is mainly used in qualitative fault-forecasting techniques, that aim at identifying, classifying and ordering the event combinations that may lead to failures, such as Fault Tree Analysis (FTA) [101], Failure Mode and Effects Analysis (FMEA) [2], Fault Propagation and Transformation Notation (FPTN) [35], Failure Propagation and Transformation Calculus (FPTC) [98] and Failure Propagation and Transformation Analysis (FPTA) [39].

Examples of this type of models are Ezhilchelvan and Shrivastava's classification of faults [33, 34] where they describe the consistency aspect of faults, and various sorts of timing faults in detail, and Bondavalli and Simoncini's time/value based error classification [16] which is used to observe the capability of system users or error detection mechanisms to detect errors in each class.

### 2.2.2 Bounded Error Models

Many fault-tolerance techniques used in real-time research [64, 77, 83, 40, 22, 44, 54, 94] use either sporadic error models in which error occurrences are assumed to be separated at least by a stated minimum inter-arrival time, or assume a maximum number of error occurrences within a stated period to specify the bounds for worst-case error scenarios. The various ways for specifying the error bounds include:

- using a parameter of the task set, or a constant value to specify the *minimum inter-arrival times*, e.g., errors are separated by at least the largest period in the task set, or two times the largest worst-case execution time (WCET) in the task set
- using a parameter of each task, the whole task set, or a constant value to specify the period during which a stated *number of errors* are allowed to occur, e.g., maximum $n$ errors may occur during each task period, or during the least common multiple (LCM) of all the task periods

Recently, Many and Doose presented a bounded error model [69] where they modeled the complex behaviour of error bursts. Their model assumes a minimum inter-arrival between the faults causing errors and each fault has a bounded interval during which errors are allowed to occur (Figure 2.4). During this interval, no information is available regarding the error arrivals, since the intensity of the faults is not modeled. Outside the fault interval, no error is assumed to occur.

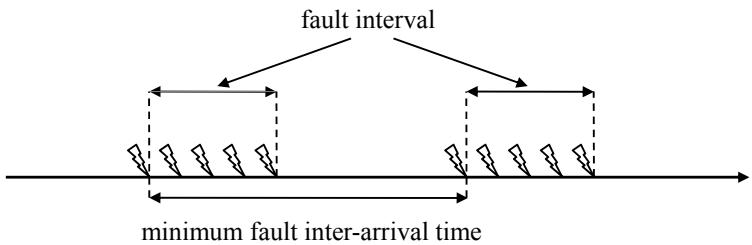


Figure 2.4: Burst error model by Many and Doose

A common issue of the fault-tolerance mechanisms that use this type of error models is the incapability of tuning the assumptions regard-

ing the error occurrence bounds, limiting their applicability in different contexts. For instance, if a more severe error scenario needs to be considered, with an increased number of recovery attempts, the fault-tolerance mechanism that has been implemented assuming the earlier worst-case scenario may not be able to provide any guarantees with the new assumptions. In an opposite case, if a system with a fault-tolerant (FT) scheduler has moved to an environment where there is a lower probability of errors, unnecessary amount of resources might have been allocated for fault-tolerance and the designer of the system may not have any idea regarding which resources to reclaim while keeping an adequate level of guarantees for the new environment.

### 2.2.3 Stochastic Error Models

The mentioned limitation of bounded error models is addressed in stochastic error models where error occurrence characteristics are modeled by random parameters, such as error occurrence rates and various time intervals. Error frequencies can be modeled without any particular bounds by using stochastic error models. Hence one can perform various sensitivity analyses to identify the error recovery thresholds, e.g., the maximum error rate at which the system can guarantee delivery of correct service.

Burns et al. [23] and Broster et al. [20, 19, 21] modeled error arrivals as a Homogeneous Poisson Process where the probability of exactly $n$ errors within an interval of $t$ is:

$$Pr_n(t) = \frac{e^{-\lambda t}(\lambda t)^n}{n!}$$

where $\lambda$ is the constant error arrival rate.

Navet et al. [73] modeled error arrivals similarly with a Poisson distribution, however, each error event is separately modeled as being either a single error or an error burst as shown in Figure 2.5. They defined error bursts as a number of errors that may hit message transmissions in the worst possible case. The number of errors are modeled with a separate distribution that depends on the operational environment of the considered embedded system. This distribution is assumed to be constructed during the system design by tests and measurements.

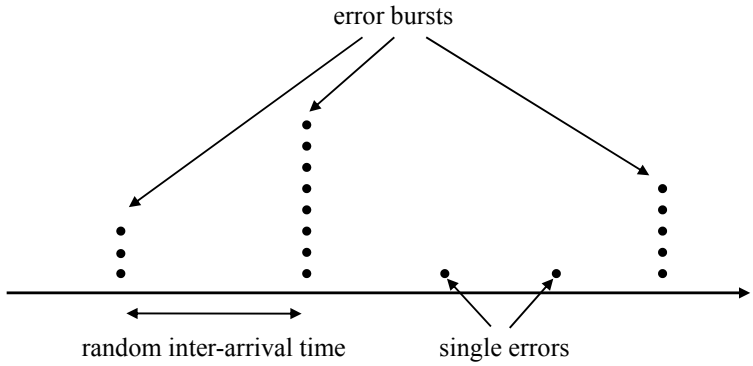


Figure 2.5: Burst error model by Navet et al.

## 2.3 Real-Time Systems

Real-time systems are computing systems whose correctness depends not only on the correctness of the outputs produced, but also on the timeliness of these outputs [92]. In *hard real-time systems*, failing to meet the timeliness requirement may result in catastrophic consequences, such as loss of human life, whereas in *soft real-time systems*, missing these requirements typically result in decreased Quality of Service (QoS), or degraded service.

Real-time systems are typically composed of a set of *tasks*, where each task performs a certain function satisfying certain *timing constraints*. The timing constraints are specified by special attributes, such as *offsets*, which specify the earliest time points at which the tasks can start executing, or *deadlines* which specify the latest time points by which the tasks should complete their executions. Tasks may have *periodic* [66], *aperiodic* or *sporadic* [71] activations which are controlled by a *scheduler*, based on a *scheduling policy*. Each periodic task consists of an infinite sequence of activations, which are called *instances*. The scheduling policy can either be *off-line* or *on-line*. In the off-line scheduling policies, the time points for each activation of task instances are decided at design-time, whereas in on-line scheduling, these decisions are made during run-time based on, e.g., task *priorities*. On-line scheduling policies can further be decomposed into *Fixed Priority Scheduling* (FPS),

and *Dynamic Priority Scheduling* (DPS) policies depending on whether the task priorities are decided during design-time or run-time [88, 7].

Fast computing or performance optimizations are not direct solutions for satisfying the timeliness requirement, since increasing the speed of computations does not mean that meeting the deadlines will be guaranteed [91]. Real-time research strives for assuring that the systems will behave predictably with respect to time, e.g., execute their tasks before their predefined deadlines, while enabling efficient usage of the limited resources such as processor and memory.

Real-time systems consist of some sort of hardware, often relatively complex real-time software and a dynamic environment that the systems interact with. Despite the advances in the production techniques of computer hardware, there still remains a possibility that the hardware may fail. Similarly, despite the advances in software engineering, bug free software development is considered as infeasible due to the costs, if at all practically possible. Furthermore, due to the non-deterministic nature of environments in which real-time systems operate, there is always a possibility of external interferences that may adversely affect the correctness or timeliness of their functioning. Therefore, special attention has to be paid to cope with such interferences, in order to have the confidence in the real-time systems at acceptable levels. This is the basic reason for the close coupling between real-time systems and dependability concerns.

## 2.4 Real-Time Communication

A real-time system typically consists of either a single processing node or a distributed set of nodes. Systems with the latter configuration, where processing nodes are interconnected over a communication network, are typically called *distributed real-time systems* and deployed in a wide range of application domains, e.g., automotive, factory automation and avionics. To satisfy the timeliness requirements in such systems, meeting the task deadlines alone is not enough which should be complemented with the timely delivery of messages between the processing nodes [95]. Real-time communication aims to satisfy the timeliness requirements of message transmissions with the help of scheduling techniques as in the case of task scheduling.

### 2.4.1 Controller Area Network (CAN)

CAN is a widely used communication protocol which was designed in the 80s at Robert Bosch GmbH [72] with a particular focus on automotive real-time requirements. It has been very popular in the automotive and automation industries due to its low cost and predictable real-time behaviour. CAN protocol provides prioritized transmission of network messages. This enables analysis of its real-time behaviour using similar techniques developed for fixed priority task sets.

CAN is a broadcast bus, which uses deterministic collision resolution to control access to the bus. The basis for the access mechanism is the electrical characteristics of the CAN bus. The dominant bit value on the bus is "0", meaning that, if more than one nodes are transmitting bits simultaneously, and one them is transmitting a "0", then the value on the bus seen by all nodes will be "0". The value on the bus becomes "1" only if all nodes transmit "1"s. This behaviour is used to resolve collusions on the bus. Each node waits until the bus is idle. Upon detection of silence, each node starts transmitting the highest priority message frame in its output queue, while simultaneously monitoring the value of the bus. Each message frame has a unique identifier acting as its unique priority. The identifier is the first part of the message frame to be transmitted and it is transmitted from the most-significant to the least-significant bit. The priority increases as the numerical value of the identifier decreases. Hence, by monitoring the value on the bus, a node detects if there is another frame being transmitted with a higher priority, when it transmits a recessive bit ("1") and sees a dominant bit ("0") on the bus. Whenever a higher priority frame transmission is detected, the node stops the transmission. Because identifiers are deemed unique within the system, a node transmitting the last bit of the identifier without detecting a collision must be transmitting the highest priority queued message frame, and hence can start transmitting the body of the message frame.

CAN frames can be transmitted at speeds of up to 1 Mbps. Each message can contain between 0 and 8 bytes of data. An 11 bit identifier is associated with each message frame. There is also an extended CAN format with a 29 bit identifier, but since this format is identical in all other respects, it will not be considered here. The identifier serves two purposes: (1) assigning a priority to the message frame, and (2) enabling receivers to filter message frames. A node filters message frames by

only receiving message frames with particular bit patterns. The CAN message frame format contains 47 bits of protocol control information (the identifier, Cyclic Redundancy Check (CRC) data, acknowledgement and synchronization bits, etc.). The data transmission uses a bit stuffing protocol which inserts a stuff bit after five consecutive bits of the same value. The frame format is specified such that only 34 of the 47 control bits are subject to bit stuffing. Hence, the maximum number of stuff bits in a message frame with $\eta$ bytes of data is $\lfloor \frac{(8\eta+34-1)}{4} \rfloor$ (since the worst-case bit pattern is '0000011110000...'). The size of a transmitted CAN message frame, denoted by $f$, is between 47 and 135 bits:

$$f = (8\eta + 47 + \lfloor \frac{(8\eta + 34 - 1)}{4} \rfloor) \tag{2.1}$$

where $\eta$ is the number of data bytes.

### Error Handling in CAN

The model underlying the basic CAN analysis assumes an error free communication bus, i.e. all message frames sent are assumed to be correctly received, which may not always be true due to the interference from the operational environment, or the faulty hardware components. To avoid erroneous transmissions, CAN designers have provided elaborate error checking and self-checking mechanisms as presented in [24], specified in the data link layer of ISO 11898 [47]. The error detection is achieved by means of transmitter-based-monitoring, bit stuffing, CRC, message frame format check, and frame acknowledgement.

To make sure that all nodes have a consistent view, errors detected in one node must be globalized. This is achieved by allowing the detecting node to transmit an error frame that is between 17 to 31 bits long (details are given in [1]). Upon reception of an error frame, each node will discard the erroneous message, which then will be automatically re-transmitted by the sender. Note that, the re-transmitted message could be subjected to arbitration during re-transmission. This implies that if any higher priority messages gets queued during the transmission and error signalling of the current message, then those messages will be transmitted before the erroneous message is re-transmitted.

Basic philosophy of these features is to identify an error as fast as possible and then retransmit the affected message. This implies that in systems without spatial redundancy of communication medium/con-

trollers, the fault-tolerance mechanism employed is temporal redundancy which addresses transient errors but could have an adverse impact on the latencies of message sets; potentially leading to violation of timing requirements. Furthermore, bursts of errors typically affect several message re-transmission attempts and contribute to potentially large response times that may deem the system unschedulable. Hence, novel schedulability analysis techniques are needed to handle complex error scenarios.

## 2.5 Real-Time Scheduling

Real-time scheduling is the procedure to control the access of real-time tasks or messages to shared resources that they are allocated on, such as processors and networks. A real-time scheduler activates the task executions or message transmissions based on the timing constraints which are translated into task or message attributes. This section presents the commonly used scheduling policies that are further addressed in this thesis.

### 2.5.1 Off-line Scheduling

Off-line scheduling is a static approach, where the order of task or message activations are pre-determined at design-time. The time points for each task activations are usually stored in scheduling tables. *Run-time dispatchers* perform a simple table-lookup to decide which task or message is granted access to the shared resources at every specified time.

This approach ease the satisfaction of complex timing constraints, such as end-to-end deadlines, precedence and instance separation, and provides deterministic execution of tasks or transmission of messages. However, it lacks the flexibility to handle non-deterministic run-time events such as performing recovery procedures in the event of errors. One can allocate certain slots for handling non-deterministic events in off-line scheduling, but this comes at the cost of suboptimal utilization or slow response, since these slots are reserved even if there is no need for them, or the scheduler needs to wait until a slot has reached, rather than having the possibility of responding immediately.

### 2.5.2 On-line Scheduling

In on-line scheduling, the scheduling decisions are made during run-time based on, e.g., task or message *priorities*. Based on whether the priorities are pre-determined at design-time, or can be changed during run-time, two major scheduling policies have been proposed, viz., FPS and DPS.

#### Fixed Priority Scheduling (FPS)

In FPS, task or message priorities are decided during design-time and remain unchanged during run-time. The scheduler gives the task or message with the highest priority, that is available in the *ready queue*, the access to the shared resource. The most well-known policy of assigning priorities to tasks is the Rate Monotonic (RM) policy proposed by Liu and Layland [66]. This policy is shown to be an *optimal* FPS policy, meaning that if there is any FPS policy that can schedule a given task set, then RM can also schedule it. RM policy assumes a periodic task model with deadlines equal to the periods and assigns priorities to tasks based on their periods, giving higher priorities to the tasks with the shorter periods. RM policy requires that the scheduler is *preemptive* which means that it suspends the currently executing task if a task with a higher priority becomes available in the ready queue, and resumes it whenever it becomes the highest priority task in the ready queue again. Leung and Whitehead proposed the Deadline Monotonic (DM) priority assignment policy [61, 6, 9] and proved that it is an optimal priority assignment policy under preemptive FPS for a set of periodic (or sporadic) tasks with deadlines that are less than their periods (or minimum inter-arrival times).

#### Dynamic Priority Scheduling (DPS)

In DPS, task or message priorities change dynamically during run-time. Earliest Deadline First (EDF) is the most well-known DPS policy [66], which assigns the highest priority to the task that has the closest deadline among the tasks in the ready queue. Dertouzos showed that [28] under the task model assumed by Liu and Layland [66], EDF is an optimal scheduler in terms of schedulability, guaranteeing schedulability for processor utilizations up to 100%, i.e., under the assumed task model, if there is a task set schedulable by any scheduler, then EDF can also schedule it.

## 2.6 Real-Time Analysis

Timing analysis of real-time systems aims to provide guarantees that every task or message in the system can meet their deadlines. The majority of the existing analysis techniques targeting hard real-time systems in the literature aims at providing deterministic guarantees using worst-case assumptions [66, 59, 77, 49]. Probabilistic analysis techniques have also been proposed targeting soft real-time systems [38, 3, 60, 50], addressing stochastic events, such as error occurrences, for which making worst-case assumptions is not always possible, and with the aim of providing more accurate analysis results for tasks whose WCET vary greatly from the average-case execution times [5, 31]. This section gives an overview of the existing deterministic real-time analysis techniques that form a basis to the proposed deterministic and probabilistic real-time analysis techniques in the thesis.

The notation used in this section, regarding the real-time task model is presented in Table 2.1.

### 2.6.1 Utilization Bounds

#### Rate Monotonic

Liu and Layland [66] showed that if the total processor utilization of all the tasks in the task set satisfies the minimum processor utilization bound, then the task set is schedulable.

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{\frac{1}{n}} - 1) \tag{2.2}$$

This total processor utilization based schedulability condition assumes that the tasks' deadlines are equal to their periods, and they are released at the beginning of their periods. This schedulability condition is sufficient but not necessary, meaning that the condition only guarantees meeting all the deadlines in case the total utilization is less than or equal to the given bound. However, if the total utilization is more than the bound, the task set may still be schedulable. The bound shown above converges to 69% for large $n$. However Lehoczky [59] showed that, in their study, the average case for the utilization bound reached up to 88% for randomly created task sets.

| | |
|---|---|
| $n$ | number of messages / tasks |
| $C_i$ | worst-case transmission/execution time of message/task $i$ |
| $\overline{C_i}$ | worst-case execution time of alternate task $i$ |
| $T_i$ | period/minimum inter-arrival time of message/task $i$ |
| $R_i$ | worst-case latency/response time of message/task $i$ |
| $D_i$ | relative deadline of message/task $i$ |
| $hp(i)$ | set of messages/tasks with priority higher than that of message/task $i$ |
| $hep(i)$ | set of messages/tasks with priority higher than or equal to that of message/task $i$ |
| $J_i$ | worst-case queuing jitter of message $i$ |
| $q_i$ | worst-case queuing delay of message $i$ |
| $B_i$ | non-preemptive transmission of a lower priority message frame, or the non-preemptive transmission of a message frame belonging to the previous instance of the message $i$ |
| $T_F$ | minimum fault inter-arrival time |

Table 2.1: Real-time task model notation

### Fault-Tolerance Adaptation of Rate Monotonic

Pandya and Malek extended the RM policy to provide recovery from single errors by re-executing the tasks hit by the errors [77]. They showed that, assuming that the inter-arrival time between any two errors is greater than largest period in the task set, the task set is schedulable even when performing re-executions, if the total processor utilization of tasks is less than or equal to 50%. This bound is better than the trivial bound obtained by simply doubling the task utilizations (34.5 %).

#### Earliest Deadline First

Liu and Layland showed that using EDF, utilizations up to 100% can be achieved while guaranteeing schedulability assuming the same task model used for deriving the RM utilization bound.

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \tag{2.3}$$

This schedulability condition is necessary and sufficient. For a more relaxed task model where deadlines are allowed to be different than the periods, it has been shown that this condition is not sufficient [62].

#### Non-preemptive Rate Monotonic on CAN

Andersson and Tovar [4] showed that using the RM policy on a CAN bus (with non-preemptive transmissions), the schedulability of message streams can be guaranteed if the total network utilization does not exceed 25%. They also proved that no greater bound can be given for the CAN bus. The messages are assumed to be sporadic and have unique priorities. The relative message deadlines are assumed to be equal to the minimum inter-arrival times.

### 2.6.2 Response Time Analysis

Response Time Analysis (RTA) is an analysis technique used to determine whether a task or a message set meets all the deadlines in the worst-case execution scenarios. For each task or message, the worst-case execution scenario is assumed as the scenario that gives the worst-case response time (WCRT) with the worst combination of task execution times [103] or message transmission times, task or message release patterns and error events.

This section presents the well-known RTA techniques for fixed priority task and message scheduling policies, that are used throughout the thesis.

#### Response Time Analysis for Task Scheduling

The traditional RTA used in FPS calculates the WCRT $R_i$ for each task $i$ (denoted by $\tau_i$) in the task set using the following equation assuming that there are no errors and, hence, no recovery attempts [49]:

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \tag{2.4}$$

The second term in the equation is the worst-case interference $I_i^{hp}$ from the higher priority tasks experienced by task $i$.

The following recurrence relation is used for solving Equation 2.4:

$$r_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{r_i^n}{T_j} \right\rceil C_j$$

where $r_i^0$ is assigned the initial value of $C_i$. $r^n$ is a monotonically non-decreasing function of $n$ and when $r_i^{n+1}$ becomes equal to $r_i^n$ then this value is the WCRT $R_i$ of task $i$. If the WCRT $R_i$ becomes greater than the deadline $D_i$, then the task cannot be guaranteed to meet its deadline, and the task set is therefore considered unschedulable.

### Response Time Analysis for Task Scheduling under Errors

If we assume an FT scheduler, where the tasks affected by errors are re-executed, then the execution of task $i$ will interfere with both errors as well as higher priority tasks. Accordingly, the WCRTs are computed [22] by using the following equation:

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j + \left\lceil \frac{R_i}{T_F} \right\rceil \max_{k \in hep(i)} (\overline{C}_k) \tag{2.5}$$

where $\overline{C}_k$ is the WCET needed by task $k$ to recover from the errors, $T_F$ is a known minimum inter-arrival time between faults that may cause errors, and $hep(i)$ is the set of tasks with priority equal to or higher than the priority of task $\tau_i$ ($hep(i) = hp(i) \cup \tau_i$). The last term calculates the worst-case interference arising from the recovery attempts and is denoted by $I_i^{err}$. This equation is also solved by a recurrence relation as in the previous case. If all $R_i$ values are less than or equal to the corresponding $D_i$ values, then the task set is guaranteed to be scheduled under the condition that no two errors occur closer than the $T_F$ value.

### Response Time Analysis for Message Scheduling in CAN

In [96] the authors present analysis to calculate the WCRT of CAN messages. This analysis is based on the RTA for task scheduling, which

is later refined in [27]. Calculating the response times requires a bounded worst-case queuing pattern of messages. The standard way of expressing this is to assume a set of traffic streams, each generating messages with a fixed priority. The worst-case behavior of each stream is to periodically queue messages.

For an ideal CAN controller (the non-ideal case is presented by [97]) the WCRT $R_i$ of a CAN message $M_i$ is defined by

$$R_i = J_i + q_i + C_i \tag{2.6}$$

where $J_i$ is the worst-case queuing jitter of message $M_i$, inherited from the sender task which queues the message. The minimum delay from the point in time $t$, relative to the time message $M_i$ is queued, is assumed to be 0 ($t$ is typically the start of the period). In other cases the term $J_i^{min}$ needs to be added to Equation 2.6, since jitter is defined as the difference between the biggest and smallest delay from $t$. The worst-case queuing delay $q_i$ is given by,

$$q_i = B_i + \sum_{j \in hp(i)} \left\lceil \frac{q_i + J_j + \tau_{bit}}{T_j} \right\rceil C_j \tag{2.7}$$

where $B_i$, in the general case, is either the non-preemptive transmission of a lower priority message frame, or the non-preemptive transmission of a message frame belonging to the previous instance of the message $M_i$ [27]. This is equivalent to the worst-case blocking time of the longest possible message frame (i.e., the worst-case transmission time of a CAN message frame with 8 bytes of data and worst-case bit stuffing). $\tau_{bit}$ caters for the difference in arbitration start times at the different nodes, due to propagation delays and protocol tolerances.

Tindell et al. [96], extended the RTA for CAN to handle the error induced interference and proposed the following equation:

$$q_i = I_i^{err} + B_i + \sum_{j \in hp(i)} \left\lceil \frac{q_i + J_j + \tau_{bit}}{T_j} \right\rceil C_j \tag{2.8}$$

where the additional term, $I_i^{err}$, denotes the maximum error interference assuming a bounded error model:

$$I_i^{err} = \left( n_{burst} + \left\lceil \frac{q_i + C_i}{T_F} \right\rceil - 1 \right) \left( max_{j \in hep(i)} + e^{max} \tau_{bit} \right) \tag{2.9}$$

In their assumed bounded error model, first an error burst of $n_{burst}$ errors is assumed to interfere with the message transmissions, and after that, one error is assumed to occur per elapsed minimum inter-arrival time $T_F$ between faults causing the errors.

In [82], the authors extended Tindell et al.'s analysis with a more general error model where a task can be interfered by a number of error bursts with different durations. When experiencing a burst error, the bus becomes unavailable and all transmissions are delayed for a duration equal to the duration of that burst.

## 2.7 Summary

This chapter provides an overview of the theoretical background for the dependability and real-time research, relevant to the thesis. It starts with describing the main concepts and terminology for the dependability research and briefly describes the commonly used error modeling approaches in the dependability and real-time research. Later, it gives an overview of real-time systems and real-time communication including a description of CAN's technical details necessary for the thesis. Finally, it presents the scheduling policies for task and message scheduling together with the real-time analysis techniques relevant to this thesis.

# Chapter 3

# Temporal and Spatial Redundancy Strategies

Embedded real-time applications typically have to satisfy high dependability requirements due to the interactions with, and possible impacts on, the environment. Ensuring dependable performance of such systems is typically ensured by fault-tolerance in their designs. Redundancy is the key fault-tolerance approach and it has been employed successfully in the spatial, temporal, information and analytical domains of a large number of critical applications [45, 102]. This chapter proposes various fault-tolerance strategies in the temporal and spatial domains that provide significant improvements over the existing strategies, in terms of better error detection, wider error coverage and more efficient resource usage.

## 3.1 Voting on Time and Value Domains

Safety and mission critical applications have been using voting strategies based on N-Modular Redundancy (NMR), most often in the well-known form of Triple-Modular Redundancy (TMR), where three nodes are used for replication [74]. The key attraction of this approach lies in its low overhead and error masking abilities, without the need for backward recovery [57]. Some of the disadvantages include the cost of redundancy and single point failure mode of the voter. Traditionally, voters

are constructed as simple electronic circuits, thus providing a very high level of reliability. Distributed voters have also been employed to take care of the single-point failure mode in highly critical systems [37, 68]. With the additional cost of increased computation time, more enhanced voting strategies, such as *plurality*, *median* and *average* voting, can be performed in software. *Plurality voters*(*m-out-of-n voters*) require $m$ matching outputs out of $n$, where $m$ is less than the majority, to reach a consensus [67, 42]. *Median voters* output the middle, and *average voters* output the average value of the replica output values. Surveys and taxonomies of voting strategies have been presented [15, 41, 58].

Replicated nodes' output values can vary slightly, resulting in a range (or a set) of values, which should be considered as correct to avoid problems of failing to reach a consensus as indicated in [18, 25]. In order to accomplish this, *inexact voting strategies* have been proposed [56, 81, 87]. This phenomenon is also observed in the time domain due to several factors, such as clock drifts, node failures, processing and scheduling variations at node level, as well as communication delays. Most of the existing voting strategies, however, focus solely on tolerating anomalies in the value domain by assuming that they are running on *tightly synchronized systems*, as pointed out in [51]. On the other hand, *loosely synchronized systems* have several advantages over the tightly synchronized systems, such as, lower overheads, reduced complexity, and the lower criticality of the synchronization mechanism itself [55]. The key issue with loosely synchronized systems is that voting is made more complicated since the differences between the replica output values may exist independent of whether errors have occurred, as each of the replicas may be receiving different inputs.

A simple approach towards tolerating both value and timing errors using the NMR approach could be adding time stamps to the replica outputs. Then, voting on the time stamps could detect possible timing errors in the replica outputs. However, this approach is unable to mask late timing errors since the voter has to wait for all the values to be delivered by the replicas. Majority voting techniques that are able to implicitly handle the errors in the time domain, have been proposed by Ravindran et al., [84, 85], and Shin et al., [89]. In these approaches, voting is performed among a quorum or a majority of the responses received, rather than waiting for all the responses, in order to be able to mask late timing errors. Both Quorum Majority Voting (QMV) and Compare Majority Voting (CMV) provide outputs within a bounded

time interval. These approaches are built on the assumptions that the number of replicas with erroneous outputs during each voting procedure does not exceed the allowed thresholds ($n$ out of $3n+1$ erroneous replica outputs for QMV, and $n$ out of $2n+1$ erroneous replica outputs for CMV). As long as the assumption holds, the ability to detect value errors is equivalent to the existing approaches. However, QMV and CMV cannot always detect whether this assumption has been violated. Hence, these approaches may produce optimistic results in the sense that, e.g., an incorrect value may be produced at a correct (or incorrect) time, and thus may not be fully suitable for hard real-time systems.

Traditionally, research efforts are less focused on scenarios and solutions beyond the stated assumptions, whereas in practice, the robustness of dependable systems can be enhanced by the provision of signalling assumption violations. For example, in the event of a violation of the underlying assumptions, a voter needs to be cautious in the provision of outputs to the environment, e.g., a "no output" together with an error signal may be a better alternative than a potentially erroneous output. This gives an enhanced capability for ensuring *fail-safe* or *fail-stop* behavior within systems.

This section outlines a conceptual design for the real-time voting strategy *Voting on Time and Value* (VTV) [14], in which voting is performed in both the time and the value domains. In particular, VTV aims to enhance the fault-tolerance abilities of NMR by ensuring the output from the voter to be both correct in value, and delivered within an admissible time interval, under specified assumptions. The detection threshold for the timing anomalies, hence the admissible time interval, can be adjusted in the voter, in order to tune the detection performance. Except for some extreme and highly unlikely scenarios, such as all node outputs having late timing errors, VTV is designed to detect and signal assumption violations in both the time and the value domains within a bounded time, while QMV and CMV can only detect assumption violations in the value domain. VTV is also designed to perform at least as good as the existing strategies targeting only tightly synchronized systems.

Table 3.1 presents an overview of various voting strategies, applicable to loosely synchronized real-time systems which are described in the following subsections.

This section ends with an evaluation that shows how different detection thresholds, error sizes, and error scenarios affect the detection

| Voter | Description | Voting domain(s) |
|---|---|---|
| QMV | 1. Wait for a quorum (2n+1 out of 3n+1 replica outputs) 2. Perform majority voting in the quorum | value |
| CMV | Wait for a majority (n+1 out of 2n+1 replica outputs) with identical values | value |
| VTV | 1. Wait for a majority (or a plurality) delivered within a predefined time window 2. Perform voting in the value domain among all the available or only the timely replica outputs (depending on the application characteristics) 3. In case there is no agreement in the value domain, return to step one, 4. If no agreement is formed in both the domains, signal disagreement | value & time |

Table 3.1: Overview of voting strategies suitable for real-time systems

capabilities. The results demonstrate the robustness of VTV, confirming that VTV outperforms CMV in all scenarios with lower false negative rates and in terms of its potential for notifying assumption violations.

### 3.1.1 System Model

In this section, a distributed real-time system is assumed, where each critical node is replicated for fault-tolerance, and voting is performed on replica outputs to ensure correctness in both value and time. Upon receiving identical requests or inputs, replicas of a node start their executions on dedicated processors whose clocks are allowed to drift from each other at most by a maximum deviation. This bound can be achieved by relatively inexpensive clock synchronization algorithms implemented in software (compared to expensive tight clock synchronization implementations). After the replicas complete their executions, the outputs are sent to a stand-alone voting mechanism. Deviation in message transfer times from the replicas to the voter is also assumed to be bounded by using reliable communication techniques.

The maximum deviation of any two replica outputs in the time domain, in an error free scenario, as perceived by the voter, is denoted by

$\delta$, which includes the maximum skew between any two non-erroneous replica clocks, and the maximum skew between any two message transmissions from replicas to the voter. Upon receiving replica outputs, the voter starts executing the voting process, which has a known Worst-Case Execution Time (WCET) $C^{voter}$, and outputs a correct value at an admissible time, or signals the non-existence of a correct output to the subsequent component in the system within a bounded time.

The maximum admissible deviation between any two voter outputs in the time domain, relative to the correct time point $t^*$ (seen by a perfect observer), is denoted by $\Delta$, and determined according to the system specifications, i.e., what the rest of the system can tolerate as per the real-time and dependability specifications. The reader should note that the maximum admissible deviation of a voter output from the correct time point, $t^*$, is $\Delta/2$ and the maximum deviation of a replica output from $t^*$ in an error free scenario is $\delta/2$.

Potential errors in the value domain are detected when two replica outputs differ more than the *maximum admissible deviation between any two replica outputs*, denoted by $\sigma$. Potential errors in the time domain are detected when the voter receives two replica outputs that are separated by more than $\delta$ multiplied by the *detector coefficient* $\alpha$. If $\alpha$ is less than one, the error detector may identify even error free outputs as erroneous increasing the false positives, but this may increase the detection of the variations in the time domain due to errors reducing the false negatives. An $\alpha$ value less than one can also be used if $\Delta/2$ is less than $\delta/2 + C^{voter}$ to detect any scenarios that may result due to inadequate synchronization and threaten system's timeliness requirements. On the other, hand if $\Delta/2$ is greater than $\delta/2 + C^{voter}$, then the system can tolerate even some of the variations due to errors. Hence, the difference between $\Delta/2$ and $\delta/2 + C^{voter}$ can be used to reduce false positives by using an $\alpha$ value greater than one. Figure 3.1 shows the relation between $\Delta$, $\delta$ and $C^{voter}$.

The output delivered by $N_i$, is specified by two domain parameters, viz., value and time (for the sake of readability, the $i^{th}$ replica of a given node is denoted by $N_i$):

$$\textit{Specified output for } N_i = < v_i^*, t_i^* >$$

where $v_i^*$ is the correct value, $t_i^*$ is the correct time point (seen by a perfect observer) when the output should be delivered.

An output delivered by $N_i$ is denoted as:

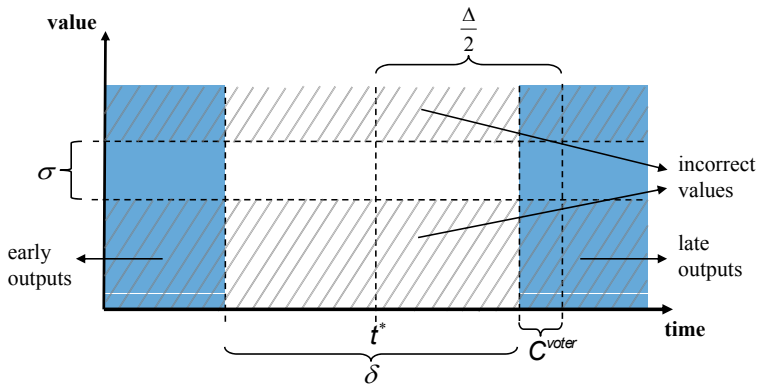


Figure 3.1: Output correctness in the time and the value domains and the relation between $\Delta$, $\delta$, and $C^{voter}$

$$\textit{Delivered output from } N_i = < v_i, t_i >$$

where $v_i$ is the value and $t_i$ is the time point at which the value was delivered. Based on the voter parameters $\sigma$, $\delta$ and $\alpha$, the output generated by replica $N_i$ is defined as *incorrect in the value domain* if:

$$v_i < v_i^* - \frac{\sigma}{2} \text{ or } v_i > v_i^* + \frac{\sigma}{2}$$

and *incorrect in the time domain* if:

$$t_i < t_i^* - \alpha\frac{\delta}{2} \; (\textit{early timing error})$$

or if

$$t_i > t_i^* + \alpha\frac{\delta}{2} \; (\textit{late timing error}).$$

The notations used for the error behavior of the replicas (seen by a perfect observer) are:

- $E_v$: the number of replicas that have only value errors
- $E_t$: the number of replicas that have only timing errors, consisting of two subcategories:

  - $E_t^e$: the number of replicas that produce early outputs with correct values
  - $E_t^l$: the number of replicas that produce late outputs with correct values

- $E_{vt}$: the number of replicas that have both value and timing errors, consisting of two subcategories:
  - $E_{vt}^e$: the number of replicas that produce early outputs with incorrect values
  - $E_{vt}^l$: the number of replicas that produce late outputs with incorrect values

  where the total number of erroneous replica outputs $E$ is:

$$E = E_t + E_v + E_{vt}$$

Finally, the notations used while describing the voting mechanisms are:

- $N$: number of replicas
- $M_t$: minimum number of replicas required to form a consensus in the time domain, as per the system specifications
- $M_v$: minimum number of replicas required to form a consensus in the value domain, as per the system specifications

**Basic assumptions:** The VTV strategy relies on the following set of basic assumptions (to a large extent based on [32]):

A1: non-erroneous nodes produce values within a specified admissible range after each computation block

A2: non-erroneous nodes produce values within a specified admissible time interval after each computation block

A3: replica outputs with incorrect values do not form (or contribute in forming) a consensus in the value domain

A4: incorrectly timed replica outputs do not form (or contribute in forming) a consensus in the time domain

A5: there exist adequate mechanisms, e.g., infrequent synchronization, which are significantly less costly than tight synchronization, to ensure a maximum permissible replica deviation from the global time

A6: the voting mechanism does not fail, as being designed and implemented as a highly reliable unit

### 3.1.2 Methodology

The correctness of the VTV approach [14] relies on a number of conditions regarding the permissible number of replicas that produce erroneous outputs:

$\mathcal{C}1$: The number of replicas that produce erroneous outputs can not exceed the difference between the total number of replicas and the minimum number of error free replicas required to achieve consensus in the value domain.

$$E_v + E_t + E_{vt} \leq N - M_v$$

$\mathcal{C}2$: The number of replicas that produce erroneous outputs in the time domain is bounded by the difference between the total number of replicas and the minimum number of error free replicas required to achieve consensus in the time domain.

$$E_t + E_{vt} \leq N - M_t$$

The goal of the VTV approach is twofold:

1. always deliver the correct value within $[t^* - \frac{\Delta}{2}, t^* + \frac{\Delta}{2}]$, if the conditions $\mathcal{C}1$ and $\mathcal{C}2$ hold
2. provide information about violation of the conditions, otherwise.

In VTV, agreement in the time domain is reached when $M_t$ out of $N$ replicas deliver their outputs within the time interval $[t^* - \alpha\delta/2, t^* + \alpha\delta/2]$ (referred to as *feasible window* henceforth).

The maximum number of sets, consisting of $M_t$ consecutive replica outputs each (out of the $N$ replicas), is $N - M_t + 1$. Since the consensus in the time domain can be reached in any of these sets, a separate feasible

window needs to be initiated upon receiving each of the first $N - M_t + 1$ replica outputs. In order to keep track of the feasible windows, simple countdown timers are utilized. Once an agreement in the time domain is obtained, then the voting is performed on the values. If an agreement in the value domain is not obtained within a particular feasible window, the process continues with subsequent feasible windows, until agreement in both the time and the value domains can be achieved, or violations of $\mathcal{C}1$ or $\mathcal{C}2$ are detected.

Depending on the application characteristics, a value produced by a node may be considered *valid* or *invalid* for the purpose of voting, in the case it is produced early. This voting dilemma is illustrated by using the scenario described in Figure 3.2. Assume, for example, an airbag control

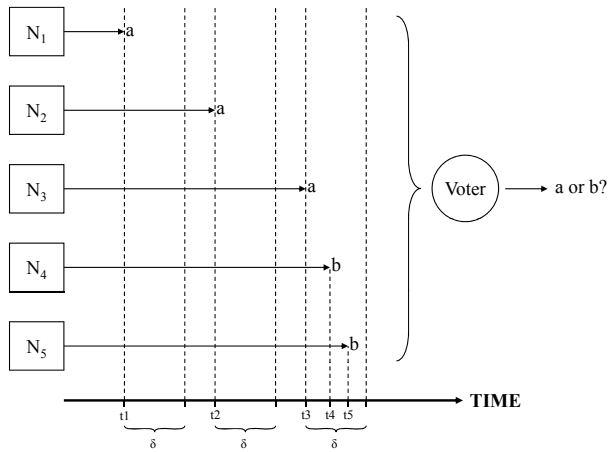


Figure 3.2: Voting dilemma

system where a collision sensor is replicated in five different nodes that produce one out of two values periodically, e.g., value $a$ in case of a collision detection or value $b$ otherwise. VTV detects the first two values as *early* and the last three are identified as *timely*. However, in this case, even an early value should be taken into consideration in the voting since an early collision detection is still a valid output with respect to the value domain. Thus, the voter outputs the value $a$. On the other hand, let the same Figure 3.2 represent an altitude sensor in an airplane, replicated by

five nodes to read and output the altitude periodically to the voter, where data freshness may be a more desirable aspect. As the correct window of time for the output is the same as described in the previous example, the only relevant values to be taken into consideration by the voter are $a$, $b$, and $b$ corresponding to the time points $t_3$, $t_4$, and $t_5$ respectively. Hence, a desirable output to be produced in this application at time $(t_5 + C^{voter})$ is $b$. The two cases are summarized as follows:

**Case 1** *Only timely outputs are considered valid.* If a plurality exists among the *timely* received values, the plurality value is delivered as the correct output.

**Case 2** *Early and timely outputs are considered valid.* If a plurality exists among *all* the received values, the plurality value is delivered as the correct output. The advantage of this case is that the number of the nodes required to mask a given number of errors (in the time and the value domains) can be significantly reduced, compared to Case 1, since replica outputs erroneous in one domain may still be used to reach consensus in the other domain. However, if the early timing errors have the potential to cause system failures, then using VTV in this configuration may be inadvisable. In this case, Condition $C1$ becomes:

$\mathcal{C}1(Case2)$ : The number of replicas that produce erroneous outputs, except the outputs with early timing errors, can not exceed the difference between the total number of replicas and the number of error-free replicas required to achieve consensus in value domain.

$$E_v + (E_t - E_t^e) + (E_{vt} - E_{vt}^e) \leq N - M_v$$

### 3.1.3 Evaluation

In this subsection, a simulation study, performed to investigate the performance of VTV as compared to CMV, is presented. As shown in Figure 3.3, five nodes are simulated where various kinds of transient errors are injected with certain probabilities along with a *reference node* that never fails. The outputs of the five nodes are voted using three different voters: (i) one implementing the VTV strategy where early generated replica outputs are considered invalid for the value voting (Case 1), (ii)

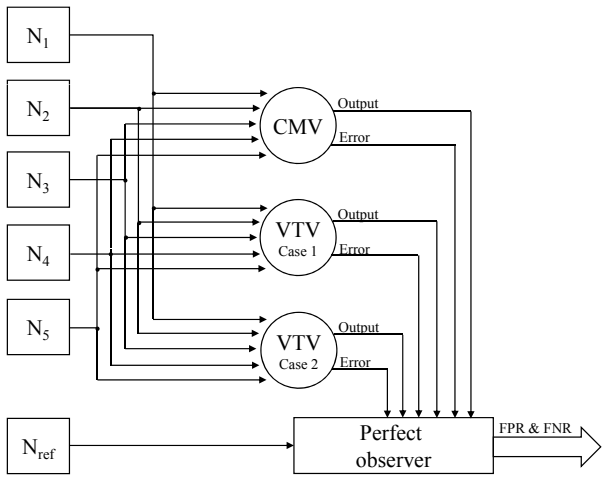


Figure 3.3: Simulation setup

one implementing the VTV strategy where early outputs are considered valid for the value voting (Case 2) and (iii) one implementing the CMV strategy. All node outputs are sent to a *perfect observer* module together with the voter outputs and the assumption violation signals from the voters, in order to determine the False Positive Rate (FPR) and the False Negative Rate (FNR) of each voting strategy.

### Experimental Design

The simulations are implemented using Matlab/Simulink. In order to evaluate the dependency on the signals input to the replicas, two different signals, viz., a sine wave and a square wave, at various frequencies are used. By using the sine wave, signals that change smoothly over time, with a value limit for the maximum change during a given time are simulated. The sine wave also allows an assessment of how specific frequencies are affected. An example for such a signal is the reading of a temperature sensor in a closed room, where the rate of change is limited based on the factors such as the heater capacity, the volume of the room, etc. A square wave also allows a range of frequencies to be assessed. By using the square wave, signals such as those that can be generated by

sensors that output boolean values are simulated. An example for such a sensor is a smoke detector which outputs either *false* to indicate that there is no smoke, or *true* otherwise. The amplitudes of the signals are set to 5 units. The sensor noise is simulated by adding a normally distributed random value with a mean of 0 and a variance of 0.2 units to the input signals.

### Injected Errors

Three types of errors are considered in the simulations.

1. **Value errors:** The node outputs the sampled value with a uniformly distributed random value offset within a given value offset range.
2. **Timing errors:** The node outputs the sampled value with a uniformly distributed random time offset within a given time offset range.
3. **Omission errors:** When an omission error (which is a special type of timing errors) is injected, the node skips outputting the sampled signal value.

In order to evaluate the sensitivity of the voting approaches, i.e. the detection capabilities of the voters depending on the errors' magnitude, the time and value offsets are randomly generated within two different ranges. The narrow time offset range is defined as $[-1ms, 1ms]$. Please note that replica outputs are assumed to be timely as long as the time difference between the output delivery times and the delivery time of an ideal replica (with no timing errors and no time drift from the real-time) is less than or equal to $\alpha\delta_{clock}/2$ where $\delta_{clock} = 0.5ms$ and $\alpha$ is selected within an interval of $[0.6, 1.4]$, hence the timing errors generated within this range are harder to detect than the timing errors generated using the wide time offset range $[-4.5ms, 4.5ms]$, which, together with the drift from the real-time, may result in replica output deliveries any time during a period. The narrow value offset range is defined as $[-1, 1]$ units and the wide value offset range is defined as $[-5, 5]$ units. Figure 3.4 shows a sampled input signal and a node output with injected errors.

Voters' abilities to work in a wide range of conditions are evaluated by injecting the errors with different combinations (no errors, only value errors, only timing errors (including omission errors), and both value and

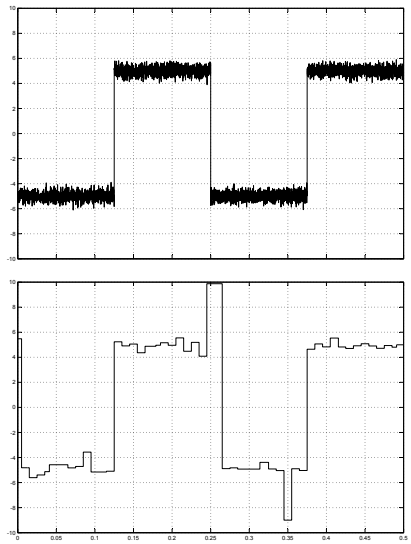

Figure 3.4: A noisy input signal and the corresponding node output with injected errors

timing of errors) as well as by injecting errors with different probabilities. Based on the error occurrence probability during a period for each type of error, two sets of experiments (one with a probability of 2% and one with a probability of 10%) are conducted.

### Task and System Properties

Each node executed an identical replica of a single task with a period $P = 10ms$ and an execution requirement $C = 5ms$. This is because, both the CMV and the VTV strategies expect parallel executions of the tasks in the replica nodes. A node samples the input signal at the beginning of its period and outputs the sampled value after the assumed execution time has elapsed in case of an error free operation. Except the timing errors, the only factor that may result in different output delivery times is the drift in the local clock of the node from the global clock. The drift is simulated by allowing local periods slightly longer or shorter than $10ms$. The execution requirement is also scaled up or down based on the local

period. Whenever the accumulated drift from the global clock, i.e. the accumulated sum of the difference between the local period and the real period, reaches the maximum admissible deviation from the real-time ($\delta_{clock}/2 = 0.5ms$), the local clock is synchronized with the global clock. This is realized by running a *synchronization period* which is shorter or longer than the real period with a value equal to the accumulated difference.

Figure 3.5 shows an example of a scenario with clock synchronization. The vertical arrows in the diagram indicate the beginning and the end

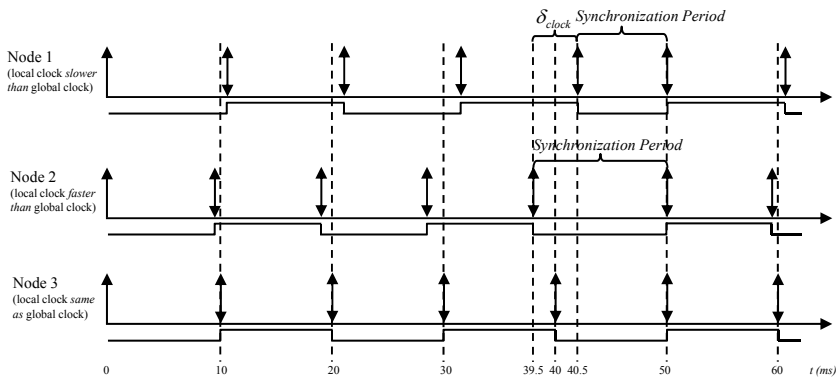


Figure 3.5: Clock synchronization

of the periods (release times and the deadlines of the tasks). In this example, node 3's clock runs in line with the real-time. However, node 1's clock runs slower and node 2's clock runs faster than the real-time. At $t = 39.5ms$, the drift of node 2's clock reaches $\delta_{clock}/2 = 0.5ms$ and it executes a synchronization period $T_2^{synch} = 10.5ms$. Similarly, at $t = 40.5ms$, the drift of node 1's clock reaches $\delta_{clock}/2 = 0.5ms$ and it executes a synchronization period $T_1^{synch} = 9.5ms$. At $t = 50ms$, all clocks are synchronized.

The time step used in the simulations is 100 simulation nanoseconds long, and each simulation is run for 100 simulation seconds. Increasing the simulation duration did not affect the statistical trend in the results, hence with the chosen error rates, a duration of 100 seconds for each run is found to be adequate for the purpose of these experiments. The

four error combinations (no error, only timing errors including omission errors, only value errors and both types of errors), the two error range combinations for both the time and the value errors, the two different error probabilities, the two different signals and the three different frequencies for each signal sum up to a total of 156 simulation runs. In addition, to show the effect of $\alpha$ on the performance of the evaluated voters, the experiments are separately run for 5 different $\alpha$ values.

#### Voters

Both the voters that use the VTV strategy and the voter that uses the CMV strategy look for a majority in the value domain ($M_v = 3$). In addition, the voters that use the VTV strategy look for a majority in the time domain ($M_t = 3$). Furthermore, the voter that uses the VTV strategy and configured for Case 1 requires that the majority of replica outputs that match in the value domain are also timely. Two outputs are assumed to be matching in the value domain if the difference between them is less than or equal to the maximum admissible deviation in the value domain between any two replica outputs ($\sigma = 0.2$ units), and they are assumed to be matching in the time domain if the difference in output delivery times is less than or equal to the maximum admissible deviation in time between any two nodes ($\delta_{clock} = 1ms$).

#### Perfect Observer

A perfect observer module is used to calculate the FPR and the FNR of each voting strategy. It compares the most recently received outputs from the voters with the output from the *reference node*. The outputs of the voters are considered to be correct if the difference between them and the reference node is less than or equal to the maximum admissible deviation in the value domain from the ideal output $\sigma/2 = 0.1$ units. This task also uses the *error signals* from the voters, i.e. the signals that indicate the assumption violations, to determine the FPR and the FNR.

#### Simulations with $\alpha = 1$

Table 3.2 shows the FPR and the FNR of the evaluated voters for the various signals and the error probabilities. Regardless from the signal types, the signal frequencies and the error probabilities, the FPR of

CMV is lower than that of VTV configured for Case 2, and the FPR of VTV configured for Case 2 is lower than that of VTV configured for Case 1. This is because the voters that use the VTV strategy identify more erroneous scenarios since, unlike the voter using the CMV strategy, they can detect the assumption violations in the time domain, and some of those timing errors are not propagated into value errors. Among the voters using the VTV strategy, the FPR is higher for the one configured for Case 1 since, similarly, it signals a greater number of assumption violations, some of which are not propagated into value errors. On the other hand, it can be seen that the FNR of CMV is the highest among all the evaluated voting strategies and the FNR of VTV configured for Case 1 is the lowest, which is much more critical than the difference in FPRs, as FNR is the rate that indicates the errors that are neither masked nor signalled.

| VOTER | *f(Hz)* | SIGNAL TYPE | FNR | | FPR | |
|---|---|---|---|---|---|---|
| | | | (2%) | (10%) | (2%) | (10%) |
| CMV | 1 | Sine w. | 0.09 | 0.39 | 1.17 | 3.92 |
| | | Square w. | 0.05 | 0.57 | 1.04 | 3.64 |
| | 4 | Sine w. | 0.56 | 5.36 | 1.08 | 3.07 |
| | | Square w. | 0.17 | 2.22 | 1.38 | 3.72 |
| | 16 | Sine w. | 1.74 | 11.27 | 0.8 | 2.49 |
| | | Square w. | 0.73 | 6.47 | 1.24 | 3.18 |
| VTV (Case 1) | 1 | Sine w. | 0.01 | 0.03 | 4.84 | 25.51 |
| | | Square w. | 0.02 | 0.09 | 5.06 | 25.87 |
| | 4 | Sine w. | 0.02 | 0.31 | 4.53 | 21.56 |
| | | Square w. | 0.03 | 0.19 | 5.01 | 25.38 |
| | 16 | Sine w. | 0.09 | 0.9 | 3.10 | 14.89 |
| | | Square w. | 0.05 | 0.34 | 4.55 | 20.98 |
| VTV (Case 2) | 1 | Sine w. | 0.09 | 0.39 | 1.17 | 3.92 |
| | | Square w. | 0.05 | 0.57 | 1.04 | 3.64 |
| | 4 | Sine w. | 0.31 | 1.19 | 2.92 | 14.74 |
| | | Square w. | 0.08 | 0.65 | 3.11 | 16.96 |
| | 16 | Sine w. | 0.93 | 3.81 | 2.38 | 11.83 |
| | | Square w. | 0.44 | 1.87 | 3 | 14.35 |

Table 3.2: FPR and FNR of CMV and VTV for various signal types and signal frequencies

Table 3.3 shows the FPR and the FNR of the evaluated voters for various error combinations. As expected, the FPR and the FNR for all the voters are zero in the absence of errors. When only value errors are injected, the FPR and the FNR are identical for all voting strategies. This is because all strategies use the same criteria for detecting the value anomalies. However, when the injected errors are only in the form of timing errors (including the omission errors), the performance of the voters using the VTV strategy outperforms the voter using the CMV strategy by a great margin. This increase in the performance is still visible when all types of errors are injected together.

| VOTER | ERROR TYPE | FNR | | FPR | |
|---|---|---|---|---|---|
| | | (2%) | (10%) | (2%) | (10%) |
| CMV | No errors | 0 | 0 | 0 | 0 |
| | Only value errors | 0.01 | 0.03 | 1.57 | 6.31 |
| | Only timing errors | 0.02 | 0.13 | 3.12 | 15.29 |
| | Both value & timing errors | 0.02 | 0.15 | 6.01 | 29.83 |
| VTV (Case 1) | No errors | 0 | 0 | 0 | 0 |
| | Only value errors | 0.01 | 0.03 | 1.57 | 6.31 |
| | Only timing errors | 0.02 | 0.13 | 3.12 | 15.29 |
| | Both value & timing errors | 0.02 | 0.15 | 6.01 | 29.83 |
| VTV (Case 2) | No errors | 0 | 0 | 0 | 0 |
| | Only value errors | 0.01 | 0.03 | 1.53 | 6.31 |
| | Only timing errors | 0.16 | 0.53 | 1.49 | 7.9 |
| | Both value & timing errors | 0.24 | 0.75 | 3.81 | 19.36 |

Table 3.3: FPR and FNR of CMV and VTV for various the error combinations

Table 3.4 shows the FPR and the FNR of the evaluated voters for different error magnitudes. Even when the injected errors become harder to detect due to their smaller magnitude, the VTV strategy outperforms the CMV strategy with respect to the FNR. However, the trend in the ratio of FNRs becomes slightly less distinct.

All of the above FPR and the FNR values are derived by the perfect observer by comparing the voter outputs with the reference node output. As stated earlier, the voter output is identified as erroneous in the value domain if the difference is greater than $\sigma/2 = 0.1$ units. Figure 3.6 shows the ratio of CMV's FNR to VTV's FNR (configured for both

| VOTER | ERROR MAGNITUDE | | FNR | | FPR | |
|---|---|---|---|---|---|---|
| | Value errors | Timing errors | (2%) | (10%) | (2%) | (10%) |
| CMV | LARGE | LARGE | 0.32 | 2.48 | 1.23 | 4.65 |
| | LARGE | SMALL | 0.2 | 2.02 | 1.44 | 4.81 |
| | SMALL | LARGE | 0.23 | 2.12 | 0.51 | 1.07 |
| | SMALL | SMALL | 0.28 | 1.76 | 0.55 | 1.26 |
| VTV (Case 1) | LARGE | LARGE | 0.02 | 0.13 | 5.8 | 28.73 |
| | LARGE | SMALL | 0.01 | 0.11 | 4.1 | 20.84 |
| | SMALL | LARGE | 0.01 | 0.2 | 4.17 | 21.27 |
| | SMALL | SMALL | 0.02 | 0.14 | 3.13 | 13.8 |
| VTV (Case 2) | LARGE | LARGE | 0.22 | 0.77 | 3.52 | 19.06 |
| | LARGE | SMALL | 0.12 | 0.64 | 3.08 | 15.73 |
| | SMALL | LARGE | 0.16 | 0.66 | 2.2 | 12.4 |
| | SMALL | SMALL | 0.19 | 0.57 | 2.12 | 10.16 |

Table 3.4: FPR and FNR of CMV and VTV for various error magnitudes

Case 1 and Case 2) for the value errors that are greater than the values identified on the X-axis. This experiments show that the masking and signalling capability of CMV decreases relative to that of VTV as the error magnitude increases, and hence is able to provide only a decreased level of fault-tolerance for the errors that are more critical. VTV performs better than CMV since the timeliness requirement of the signals limits the amount of deviation from the correct signal.

### Simulations Exploring the Variation of $\alpha$

Table 3.5 shows the FPR and the FNR of the evaluated voters for the given $\alpha$ values selected from the interval $[0.6, 1.4]$. Changing the value of $\alpha$ does not have any effect on the performance of CMV since no specific error detection is performed in the time domain. Nevertheless, the order among the FPR and the FNR values for all the three types of voters are preserved for all the chosen values of $\alpha$. For both configurations of VTV, as $\alpha$ increases, the FPR decreases and the FNR increases. This is because, for small values of $\alpha$, a greater number of voter outputs are detected as potentially erroneous in the time domain. Among these detected outputs, there are scenarios both where these anomalies are occurred due to errors and due to clock drifts. If the anomalies are

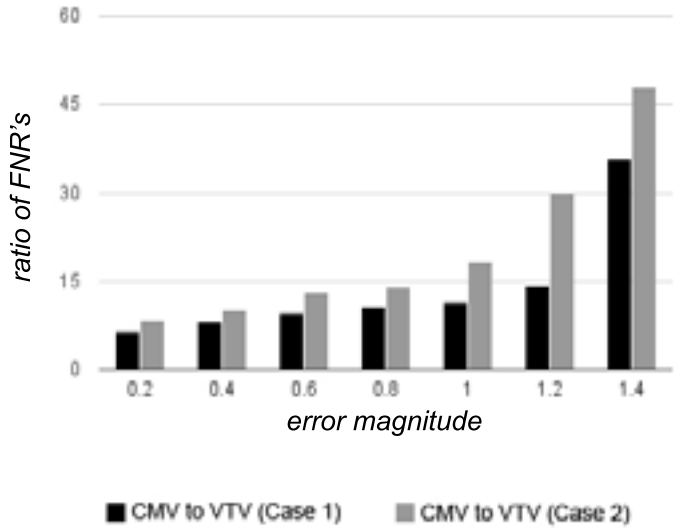


Figure 3.6: Ratio of CMV's FNR to VTV's FNR (configured for Case 1 and Case 2) with increasing error magnitudes

caused by errors, then their detection contributes to the reduction of the FNR. If they are caused by clock drifts, then their detection causes an increase in the FPR.

The experiment results confirm that VTV outperforms CMV in all scenarios, showing a lower percentage of errors that are neither masked nor signalled. The results indicate the overcautious nature of VTV by showing a higher percentage of false positives due to the fact that it does not account the untimely values for arriving at a majority. The results also show that the ratio of the error masking and signalling capability of VTV to that of CMV increases as the magnitude of the value errors in-

| VOTER | $\alpha$ | FNR | | FPR | |
|---|---|---|---|---|---|
| | | (2%) | (10%) | (2%) | (10%) |
| CMV | 0.6 | 0.01 | 0.11 | 5.64 | 27.5 |
| | 0.8 | 0.01 | 0.18 | 5.18 | 26.43 |
| | 1 | 0.02 | 0.21 | 4.64 | 24.61 |
| | 1.2 | 0.02 | 0.22 | 4.3 | 21.82 |
| | 1.4 | 0.25 | 2.12 | 1.09 | 3.49 |
| VTV (Case 1) | 0.6 | 0.01 | 0.11 | 5.64 | 27.5 |
| | 0.8 | 0.01 | 0.18 | 5.18 | 26.43 |
| | 1 | 0.02 | 0.21 | 4.64 | 24.61 |
| | 1.2 | 0.02 | 0.22 | 4.3 | 21.82 |
| | 1.4 | 0.02 | 0.22 | 4.06 | 20.92 |
| VTV (Case 2) | 0.6 | 0.02 | 0.16 | 4.24 | 19.8 |
| | 0.8 | 0.04 | 0.23 | 4 | 19.47 |
| | 1 | 0.13 | 0.76 | 2.89 | 16.27 |
| | 1.2 | 0.16 | 0.84 | 2.57 | 14.98 |
| | 1.4 | 0.19 | 0.85 | 2.47 | 14.66 |

Table 3.5: FPR and FNR of CMV and VTV for various error detector coefficients

crease. Hence, VTV provides better error detection than CMV for value errors of greater magnitude which are originated by timing anomalies.

The goal of using redundancy schemes is to boost the reliability of the system to a level that the system specifications meet the dependability requirements. The evaluations show that for a given redundancy scheme, it is possible to tune its performance by choosing (i) an appropriate $\delta$ (by adjusting the level of clock synchronization) and (ii) an appropriate $\alpha$ that in combination would enable reaching a desired level of the FNR. On the other hand, the FPR needs to be bounded at a certain level so that the recovery actions taken due to the false positives would not jeopardize the real-time requirements.

## 3.2 Fault-Tolerant Scheduling of Tasks With Mixed Criticality

Real-time systems typically have to satisfy complex requirements, mapped to the task attributes, eventually guaranteed by the underlying scheduler. In such systems, due to resource constraints, the error recovery has to be performed in a prioritized way depending on the task criticality levels. Moreover, as the relative criticality levels of the tasks could undergo changes during the evolution/life time of these systems, the designer might have the tedious task of making new schedules to reflect such changes. This is especially relevant in the case of 'system of systems' or component based systems where the integrator needs to make judicious choices for assigning/fine-tuning the priorities for scheduling the tasks of subsystems within the global context. Temporal redundancy techniques are preferred in many embedded applications where it is not feasible to provide spatial redundancy due to weight, space and cost considerations, hence, it is extremely important to devise appropriate methodologies for scheduling real-time tasks under error assumptions so that the exploitation of temporal redundancy does not jeopardize the timeliness requirements on critical tasks.

This section presents a methodology to provide a priori guarantees in fixed priority scheduling (FPS) such that the system will be able to tolerate one error per every critical task instance. The methodology utilizes Integer Linear Programming (ILP) to derive task attributes that guarantee the re-execution of every critical task instance before its deadline, while keeping the associated costs minimized. Unlike many previous works, this method guarantees *all primaries' and all alternates' feasible execution, up to 100% utilization, in FPS* while providing recovery from up to one error per critical task instance, without any on-line computational overhead or major modifications to the underlying scheduler. Additionally, in case the system load permits, non-critical tasks can feasibly coexist with critical ones at high priority levels. The effectiveness of the approach is evaluated, through simulation studies, by comparing it with a well-known fault-tolerant (FT) adaptation of the Rate Monotonic (RM) scheduling policy which is able to provide recovery from a less severe worst-case error scenario, i.e., one error per longest task period.

In this approach, the term 'FT feasibility' of a schedule indicates whether it is guaranteed to meet the deadlines of all critical tasks under

a specified fault hypothesis. The fault-tolerance strategy employed in this approach is the re-execution of the affected task, or execution of an alternate task in the event of errors.

The error-induced additional timing requirements are analyzed at the task instance level and appropriate task execution windows are derived satisfying these requirements. Based on these windows, fixed task priorities are derived using ILP in order to obtain FT feasible schedules. In some cases, e.g., when the fault-tolerance requirements can not be expressed directly by FPS attributes, artifacts are introduced by splitting tasks into instances to obtain a new task set with consistent FPS attributes. The number of artifacts is bounded by the total number of instances in the schedule within the hyperperiod (LCM). The presented approach guarantees to find a solution, i.e., FT feasible FPS attributes, under given assumptions, and is optimal in the sense that it minimizes the number of artifacts, which is the main element of cost.

This approach is highly applicable in safety critical RT systems design, in legacy applications (where one needs to preserve the original scheduler and scheduling policy), during system evolution (where criticalities and priorities could undergo changes), or during subsystem integration (as in embedded software present in Electronic Control Units (ECUs) in automotive applications. For example, in the case of two ECUs, developed with pre-assigned priorities for tasks from specified priority bands, one may want to fine-tune and get a better schedule considering the global context during integration. One can envisage many possible variations to the error model and fault-tolerance strategies. Though the present work does not categorically mention each of them, the approach is designed in such a way as to accommodate future anticipated changes in the error model and fault-tolerance strategies.

### 3.2.1 System Model

This approach assumes a periodic task set, denoted by $\Gamma$, where each task $i$ represents a real-time thread of execution and has a deadline equal to its period. The task set consists of critical and non-critical tasks where the criticality of a task could be seen as a measure of the impact of its correct (or incorrect) functioning on the overall system correctness. Each critical task $i$ has an alternate task, denoted by $\overline{\tau}_i$, with a WCET $\overline{C}_i \leq C_i$ and a deadline $\overline{D}_i = D_i$. This alternate can typically be a re-execution of the same task, a recovery block, an exception handler or

an alternate with imprecise computations.

$\Gamma_c$ represents the subset of critical tasks out of the original task set and $\Gamma_{nc}$ represent the subset of non-critical tasks, so that $\Gamma = \Gamma_c \cup \Gamma_{nc}$. $\overline{\Gamma}_c$ represents the set of critical task alternates. While this framework permits varying criticality levels for tasks, in this section, to simplify the illustration, only binary values are assumed for criticality levels.

For each $j^{th}$ instance of task $i$ ($\tau_i^j$), an *original feasibility window* is defined which is delimited by the original earliest start time $est(\tau_i^j)$ and deadline $D_i^j$ relative to the start of the LCM.

Obviously, the maximum utilization of the original critical tasks together with their alternates can never exceed 100%. This will imply that, during error recovery, execution of non-critical tasks cannot be permitted as it may result in overload conditions. The scheduler is assumed to have adequate support for flagging non-critical tasks as unschedulable during such scenarios, along with appropriate error detection mechanisms in the operating system.

The primary concern is providing schedulability guarantees to all critical tasks in FT real-time systems which employ temporal redundancy for error recovery. The basic assumption here is that the effects of a large variety of transient and intermittent hardware faults can effectively be tolerated by a simple re-execution of the affected task whilst the effects of software design faults could be tolerated by executing an alternate action such as recovery blocks or exception handlers. Both of these situations could be considered as execution of another task (either the primary itself or an alternate) with a specified computation time requirement.

Each error is assumed to adversely affect at most one task at a time and is detected before the termination of the current execution of the affected task instance. This would naturally include error detection before any context switches due to release of a high priority task. Although somewhat pessimistic, this assumption is realistic since in many implementations, errors are detected by acceptance tests which are executed at the end of task execution or by watchdog timers that interrupt the task once it has exhausted its budgeted WCET. In case of tasks communicating via shared resources, acceptance tests are assumed to be performed before passing an output value to another task to avoid error propagations and subsequent domino effects.

### 3.2.2 Methodology

As the original feasibility windows and original priority assignment (if any, e.g., in case of a legacy system) may not express the various fault-tolerance requirements, the first step involves derivation of new feasibility windows for each task instance $\tau_i^j \in \Gamma$ to reflect the fault-tolerance requirements. While executing non-critical tasks in the background can be a safe and straight forward solution, this approach aims to provide non-critical tasks a better service that background scheduling. Hence, depending on the criticality of the original tasks, the new feasibility windows differ as:

1. *Fault-Tolerant* (FT) feasibility windows for critical task instances
2. *Fault-Aware* (FA) feasibility windows for non-critical ones

While critical task instances must execute within their FT feasibility windows to be able to re-execute feasibly upon an error, the derivation of FA feasibility windows has two purposes: to prevent non-critical task instances from interfering with critical ones, i.e., to cause a critical task instance to miss its deadline, while ensuring their execution at high priority levels, i.e., not only executing in the background. Since the size of the FA feasibility windows depend on the size of the FT feasibility windows, first the FT feasibility windows are derived followed by the derivation of the FA feasibility windows. Finally, FPS attributes are derived that ensures the task executions within their newly derived feasibility windows.

In some cases, however, FPS cannot to express all the specified fault-tolerance requirements and the error assumptions with the same priorities for all instances directly. General fault-tolerance requirements may require that instances of a given set of tasks need to be executed in different order on different occasions. Obviously, there exist no valid FPS priority assignment that can achieve these different ordering. This approach detects such situations, and circumvents the problem by splitting a task into its instances. Then, it assigns different priorities to the newly generated "artifact" tasks, the former instances. Key issues in resolving the priority conflicts are the number of artifact tasks created, and the number of priority levels. Depending on how the priority conflicts are resolved, the number of resulting tasks may vary, i.e., based on the size of the periods of the split tasks. This approach uses ILP in order to solve the priority assignment problem, which at the same time minimizes the

number of artifact tasks created in case of priority conflicts. The major steps of the proposed methodology are shown in Figure 3.7.

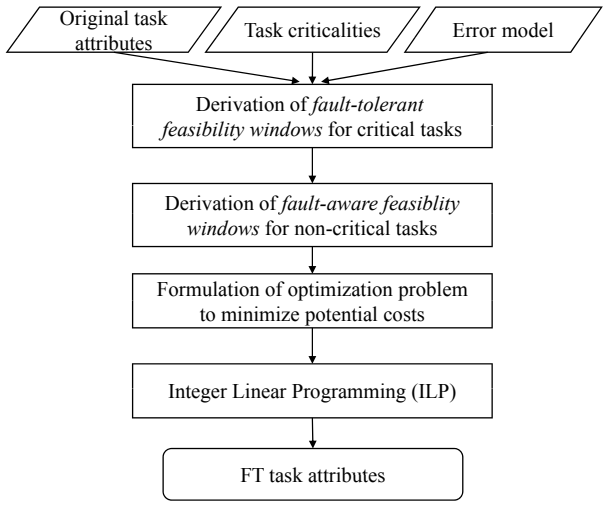


Figure 3.7: Methodology overview - scheduling tasks with mixed criticality

A simple example is used throughout the description of the approach. Let the task set consist of 2 tasks, A and B, where $T_A = 3$, $T_B = 6$, $C_A = 2$ and $C_B = 2$, scheduled according to the RM policy (Figure 3.8), where B is the critical task subject to failures. Here, the earliest start times and the deadlines are represented by up- and down arrows respectively. The fault-tolerance strategy used in this example is re-execution of erroneous task instances.

To be able to re-execute B upon an error occurrence, B must complete before $D_B - C_B$. In this case, B's new deadline will be 4. One possibility is to assign B a higher priority than A. However, by doing so, the first instance of A will always miss its deadline, even in error free scenarios (Figure 3.9). Moreover, raising the priority of critical tasks may not always ensure fault-tolerance in the assumed error scenarios, i.e., one error per task instance, while the processor utilization approaches 100%.

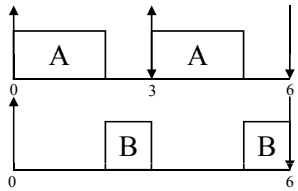


Figure 3.8: Original task set

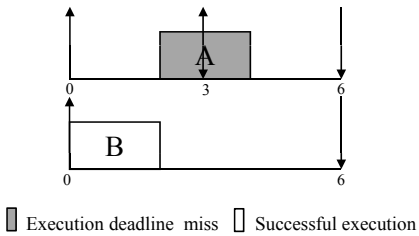


Figure 3.9: Task B fault-tolerant - task A always misses its deadline

The first part of the approach is the derivation of FT and FA feasibility windows for critical and non-critical task instances respectively. FT deadlines for the primary versions of the critical task instances are derived in a way that, in case of a critical task error, an alternate version of that instance can be executed without missing its original deadline. FA deadlines for the non-critical task instances are then derived so that the provided fault-tolerance for the critical ones is not jeopardized. During these steps the goal is to keep the FT and FA deadlines as late as possible in order to maximize the flexibility for the second part of the approach, which is the FPS attribute assignment using an ILP solver.

**Derivation of FT deadlines:** The aim of this step is to reserve sufficient resources for the executions of the critical task alternates in the schedule. While one can use any method to achieve that, the goal of this approach is to provide guarantees in scenarios where the processor utilization can reach 100%. Thus, the approach proposed by Chetto and Chetto [26] is used for finding the latest possible start of executions for

critical task alternates. Specifically, by scheduling only the set of critical tasks $\Gamma_c$ and their alternates $\overline{\Gamma}_c$ according to Chetto and Chetto's approach, the FT deadline for each critical task instance, $D_i^j(FT)$, is defined as being equal to the latest start time of its alternate $\overline{\tau}_i^j$.

In this way sufficient resources are reserved for each critical task instance alternate, assuming that the cumulative processor utilization of the primaries together with their alternates does not exceed 100% over LCM. In the example, the FT deadline of B is 4.

**Derivation of FA deadlines:** The main purpose of FA deadline derivation to non-critical task instances is to protect critical ones from being adversely affected. As a part of recovery action upon errors, an underlying on-line mechanism checks if there is enough time left for the non-critical task instances to complete before their new deadlines. If not, these instances are not executed.

The same process as in FT deadline derivation is applied to derive the FA deadlines, on the set of non-critical tasks, $\Gamma_{nc}$, but *in the remaining slack* after the critical task primaries are scheduled to execute as late as possible. This is done so due to two reasons, (i) to prevent non-critical tasks from delaying the execution of critical primaries beyond their FT deadlines, and (ii) to allow non-critical tasks to be executed at high priority levels.

In the example, the derived FT and FA deadlines are illustrated in Figure 3.10).

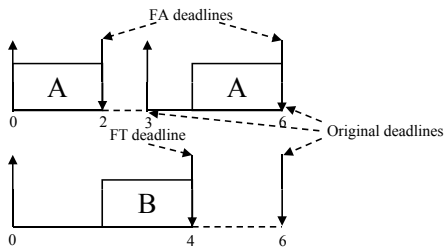


Figure 3.10: FT and FA task deadlines

In some cases, finding valid FA deadlines for some non-critical task instances may not be possible. If $D_i^j(FA) - est(\tau_i^j) < C_i^j$, the FA dead-

line, $D_i^j(FA)$, is denoted as *invalid*. This scenario could occur since the task set consists now of tasks with deadlines less than periods. In such a case, the original deadline is kept as is, and task is given a background priority, i.e., a lower priority than any other critical task, and any other non-critical task with a *valid FA deadline*.

### FPS Attribute Assignment

At this step, priority relations are identified for each point in time $t_k$ at which at least one task instance is released for the task set with the new FT and FA deadlines. This is done by deriving priority inequalities between instances to ensure their execution within their derived FT and FA feasibility windows. By solving the inequalities, the FPS attributes are obtained that guarantees meeting the specified fault-tolerance requirements.

At this stage, the task model consists of four types of task instances: critical task instances consisting of primaries $\Gamma_c$ and alternates $\overline{\Gamma}_c$, and non-critical task instances with and without valid FA deadlines, $\Gamma_{nc} = \Gamma_{nc}^{FA} \cup \Gamma_{nc}^{non\_FA}$.

For every $t_k \in [0, LCM)$ where $t_k$ equals the release time of at least one task instance, a subset of the task set $\Gamma_{t_k} \subseteq \Gamma$ is considered that consists of:

1. $\{current\_instances\}_{t_k}$ - instances $\tau_i^j$ of tasks $\tau_i$, released *at* the time $t_k$: $est(\tau_i^j) = t_k$
2. $\{interfering\_instances\}_{t_k}$ - instances $\tau_s^q$ of task $\tau_s$ released *before* $t_k$ but potentially executing *after* $t_k$: $est(\tau_s^q) < t_k < D_s^q$, where

$$D_s^q = \begin{cases} D_s^q(FT), & if\ \tau_s^q \in \Gamma_c \\ \overline{D}_s^q(FT), & if\ \tau_s^q \in \overline{\Gamma}_c \\ D_s^q(FA), & if\ \tau_s^q \in \Gamma_{nc}^{FA} \\ D_s^q, & if\ \tau_s^q \in \Gamma_{nc}^{non\_FA} \end{cases}$$

Priority relations are derived within each subset $\Gamma_{t_k}$ based on the derived FT and FA deadlines, such that the instance with the shortest relative deadline will get the highest priority in each inequality.

$\forall t_k, \forall \tau_i^j, \tau_s^q \in \Gamma_{t_k}$, where $i \neq s$.

1. if $\tau_i^j, \tau_s^q \in \Gamma_c \cup \Gamma_{nc}^{FA}$, or if $\tau_i^j, \tau_s^q \in \Gamma_{nc}^{non\_FA}$

$$P_i^j > P_s^q,\ where\ D_i^j < D_s^q$$

2. if $\tau_i^j \in \Gamma_c \cup \Gamma_{nc}^{FA}$ and $\tau_s^q \in \Gamma_{nc}^{non\_FA}$

$$P_i^j > P_s^q$$

In tie situations, i.e. when the instances $\tau_i^j$ and $\tau_s^q$ have same deadlines, the task with the earliest start time is given a higher priority. In cases where even the earliest start times are equal, the priority inequalities are derived consistently.

The goal of the approach is to provide tasks with fixed offsets and fixed priorities. When the priority inequalities are solved, it may happen that different priorities need to be assigned to different instances of the same task. These cases cannot be expressed directly with fixed priorities and are the sources for *priority assignment conflicts*. Such a conflict is resolved by splitting the task with the inconsistent priority assignment into a number of new periodic tasks with different priorities. The instances of the new tasks comprise all instances of the original task. The priorities that resolve the conflicts and the splits that yield the smallest number of FPS tasks is found by ILP.

### ILP Formulation

The goal of the attribute assignment problem is to find the minimum number of tasks together with their integer priorities, that fulfil the priority relations derived so far. As mentioned above, each task of the task set is either one of the original tasks or an artifact task created from one of the instances of an original task selected for splitting. The goal function $G$ to be minimized computes the number of tasks to be used in the FPS scheduler.

$$G = n + \sum_{i=1}^{n}(n_i - 1)b_i + \sum_{i=1}^{n}\sum_{j=1}^{n_i}\bar{b}_i^j$$

where $n$ is the number of original tasks, $n_i$ is the number of instances of $\tau_i$ over LCM, $b_i$ is a binary integer variable that indicates if $\tau_i$ needs to be split into its instances and $\bar{b}_i^j$ is a binary integer variable that indicates if the alternate of the critical task instance $\tau_i^j$ can be executed at the same priority as it primary.

The constraints of the ILP problem reflect the restrictions on the task priorities as imposed by scheduling problem. To account for the

case of priority conflicts, i.e., when tasks have to be split, the constraints between the original tasks, including task re-executions, are extended to include the constraints of the artifact tasks. Thus each priority relation $P_i^j > P_p^q$ between two tasks is translated into an ILP constraint:

$$p_i + p_i^j > p_p + p_p^q,$$

where the variables $p_i$ and $p_p$ stand for the priorities of the FPS tasks representing the original tasks or alternates, $\tau_i$ and $\tau_p$ respectively, and $p_i^j$, $p_p^q$ stand for the priorities of the artifact tasks, $\tau_i^j$ and $\tau_p^q$, in case it is necessary to split the original tasks or to run an alternate at a different priority. Although this may look like a constraint between four tasks ($\tau_i$, $\tau_i^j$, $\tau_p$, $\tau_p^q$) it is in fact a constraint between two tasks – for each task only its original ($\tau_i$ resp. $\tau_p$) or its artifact tasks ($\tau_i^j$ resp. $\tau_p^q$) can exist in the FPS schedule. In case the priority relation involves task re-executions, e g., $\overline{P}_i^j > P_p^q$ the translated constraint is:

$$\overline{p}_i^j > p_p + p_p^q,$$

The goal is to be able to re-execute a task instance without changing its priority.

A further set of constraints for each task ensures that only either the original tasks or their instances (artifacts) are assigned valid priorities (greater than 0) by the ILP solver. All other priorities are set to zero.

$$\begin{aligned} p_i &\leq (1 - b_i)M \\ \forall j : p_i^j &\leq b_i M \end{aligned}$$

While both primaries and alternates can coexist at different valid priorities, the last set of constraints aims to yield same priorities for both of them. Otherwise, the alternate will be assigned a different priority than its primary.

$$\left|(p_i + p_i^j) - \overline{p}_i^j\right| \leq \overline{b}_i^j M$$

In these constraints $M$ is a large number, larger than the total number of instances and alternates in the original task set. The variable $b_i$ for task $i$, which also occurs in the goal function, indicates if $\tau_i$ has to be split, i.e., $b_i$ allows only a task or its artifact tasks to be assigned valid priorities. On the other hand, the variable $\overline{b}_i^j$ is a binary variable that indicates if the alternate of $\tau_i^j$ can be scheduled at the same priority as

its primary. Since the goal function associates a penalty for each $b_i$ and $\overline{b}_i^j$ that has to be set to 1, the ILP problem indeed searches for a solution that produces a minimum number of task splits. The constraints on the binary variables complete the ILP constraints:

$$\forall i, j: \; b_i, \overline{b}_i^j \leq 1$$

The solution of the ILP problem yields the total number of tasks as the result of the goal function. The values of the variables represent a priority assignment for tasks and artifact tasks that satisfies the priority relations of the scheduling problem.

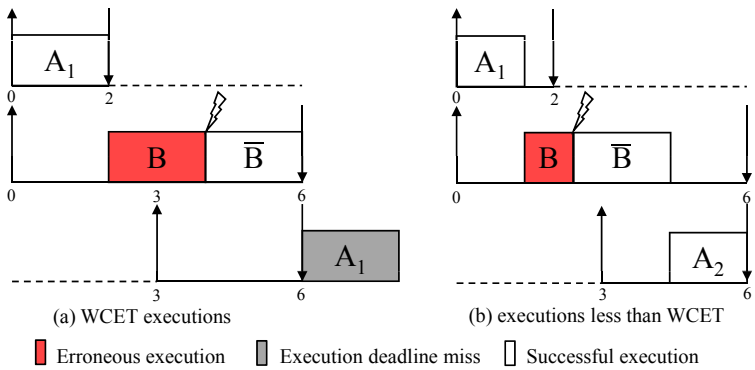


Figure 3.11: FT feasible task set

### Periods and Offsets

The priority assignment to the tasks by the ILP-solver is followed by the assignment of periods and offsets. Based on the information provided by the solver, periods and offsets are assigned to each task in order to ensure the run time execution under FPS within their respective FT feasibility windows, as following:

$$\begin{aligned} for \quad & 1 \leq i \leq n \\ & T_i = \frac{LCM}{n_i} \\ & O_i = est(\tau_i^1)) \end{aligned}$$

The final set of tasks executing under FPS is presented in Figure 3.11. A1 has the highest priority and A2 the lowest. In Figure 3.11 (a), the tasks execute the worst-case scenario, i.e., task execution equal to WCET and errors occurring at the end of the executions. In this case, A2 will be shed by the scheduler due to the system overload. However, at run-time, tasks will most likely execute for less than their WCETs. In such scenarios, B can feasibly re-execute as well as the non critical tasks A1 and A2 can complete before their deadlines (Figure 3.11 (b)).

### 3.2.3 Example

The approach is illustrated by an example where the task set shown in Table 3.6 is scheduled by the RM scheduling policy as shown in Figure 3.12.

| Task | T | C | P | Criticality |
|---|---|---|---|---|
| A | 3 | 1 | 3 (highest) | 0 (non-critical) |
| B | 4 | 1 | 2 | 1 |
| C | 12 | 3 | 1 | 1 |

Table 3.6: Original task set

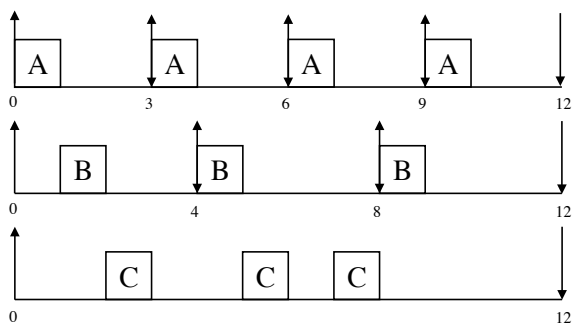


Figure 3.12: Original RM schedule

In this example, B and C are assumed to be the critical tasks. Here, RM priority assignment cannot guarantee fault-tolerance execution of

every critical task instance. For instance, if all the instances of B are hit by errors and have to be re-executed, the primary version of C will always miss its deadline. (3.13)

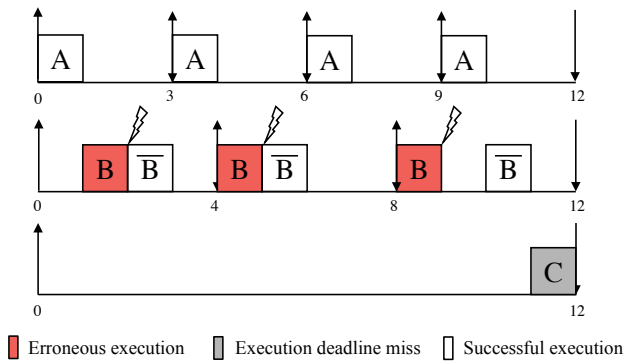


Figure 3.13: RM schedule under errors - C misses its deadline

The first step is to derive FT feasibility windows for the critical tasks. The method proposed by Chetto and Chetto [26] is used to calculate the latest possible start of execution for critical tasks and alternates (Figure 3.14). As previously mentioned, the earliest start times and the deadlines are represented by up- and down arrows respectively. The dashed blocks represent the re-execution of the critical tasks instances. Accordingly, the FT feasibility windows for the critical tasks are presented in Figure 3.15.

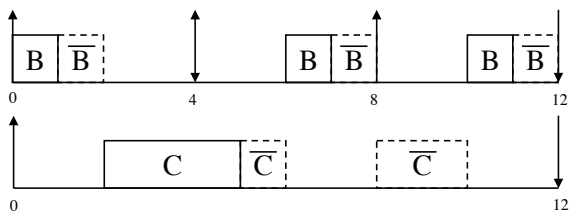


Figure 3.14: Latest possible executions for critical tasks and alternates

At this point, FA feasibility windows are derived for non-critical task

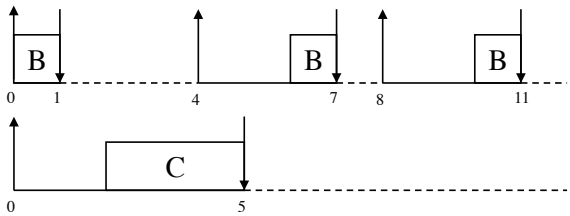


Figure 3.15: FT feasibility windows for critical tasks (B and C)

instances (in this example, for the instances of A), by scheduling them *as late as possible* [26], *together* with the critical ones and associated FT feasibility windows. The resulting FA feasibility windows are shown in Figure 3.16.

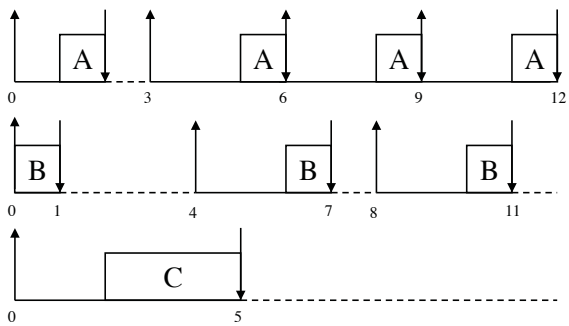


Figure 3.16: FA feasibility windows for the non-critical task (A)

Based on the derived FT and FA feasibility windows for the critical and non-critical tasks respectively, the sets of current and interfering instances are analyzed for each release time in the task set and the priority relations between the instances are derived as described in Section 3.2.2. The resulting priority inequalities are presented in Table 3.7.

Next, the optimization problem is formulated as described in Section

| $t_k$ | $\{\ currentinst.\ \}_{t_k}$ | $\{\ interferinginst.\ \}_{t_k}$ | inequalities |
|---|---|---|---|
| 0 | $A^1, B^1, \overline{B}^1, C^1, \overline{C}^1$ | None | $P_B^1 > P_A^1$<br>$\overline{P}_B^1 > P_C^1$<br>$P_A^1 > P_C^1$ |
| 3 | $A^2$ | $C^1$ | $P_C^1 > P_A^2$ |
| 4 | $B^2, \overline{B}^2$ | $A^2, C^1, \overline{C}^1$ | $P_C^1 > P_A^2$<br>$P_A^2 > P_B^2$<br>$P_B^2 > \overline{P}_C^1$<br>$\overline{P}_B^2 > \overline{P}_C^1$ |
| 6 | $A^3$ | $B^2, \overline{B}^2\overline{C}^1$ | $P_B^2 > P_A^3$<br>$P_B^2 > \overline{P}_C^1$<br>$\overline{P}_B^2 > \overline{P}_C^1$ |
| 8 | $B^3, \overline{B}^3$ | $A^3, \overline{C}^1$ | $P_A^3 > P_B^3$<br>$\overline{P}_C^1 > P_B^3$<br>$\overline{P}_C^1 > \overline{P}_B^3$ |
| 9 | $A^4$ | $B^3, \overline{B}^3, \overline{C}^1$ | $P_B^3 > P_A^4$<br>$\overline{P}_C^1 > P_B^3$<br>$\overline{P}_C^1 > \overline{P}_B^3$ |

Table 3.7: Derivation of inequalities

3.2.2. The terms in the ILP goal function, i.e.,

$$G = n + \sum_{i=1}^{n}(n_i - 1)b_i + \sum_{i=1}^{n}\sum_{j=1}^{n_i}\overline{b}_i^j$$

are:

$$n = 3$$
$$\sum_{i=1}^{n}(n_i - 1)b_i = 3b_A + 2b_B + 0b_C,\ and$$
$$\sum_{i=1}^{n}\sum_{j=1}^{n_i}\overline{b}_i^j = \overline{b}_B^1 + \overline{b}_B^2 + \overline{b}_B^3 + \overline{b}_C^1$$

subject to the constraints derived from the priority inequalities. For example, $P_B^1 > P_A^1$ is translated into the constraint C1:

$$C1 : p_B + p_B^1 > p_B + p_B^1$$

The ILP solver provides a set of FT tasks schedulable by FPS to which periods and offsets are assigned, as described in Section 3.2.2. The resulting task set is presented in Table 3.8.

| $\overline{\tau_i}$ | T | C | O | D | P |
|---|---|---|---|---|---|
| $A^1$ | 12 | 1 | 0 | 3 | 7 |
| $A^2$ | 12 | 1 | 3 | 6 | 5 |
| $A^3$ | 12 | 1 | 3 | 9 | 2 |
| $A^4$ | 12 | 1 | 9 | 12 | 0 |
| $B^1$ | 12 | 1 | 0 | 1 | 8 (highest) |
| $B^2$ | 12 | 1 | 4 | 7 | 4 |
| $B^3$ | 12 | 1 | 8 | 11 | 1 |
| $C$ | 12 | 3 | 0 | 5 | 6 |
| $\overline{C}$ | 12 | 3 | 0 | 10 | 3 |

Table 3.8: FT FPS tasks

In this example, since the utilization is already 100% without any errors, the LP solver yields a solution consisting of 9 tasks, i.e., 8 from the original tasks instances, and one additional consisting of the alternate task belonging to C that has to be executed at a lower priority than C. The resulting task set is directly schedulable by the original scheduler while the critical tasks can tolerate one error per instance. Non-critical tasks are allowed to be executed at a higher priority than critical ones within their derived FA feasibility windows without jeopardizing the FT feasibility of the critical tasks. In case of an error, however, non-critical tasks will be suspended by the underlying scheduler until the erroneous task has been re-executed.

### 3.2.4 Evaluation

In real-time systems where both critical and non-critical tasks co-exist, missing a single deadline of a critical task instance can result in more severe consequences than missing several deadlines of non-critical task instances. Based on this point of view, the primary success criteria is defined as the percentage of successfully met critical deadlines in the evaluation. Meeting the deadlines of non-critical task instances is assumed to be the secondary success criteria and amount of deadline misses of such tasks can be seen as the cost of meeting more critical deadlines.

In this section the performance of the presented approach is evaluated in comparison with the well-known FT adaptation of the RM scheduling policy. The evaluation is performed by simulating the worst-case scenario of detecting one error at the end of each critical task instance execution and then re-executing the task instance. 2000 task sets are generated where the total number of tasks in every task set is 10 and the number of critical tasks is varying randomly from 1 to 10. The LCM is chosen randomly between 20 and 200 time units which seems sufficient to compare the behavior of two approaches. One reason of choosing the constant value 10 for the number of tasks is related to the LCM range. With this constant value and the given LCM range, tasks sets are created with a wide range of total utilization from 0.5 to 1 even when the LCM is selected as minimum. Furthermore, the limited number of tasks increases the traceability of the scheduling decisions made by the approaches under observation. After finding the LCM, task periods are randomly chosen among the divisors of LCM. Randomization is realized by Mersenne Twister pseudorandom number generator with 32-bit word length [70]. Total processor utilizations of the task sets are kept within intervals of 0.1 for every group of 500 task sets starting from the range 0.6-0.7. Within each group, processor utilizations of the critical tasks are also kept within intervals of 0.1 for every sub-group of 100 task sets varying between the range 0-0.1 and 0.4-0.5. The average execution time of the implementation to create FT feasible task attributes is around 100 milliseconds on a 1GHz PC when a task set generated as described above is used as an input. Figures 3.17 to 3.20 show the average percentage of successfully met deadlines with respect to critical task utilization.

Each figure shows a different range of total processor utilization starting from the range 0.6-0.7. As the processor utilization increases, it can be seen that the success of the presented approach also increases with the cost of missing more non-critical deadlines.

In the processor utilization range 0.6-0.7, the presented approach starts to give better results than RM when critical task utilization is above 0.3 (Figure 3.17). In the range 0.8-0.9 this threshold decreases to 0.2 (Figure 3.19). When the processor utilization is between 0.9 and 1 (Figure 3.20), critical task instances scheduled by RM start to miss their deadlines even the critical task utilization very low while the presented approach still guarantees meeting all the critical deadlines.

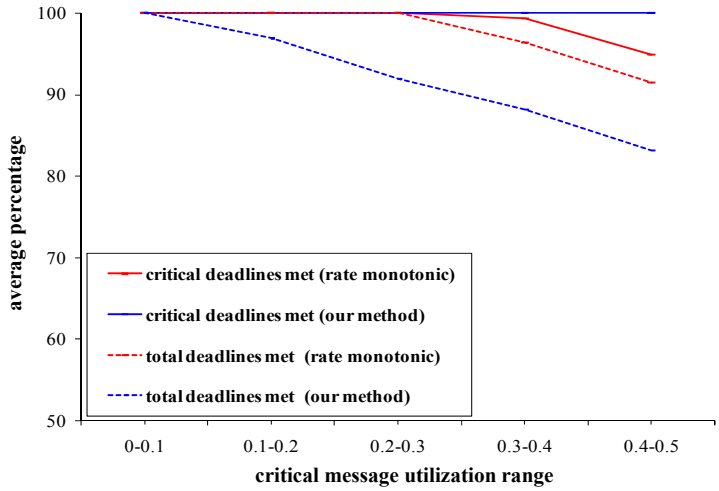


Figure 3.17: Total processor utilization between 0.6 - 0.7

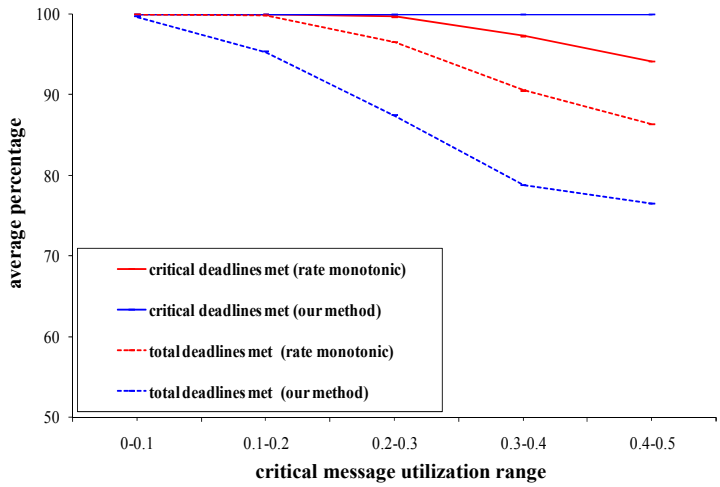


Figure 3.18: Total processor utilization between 0.7 - 0.8

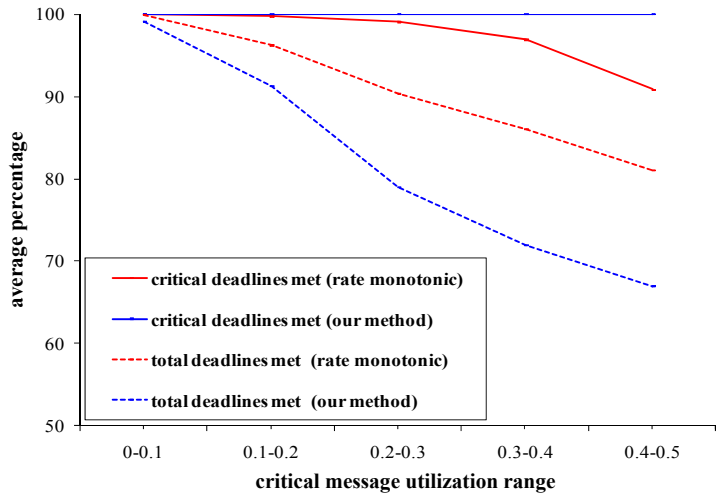


Figure 3.19: Total processor utilization between 0.8 - 0.9

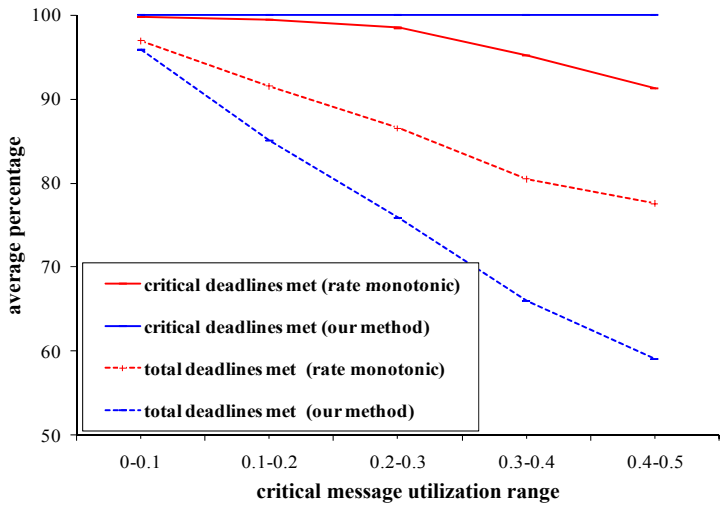


Figure 3.20: Total processor utilization between 0.9 - 1

## 3.3 Fault-Tolerant Scheduling of Messages with Mixed Criticality

Dependable communication is becoming a critical factor due to the pervasive usage of networked embedded systems that increasingly interact with human lives in many real-time applications. Similar to task scheduling, the communication system often needs to transmit messages with different criticality levels as well. Mixed criticality levels of messages imply different reliability requirements, e.g., non critical messages do not need to be retransmitted at all while critical ones require a specified level of fault-tolerance. Hence, the native message retransmission mechanism, that assumes that all messages are equally critical, can not handle the above scenarios in an efficient way, may result in the inability to meet user defined reliability requirements, and, in some cases, even leads to violation of timing requirements. Therefore there is a need for mechanisms to address the multi-criticality levels of messages.

The approach presented in this section enables the provision of appropriate guarantees in Controller Area Network (CAN) scheduling of messages with mixed criticality levels which involves identifying FT feasibility windows of execution for critical messages, and off-line derivation of optimal message priorities that fulfil the user specified level of fault-tolerance.

### 3.3.1 System Model

This approach assumes a distributed real-time architecture consisting of sensors, actuators and processing nodes communicating over CAN. The communication is performed via a set of periodic messages, $\Gamma = \{M_1, M_2, \ldots, M_n\}$, with mixed criticality levels. The criticality of a message indicates the severity of the consequences caused by its failure and corresponds to the amount of resources allocated for error recovery in terms of guaranteed re-transmissions. The basic assumption here is that the effects of a large variety of transient and intermittent hardware faults can effectively be tolerated by a simple re-transmission of the affected frames. An error is assumed to adversely affect at most one message frame at a time and be detected by the nodes in the network. $\Gamma_c$ represents the subset of critical messages out of the original message set and $\Gamma_{nc}$ represents the subset of non-critical messages, so that $\Gamma = \Gamma_c \cup \Gamma_{nc}$.

In this approach, single-shot transmission on CAN is assumed, i.e., the automatic re-transmission mechanism is disabled. This particular feature is available on a number of commercial controllers, e.g., Atmel T89C51CCO2, Philips SJA1000 and Microchip MCP2515.

A message consists of $m$ frames, $m \geq 1$, and the network communication is non-preemptive during the frame transmissions. However, messages composed of more than 2 frames can preempt each other at frame boundaries. Additionally, the non-preemptiveness of message frames may cause a higher priority message to be blocked by a lower priority message for at most one frame length, if the high priority message is released during the transmission of a lower priority frame. This priority inversion phenomenon can affect all messages except the lowest priority one, and only once per message period, before the transmission of the first message frame [29].

Each CAN message, $M_i$, has a unique priority $P_i$ (defined by the message identifier), a period $T_i$, a relative deadline $D_i$, which is assumed to be equal to the period, and a re-transmission requirement $r_i$ which represents the number of frames that are required to be guaranteed for re-transmission. Note that for non-critical messages $r_i = 0$.

In an error free scenario, the worst-case transmission time $C_i$ of message $i$ is given by:

$$C_i = m_i \ f^{max} \ \tau_{bit} \tag{3.1}$$

where $f^{max}$ is the worst-case frame size.

For each *message instance*, $M_i^j$, an *original feasibility window* is defined, delimited by its original earliest start time $est(M_i^j)$ and deadline $D_i^j$ relative to the start of the LCM.

Obviously, the maximum utilization of the original critical messages together with the re-transmissions can never exceed 100%. This will imply that, during error recovery, transmission of non-critical messages cannot be permitted as it may result in overload conditions, except in situations where a non-critical frame is blocking a higher priority critical message due to priority inversion. Hence, upon receiving an error frame, the nodes transmitting non-critical messages suspend their transmissions until the end of the failed message's period. This will require that all nodes transmitting non-critical messages have knowledge about critical messages' periods.

### 3.3.2 Methodology

The first step of the methodology involves the derivation of new feasibility windows for each message instance $M_i^j \in \Gamma$, to guarantee meeting the FT requirements. While transmitting non-critical messages using a background priority band can be a safe and straightforward solution, the goal of this approach is to provide non-critical messages a better service than background scheduling. Hence, depending on the criticality of the original set of messages, the new feasibility windows differ as:

1. *Fault-Tolerant* (FT) feasibility windows for critical messages
2. *Fault-Aware* (FA) feasibility windows for non-critical messages

While critical messages need to be entirely transmitted within their FT feasibility windows to be able to be feasibly retransmitted upon an error, according to the reliability requirements, the derivation of FA feasibility windows has two purposes: (i) to prevent non-critical messages from interfering with critical ones, by causing a critical message to miss its deadline, while (ii) enabling the transmission of the non-critical messages at high priority levels in error free situations. Since the size of the FA feasibility windows depends on the size of the FT feasibility windows, first the FT feasibility windows are derived, followed by the derivation of the FA feasibility windows. Then, the message priorities are derived that ensure the message transmissions within the newly derived feasibility windows.

A high priority message can be blocked by a low priority message at most one frame at the beginning of its transmission. Since the derivation of the FT/FA feasibility windows requires knowledge about the worst-case message sizes, the blocking frame needs to be accounted for in every message transmission by adding one additional frame during the analysis.

In some cases, however, a fixed priority scheme cannot express all the assumed FT requirements. General FT requirements may require that instances of a given set of periodic messages needs to be transmitted in different order on different occasions. Obviously, there exists no valid fixed priority assignment that can achieve such an ordering. The presented approach proposes a priority allocation scheme at message instance level by using ILP to guarantee the message transmissions within their FT/FA Feasibility Windows that efficiently utilizes the resources while minimizing the priority levels. The major steps of the proposed methodology are shown in Figure 3.21.

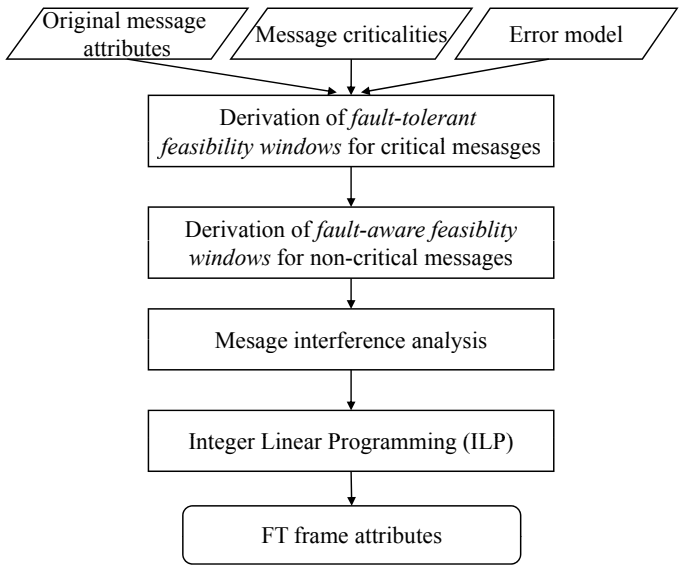


Figure 3.21: Methodology overview - scheduling messages with mixed criticality

A simple example is used throughout the description of the approach. Let the message set consist of 2 messages, A and B, sent from 2 nodes, N1 and N2, where $T_A = 8,\ T_B = 16,\ m_A = 3$ and $m_B = 6$, i.e., message A is allocated to 3 frames that need to be transmitted during one period and message B is allocated to 6 frames. Moreover, let message B be the only critical message with a re-transmission requirement $r_B = 5$ frames. The transmission scenario is illustrated in Figure 3.22) assuming that the message set scheduled on CAN according to the RM scheduling policy. To ease the readability, in this example, the blocking frames are assumed to be included in the size of the messages. The earliest start times and the deadlines are represented by up- and down arrows respectively.

To be able to re-transmit 5 frames before its deadline, B must complete before $D_B - r_B$. In this case, B's new deadline will be 11. One possibility is to assign B a higher priority than A. However, by doing so, the first instance of A will always miss its deadline, even in error free scenarios (Figure 3.23). Moreover, this solution is not useful if a larger

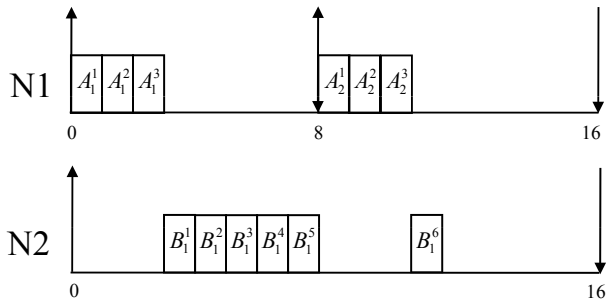


Figure 3.22: Original message set

number of critical messages need to be feasibly transmitted on the bus.

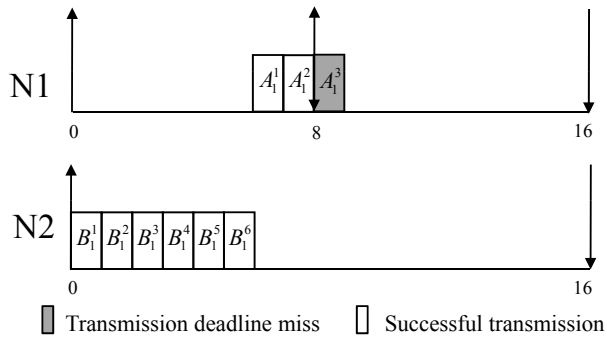


Figure 3.23: Message B fault-tolerant - message A always misses its deadline

### Derivation of FT and FA Feasibility Windows

The first part of the approach is the derivation of FT and FA feasibility windows for critical and non-critical message instances respectively. The first step is the derivation of the FT deadlines for the critical message instances in a way that, in case of errors, erroneous frames can be retransmitted before the original message deadline. Then, FA deadlines for

the non-critical message instances are derived so that the provided fault-tolerance guarantees for the critical ones are not jeopardized. During these steps the goal is to keep the FT and FA deadlines as late as possible in order to maximize the flexibility for the second part of the approach, which is the priority assignment using an ILP solver.

**Derivation of FT deadlines:** The aim of this step is to reserve sufficient resources for the re-transmission of erroneous frames in the schedule. The main goal is to provide guarantees while maximizing the bus utilization. To do so, the approach proposed by Chetto and Chetto [26] is used to schedule all the critical messages and their worst-case retransmissions as late as possible. Then, FT deadlines are determined for each critical message instance, $D_i^j(FT)$, as the latest start time of each critical task's first re-transmitted frame instance. In this way sufficient resources are reserved for the recovery of each critical message instance, assuming that the cumulative resource utilization of the critical messages and re-transmissions, including the blocking from lower priority messages, does not exceed 100% over LCM. In this example, the FT deadline of message B is 11.

**Derivation of FA deadlines:** The aim of this step is to provide FA deadlines to non-critical message instances to protect critical ones from being adversely affected. However, as a part of recovery action upon errors, the sending node should check if there is enough time left for the non-critical messages to be sent before their new deadlines. If not, the message is not transmitted.

To derive the FA deadlines, the same process as in FT deadline derivation is applied, on the set of non-critical messages but *in the remaining slack* after the critical messages (without re-transmissions) are scheduled to be transmitted as late as possible. This is done so due to two reasons, (i) to prevent non-critical messages from delaying the transmission of critical messages beyond their FT deadlines in case of critical frame failures, and (ii) to allow non-critical messages to be transmitted at high priority levels in error free scenarios. In this example, the derived FT and FA deadlines are illustrated in Figure 3.24, where the FA deadlines for the instances of A are 5 and 16 respectively.

In some cases, finding valid FA deadlines for some non-critical messages instances may not be possible. If $D_i^j(FA) - est(M_i^j) < C_i^j$, the FA

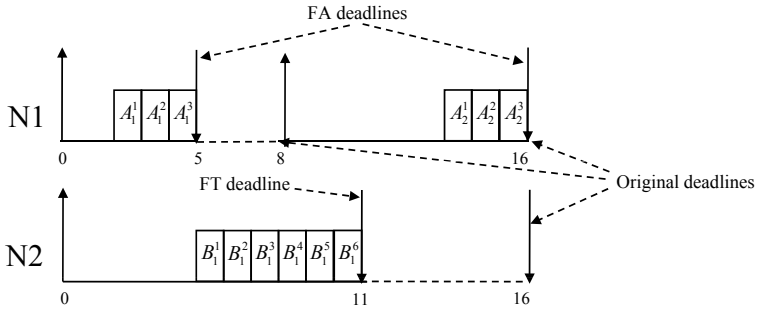


Figure 3.24: FT and FA message deadlines

deadline, $D_i^j(FA)$, is marked as be *invalid* . This scenario could occur since the messages now may have deadlines less than periods after the derivation of FT deadlines. In such a case, the original deadline is kept as is, and the priority assignment mechanism assigns the non-critical message a background priority, i.e., lower than any other critical message, and any other non-critical message with a *valid FA deadline*.

### FPS Attribute Assignment

At this step, priority relations are identified for each point in time $t_k$ at which at least one message instance (i.e., the first frame of the message) is released on the bus and the priority inequalities between messages are derived to ensure their transmission within their derived FT and FA feasibility windows. By solving the inequalities, the scheduling attributes are obtained that guarantees meeting the specified FT requirements for the message set $\Gamma_{FPS}$.

At this point, the message set consists of four types of message frames: critical messages consisting of primary frames $\Gamma_c$ and re-transmitted frames $\overline{\Gamma}_c$, and non-critical messages, consisting of non-critical frames, with and/or without valid FA deadlines, $\Gamma_{nc} = \Gamma_{nc}^{FA} \cup \Gamma_{nc}^{non\_FA}$. For every $t_k \in [0, LCM)$ where $t_k$ equals the release time of at least one message instance, a subset of the message set $\Gamma_{t_k} \subseteq \Gamma$ is considered that consists of:

1. $\{current\_instances\}_{t_k}$ - instances of message $i$, released *at* the time $t_k$: $est(M_i^j) = t_k$

2. $\{interfering\_instances\}_{t_k}$ - instances of message $s$ released *before* $t_k$ but potentially executing *after* $t_k$: $est(M_s^q) < t_k < D_s^q$, where

$$D_s^q = \begin{cases} D_s^q(FT), & if\ M_s^q \in \Gamma_c \\ \overline{D}_s^q(FT), & if\ M_s^q \in \overline{\Gamma}_c \\ D_s^q(FA), & if\ M_s^q \in \Gamma_{nc}^{FA} \\ D_s^q, & if\ M_s^q \in \Gamma_{nc}^{non\_FA} \end{cases}$$

Priority relations within each subset $\Gamma_{t_k}$ based on the derived FT and FA deadlines, such that the message with the shortest relative deadline will get the highest priority in each inequality:

$\forall t_k, \forall M_i^j, M_s^q \in \Gamma_{t_k}$, where $i \neq s$:

1. if $M_i^j, M_s^q \in \Gamma_c \cup \Gamma_{nc}^{FA}$, or if $M_i^j, M_s^q \in \Gamma_{nc}^{non\_FA}$

$$P_i^j > P_s^q,\ where\ D_i^j < D_s^q$$

2. if $M_i^j \in \Gamma_c \cup \Gamma_{nc}^{FA}$ and $M_s^q \in \Gamma_{nc}^{non\_FA}$

$$P_i^j > P_s^q$$

In tie situations, i.e., when the message instances $M_i^j$ and $M_s^q$ have same deadlines, the message with the earliest start time is given a higher priority. In cases where even the earliest start times are equal, the priority inequalities are derived consistently.

The goal of this approach is to provide fixed priorities to all messages. When the derived priority inequalities are solved, however, it may happen that different instances of the same message need to be assigned different priorities, due to the Earliest Deadline First (EDF) heuristic used in the approach. These cases cannot be expressed directly with fixed priorities and are the sources for *priority assignment conflicts*. This issue is resolved by splitting the messages with inconsistent priority assignments into a number of new periodic messages with different priorities. The new message instances comprise all instances of the original set of messages. As a major concern is the number of priorities that may increase, ILP is used to find the priorities and the splits that yield the lowest number of messages that satisfy the inequalities, and implicitly the lowest number of priority levels. A full description of the ILP problem formulation that is adapted to CAN scheduling is found in the previous section.

A major difference in CAN scheduling compared to task scheduling on processors, is that frames are re-transmitted as soon they are identified as erroneous, rather than after the transmission of the whole message. Hence, a message consisting of $m$ primary frames and $r$ re-transmitted frames may need to be transited at a priority level $p$ the first $m$ frames, and at a priority $p'$ for the rest $r$ frames. However, ILP will make sure that, if possible, $p = p'$.

The final set of messages feasibly scheduled on CAN is presented in Figure 3.25. A1 has the highest priority and A2 the lowest. In Figure 3.25, maximum number of re-transmissions are performed upon transmission errors. In this case, due to the overload, A2 will be either shed by the scheduler or only partially transmitted, i.e, 2 out of 3 frames, if the message validity is still acceptable.

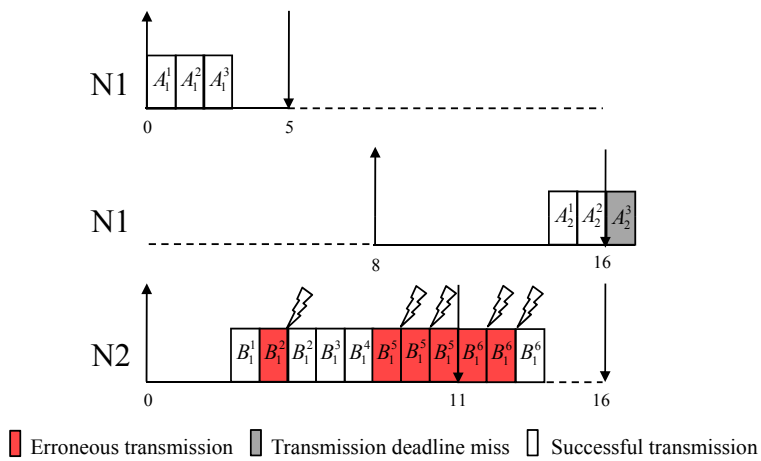


Figure 3.25: FT feasible message set

### 3.3.3 Evaluation

In network communications where both critical and non-critical messages co-exist, missing a single deadline of a critical message instance can result in more severe consequences than missing several deadlines of non-critical message instances. Based on this point of view, the primary success

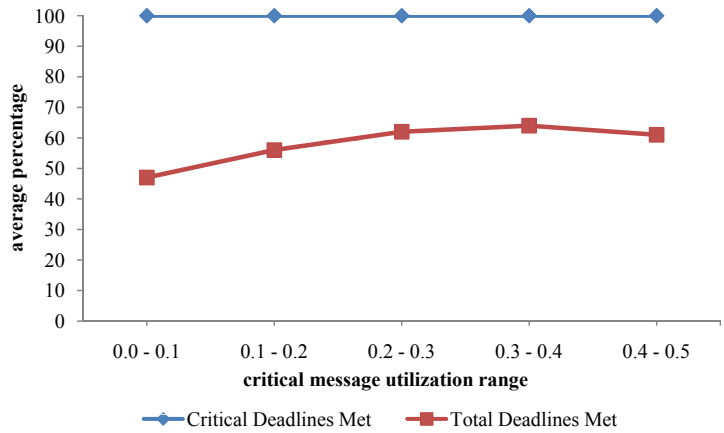


Figure 3.26: Worst-case error scenario - total utilization 0.4 - 0.6

criteria is selected as the percentage of successfully met critical deadlines (including re-transmissions) in the evaluations. Meeting the deadlines of non-critical message instances is assumed to be the secondary success criteria and the amount of deadline misses of such tasks can be seen as the cost for guaranteeing all critical deadlines together with their re-transmissions.

In this section the performance of the presented approach is evaluated under different error scenarios. First, the worst-case error occurrence scenario is simulated that leads to the maximum number of re-transmitted frames, according to the reliability specifications per each critical message. In the next series of runs, the scenario where every other message instance is hit by errors is simulated. This shows the improvement on the non-critical message performance in a less than worst-case error scenario. In all cases, however, the simulation results show that the all critical message frames are feasibly re-transmitted before their deadlines, according to their user specified reliability constraints.

1000 message sets are generated, where the total number of messages in every message set ranges from 5 to 10, and the number of critical messages ranges from 1 to 5. The periods vary between 5 to 50 time units, where one unit is equal to the largest possible frame length. The number of frames in each message, as well as the maximum number

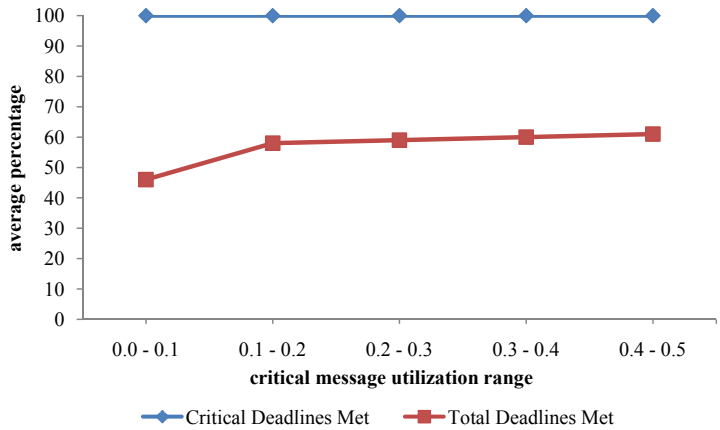


Figure 3.27: Worst-case error scenario - total utilization 0.6 - 0.8

of frame re-transmissions in critical messages, range from 1 to 5. The maximum number of message instances over LCM is limited to 1500. Results are grouped with respect to the total network utilization of the message sets.

Figures 3.26 and 3.27 show the percentage of successfully deadlines met in the worst-case error scenario of maximum specified number of re-transmissions for each critical message instance. The total network utilization ranges are 0.4-0.6 and 0.6-0.8 respectively, and the X-axis shows the network utilizations by the critical messages. Figures 3.28 and 3.29 show the results from the simulation runs where every other critical message instance is hit by errors and re-transmitted according to its reliability constraints. However, as the network utilization increases, it can be seen that the cost of meeting the critical deadlines increases as well.

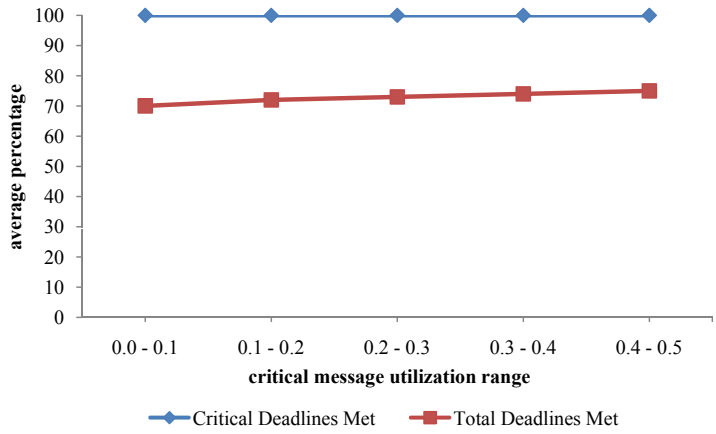


Figure 3.28: Less severe error scenario - total utilization 0.4 - 0.6

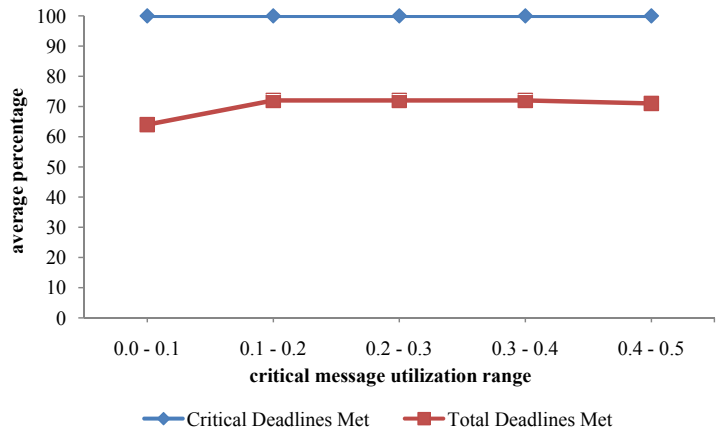


Figure 3.29: Less severe error scenario - total utilization 0.6 - 0.8

## 3.4 Summary

This chapter proposes three FT strategies. First strategy is a spatial redundancy approach (VTV), which is particularly designed for real-time systems. Its performance is evaluated against the well-known voting strategy, CMV, confirming that VTV outperforms CMV in all scenarios by showing a lower percentage of errors that are neither masked nor signalled. The experiments demonstrate the overcautious nature of VTV by showing a higher percentage of false positives due to the fact that it does not account the untimely values for arriving at a majority. The experiments also show that, for a given redundancy scheme, it is possible to tune the voter performance by choosing (i) an appropriate $\delta$ (by adjusting the level of clock synchronization) and (ii) an appropriate $\alpha$ that in combination would enable reaching a desired level of FNR while keeping the FPR bounded at a certain level so that the recovery actions taken due to the false positives will not jeopardize the real-time requirements.

The other two FT strategies are temporal redundancy approaches for fault-tolerant scheduling of real-time tasks and messages respectively, addressing the mixed criticality levels in a resource efficient manner. The task scheduling approach guarantees a recovery attempt for every critical task instance upon an error occurrence before the deadlines, provided the combined utilization of primaries and alternates is less than or equal to 100%. The message scheduling approach enables the user to guarantee a variable level of redundancy for critical messages depending on their criticality levels. The non-critical tasks and messages in both approaches are allowed have priorities higher than the priorities of the critical ones. Hence, the non-critical tasks and messages are provided a better service than background scheduling.

# Chapter 4

# Probabilistic Real-Time Analysis (PRTA)

In dependable real-time systems, provision of schedulability guarantees for task sets under realistic fault and error assumptions is an essential requirement, though complex and tricky to achieve. Timing analysis of real-time systems is typically performed to provide on worst-case guarantees, by investigating the schedulability of systems in the case of worst error arrival patterns together with the longest possible execution times. In the cases where it is not feasible/possible to derive an absolute worst-case pattern or an execution time, upper bounds are used. This keeps the worst-case guarantees valid, although it provides somewhat pessimistic results depending on the difference between the worst-case value and the upper bound used. However, in the case of fault and error events, such bounds may not exist due to the random nature of such events, and assuming a rigid worst-case occurrence scenario may either provide inaccurate or excessively pessimistic analysis results. Hence, it is advisable to combine the timing analysis with probabilistic argumentations. Wang et al. [100] addressed the trade-offs between task schedulability and task reliability under random error events to find an optimal resource allocation for fault-tolerance where optimality is decided based on the penalties given to missing deadlines due to insufficient resources and task failures due to errors. Later, Burns et al. [23] and Broster et al. [20, 19, 21] used random error models to provide probabilistic ar-

gumentations on task and message scheduling. This chapter proposes a more general stochastic error model than the models used in previous approaches, and various probabilistic schedulability analysis techniques targeting scheduling on both processors and networks.

## 4.1 Stochastic Error Model

A generic stochastic *fault and error model* that is capable of modeling single errors as well as error bursts caused by the a single fault with improvements over the existing methods is proposed in this subsection. This model consists of the following three parameters and illustrated in Figure 4.1:

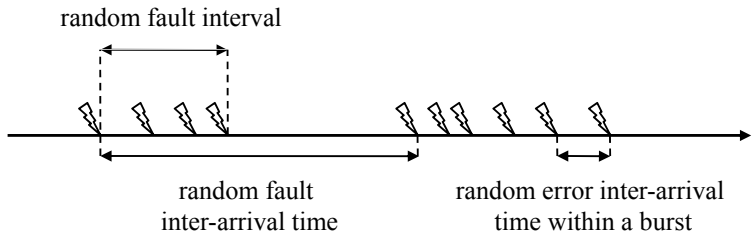


Figure 4.1: Stochastic error model

1. **Fault inter-arrival time**: This is the random duration between two consecutive exposures to independent faults. This time not only depends on the system itself, but also on the type of environment in which the system is operating, due to the various sources of faults. A constant fault arrival rate $\lambda$ is assumed in this error model.

2. **Fault duration**: Once an error occurs, it is likely that the fault causing this error will be in effect for a certain duration and will cause more errors during that period. Consider, for example, a vehicle as the system under observation, which passes through a field with strong electromagnetic interference (EMI). The duration of the exposure to this fault is related to the the velocity of the vehicle, the path through and the area of the field under EMI.

The duration of the faults is very much application specific, and obtaining information regarding the probability distribution of the fault durations, requires again the consideration of not only the application itself but also the operational environment.

3. **Error inter-arrival time within a burst**: This is the random duration between two consecutive errors caused by a single fault. This time depends on the *intensity* of a fault, together with various other factors, such as the resistance of the hardware to the fault, and the existing fault detection and fault-tolerance mechanisms. In this chapter, a constant error arrival rate $\lambda^{error}$, is assumed for all faults. However the error model supports a distribution of various error arrival rates.

### 4.1.1 Using Stochastic Error Models as Bounded Error Models in the Response Time Analysis

In order to perform a joint dependability and timing analysis, the random nature of error events needs to be considered with the usage of stochastic error models. One way to do this is to convert stochastic error models to bounded error models to use in timing analyses, and derive the probabilities that the bounds are held. With these probabilities, probabilistic arguments can be made regarding the results of timing analyses. For each fault $F_i$, a fault duration ($l_i$) is assumed for the timing analysis. The stochastic error model provides a probability mass function for the fault durations that is denoted by $f(l)$. This function gives the probabilities of all possible fault durations from the range of $l_i$. Therefore, no conversion is needed. The random parameters that are converted into worst-case bounds are listed as follows:

1. From the constant fault arrival rate $\lambda$, together with the mission time $L$ of the system, a minimum inter-arrival time $T_F$ between two *independent faults* materializing into either error bursts or single errors can be derived with an associated probability using the Poisson distribution.

   One should note that the mission time $L$ varies largely depending on the domain, typically ranging from minutes for a car to take a short trip to years for a satellite to complete its mission. Furthermore, the number of error events in a unit time $\lambda$ not only depends

on the system but also on the type of environment. For a given system, the common values for $\lambda$ range from $10^2$ errors per hour in aggressive environments to $10^{-2}$ errors per hour in lab conditions, as indicated by Ferreira et al. [36] and Rufino et al. [86].

2. Form the assumed constant error rate $\lambda^{error}$, and the assumed fault duration $l_i$, a minimum inter-arrival time $T_E$ between two errors caused by the same fault can be derived with an associated probability using the Poisson distribution.

   Depending on the fault duration and the fault intensity, a fault can materialize into a burst of errors, only a single error, or no error at all during its length. However, the worst-case scenario always assumes that at least one error occurs during each fault exposure.

## 4.2 PRTA for Mixed Criticality Real-Time Systems

Modern dependable embedded systems typically consist of a mix of hard and soft tasks with varying criticality levels as well as associated fault-tolerance requirements. To ensure efficient resource usage, allocation of resources for providing fault-tolerance should be prioritized according to the criticality levels of the tasks. One way of handling multiple criticality levels is by direct assignment to resources per task based on a pre-defined maximum number of feasible recovery attempts [12], that would deterministically ensure schedulability. However, from the designers' perspective, the most natural way to specify the task criticality levels is by expressing reliability requirements at task level, preferably without having to deal with low level decisions, such as specifying the maximum number of feasible recovery attempts. Hence, it is extremely important to devise methods for translating the high-level requirement specifications for each task into the low-level scheduling decisions needed for the fault-tolerance mechanism to function efficiently and correctly.

This section presents a method, built upon the approach introduced in [23], which allows the system designer to specify task-level reliability requirements and provides a priori probabilistic scheduling guarantees for real-time tasks with mixed-criticality levels in the context of preemptive fixed-priority scheduling where the exploited fault-tolerance strategy is redundancy in the temporal domain.

In this section, a single processor platform is assumed on which a set of sporadic tasks with deadlines equal to or less than their minimum inter-arrival times is allocated. The task set allocated on the processor consists of critical and non-critical tasks, where the criticality of a task indicates the severity of the consequences caused by its failure and corresponds to the amount of resources allocated for error recovery in terms of guaranteed recovery attempts. Each critical task has an alternate task with a Worst-Case Execution Time (WCET) less than or equal to the original WCET of its primary, a deadline equal to the original deadline and a minimum inter-arrival time equal to the minimum inter-arrival time of its primary. This alternate can typically be a re-execution of the same task, a recovery block, an exception handler or an alternate with imprecise computations. Errors are assumed to be caused by faults that arrive with a random inter-arrival time and detected just before the completion of the affected task instances. In this section, only single errors are assumed. In case of tasks communicating via shared resources, an acceptance test is executed before passing an output value to another task to avoid error propagations and subsequent domino effects.

### 4.2.1 Dependability Requirements Specification

During the initial phase of the design of fault-tolerance mechanisms, the designers specify the criticality levels for each task in terms of failure probability (or reliability) per hour. These reliability figures are then used to derive the fault inter-arrival time thresholds, $T_{F_i}$, for each task $i$. These thresholds determine the maximum number of permitted error recovery actions. If the actual fault inter-arrival times are greater than or equal to these thresholds then the specified reliability requirements are guaranteed to be satisfied, i.e. deadlines will be met even under error occurrences.

In [23], a single $T_F$ value is valid for the whole task set (based on a single level of criticality), which means that the reliability requirement is specified for the whole system. It is assumed that if the actual shortest interval between two faults in a mission time $W$ is less than $T_F$, then the task set is unschedulable and the probability of unschedulability $Pr(U)$ is shown to be equal to $Pr(W < T_F)$. In the next subsection, the methodology for providing the probabilistic guarantees for mixed-criticality task sets is presented where the the specified reliability requirements are used as inputs.

### 4.2.2 Methodology Overview

The ultimate goal is to provide a schedulability analysis for the mentioned selective fault-tolerant (FT) scheduling technique for scheduling the tasks with mixed reliability requirements, ranging from non-critical tasks, failure of which does not adversely affect the systems' correct and dependable functioning, to highly critical tasks, where one or more times of recovery actions might be necessary to perform in order to ensure dependability. The methodology consists of two steps:

1. The first step is to determine the minimum fault inter-arrival time, $T_{F_i}$, for each critical task $i \in \Gamma_c$ using the reliability figures given by the system designers.

2. The second step involves checking whether the schedulability of these tasks can be guaranteed even in the worst-case error scenarios specified by the derived minimum inter-arrival times.

### 4.2.3 Proposed Approach

In this subsection, the approach is presented with the help of an example. Let the task set for this example consists of 4 tasks, as shown in Table 4.1 where columns $P, T, C, D, \overline{C}$ represent the priority, period, WCET, deadline and the worst-case recovery time respectively and the time unit is *milliseconds*. Priorities are ordered from 1 to 4 where 4 is the lowest priority. Note that task B is a non-critical task, and hence it does not have a worst-case recovery time.

| **Task** | **P** | **T** | **C** | **D** | $\overline{C}$ |
|---|---|---|---|---|---|
| A | 1 | 100 | 15 | 100 | 15 |
| B | 2 | 175 | 10 | 175 | - |
| C | 3 | 200 | 15 | 200 | 15 |
| D | 4 | 300 | 20 | 300 | 20 |

Table 4.1: Example mixed criticality task set

### Derivation of $T_{F_i}$ Values

The following equations, proposed by Burns et al. [23], are used for deriving the upper and lower bounds of $Pr(W < T_F)$. These equations are derived assuming a Poisson probability distribution to find the probability of a number of events that occurs in a fixed time period, provided that the occurrence rate ($\lambda$) is constant and the events are independent.

**Upper bound:** If $L/(2T_F)$ is a positive integer then

$$Pr^{ub}(W < T_F) = 1 + [e^{-\lambda T_F}(1 + \lambda T_F)]^{\frac{L}{T_F}-1} - 2[e^{-2\lambda T_F}(1 + 2\lambda T_F)]^{\frac{L}{2T_F}} \quad (4.1)$$

**Lower bound:** If $L/(2T_F)$ is a positive integer then

$$Pr^{lb}(W < T_F) = 1 - [e^{-\lambda T_F}(1 + \lambda T_F)]^{\frac{L}{T_F}} \quad (4.2)$$

Burns et al. [23] also derived the following two useful approximations for the upper and lower bounds:

**Approximation 1** An approximation for the upper bound on $Pr(W < T_F)$ is given by Equation 4.1 is $\frac{3}{2}\lambda^2 L T_F$ provided that $\lambda T_F$, $\lambda^2 L T_F$ are small and $L >> T_F$.

**Approximation 2** An approximation for the lower bound on $Pr(W < T_F)$ is given by Equation 4.2 is $\frac{1}{2}\lambda^2 L T_F$ provided that $\lambda T_F$, $\lambda^2 L T_F$ are small and $L >> T_F$.

In this approach, the approximation for the upper bound is used to calculate the $T_{F_i}$ values for each critical task $i \in \Gamma_c$ from the reliability requirements, since the goal of this approach is to find an upper bound for unschedulability (or a lower bound for schedulability).

Let $\lambda$ value for the operational environment be $10^{-2}$ and the lifetime $L$ of the system be 1 hour. The probability of any fault happening during $L$ is calculated as approximately $10^{-2}$ by using the Poisson probability distribution. This would mean that the parts of the system where no fault-tolerance is implemented would fail with a probability of $10^{-2}$, i.e., any non-critical task will have a reliability of approximately 0.99. Let the reliability requirements be given as shown in Table 4.2 for the

| Task | Reliability requirement |
|---|---|
| A | 1 - 1 x $10^{-8}$ |
| C | 1 - 1.25 x $10^{-9}$ |
| D | 1 - 5.85 x $10^{-9}$ |

Table 4.2: Reliability requirements for the critical tasks

critical tasks in the example task set (the numbers shown in the table are selected for presentation purposes only).

By using Approximation 1 and specifying $1-R(i)$ as the upper bound for the error probability of each task, where $R(i)$ is the reliability requirement of task $i$, the $T_{F_i}$ values are calculated as shown in Table 4.3 (in *milliseconds*).

| Task | $T_F$ |
|---|---|
| A | 240 |
| C | 30 |
| D | 140 |

Table 4.3: Derived minimum fault inter-arrival times for critical tasks

### Schedulability Test

Table 4.4 shows the Worst-Case Response Times (WCRT) that are calculated by the traditional Response Time Analysis (RTA) assuming no errors occur during task executions. For the sake of presentation, blocking times are assumed to be zero. As all the WCRTs are less than the corresponding deadlines, this task set is schedulable under no-error scenarios.

If an FT scheduler is assumed where the failed tasks are re-executed, then the execution of task $i$ will be affected by errors in task $i$ or any higher priority task. Based on this assumption, the WCRTs are computed [22] by using the following equation:

$$R_i = C_i + \sum_{j \in hp(\tau_i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j + \left\lceil \frac{R_i}{T_F} \right\rceil \max_{k \in hep(\tau_i)} (\overline{C}_k) \qquad (4.3)$$

| Task | P | T | C | D | R |
|---|---|---|---|---|---|
| A | 1 | 100 | 15 | 100 | 15 |
| B | 2 | 175 | 10 | 175 | 25 |
| C | 3 | 200 | 15 | 200 | 40 |
| D | 4 | 300 | 20 | 300 | 60 |

Table 4.4: Worse-case response times - no error scenario

where $\overline{C}_k$ is the extra computation time needed by task $k$. The last term calculates the worst-case interference arising from the recovery attempts.

Table 4.5 shows the response times that are calculated by Equation 4.3 assuming a minimum fault inter-arrival of $T_F = 75ms$. The task set is guaranteed to be schedulable under this fault rate assumption, assuming no two faults arrive closer than the $T_F$ value, as all the WCRTs are less than the corresponding deadlines.

| Task | P | T | C | D | $\overline{C}$ | R ($T_F = 75$) |
|---|---|---|---|---|---|---|
| A | 1 | 100 | 15 | 100 | 15 | 30 |
| B | 2 | 175 | 10 | 175 | 10 | 40 |
| C | 3 | 200 | 15 | 200 | 15 | 55 |
| D | 4 | 300 | 20 | 300 | 20 | 100 |

Table 4.5: Worse-case response times - single criticality level

To address the multiple task criticality levels, the FT scheduling paradigm presented in [23] is extended by enabling recovery attempts only for the critical tasks and limiting these attempts to once per an interval equal to the derived minimum inter-arrival times for each critical task. Accordingly, WCRTs are computed by the following equation adapted for this scheduler:

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j + I_i^{err} \tag{4.4}$$

In this equation, the term $I_i^{err}$ is a function that computes the worst-case interference, arising from the recovery attempts, during the execution of task $i$ for $R_i$. Algorithm 1 shows the details of this function.

Inputs to the algorithm are task $i$, the response-time $r_i$ in the current

**Algorithm 1:** $I_i^{err}$

**input** : $\tau_i$, $r_i$, $\forall k \in hepc(i)$ $T_{F_k}$, $\overline{C}_k$
**output**: worst-case interference for task $i$ in response time $r_i$
**begin**
  $output \leftarrow 0$;
  $n \leftarrow \left\lceil \frac{r_i}{min(T_{F_k})|k \in hepc(i)} \right\rceil$ ;
  **foreach** $k \in hepc(i)$ **do**
    put $\tau_k$ in Array $A$;
  $j, k, l \leftarrow 0$;
  sort $A$ by decreasing $\overline{C}$;
  **while** $j < n$ **do**
    $output \leftarrow output + \overline{C}_{A[k]}$;
    $j \leftarrow j + 1$;
    $l \leftarrow l + 1$;
    **if** $l \geq \left\lceil \frac{R_i}{T_{F_{A[k]}}} \right\rceil$ **then**
      $k \leftarrow k + 1$;
      $l \leftarrow 0$;
  return $output$;

iteration of the recurrence relation, and $T_{F_k}$ and $\overline{C}_k$ values for each task $k \in hepc(i)$, where $hepc(i)$ is the set of *critical* tasks with priority equal to or higher than the priority of task $i$. The algorithm first calculates the worst-case number of interferences from the following equation:

$$n = \left\lceil \frac{R_i}{min(T_{F_k})|k \in hepc(i)} \right\rceil \tag{4.5}$$

Unlike in Equation 4.3, multiplying the worst-case number of interferences with the largest $F_k$ value would generate an unnecessary pessimism, since task $k$ that has the largest $\overline{C}_k$ may have a larger $T_{F_k}$ than the $T_F$ value used in Equation 4.5. In that case, the worst-case number of interferences $n$ in $r_i$ can be greater than the worst-case number of interferences by task $k$ in $r_i$ which is calculated by $\left\lceil \frac{r_i}{T_{F_k}} \right\rceil$. Therefore,

after $\left\lceil \frac{r_i}{T_{F_k}} \right\rceil$ additions of $\overline{C}_k$, the interferences by task $l$ that has the next largest worst-case recovery time should be added either for $\left\lceil \frac{r_i}{T_{F_l}} \right\rceil$ times or $n - \left\lceil \frac{r_i}{T_{F_k}} \right\rceil$ times, whichever is smaller. This procedure is continued until $n$ interference values are added, and finally, the algorithm outputs the accumulated worst-case interference for task $i$ in response time $r_i$.

In the example, the response times are calculated as shown in Table 4.6. As all the WCRTs are less than the corresponding deadlines, it can be concluded that the task set is schedulable under the specified error hypothesis.

| **Task** | **P** | **T** | $T_F$ | **C** | **D** | $\overline{C}$ | **R** |
|---|---|---|---|---|---|---|---|
| A | 1 | 100 | 240 | 15 | 100 | 15 | 30 |
| B | 2 | 175 | - | 10 | 175 | - | 40 |
| C | 3 | 200 | 30 | 15 | 200 | 15 | 85 |
| D | 4 | 300 | 140 | 20 | 300 | 20 | 175 |

Table 4.6: Worse-case response times - multiple criticality levels

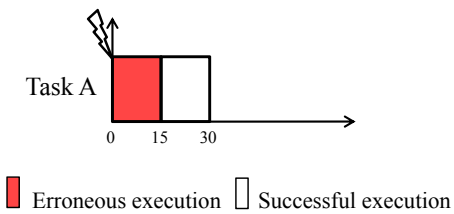


Figure 4.2: Worst-case interference for task A

Figures 4.2 to 4.5, show the worst-case interference for the individual tasks. In the worst-case, task $A$ has one interference in a computational window of $R_A = 30$, and it is the interference by task $A$ itself, since $n$ in Equation 4.5 is 1 ($\lceil \frac{30}{240} \rceil$) and task $A$ is the only task in $hepc(A)$. Figure 4.2 shows the worst-case interference scenario for task $A$.

Task $B$ has also one interference in a computational window of $R_B = 40$ in the worst-case, and it is also the interference by task $A$, since $n$ in

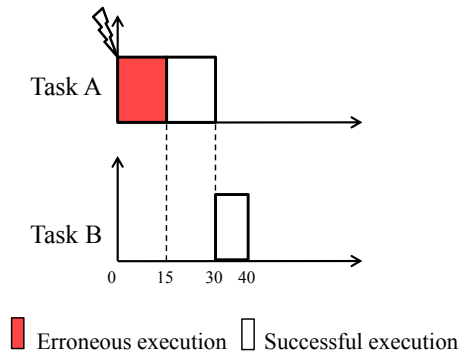


Figure 4.3: Worst-case interference for task B

Equation 4.5 is 1 ($\lceil \frac{40}{240} \rceil$) and task $A$ is again the only task in $hepc(B)$. Figure 4.3 shows the worst-case interference scenario for task $B$.

Task $C$ can be interfered for maximum three times in a computational window of $R_C = 85$ ($\lceil \frac{85}{30} \rceil$). As the maximum of the $\overline{C}$ values for the tasks in $hepc(C)$ is $\overline{C}_A = \overline{C}_C = 15$, the algorithm returns three times this value ($I_C^{err} = 45$). Figure 4.4 shows the worst-case interference scenario for task $C$.

Task $D$ has a worst-case of 6 interferences in a computational window of $R_D = 175$ ($\lceil \frac{175}{30} \rceil$). The maximum of the $\overline{C}$ values for the tasks in $hepc(D)$ is task $D$'s worst-case recovery time $\overline{C}_D = 20$, however, the recovery of task $D$ can interfere task $D$'s execution maximum 2 times, as $T_{F_D} = 140$ and $\lceil \frac{175}{140} \rceil = 2$. The other 4 interferences can either come from task $A$ or task $C$ which have the next largest $\overline{C}$ value. The algorithm returns $I_D^{err} = 100$. Figure 4.5 shows the worst-case interference scenario for task $D$.

The analysis shows that for the given task set, all deadlines are met with probabilities that satisfy the reliability requirements specified by the designers even in the worst-case error scenario for the given error rate $\lambda = 10^{-2}$ and mission length $L = 1$ hour.

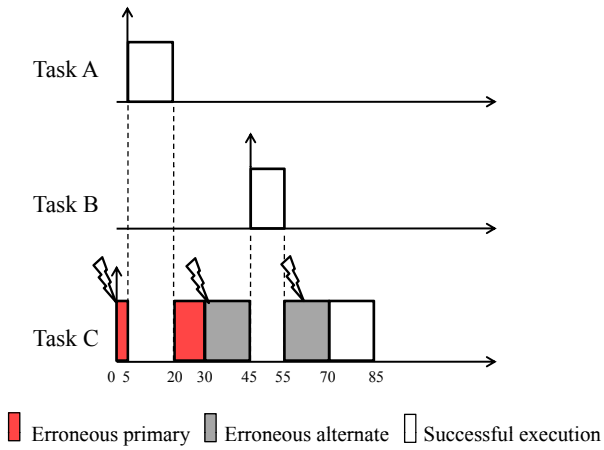


Figure 4.4: Worst-case interference for task C

## 4.3 PRTA for Task Scheduling under Error Bursts

This section presents an approach that combines a schedulability analysis for real-time tasks scheduled under the Fixed Priority Scheduling (FPS) policy using an error model that is capable of modeling of error bursts with random attributes, and a sensitivity analysis in order to derive accurate probabilistic schedulability guarantees for FT real-time tasks.

The approach begins with performing a set of schedulability analyses that accounts for a range of worst-case scenarios generated by stochastic error burst occurrences on the response times of tasks scheduled under the FPS policy. Then the probabilistic schedulability guarantees are calculated as a weighted sum of the conditional probabilities of schedulability under specified error burst characteristics.

In this section a single processor platform is assumed on which a sporadic task set is allocated which have deadlines equal to or less than their minimum inter-arrival times. Whenever an error is detected within a task, the affected task $i$ executes an alternate task with a WCET less than or equal to the original WCET of its primary, a deadline equal to the original deadline and a minimum inter-arrival time equal to the minimum

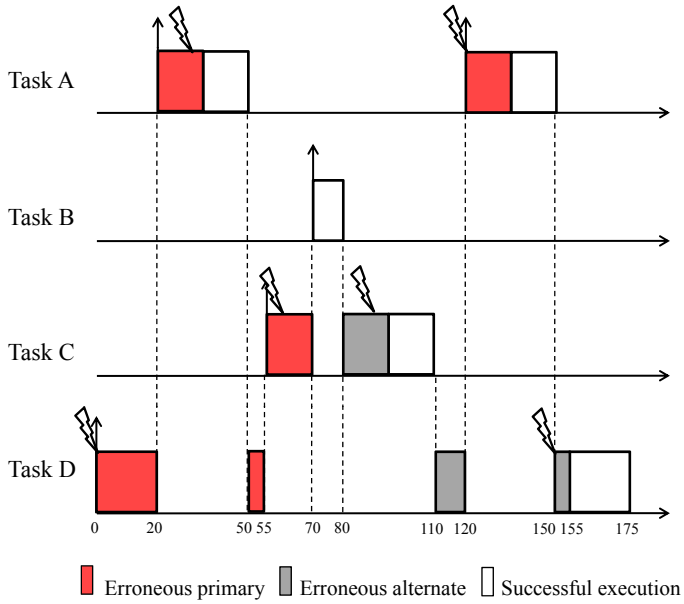


Figure 4.5: Worst-case interference for task D

inter-arrival time of its primary. This alternate can typically be a re-execution of the same task, a recovery block, an exception handler, or an alternate task with imprecise computations. Errors are assumed to be detected just before the completion of the affected task instances.

The main sources of errors are assumed to be EMI (external faults), and transient hardware faults (internal faults) that affect, e.g. the sensors and the network systems. Examples to the considered errors are incorrect input values from sensors, or failure in delivering the output values via network messages. Errors that are propagated into tasks are detected at the end of task executions by observing, e.g., the out-of-range output values or omitted outputs. Examples to the assumed error detection mechanisms are usage of sanity checks, range checks, checksums for the value correctness and the usage of watchdog timers for the time correctness. Watchdog timers are assumed to be implemented as simple hardware units that run in parallel with the tasks and interrupt

in case of detected errors and the overhead of the value error detectors are included in the WCETs of the respective tasks.

Each fault may result in errors encapsulated in an error burst for a random duration. The distribution regarding the duration of the faults is very much domain specific, and, in this section, it is assumed that the information regarding the probability distribution of the fault durations is available. Other parameters assumed to be given related to the error model is the error rate $\lambda$ and the the mission time $L$. In this section, it is assumed that no task can successfully finish between errors within a burst. Furthermore, any task instance scheduled even partially under the error burst will be considered as affected by the error.

### 4.3.1 Methodology Overview

The goal of this approach is to find the probability that the given task set is schedulable during a mission time $L$ under the specified error model. This probability is dependent on the error characteristics (the minimum inter-arrival time between faults, $T_F$, the possible values for the fault duration $l_j$, and the probability distribution $f(l)$) and can be derived from the conditional probabilities that the task set is schedulable under specific sets of values for these parameters.

The analysis begins with finding the maximum number of error bursts, $n$, that can hit any task in the task set. Considering the interplay between $T_F$ and $l_j$, a set of sensitivity analyses is performed to derive the minimum inter-arrival times between faults ($T_F$) for each possible combination of $n$ fault durations by assuming the worst-case task executions and error overheads. One should note that the derived minimum inter-arrival times are actually *upper bounds* which may never be reached. This is due to the nature of the inexact worst-case assumptions, such as the WCETs of the tasks, which correspond to upper bounds rather than exact worst-case values. The fault duration combinations and the corresponding upper bound $T_F$ values are then used to find the *conditional* probabilities of schedulability which are actually lower bounds for the exact probabilities. Finally, the lower bound probability of schedulability is computed as a cumulative sum of these individual conditional lower bound probabilities, i.e. by unconditioning the probability of schedulability with respect to the fault durations. The steps involved in the methodology are illustrated in Figure 4.6 and briefly described below.

STEP 1: The analysis begins by finding an upper bound for the

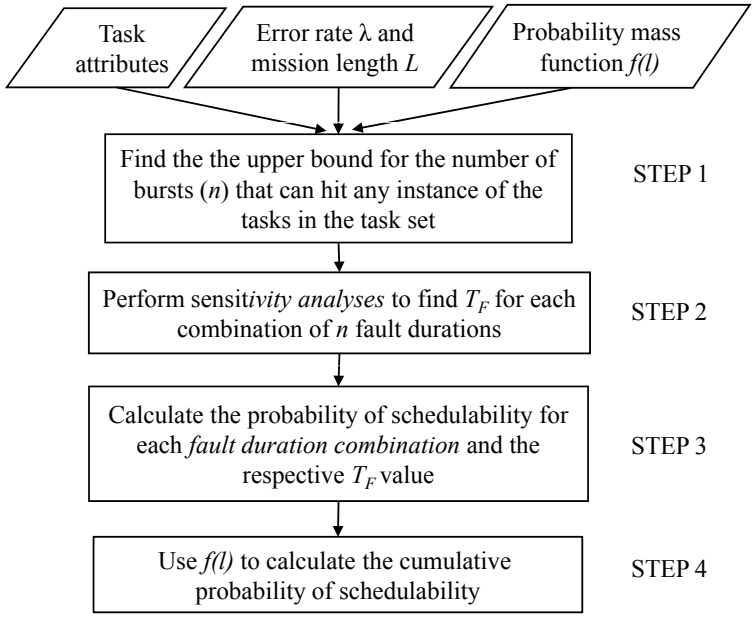


Figure 4.6: Methodology overview - PRTA for task scheduling under error bursts

maximum number of error bursts that can hit any task in the task set while the task set is still schedulable. To do this, a sensitivity analysis is performed to derive the maximum number of bursts that can hit the task task with the *longest deadline* by assuming that all faults have the *shortest possible duration*.

STEP 2: In this step, a set of sensitivity analyses is performed for each combination of $n$ fault durations specified in the probability mass function $f(l)$ in order to derive the minimum inter-arrival time between faults ($T_F$) under which the task set is still schedulable.

STEP 3: The goal of this step is to derive the probabilities that the actual inter-arrival times between bursts will not be shorter than the calculated minimum inter-arrival times by taking into account

$\lambda$ and the mission time $L$.

STEP 4: Finally, based on the probability mass function $f(l)$, as well as the derived probabilities for each fault duration combination, the cumulative probability of schedulability is derived.

### 4.3.2 Response Time Analysis under Error Bursts

In this subsection, an RTA is presented that identifies whether a given task set is schedulable when affected by faults with *random durations* and a minimum inter-arrival time $T_F$. One should note that if the fault duration is greater than or equal to the minimum inter-arrival time between bursts, every burst can start before the end of the previous one, hence the bursts can potentially affect the whole mission time. If this is the case, or if the fault duration is greater than the minimum inter-arrival time of the task whose WCRT is to be calculated, the schedulability of this task cannot be guaranteed.

The main differences between the error characteristics in the traditional single error model and the burst model are:

- An error burst may consist of multiple errors
- An error burst may affect multiple tasks

Hence, the worst-case scenario required for calculating the WCRTs is not the same in case of error bursts as compared to the model introduced in [23].

The set of faults interfering with task $i$, i.e., arriving after the release of $\tau_i$, is denoted by $\{F_j |\, j = 1, 2, ..., n\}$.

**Definition 1.** *The **worst-case error overhead** $I_i^{err_j}$ for task $i$ caused by fault $F_j$ is the largest amount of time required by the task alternates, $\overline{\tau}_k \in \overline{\Gamma}$, to recover from the effects of fault $F_j$, in the interval between the release time of task $i$ and its completion.*

**Remark 4.1.** *An important observation is that, while the worst-case error overhead accounts for all the alternates required for recovery (including the successful ones), it excludes all the primaries, since, although affected by errors, those are already taken into account in the interference from the higher priority tasks and task $i$'s own primary execution, i.e., $C_i$ and $\sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$.*

**Remark 4.2.** *Another observation following Definition 1 is that, in the general case, a burst causes its worst-case error overhead when its interference (i.e., overlap) with the executions of the first and last task it affects is minimized. Hence, it has to start $\epsilon$ before the completion of the first affected task, and end $\epsilon$ after the start of the last affected task (where $\epsilon$ is an arbitrarily small real number).*

**Lemma 4.1.** *The worst-case error overhead $I_i^{err_j}$ caused by fault $F_j$ with duration $l_j$ on the highest priority task $\tau_h \in \Gamma$ is given by*

$$I_h^{err_j} = 2\overline{C}_h + l_j - \epsilon \tag{4.6}$$

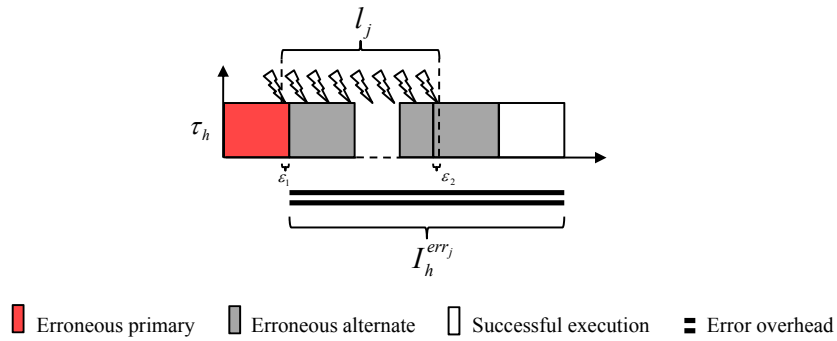


Figure 4.7: Worst-case error overhead for the highest priority task $\tau_h$

*Proof.* In this case, following the Remark 4.2, the worst-case error overhead occurs in the scenario when the burst starts very close ($\epsilon$ before) the completion of the primary version of the task, and ends $\epsilon$ after the start of the last failed execution. The scenario is illustrated in Figure 4.7 where the sum of the computation requirements of all alternates except last one equals $l - \epsilon$. Hence, $I_i^{err_j} = 2\overline{C}_h + l_j - \epsilon$. (In Figure 4.7, $\epsilon = \epsilon_1 + \epsilon_2$) □

**Lemma 4.2.** *If a burst hits task i, where i gives the priority level of the task, its worst-case error overhead occurs under the following three conditions:*

1. *the burst starts $\epsilon$ before the completion of $\tau_i$ and ends $\epsilon$ after the start of execution of the last failed execution,*
2. *the burst hits all tasks with higher priorities than task $i$, if any,*
3. *$\overline{\tau}_i$ and all its higher priority tasks are preempted by their respective next higher priority task (if any), $\epsilon$ after their start of execution.*

*Proof.* The basic idea is that the burst, in order to cause the most damage, should leave outside the fault duration as much wasted computations as possible. The first condition is a direct result of the Remark 4.2 and Lemma 4.1. In this case, the largest amount of wasted computation on task $i$ outside the burst occurs when the burst hits $\tau_i$, $\epsilon$ before its completion and ends $\epsilon$ after the start of the last failed execution. Now, in order to maximize the damage, the burst should hit as many tasks as possible during its occurrence, since each additional task hit by the burst will eventually recover and, hence, contribute with at least one extra successful alternate (except the failed ones) to the worst-case error overhead. The maximum number of tasks the burst can hit in addition to $\tau_i$ is given by all higher priority tasks than $\tau_i$ (proving condition 2). This scenario, in its turn, is only possible if all higher priority tasks preempt each other in increasing priority order, and all preemption points occur during the burst. Additionally, in order to leave as much wasted computation as possible outside the burst. i.e., after the end of the burst, all preemptions should occur as early as possible during the affected tasks' executions. □

**Theorem 4.3.** *The worst-case error overhead for task $i$ caused by fault $F_j$ with duration $l_j$ is:*

$$I_i^{err_j} = \max_{k \in hep(i)} (\overline{C}_k + \sum_{m \in hep(k)} \overline{C}_m + \alpha) \tag{4.7}$$

*where*

$$\alpha = \begin{cases} 0, & if\ k \neq h \\ & \quad and\ C_h - (l_j - \epsilon) \geq \overline{C}_h \\ l_j - \epsilon + \overline{C}_h - C_h, & if\ k \neq h \\ & \quad and\ C_h - (l_j - \epsilon) < \overline{C}_h \\ l_j - \epsilon, & otherwise \end{cases}$$

*$\tau_h$ is the highest priority task in the task set $\Gamma$ and $\epsilon$ is an arbitrarily small positive real number.*

*Proof.* Following Lemma 4.2, the worst-case error overhead $I_i^{err_j}$ consists of three elements: a) the longest remaining execution of the failed alternate of the first task hit by the burst, b) one successful alternate from each higher or equal priority task, and c) the additional contribution $\alpha$ to $I_i^{err_j}$ from the highest priority task $\tau_h \in \Gamma$. While a) and b) are given by the worst-case execution requirements of the tasks' alternates, $\alpha$ may vary depending on the relation between the fault duration and the WCETs of the highest priority task $\tau_h$ and its alternate.

Following Lemma 4.1, in the general case, the largest contribution $\alpha$ of $\tau_h$ to $I_i^{err_j}$ cannot exceed the fault duration $l_j$ minus $\epsilon$.

$$\alpha = l_j - \epsilon \tag{4.8}$$

However, if $\tau_h$ is not the first task hit by the burst (in which case its $\alpha$ would be given by Equation 4.8, following Lemma 4.1) $\alpha$ can be further refined depending on the relation between the fault duration, the WCETs of the highest priority task $\tau_h$ and its alternate. The reason is twofold:

1. If $l > C_h + \epsilon$, the burst *always* ends after the completion of $\tau_h$. In this case, from the duration of the fault (that started $\epsilon$ before $\tau_h$) $C_h$ is subtracted (that is accounted in the $I_i^{hp}$ in the RTA), and the last failed alternate is added (that started $\epsilon$ after the end of the burst) as shown in Figure 4.8.

$$\alpha = l_j - \epsilon + \overline{C}_h - C_h \tag{4.9}$$

   From the equation above it is obvious that, if the WCET of $\tau_h$ is grater than the one of its alternate, i.e., $C_h > \overline{C}_h$, $\alpha$ is less than given by Equation 4.8.

2. If $l \leq C_h + \epsilon$, the burst *may* end before $\tau_h$ completes (note that the burst still can end after the completion of $\tau_h$, in case $\tau_h$ executes less than its WCET). Here there are two possible cases that potentially give the worst-case error overhead:

Case 1: The burst ends before the completion of $\tau_h$, and $\tau_h$ executes for its WCET. In this case, the first alternate of $\tau_h$ will successfully complete its execution as illustrated in Figure 4.9. Hence,

$$\alpha = 0 \tag{4.10}$$

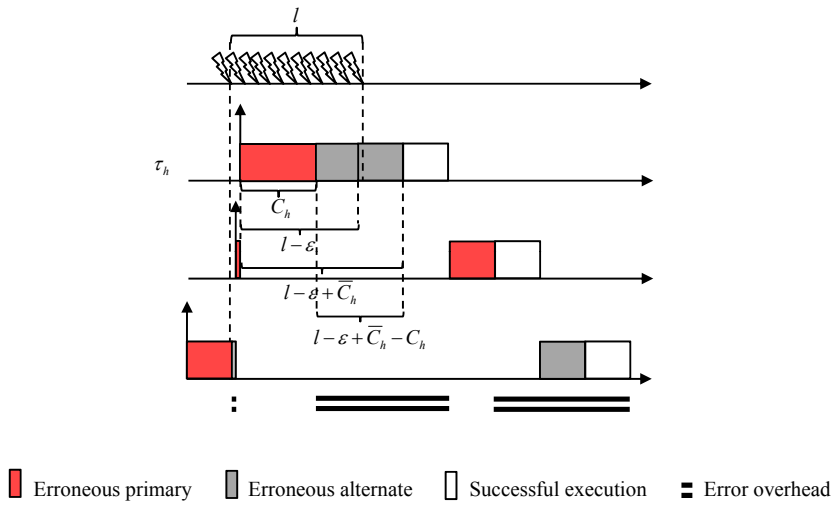


Figure 4.8: Error overhead when $l > C_h + \epsilon$

Case 2: The burst ends after the completion of $\tau_h$ due to $\tau_h$'s execution less than its WCET. According to the Lemma 4.2, in order to generate the worst-case error overhead, the burst needs to end $\epsilon$ after the start of $\overline{\tau}_h$. Hence, $\alpha$ consists of the failed alternate that started $\epsilon$ before the end of the burst, from which the difference between the worst-case execution requirements $C_h$ and $l$ must be subtracted (that is already accounted in the $I_i^{hp}$ in the RTA),

$$\alpha = \overline{C}_h - (C_h - (l_j - \epsilon)) \tag{4.11}$$

and it is identical with Equation 4.9. This scenario is illustrated in Figure 4.10.

$\alpha$ is obtained from equations 4.8, 4.9 and 4.10 as follows:

$$\alpha = \begin{cases} 0, & if\ k \neq h \\ & \quad and\ C_h - (l_j - \epsilon) \geq \overline{C}_h \\ l_j - \epsilon + \overline{C}_h - C_h, & if\ k \neq h \\ & \quad and\ C_h - (l_j - \epsilon) < \overline{C}_h \\ l_j - \epsilon, & otherwise \end{cases} \tag{4.12}$$

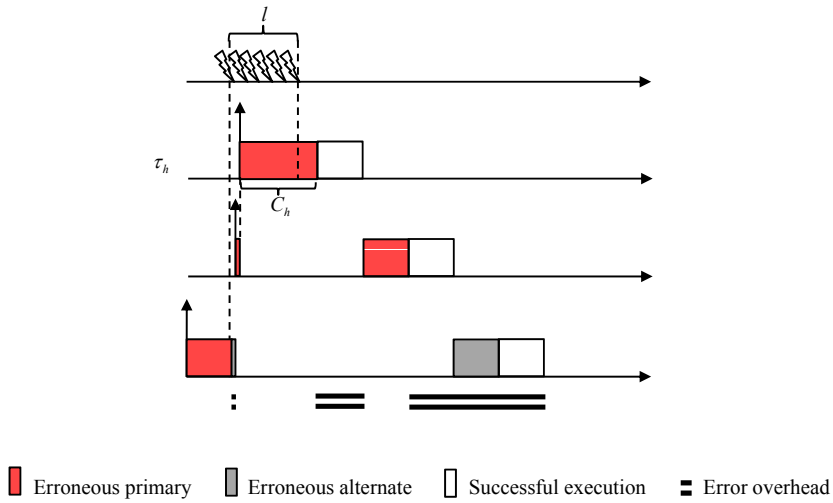


Figure 4.9: Error overhead when $l \leq C_h + \epsilon$ and the error burst ends before $\tau_h$ completes

However, the worst-case error overhead $I_i^{err_j}$ does not necessarily occur when $\tau_i$ is the first task hit by Fault $F_j$. Figure 4.11 shows an example of this phenomenon while calculating the worst-case error overhead caused by the burst on task $C$. In Figure 4.11 a), if task $C$ is the first task hit by the burst, the error overhead is equal to 13, leading to a response time of C equal to 19. However, in this scenario, the worst-case error overhead $I_C^{err}$ occurs when the first task hit by the burst is $B$, and equals 15. Implicitly, the response time of $C$ is 21. Hence, the worst-case error overhead for task $i$ caused by a fault with duration $l_j$ is given by the maximum function of a) the remaining execution of the failed alternate of the first task $\overline{\tau}_k \in hep(i)$ hit by the burst, b) one successful alternate from each $\tau_m \in hep(k)$ , and c) the additional contribution $\alpha$ to $I_i^{err_j}$ from the highest priority task $\tau_h \in \Gamma$:

$$I_i^{err_j} = \max_{k \in hep(i)} (\overline{C}_k + \sum_{m \in hep(k)} \overline{C}_m + \alpha)$$

where $\alpha$ is given by Equation 4.12, which concludes the proof.

□

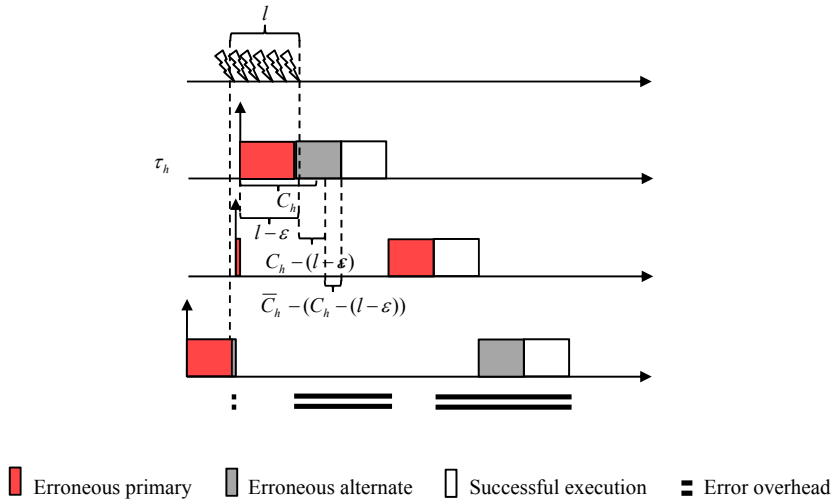


Figure 4.10: Error overhead when $l \leq C_h + \epsilon$ and the error burst ends $\epsilon$ after $\tau_h$ completes

The total interference $I_i$ experienced by task $i$ is the sum of the maximum interference caused by the higher priority tasks, $I_i^{hp}$, and the maximum interference caused by error bursts $I_i^{err}$.

$$\forall \tau_i \ \in \Gamma, I_i = I_i^{hp} + I_i^{err}$$

Note that $I_i^{hp}$ is given by the traditional RTA [8, 49]:

$$I_i^{hp} = \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

Consequently, the worst-case error interference that needs to be accounted for, in the RTA, is obtained by the summing up the the worst-case error overheads, $I_i^{err_j}$, of each Fault $F_j$ that is assumed to interfere with task $i$'s execution. In this case, the maximum interference caused by the error bursts caused by faults with a minimum inter-arrival time

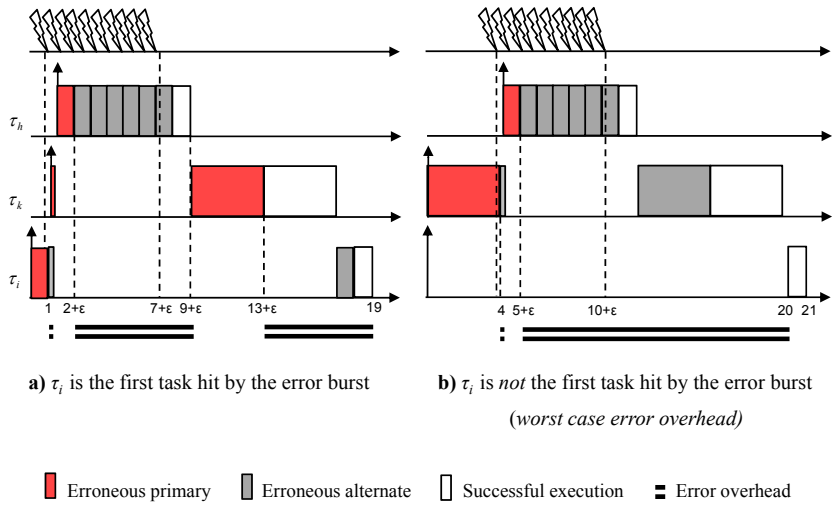


Figure 4.11: worst-case error overhead occurs when $\tau_i$ is not the first task hit by the error burst

$T_F$ on task $i$ in the interval $(0, R_i]$ is,

$$I_i^{err} = \sum_{j=1}^{\left\lceil \frac{R_i}{T_F} \right\rceil} I_i^{err_j}$$

Hence, the equation that gives the WCRT for task $i$ under error bursts is:

$$R_i = C_i + B_i + I_i^{hp} + I_i^{err} \tag{4.13}$$

### 4.3.3 Probabilistic Schedulability Bounds

In this section, it is assumed that, during a mission, if the actual shortest interval between any two error bursts $W$ is less than the derived minimum inter-arrival time between faults $T_F$, then the task set is unschedulable. Hence, the probability of unschedulability $Pr(U)$, is equal to $Pr(W < T_F)$, i.e., the probability of schedulability of a given task set is translated to the derivation of the probability that, during the mission

time $L$, no two consecutive error bursts arrive with an inter-arrival time shorter than the derived $T_F$.

As outlined in the beginning of this section, Equation 4.1, which derives an upper bound of $Pr(W < T_F)$, is used to calculate the lower bound for the probability of schedulability (or the upper bound for the probability of unschedulability) for each derived $T_F$ value. Later, the derived lower bounds for the probability of schedulability, corresponding to each fault duration combination, $Pr^{lb}(S|\, combo_i) = 1 - Pr^{ub}(U|\, combo_i)$, as well as the probability values for each fault duration combination derived from the probability mass function $f(l)$ are used to calculate the lower bound for the cumulative probability of schedulability $Pr^{lb}(S)$ for the given task set.

$$Pr^{lb}(S) = \sum_{combo_i} (Pr^{lb}(S \mid combo_i) Pr(combo_i) \tag{4.14}$$

### 4.3.4 Illustrative Example

The proposed methodology is illustrated with a simple example. A single processor system is assumed on which a task set consisting of 4 tasks as shown in Table 4.7 is allocated. Priorities are ordered from 1 to 4 where 4 is the lowest priority. The time unit is milliseconds. The mission

| $Task$ | $P$ | $C$ | $C^{alt}$ | $T = D$ |
|---|---|---|---|---|
| A | 1 | 4 | 4 | 80 |
| B | 2 | 4 | 4 | 80 |
| C | 3 | 2 | 2 | 60 |
| D | 4 | 6 | 3 | 100 |

Table 4.7: Example task set

time is assumed to be one hour ($L = 1h$), and a discrete probability distribution for the fault duration $l$ is given as shown in Figure 4.12. Expected number of bursts in unit time is assumed as $\lambda = 5$.

STEP 1: First step is to derive an upper bound for the maximum number of bursts ($n$) that can hit any task in the task set for which the task set is guaranteed to be schedulable. In order to do that, a sensitivity analysis is performed assuming the shortest possible

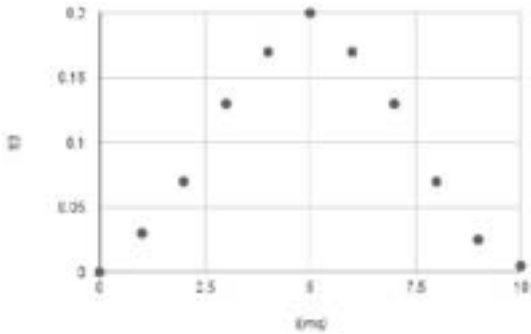

Figure 4.12: Example probability mass function $f(l)$

fault duration ($1ms$) for each burst occurrence. The analysis shows that the shortest feasible minimum inter-arrival time is $T_F = 38ms$. The response times of the tasks in the example task set are shown in the third column of Table 4.8 assuming $T_F = 38ms$. If a slightly

| $Task$ | $D = T$ | R ($T_F = 38$) | R ($T_F = 37$) |
|---|---|---|---|
| A | 80 | 28 | 28 |
| B | 80 | 36 | 36 |
| C | 60 | 32 | 32 |
| D | 100 | 76 | 144 (UNSCH) |

Table 4.8: Worse-case response times for $T_F = 38$ and $T_F = 37$

shorter minimum inter-arrival time ($T_F = 37ms$) is assumed, then the analysis shows that the schedulability of task $D$ cannot be guaranteed as shown in the last column of Table 4.8. The analysis performed in this step would indicate that, if all task deadlines are met, no more than three bursts can hit any task instance during the mission:

$$\max_{i \in \Gamma} \left( \left\lceil \frac{D_i}{T_F} \right\rceil \right) = \frac{100}{38} = 3$$

STEP 2: Then Equation 4.13 is used to perform a set of sensitivity analyses for each combination of 3 fault durations and found the

minimum inter-arrival times between bursts, at which the task set is guaranteed to be schedulable as shown in the fourth column of Table 4.9. Note that the maximum number of bursts derived in the first step ($n = 3$) is an upper bound, which is actually never reached in this example. This is the reason why changing the values of $l_3$ does not vary the $T_F$ value.

STEP 3: Using Equation 4.1, a lower bound for the probabilities of schedulability is derived for each fault duration combination and the corresponding $T_F$'s as shown in the last column of Table 4.9.

| $l_1$ | $l_2$ | $l_3$ | $T_F$ | $P^{lb}(S \mid combo_i)$ |
|---|---|---|---|---|
| 1 | 1 | 1-10 | 38 | $9.99604\text{x}10^{-1}$ |
| 1 | 2 | 1-10 | 40 | $9.99583\text{x}10^{-1}$ |
| 1 | 3-10 | 1-10 | 45 | $9.99553\text{x}10^{-1}$ |
| 2 | 1 | 1-10 | 40 | $9.99583\text{x}10^{-1}$ |
| 2 | 2 | 1-10 | 46 | $9.99521\text{x}10^{-1}$ |
| 2 | 3 | 1-10 | 48 | $9.995\text{x}10^{-1}$ |
| 2 | 4-10 | 1-10 | 49 | $9.9949\text{x}10^{-1}$ |
| 3 | 1 | 1-10 | 46 | $9.99521\text{x}10^{-1}$ |
| 3 | 2 | 1-10 | 48 | $9.995\text{x}10^{-1}$ |
| 3 | 3 | 1-10 | 50 | $9.99479\text{x}10^{-1}$ |
| 3 | 4-10 | 1-10 | 53 | $9.99448\text{x}10^{-1}$ |
| 4 | 1 | 1-10 | 48 | $9.995\text{x}10^{-1}$ |
| 4 | 2 | 1-10 | 56 | $9.99417\text{x}10^{-1}$ |
| 4 | 3-10 | 1-10 | 57 | $9.99406\text{x}10^{-1}$ |
| 5 | 1 | 1-10 | 50 | $9.99479\text{x}10^{-1}$ |
| 5 | 2-10 | 1-10 | 63 | $9.99344\text{x}10^{-1}$ |
| 6 | 1-10 | 1-10 | 67 | $9.99302\text{x}10^{-1}$ |
| 7 | 1-10 | 1-10 | 71 | $9.99261\text{x}10^{-1}$ |
| 8 | 1-10 | 1-10 | 75 | $9.99219\text{x}10^{-1}$ |
| 9 | 1-10 | 1-10 | - | 0 |
| 10 | 1-10 | 1-10 | - | 0 |

Table 4.9: Lower bound probabilities of schedulability

STEP 4: Finally, Equation 4.14 is used to find a lower bound for the cumulative probability of schedulability. Based on the derived

lower bounds for the probabilities of schedulability for each fault duration combination, as well as the probability of each of the combination derived from the individual fault duration probabilities, the cumulative probability of schedulability is calculated as $P(S) = 0.9702797789$. This analysis shows that the example task set is schedulable at least with a probability of 0.9702797789 during a $1h$ mission where $\lambda = 5$, for the fault duration probabilities given by $f(l)$.

## 4.4 PRTA for Message Scheduling under Error Bursts

Networked embedded systems used in many critical real-time applications rely on dependable communication. Research so far has focussed on rather simplistic error models which assume only singleton errors separated by a minimum inter-arrival time. However, these systems are often subject to faults that manifest as error bursts of various lengths during message transmissions and these have an adverse effect on the message response times that needs to be accounted for.

This section presents an approach that combines a schedulability analysis for real-time message scheduling instantiated to Controller Area Network (CAN) and a sensitivity analysis in order to derive accurate probabilistic schedulability guarantees for FT real-time messages. The schedulability analysis presented in this section extends the existing RTA for CAN [96, 20, 19, 73, 69] to cope with burst errors modeled with an improved accuracy that enables the specification of a range of new parameters including e.g., fault duration and intensity.

In this section a distributed real-time architecture is assumed, consisting of sensors, actuators and processing nodes communicating over CAN. The communication is performed via a set of periodic messages. For the sake of generality, a message $i$ is assumed to include $m_i$ frames, hence the worst-case transmission time $C_i$ of the message in an error free scenario is:

$$C_i = m_i \; f^{max} \; \tau_{bit} \tag{4.15}$$

where $f^{max}$ is the maximum frame size in number of bits, and $\tau_{bit}$ is the time it takes to transmit a single bit on CAN. However, the analysis presented in this section applies to the particular case of single frame messages as well.

While CAN communication is non-preemptive during the frame transmissions, messages composed of more than one frame can preempt each other at the frame boundaries. Additionally, the non-preemptiveness of message frames may cause a higher priority message to be blocked by a lower priority message for at most one frame length, if the high priority message is released during the transmission of a lower priority frame. This priority inversion phenomenon can affect all messages except the lowest priority one, and only once per message period, before the transmission of the first message frame [29].

Each fault may affect the system for a certain duration. Depending on the duration of a fault and the minimum inter-arrival time between errors within a fault, a fault can materialize into a burst of errors, only a single error, or no error at all during its length. However it is assumed that at least one error occurs during each fault exposure, since analysis assumes the worst-case scenario. For the sake of presentation, the term *error burst* is used for both error bursts and single errors.

The duration of the faults is very much domain specific, and in this section, it is assumed that the information regarding the probability distribution of the fault durations is available. Errors may occur any time during the fault as long as they satisfy the minimum inter-arrival time condition derived from the sensitivity analyses. In this section, each error in message frames is assumed to be detected as soon as it occurs by the built in CAN error detection mechanisms and upon each error in a frame, an identical frame to the erroneous frame is scheduled for re-transmission following the error frame.

Other error model related parameters that are assumed to be given are the rate that the observed system is hit by errors caused by independent faults $\lambda$ and the the mission time $L$ of the system. This section assumes that at most one burst may hit any message instance, hence $T_F$ is equal to the largest period of all the messages in the message set.

### 4.4.1 Methodology Overview

The ultimate goal of this approach is to find the probability that the message set is schedulable under a given fault and error hypothesis. The methodology is outlined in the following steps, and illustrated in Figure 4.13.

- STEP 1: In this step, a series of sensitivity analyses is performed for each $l$ in the probability mass function $f(l)$ in order to derive

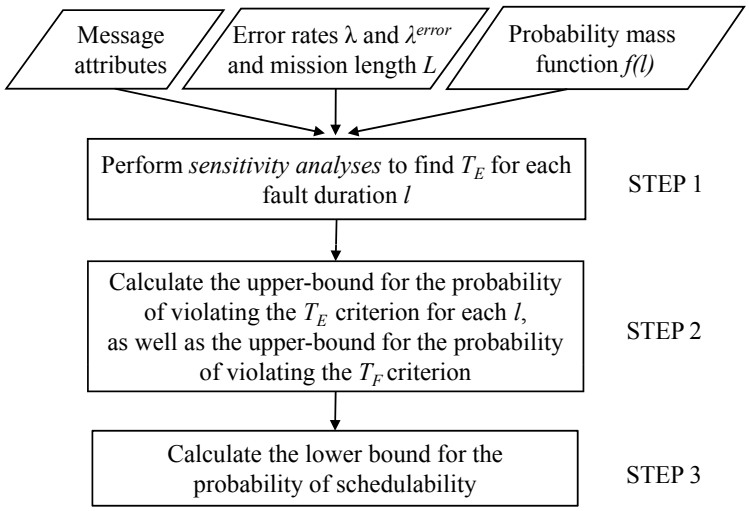


Figure 4.13: Methodology overview - PRTA for message scheduling under error bursts

the minimum inter-arrival times of errors within error bursts, $T_E$, for which the message set is guaranteed to be schedulable.

- STEP 2: In this step, first an upper-bound for the probability of violating the minimum inter-arrival time requirement between errors within a burst, $T_E$, for each fault duration $l$ is calculated. Then, this probability bound on the *fault duration* is unconditioned and an upper-bound for the probability of violating the minimum inter-arrival time requirement between errors within bursts under faults of random length, during the whole mission is derived. In this step, separately, an upper-bound for the probability of violating the minimum inter-arrival time requirement between faults, $T_F$, during the whole mission is derived.

- STEP 3: Finally, in the last step, an upper-bound for probability of unschedulability, i.e. lower bound for the probability of schedulability, which is shown to be the union of the upper-bounds of the probabilities of at least one occurrence of any two faults arriving

less than $T_F$ apart *or* at least one occurrence of any two errors within a burst, arriving less than $T_E$ apart during a mission of length $L$ is calculated.

In the next subsection, a schedulability analysis under error bursts is presented which is the main tool to perform the outlined analysis.

### 4.4.2 Response Time Analysis under Error Bursts

In this subsection, an RTA is presented that identifies whether a given message set is schedulable when affected by error bursts caused by faults with a given duration $l$ and a combination of error inter-arrival time thresholds (minimum inter-arrival time of faults $T_F$ and errors within a burst $T_E$). The presented RTA is based on the RTA of CAN under periodic messages and sporadic faults introduced by Tindell et al. [96], where an additional term, $I_i^{err}$, for the maximum error interference is added to Equation 4.16:

$$q_i = I_i^{err} + B_i + \sum_{j \in hp(i)} \left\lceil \frac{q_i + J_j + \tau_{bit}}{T_j} \right\rceil C_j \tag{4.16}$$

Assuming the burst error model, the WCRT calculations will differ in the following cases depending on the minimum inter-arrival time of the errors within an error burst $T_E$:

CASE 1: $T_E \leq (e^{max} + f^{max})\tau_{bit}$: In this case, if the errors within an error burst occur with a separation of $T_E$, it may not be possible to transmit any frame between any two consecutive errors during the burst. Therefore, the worst-case error overhead $I_i^{err}$ in Equation 4.16 becomes:

$$I_i^{err} = (f^{max} + e^{max})\tau_{bit} + l \tag{4.17}$$

The error overhead includes the transmission time of the largest frame in the worst-case scenario, i e., when the first error in the burst hits its last bit. The other components of error overhead are the transmission time of the largest error frame and the whole duration of the fault, since in the worst-case, no frame can be transmitted during this time. Figure 4.14 shows a worst-case scenario in Case 1. The largest message frame and the largest error frame in Equation 4.17 are the frames before and after the error burst respectively.

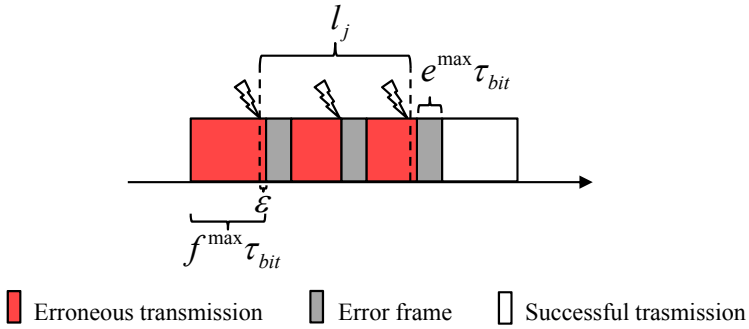


Figure 4.14: Worst-case error overhead (Case 1)

CASE 2: $T_E > (e^{max} + f^{max})\tau_{bit}$: In this case, one or more frames can successfully be transmitted between two errors within an error burst. Therefore only certain sections during the exposure to the fault may contribute to the error induced overhead. The worst-case error overhead, $I_i^{err}$, in this case, is given by:

$$\begin{aligned} I_i^{err} &= (f^{max} + e^{max})\tau_{bit} \\ &+ \left( \left\lfloor \frac{l}{T_E} \right\rfloor (e^{max}\tau_{bit} \right. \\ &\left. + (T_E - e^{max}\tau_{bit} - \epsilon) \ mod \ f^{max}\tau_{bit} + \epsilon) \right) \\ &+ x \end{aligned} \tag{4.18}$$

where $T_E > 0$, $\epsilon < \tau_{bit}$ and

$$x = \begin{cases} a, & if \ a \leq \left\lfloor \frac{l}{T_E} \right\rfloor b \\ \left\lfloor \frac{l}{T_E} \right\rfloor b, & if \ \left\lfloor \frac{l}{T_E} \right\rfloor b < a \end{cases}$$

$$\begin{aligned} a &= l \ mod \ T_E \\ b &= f^{max}\tau_{bit} \\ &- (T_E - e^{max}\tau_{bit} - \epsilon) \ mod \ f^{max}\tau_{bit} + \epsilon \end{aligned}$$

The error overhead in this case includes the transmission time of the largest frame, the largest error frame, and the error overhead *during* $l$. Note that in this case, the error overhead during $l$ is strictly less than the fault duration $l$, however, Equation 4.18 is written in a general form and can be used for both cases.

The first term $(f^{max} + e^{max})\tau_{bit}$ in Equation 4.18 gives the worst-case error overhead caused by the first error in the burst and is equal to the sum of the largest message frame and the largest error frame. The second term gives the worst-case error overhead caused by a single error during the burst (except the first error) multiplied by the maximum number of errors that can occur during the error burst minus one (the first error) *assuming that the errors arrive with an exact inter-arrival time of* $T_E$. The product term $\lfloor l/T_E \rfloor$ of the second term in Equation 4.18 gives the maximum number of errors that can occur during an error burst minus one. The product term $e^{max}\tau_{bit} + (T_E - e^{max}\tau_{bit} - \epsilon)\ mod\ f^{max}\tau_{bit} + \epsilon$ of the second term includes the transmission time of the largest error frame and the largest message frame that can contribute to $I_i^{err}$. The last term $x$ in Equation 4.18 gives the additional overhead caused by the errors whose relative arrival times are larger than $T_E$. One should note that, the error overhead for a single error arrived with the minimum inter-arrival time $T_E$, plus the additional overhead per error caused by late arrivals can at most be equal to $(f^{max} + e^{max})\tau_{bit}$. Therefore, the worst-case value for $x$ is equal to either the total amount of time that can be distributed to the error inter-arrival times for late arrivals $(a)$, or the difference between the overhead assuming all errors hit the largest possible message in the last bit and the overhead assuming all errors arrive with the minimum inter-arrival time between errors within a burst $(\lfloor l/T_E \rfloor b)$, whichever is smaller. Figure 4.15 shows three different scenarios in Case 2. In the first scenario $(a = 0)$ the fault duration is an integer multiple of the minimum inter-arrival time between the errors in the burst $(l/(T_E) \in \mathbb{N}^*)$, hence in the worst-case all errors arrive with a relative arrival time equal to $T_E$ and $x = 0$. In the second scenario $(a \leq 2b)$, if the errors arrive with arrival times equal to the minimum inter-arrival time $T_E$, then after the last error in the burst there is still a time equal to $a$ until the end of the fault. As shown in Figure 4.15, in the worst-case scenario, the errors arrive later than their relative minimum inter-arrival times,

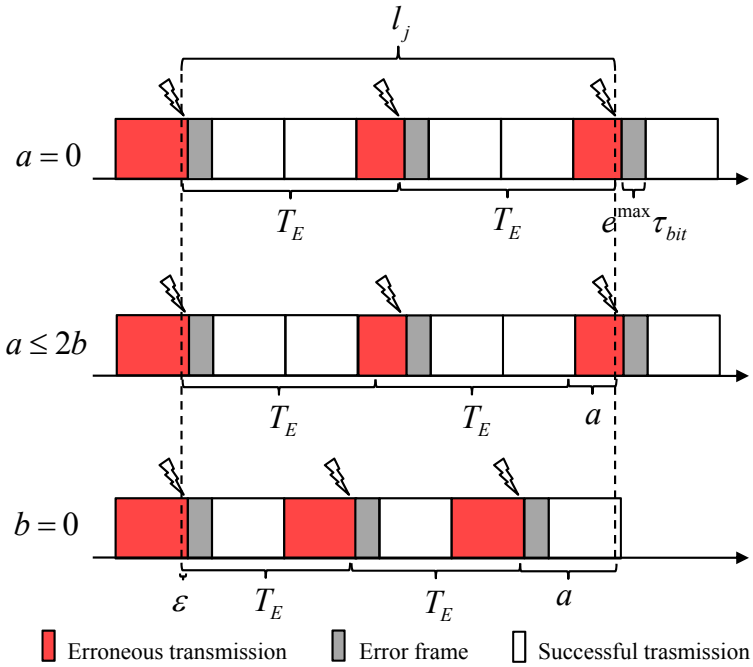


Figure 4.15: Worst-case error overhead (Case 2)

and the sum of the differences between the worst-case arrival times and the minimum inter-arrival times is equal to $a$. However, in the third scenario ($b = 0$), even though there is still a time after the last error arrival until the end of the fault, late arrivals cannot increase the overhead as it is already at maximum ($(f^{max} + e^{max})\tau_{bit}$ per error). In a scenario where $b > 0$ and $\lfloor l/T_E \rfloor b < a$, the additional overhead caused by late arrivals can at most be equal to $\lfloor l/T_E \rfloor b$.

In this section, all successfully transmitted frames between two errors in a burst are assumed to have the maximum frame size. If these frames are shorter than the maximum frame size, the error related interference may be larger than the value calculated by Expression 4.17. However, this increase in the error interference is bounded by the total sum of the differences between the actual frame sizes and the maximum frame

size, i.e, the increase in the error interference is never larger than the cumulative reduction in the frame sizes. Hence, the RTA holds for the general case when message frame sizes are less than maximum, i.e., the analysis never calculates an optimistic value.

### 4.4.3 Probabilistic Schedulability Bounds

This subsection makes the same assumption as Burns et al. made in [23], which states that, during a mission time $L$, if the actual interval between any two error arrivals $W$ is less than the assumed minimum inter-arrival time $T_F$, then the system is unschedulable. They presented an upper bound for the probability of any two error arrivals violating this minimum inter-arrival time constraint, assuming that $L$ is an even integer multiple of $T_F$ ($L/(2T_F) \in \mathbb{N}^*$), $Pr^{ub}(W < T_F) > Pr(W < T_F)$, as shown below:

$$\begin{aligned} Pr^{ub}(W < T_F) = 1 &+ [e^{-\lambda T_F}(1 + \lambda T_F)]^{\frac{L}{T_F}-1} \\ &- 2[e^{-2\lambda T_F}(1 + 2\lambda T_F)]^{\frac{L}{2T_F}} \end{aligned} \tag{4.19}$$

In addition to the unschedulability criterion given above, the system is also assumed to be unschedulable if the actual shortest time interval between any two error arrivals *within an error burst* is less than the assumed minimum inter-arrival time $T_E$ for that burst. First step towards finding an upper bound for the probability of this unschedulability criterion is to find an upper bound for the probability of violating the minimum inter-arrival time between errors *within a single burst caused by a fault of a given duration.*

**Lemma 4.4.** *Let the fault duration $l$ be an even integer multiple of the minimum inter-arrival time $T_E$ ($l/(2T_E) \in \mathbb{N}^*$). Then an upper bound for the probability of any two errors arriving less than $T_E$ apart within a duration $l$ is:*

$$\begin{aligned} Pr^{ub}_{single\_burst(l)}(W^{burst} < T_E) &= 1 \\ &+ [e^{-\lambda^{error} T_E}(1 + \lambda^{error} T_E)]^{\frac{l}{T_E}-1} \\ &- 2[e^{-2\lambda^{error} T_E}(1 + 2\lambda^{error} T_E)]^{\frac{l}{2T_E}} \end{aligned} \tag{4.20}$$

*Proof.* The proof is similar to the proof made in [23] for the upper bound of $Pr(W < T_F)$ with the following differences:

- The minimum inter-arrival time between faults $T_F$ is substituted with the minimum inter-arrival time between errors *within* the burst $T_E$
- The mission time $L$ is substituted with the fault duration $l$.

□

Next step is to uncondition on the *fault duration* and to find an upper bound for the probability of violating the minimum inter-arrival time requirement between errors within bursts caused by faults with *random durations*, during the *whole mission*.

**Theorem 4.5.** *Let $T_E$ be the minimum inter-arrival time for errors within a burst caused by a fault of duration $l$, $l$ be an even integer multiple of $T_E$ ($l/(2T_E) \in \mathbb{N}^*$) and $f(l)$ be the probability mass function for the fault durations. Then an upper bound for the probability of violating the minimum inter-arrival requirement between errors within a burst caused by a fault of* random duration*, during a mission of length $L$ is:*

$$Pr^{ub}(W^{burst} < T_E) = 1 - \left(1 - \sum_{l=l_{min}}^{l_{max}} f(l) Pr^{ub}_{single\_burst(l)}(W^{burst} < T_E)\right)^{\lambda L} \tag{4.21}$$

*where $l_{min}$ and $l_{max}$ are the lengths of the shortest and the longest faults in $f(l)$ respectively.*

*Proof.* Error bursts arrive as a Poisson process with rate $\lambda$. Once a burst arrives, errors within a burst arrive according to a Poisson process at rate $\lambda^{error}$ until the burst ends, with the duration of the fault causing the burst being distributed according to $f(l)$. Lemma 1 shows the upper bound for the probability that two errors within a single burst are separated by less that $T_E$ under the condition that the fault duration is $l$. To remove the condition on the fault duration, i.e., in order to find the probability that two errors arrive less than $T_E$ apart within a single burst caused by a *fault of random duration*, all the probabilities $Pr^{ub}_{single\_burst(l)}(W^{burst} < T_E)$ for each fault duration $l$ multiplied by the

probability of that fault duration $f(l)$ is summed:

$$Pr^{ub}_{single\_burst(random)}(W^{burst} < T_E) = \sum_{l=l_{min}}^{l_{max}} f(l) Pr^{ub}_{single\_burst(l)}(W^{burst} < T_E)$$

In order to find the upper bound for the probability of any anomalies within bursts in a mission, the expected number of error bursts during the mission, $\lambda L$, is used which is also a random variable following a Poisson process with the arrival rate $\lambda$ during a mission of length $L$. First, the lower bound for the probability of *no anomalies during a single burst caused by a fault of random duration* is calculated:

$$1 - \sum_{l=l_{min}}^{l_{max}} f(l) Pr^{ub}_{single\_burst(l)}(W^{burst} < T_E)$$

Then the lower bound for the probability of *no anomalies during the whole mission* is found:

$$\left(1 - \sum_{l=l_{min}}^{l_{max}} f(l) Pr^{ub}_{single\_burst(l)}(W^{burst} < T_E)\right)^{\lambda L}$$

Finally, the upper bound for the probability of *at least one anomaly during the entire mission* is shown to be:

$$1 - \left(1 - \sum_{l=l_{min}}^{l_{max}} f(l) Pr^{ub}_{single\_burst(l)}(W^{burst} < T_E)\right)^{\lambda L}$$

□

Finally, the derivation of an upper bound for the probability of unschedulability is presented which is the union of the upper bounds of the two presented criteria of unschedulability, viz. the probability of any two faults arriving less than $T_F$ apart *or* any two errors within a burst arriving less than $T_E$ apart (where $T_E$ is defined according to the length of the fault) during a mission of length $L$, $(Pr(W < T_F) \cup Pr(W^{burst} < T_E))$. In the proposed approach, the cause(s) of these two events are assumed to be independent. As an example, the probability of driving a car

through a path close to police stations or airports, thus exposing it to EMI which in turn result in error bursts is assumed to be independent from the factors contributing to potential occurrence of errors within a burst such as the intensity of the EMI, and the distance to the source of EMI.

**Theorem 4.6.** *Let* $((L/(2T_F) \in \mathbb{N}^*)$ *and* $(l/(2T_E) \in \mathbb{N}^*))$ *for each* $(l, T_E)$ *pair. Then an upper bound for* $Pr(W < T_F) \cup Pr(W^{burst} < T_E)$ *is:*

$$\begin{aligned} &Pr(W < T_F) \cup Pr(W^{burst} < T_E) < \\ &Pr^{ub}(W < T_F) + Pr^{ub}(W^{burst} < T_E) \\ &- Pr^{ub}(W < T_F) Pr^{ub}(W^{burst} < T_E) \end{aligned} \tag{4.22}$$

*Proof.* The general rule for finding the union of two events A and B, i.e., the probability of A *or* B occurring, can be expressed as follows:

$$Pr(A) \cup Pr(B) = Pr(A) + Pr(B) - Pr(A) \cap Pr(B)$$

As the two events are assumed to be independent, the intersection of A and B, i.e., the probability of A *and* B occurring, can be rewritten as follows:

$$Pr(A) \cap Pr(B) = Pr(A)Pr(B)$$

So the union of A and B becomes:

$$Pr(A) \cup Pr(B) = Pr(A) + Pr(B) - Pr(A)Pr(B)$$

Since the upper bounds for each event is known, the theorem is proved if the following inequality can be shown to be correct:

$$\begin{aligned} &Pr(A) + Pr(B) - Pr(A)Pr(B) < \\ &Pr^{ub}(A) + Pr^{ub}(B) - Pr^{ub}(A)Pr^{ub}(B) \end{aligned} \tag{4.23}$$

It is known that $\{Pr(A), Pr(B), Pr^{ub}(A), Pr^{ub}(B)\} \in [0, 1]$. From the following given inequalities,

$$Pr(A) < Pr^{ub}(A)$$

$$Pr(B) < Pr^{ub}(B)$$

the below equalities are derived:

$$Pr(A) + \epsilon_A = Pr^{ub}(A)$$
$$Pr(B) + \epsilon_B = Pr^{ub}(B) \tag{4.24}$$

where $\epsilon_A, \epsilon_B > 0$. $Pr^{ub}(A)$ and $Pr^{ub}(B)$ in the Inequality 4.23 are substituted with $Pr(A) + \epsilon_A$ and $Pr(B) + \epsilon_B$ respectively:

$$Pr(A) + Pr(B) - Pr(A)Pr(B) <$$
$$Pr(A) + \epsilon_A + Pr(B) + \epsilon_B - (Pr(A) + \epsilon_A)(Pr(B) + \epsilon_B)$$

$$Pr(A) + Pr(B) - Pr(A)Pr(B) <$$
$$Pr(A) + \epsilon_A + Pr(B) + \epsilon_B - Pr(A)\epsilon_B - Pr(B)\epsilon_A$$
$$- Pr(A)Pr(B) - \epsilon_A\epsilon_B$$

$Pr(A) + Pr(B) - Pr(A)Pr(B)$ is subtracted from both sides of the inequality, and it is rearranged as follows:

$$0 < \epsilon_A + \epsilon_B - Pr(A)\epsilon_B - Pr(B)\epsilon_A - \epsilon_A\epsilon_B$$
$$0 < \epsilon_A(1 - Pr(B)) + \epsilon_B(1 - Pr(A)) - \epsilon_A\epsilon_B$$
$$\epsilon_A(1 - Pr(B)) + \epsilon_B(1 - Pr(A)) > \epsilon_A\epsilon_B$$

Both sides of the inequality is divided by $\epsilon_A\epsilon_B$:

$$\frac{(1 - Pr(B))}{\epsilon_B} + \frac{(1 - Pr(A))}{\epsilon_A} > 1$$

Here, it is enough to show that each of the terms on the left hand side of the inequality is greater than or equal to 1:

$$1 - Pr(B) >= \epsilon_B$$
$$1 - Pr(A) >= \epsilon_A$$

If $\epsilon_A$ and $\epsilon_B$ are substituted with $Pr(A)^{ub} - Pr(A)$ and $Pr(B)^{ub} - Pr(B)$ respectively (using Equation set 4.24), the following inequalities are obtained:

$$Pr^{ub}(A) \leq 1$$
$$Pr^{ub}(B) \leq 1$$

which is always true. Hence, the theorem is proved. □

### 4.4.4 Illustrative Example

In this example, a distributed embedded system is assumed where 4 devices exchange messages over CAN. The message set consists of 4 messages (1 per device) as shown in Table 4.10 where columns P, N, T and D represent the priority, number of message frames, period and the deadline respectively. Priorities are ordered from 1 to 4 where 4 is the

| Task | P | N | T | D |
|---|---|---|---|---|
| A | 1 | 8 | 8 | 8 |
| B | 2 | 4 | 8 | 8 |
| C | 3 | 6 | 10 | 10 |
| D | 4 | 6 | 15 | 15 |

Table 4.10: Example message set

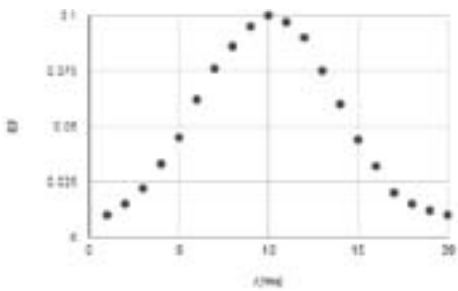

Figure 4.16: Example probability mass function $f(l)$

lowest priority. The time unit is milliseconds. Each message consists of 8 data bytes, hence $f^{max} = 135$ and $e^{max} = 31$ bits. The mission time is 1 hour ($L = 1h$), the bus speed is 0.5 Mbit/s ($\tau_{bit} = 2\mu s$), and a discrete probability distribution of the fault durations is given as in Figure 4.16.

Expected number of error bursts and errors within a burst in unit time are assumed to be $\lambda = 10^{-1}$ and $\lambda^{error} = 10^4$ respectively.

- STEP 1: First, the presented schedulability test is used to perform a series of sensitivity analyses for each error fault duration $l$ in the

probability mass function $f(l)$ and the corresponding minimum error inter-arrival times within bursts are derived for which the message set is guaranteed to be schedulable as shown on Table 4.11.

| $l$ | $T_E$ | $l$ | $T_E$ |
|---|---|---|---|
| 1 | 0 | 11 | 0.872 |
| 2 | 0 | 12 | 1.142 |
| 3 | 0 | 13 | 1.142 |
| 4 | 0.332 | 14 | 1.4 |
| 5 | 0.332 | 15 | 1.412 |
| 6 | 0.6 | 16 | 1.412 |
| 7 | 0.602 | 17 | 1.682 |
| 8 | 0.602 | 18 | 1.682 |
| 9 | 0.872 | 19 | 1.682 |
| 10 | 0.872 | 20 | 1.952 |

Table 4.11: Minimum inter-arrival times between errors within a burst

- STEP 2: In the next step, the statistical formulas presented in Section 4.4.3 is used to derive the probabilities of unschedulability for each $(l, T_E)$ pair. As $l$ is assumed to be an even integer multiple of $T_E$, the probabilities are calculated by using the smallest value greater than $T_E$ satisfying this condition. The upper bounds for the probabilities of violating the minimum inter-arrival time requirement between errors within a burst caused by a fault of the specified duration are found as shown in Table 4.12.

  In order to uncondition the probability of unschedulability on the *fault duration* and to find an upper bound for the probability of violating the minimum inter-arrival time requirement between errors within bursts caused by faults with *random durations*, during the *whole mission*, Equation 4.21, the probability mass function $f(l)$ and the probability bounds derived in the previous step are used. The upper bound for this probability is calculated as 1.3986 x $10^{-7}$.

  The upper bound probability of violating the minimum inter-arrival time requirement between error bursts during the whole mission is

| $l$ | $Pr^{ub}_{single\_burst(l)}(U)$ | $l$ | $Pr^{ub}_{single\_burst(l)}(U)$ |
|---|---|---|---|
| 1 | - | 11 | 1.4465 x $10^{-6}$ |
| 2 | - | 12 | 1.7213 x $10^{-6}$ |
| 3 | - | 13 | 2.0201 x $10^{-6}$ |
| 4 | 1.9133 x $10^{-7}$ | 14 | 2.3427 x $10^{-6}$ |
| 5 | 2.9893 x $10^{-7}$ | 15 | 2.6893 x $10^{-6}$ |
| 6 | 4.3045 x $10^{-7}$ | 16 | 3.0576 x $10^{-6}$ |
| 7 | 5.8586 x $10^{-6}$ | 17 | 3.4539 x $10^{-6}$ |
| 8 | 7.6517 x $10^{-6}$ | 18 | 3.8720 x $10^{-6}$ |
| 9 | 9.6838 x $10^{-6}$ | 19 | 4.3140 x $10^{-6}$ |
| 10 | 1.1955 x $10^{-6}$ | 20 | 4.7799 x $10^{-6}$ |

Table 4.12: Upper bound probabilities of unschedulability

calculated as 8.3333 x $10^{-8}$ assuming $T_F = 20ms$ (largest period) using Equation 4.19.

- STEP 3: Finally, in the last step, an upper bound for the probability of unschedulability (hence a lower bound for the schedulability) is calculated, which is shown to be the union of the upper bounds of the probability of at least one occurrence of any two faults arriving less than $T_F$ apart *or* at least one occurrence of any two errors within a burst, arriving less than $T_E$ apart (where $T_E$ is determined based on the fault duration) during a mission of length $L$.

This analysis shows that the example message set is schedulable with a probability of *at least* 0.999997768 during a 1 hour mission where $\lambda = 10^{-1}$, $\lambda^{error} = 10^4$ and fault duration probabilities are as given by the $f(l)$.

## 4.5 Summary

Design of dependable real-time systems demands advances in both dependability modeling and scheduling theory in tandem, to provide system level guarantees that potential error scenarios are addressed in an effective as well as efficient manner. This chapter proposes three proba-

bilistic schedulability analysis frameworks, (i) one for tasks with different criticality levels, (ii) one for network messages under burst error scenarios and (iii) one for tasks under burst error scenarios. In each framework, a sufficient analysis is presented that accounts for the worst-case interference caused by errors on the response times of tasks or messages using a stochastic error model. Further, methods for deriving probabilistic scheduling guarantees from the stochastic behaviour of errors are presented by performing a joint real-time– and reliability analysis.

# Chapter 5

# A Cascading Redundancy Approach

Temporal and spatial redundancy are widely used fault-tolerance approaches to improve the dependability of real-time systems. Each approach has a long and successful usage history in various mission/safety critical applications. However, there have been only a few attempts to combine these approaches as a unified fault-tolerant design to bring out their synergies, and appropriate design and analysis tools are still needed to achieve this goal. An earlier attempt in combining these two redundancy approaches [11] assumed the existence of perfect error detectors, an overly pessimistic and inflexible bounded error model, and was lacking a proper response time- and reliability analysis.

The framework described in this chapter addresses those issues and elaborates the cascading redundancy approach, bringing out the synergetic effects of an appropriate combination of the temporal and spatial redundancy techniques within a fault-tolerant scheduling framework. The spatial redundancy stage provides fault-tolerance in both the time and the value domains with the usage of the real-time voting strategy Voting on Time and Value (VTV) [14]. It also works as an error detector in scenarios where the total number of errors exceeds the masking capability of the spatial redundancy stage. In such scenarios, the temporal redundancy stage takes over, assuming that there is sufficient time to re-execute all the replicas before their deadlines. The framework includes a joint response time and reliability analysis for both an ideal voter, which

does not yield any *false negatives* or *false positives*, and real voters, in case information regarding the real-world performance is available for the voter.

## 5.1 System Model

Figure 5.1 illustrates the N-modular redundant configuration that will be used in this chapter. In this configuration, on each node $N_j$, a task

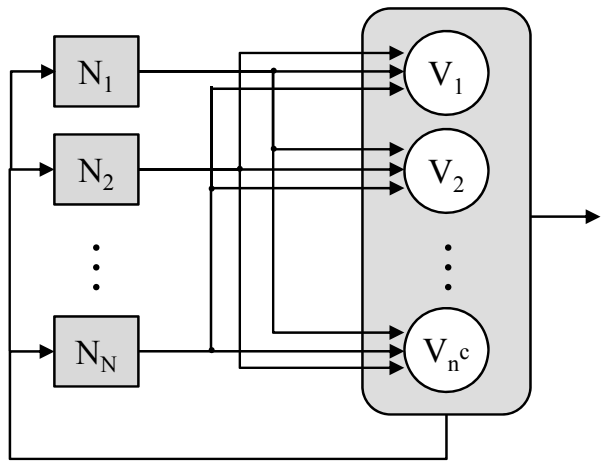


Figure 5.1: N-modular redundant configuration with temporal redundancy

set $\Gamma_j$ is allocated, where tasks represent real-time threads of execution to be scheduled by a fixed-priority scheduler. Each task $i$ has a minimum period $T_i$, known worst and best-case execution times (WCET and BCET) denoted by $C_i$ and $C_i^{min}$ respectively, a deadline $D_i$, and a unique priority $P_i$. Tasks are assumed to have deadlines equal to, or less than their periods. The task sets consist of critical and non-critical tasks where the criticality of a task could be seen as a measure of the impact of its correct (or incorrect) functioning on the overall system correctness. To provide fault-tolerance, critical tasks are replicated over a number of processing nodes.

The outputs of $n^c$ critical tasks are voted by a separate voter corresponding to each critical task, $V_k$ $(k = 1, 2, ..., n^c)$, running on a stand alone node. Upon receiving the outputs from a critical task $i$, the corresponding voter process starts executing the voting algorithm which has a known WCET denoted by $(C_i^{voter})$, and either outputs the correct value at an admissible time, or signals the non-existence of a correct output, if detected, back to the nodes for a recovery attempt by re-execution. In this chapter, the voters are assumed to be able to execute in parallel for the sake of presentation.

Nodes' clocks are allowed to drift from each other by at most a maximum deviation $\delta$ which is assured by clock synchronization mechanisms. The voter performs *inexact voting* [56, 81, 87] in the value domain assuming a maximum admissible deviation between any two non-erroneous outputs, $\sigma$. Error detection in the time domain is performed by checking whether any of the two replica outputs are separated by a time greater than the *worst-case voting jitter* $VJ_i$ of a critical task $i$ multiplied by the detector coefficient $\alpha$. If $\alpha$ is less than one, the detection of the variations in the time domain caused by faults can be improved which reduces the false negatives. However, the error detector may identify even error-free outputs as erroneous, increasing the false positives. An $\alpha$ value less than one can also be used to shorten the *worst-case response time (WCRT) of the voter*. Tuning the $\alpha$ value is an application dependent delicate task, and the analysis techniques proposed in this chapter aims to assist the system designer in performing this task.

Errors are assumed to occur randomly with a *rate* $\lambda$ per unit time which not only depends on the system but also on the type of environment. Examples of the considered errors are task outputs outside the specified value ranges, caused by incorrect input values from faulty sensors, or outside the specified time windows due to EMI on the transmissions of messages that carry the sensor values.

## 5.2 Methodology Overview

The goal of this approach is to improve the dependability of systems using the proposed fault-tolerant architecture. The analysis framework for achieving this goal is outlined in Figure 5.2. The inputs to the analysis framework consist of the task sets allocated to the distributed processing nodes, the error rate $\lambda$ determined by the system, the environment char-

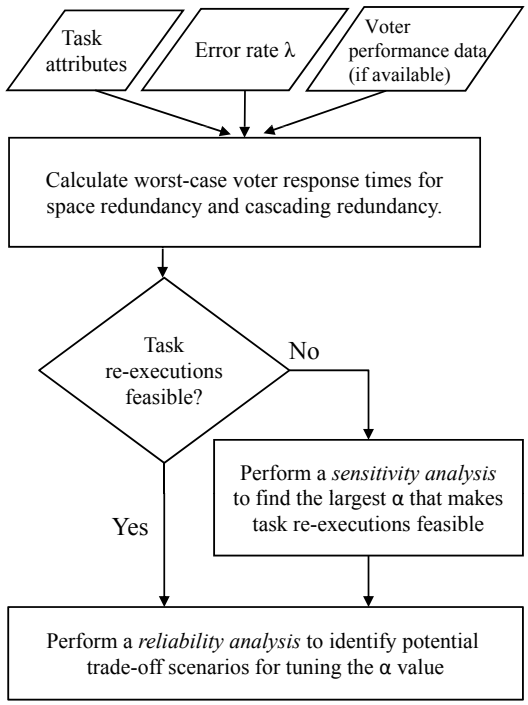


Figure 5.2: Methodology overview

acteristics, and, if available, data regarding the voter performance. The analysis begins with finding the WCRTs of the voters for each critical task for spatial redundancy and cascading redundancy configurations. If performing cascading redundancy is infeasible due to deadline misses, a sensitivity analysis is performed to identify the largest $\alpha$ value that enables meeting the deadlines. In this chapter, one global $\alpha$ value that makes all the critical tasks schedulable has been used for the sake of presentation. Since the $\alpha$ value is configured individually for each voter, one can also perform a set of sensitivity analyses to find the largest $\alpha$ values for each task. The final step is the reliability analysis performed for each task and $\alpha$ value that enables the system designer to decide whether cascading redundancy is necessary/useful or if spatial redundancy alone is

sufficient for each individual critical task.

## 5.3 Response Time Analysis

This section proposes a Response Time Analysis (RTA) for the fault-tolerant unit equipped with both the temporal and the spatial redundancy mechanisms. The conventional RTA calculates the WCRT $R_i$ for each task $i$ on a single processing node using the following equation assuming that there are no errors and, hence, no recovery attempts [49]:

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \tag{5.1}$$

On a loosely synchronized single node whose local clock is allowed to drift from the real-time by a bounded value ($\delta/2$), the $R_i$ of task $i$, relative to time 0, can further be delayed with an amount equal to the worst-case value of this drift:

$$\overline{R}_i = R_i + \frac{\delta}{2} \tag{5.2}$$

The WCRT of the voter for task $i$ ($R_i^{voter}$), in case no voter signals disagreement during the interval in which all the replicas of task $i$ that contribute to the agreement execute, is equal to the sum of the WCET of task $i$'s voter process, $C_i^{voter}$, and the largest WCRT among task $i$'s replicas, since in the worst-case, the voter needs to wait for the replica output with the largest WCRT and then finish the execution of the voting algorithm:

$$R_i^{voter} = \max_{k \in \{1..N\}} (\overline{R}_{ik}) + C_i^{voter} \tag{5.3}$$

where $\overline{R}_{ij}$ is the WCRT of the task $i$'s replica executing on the loosely synchronized node $j$.

Bril et al. [17] present the following recursive equation that calculates the *best-case response time* $R_i^{min}$ of task $i$:

$$R_i^{min} = C_i^{min} + \sum_{j \in hp(i)} \left( \left\lceil \frac{R_i^{min}}{T_j} \right\rceil - 1 \right) C_j^{min} \tag{5.4}$$

Similar to the above consideration for the WCRTs on processing nodes that are allowed to drift from the real-time, the best-case response times may also be affected by this drift with an amount equal to the worst-case value of the drift:

$$\overline{R}_i^{min} = R_i^{min} - \frac{\delta}{2} \tag{5.5}$$

**Definition 2.** *The* ***worst-case voting jitter*** $\mathrm{VJ}_{ij}$ *for a critical task replica $i$ on node $j$ (denoted by $\tau_{ij}$) is the largest response time of any replicas of task $i$ except the one on node $j$, relative to the* best-case response time $\overline{R}_{ij}^{min}$ *of the replica of task $i$ on node $j$, in an error free scenario.*

$$VJ_{ij} = \max_{k \in \{1..N\} \setminus j} (\overline{R}_{ik}) - \overline{R}_{ij}^{min} \tag{5.6}$$

A voter signals a disagreement if the total number of detected node errors exceeds the number of errors that the voter can tolerate. In the worst-case, disagreement occurs with a late timing or an omission error. In that case, the voter needs to wait for the maximum allowed time deviation between any two replica output delivery times. This value is equal to the worst-case voting jitter multiplied by the detector coefficient $\alpha$.

The WCRT of task $i$'s voter, in the case a voter signals a disagreement and the corresponding task is re-executed (denoted by $R_i^{voter}(cascading)$) includes the following:

- the WCET of task $i$'s voter process.

and the larger of the following two groups:

- if task $i$'s voter has reached a disagreement
  - the largest sum of the worst-case voting jitter multiplied by $\alpha$ for task $i$ and two times the largest WCRT among task $i$'s replicas, one for the initial execution and one for the re-execution,
  - the WCET of task $i$'s voter process for the second round of voting,
- otherwise

– the largest WCRT among task $i$'s replicas assuming re-executions of the replicas of a higher priority critical task,

$$R_i^{voter}(cascading) = \max\left( \max_{k\in\{1..N\}}(2\overline{R}_{ik} + \alpha\, VJ_{ik}) + C_i^{voter}, \max_{k\in\{1..N\}}(\overline{R}_{ik}(recovery\ in\ hp)) \right) + C_i^{voter} \tag{5.7}$$

where the WCRT of task $i$ assuming a recovery in one of the higher priority tasks, $R_i(recovery\ in\ hp)$, is calculated as follows:

$$R_i(recovery\ in\ hp) = C_i + \sum_{j\in hp(i)} \left\lceil \frac{R_i(recovery\ in\ hp)}{T_j} \right\rceil C_j + \max_{k\in hpc(i)} C_k$$

where $hpc(i)$ is the set of critical tasks with higher priorities than the priority of task $i$.

In Equation 5.7, by assuming two times the largest WCRT among the WCRTs of task $i$'s replicas, certain pessimism may be introduced to $R_i^{voter}(cascading)$ due to an unnecessary addition of interference from higher priority tasks depending on their attributes, e.g., in case a higher priority task is not re-released during the re-execution of task $i$.

As an alternative approach, one can add the largest sum of the *worst-case voting jitter multiplied by* $\alpha$ and the *worst-case re-execution time* of the tasks in $hepc(i)$, which is the set of critical tasks with priorities higher than or equal to the priority of task $i$,

$$\max_{k\in hepc(i)}(\alpha\, VJ_k + C_k)$$

to Equation 5.1, similar to Burns et al.'s [23] error interference addition to the conventional RTA [49]. However, differently from Burns et al.'s approach, this would bring the pessimism of unnecessarily adding a value equal to the *worst-case voting jitter multiplied by* $\alpha$ to the WCRT, in case the re-executed task is not task $i$. This is because task $i$'s response time is

delayed due to the voting jitter only if task $i$'s voter is causing the delay. If the disagreement is signalled by another voter, then during the time that the voter waits for detection of that disagreement, the processors can be used for serving task $i$'s replicas. While both approaches are safe, they can result in different levels of pessimism depending on the characteristics of the task sets.

## 5.4 Reliability Analysis

This section presents the reliability analysis for a triple-modular redundant unit using the temporal and spatial redundancy approach equipped with a voter using the VTV strategy. In this configuration, an error in one node during the voting process can be masked by the voter and errors in two nodes can be signalled within a bounded time. If all nodes are erroneous, no guarantee on detection or signalling the errors can be given [14].

### 5.4.1 Ideal Voter

An ideal Triple-Modular Redundancy (TMR) voter implementing the VTV strategy masks all single node errors, as well as signals all double node errors. However, it cannot provide any guarantees in a triple node error scenario, hence, it is assumed to fail assuming the worst case [14]. The VTV strategy relies on the following set of assumptions:

A1: error free nodes produce values within a specified admissible value *range* after each computation block

A2: error free nodes produce values within a specified admissible time *interval* after each computation block

A3: replica outputs with incorrect values do not form (or contribute in forming) a consensus in the value domain

A4: incorrectly timed replica outputs do not form (or contribute in forming) a consensus in the time domain

A5: there exist adequate mechanisms, which are significantly less costly than tight synchronization, that ensure a maximum permissible replica deviation from the global time

A6: the failure rate of the voting mechanism is negligibly low, as being designed and implemented as a highly reliable unit

The reliability of this voter is calculated using the *Homogeneous Poisson Process* assuming a constant error arrival rate denoted by $\lambda$.

Let $I_i^0$ be the longest time interval for task $i$, which spans from the start to the completion of any of its replicas assuming an error free scenario as shown in Figure 5.3. This time interval is calculated by the

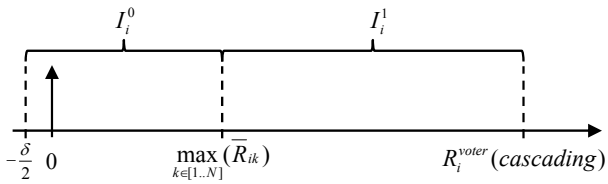


Figure 5.3: $I_i^0$ and $I_i^1$

summing up of the largest WCRT among the WCRTs of all the replicas $\max_{k\in\{1..N\}}(R_{ik})$ and the worst-case drift of the local clock from the real-time $\delta/2$.

$$I_i^0 = \max_{k\in\{1..N\}}(\overline{R}_{ik}) + \frac{\delta}{2} \tag{5.8}$$

Let $Y_i^n$ be a Poisson random variable with parameter $\lambda I_i^n$. The probability of having a correct voter output (an agreement in both the time and the value domains) for task $i$ after the first round of voting is equal to the probability of having at most one node error during the time interval in which the nodes execute the replicated tasks. If we assume a constant length for this interval whose value is equal to its worst-case ($I_i^0$), the *lower bound* probability of having a correct voter output for task $i$ after the first round of voting becomes:

$$Pr_{i,agreement}^{lb}(0) = Pr(Y_i^0 \leq 1) = e^{-\lambda I_i^0}(1 + \lambda I_i^0) \tag{5.9}$$

If exactly two errors have occurred during the first round of voting, and if there is enough time to either re-execute all the three replicas before their deadlines, or wait for the re-execution of a higher priority task's replicas (whichever takes more time), then the lower bound probability

of having a correct output becomes:

$$Pr^{lb}_{i,agreement}(1) =$$
$$Pr(Y_i^0 \leq 1 \cup (Y_i^0 = 2 \ \cap \ Y_i^1 \leq 1)) =$$
$$e^{-\lambda I_i^0}(1 + \lambda I_i^0) + \frac{e^{-\lambda I_i^0}(\lambda I_i^0)^2}{2!} e^{-\lambda I_i^1}(1 + \lambda I_i^1) \tag{5.10}$$

where $I_i^1$ is the length of the interval starting from $\max_{k \in \{1..N\}}(\overline{R}_{ik})$ and lasting until $R_i^{voter}(cascading)$ (Figure 5.3):

$$I_i^1 = R_i^{voter}(cascading) - \max_{k \in \{1..N\}}(\overline{R}_{ik}) \tag{5.11}$$

In the above statement, the probabilities $Pr(Y_i^0 \leq 1)$ and $Pr(Y_i^0 = 2 \cap Y_i^1 \leq 1)$ are mutually exclusive since if there is at most one error during the first interval whose length is equal to $I_i^0$, the second round of voting is never performed. Hence, for calculating the union of these probabilities, there is no need to subtract their intersection from their sum.

### 5.4.2 Real Voter

A real voter does not always act like an ideal one due to various reasons. For example, assumptions A1 and A2 may not always hold even with correct execution of the tasks due to, e.g., noise in sensor values. Special attention during system design and development can be paid to ensure that the assumptions A5 and A6 will hold for a required extent. However, the same cannot be done for assumptions A3 and A4, as they are truly random events. Furthermore, *positive rates* of voter outputs may be different than theoretically calculated rates since even erroneous outputs may end up contributing to an agreement in a correct output if the magnitude of the error is small, and/or the error is only in one of the value and the time domains. Obviously, the choice of the $\alpha$ value contributes to this difference as well.

The lower bound probability of having a correct voter output (a true agreement in both the time and the value domains) for task $i$ can be derived from the following probabilities:

- $Pr_{i,FP}(n, \alpha)$: The probability that the voter signals a disagreement where it was possible to reach an agreement (*false positive*), during an interval equal to $I^n$ for a given $\alpha$.

- $Pr_{i,TP}(n,\alpha)$: The probability that the voter signals a disagreement which is the correct action to be taken by the voter (*true positive*), during an interval equal to $I^n$ for a given $\alpha$.
- $Pr_{i,FN}(n,\alpha)$: The probability that the voter signals an agreement, where it is supposed to signal a disagreement according to the specifications (*false negative*), during an interval equal to $I^n$ for a given $\alpha$.

$$Pr^{lb}_{i,agreement}(0,\alpha) = 1 - (Pr_{i,FP}(0,\alpha) + Pr_{i,TP}(0,\alpha) + Pr_{i,FN}(0,\alpha)) \tag{5.12}$$

For the case of performing cascading redundancy, the probability of a correct voter output after re-executions is calculated by the following equation:

$$Pr^{lb}_{i,agreement}(1,\alpha) = Pr^{lb}_{i,agreement}(0,\alpha) + (Pr_{i,FP}(0,\alpha) + Pr_{i,TP}(0,\alpha)) (1 - (Pr_{i,FP}(1,\alpha) + Pr_{i,TP}(1,\alpha) + Pr_{i,FN}(1,\alpha))) \tag{5.13}$$

Note that the above equation is a theoretical assessment of the reliability, although it can be used to derive reliability estimates of a real voter in case information regarding the voter performance is either available from usage statistics or derived via simulations/experiments.

## 5.5 Illustrative Example

The proposed analysis is illustrated with an example. Three processing nodes are configured as a triple-modular redundant unit as shown in Figure 5.4. On each node, three tasks are allocated and scheduled by the fixed-priority scheduling policy, two of which are the replicas of the two critical tasks ($A$ and $B$) in the system. These critical tasks have slow dynamics ($T = 200ms$), while the third task on each node is a non-critical task with fast dynamics ($T = 10ms$). The task sets are shown in Table 5.1. $L$, $M$ and $H$ represent the lowest, middle and the highest priority levels respectively. The time unit is milliseconds. The non-critical tasks have deadlines equal to their periods, and the critical tasks have deadlines less than their periods.

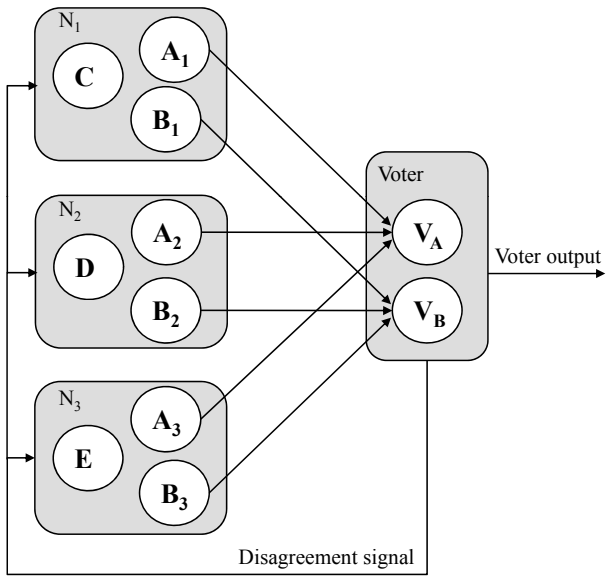


Figure 5.4: Triple-modular redundant configuration with temporal redundancy

### 5.5.1 Response Time Analysis

The worst and best-case response times and the worst-case voting jitter are calculated as shown in Table 5.2. The first two columns show the response times with respect to each local clock (assuming $\delta = 0$), and the next two columns show the response times, and the last column shows the voting jitter with respect to the real-time (assuming $\delta = 2ms$).

The response times of the voting processes in cases of only spatial redundancy and cascading redundancy are calculated by equations 5.3 and 5.7 (for $\alpha = 0.5$ and $\alpha = 1$) as shown in Table 5.3. In this example, the WCETs of the voting processes are assumed to be $C_A^{voter}, C_B^{voter} = 1ms$. For $\alpha = 1$, neither task A nor task B can complete before their deadlines. However, for $\alpha = 0.5$, they both have sufficient time to re-execute before their deadlines. If an agreement has been reached for the replicas of each critical task, then the deadlines are met for both of them.

| $Node$ | $Task$ | $P$ | $C^{min}$ | $C$ | $D$ | $T$ |
|---|---|---|---|---|---|---|
| 1 | $A_1$ | $L$ | 2 | 3 | 65 | 100 |
| | $B_1$ | $M$ | 10 | 12 | 42 | 100 |
| | $C$ | $H$ | 1 | 2 | 10 | 10 |
| 2 | $A_2$ | $L$ | 2 | 3 | 65 | 100 |
| | $B_2$ | $M$ | 10 | 12 | 42 | 100 |
| | $D$ | $H$ | 2 | 3 | 10 | 10 |
| 3 | $A_3$ | $L$ | 2 | 3 | 65 | 100 |
| | $B_3$ | $M$ | 10 | 12 | 42 | 100 |
| | $E$ | $H$ | 2 | 2 | 10 | 10 |

Table 5.1: Example task sets on three nodes

### 5.5.2 Reliability Analysis of an Ideal Voter

In this example the expected number of errors is assumed as $\lambda = 1$ per $1s$. The interval lengths $I_i^0$ and $I_i^1$ for tasks A and B are calculated as $I_A^0 = 27, I_A^1 = 35.5$ and $I_B^0 = 21, I_B^1 = 22$ respectively using equations 5.8 and 5.11. Assuming an ideal voter, the probabilities of reaching an agreement in different rounds of voting are calculated using equations 5.9 and 5.10 as shown in Table 5.4.

From these results, it can be concluded that using the cascading redundancy approach brings significant reliability improvements over using the spatial redundancy approach alone.

### 5.5.3 Reliability Analysis of a Simulated Voter

To simulate a real voter, the setup shown in Figure 5.5 is implemented using Matlab/Simulink. Three nodes forming a triple-modular redundant unit and a *reference node* are simulated, which sample a signal with random noise and output it after a delay, chosen differently for tasks A and B, simulating the execution of these tasks. Various kinds of transient errors (value, timing and omission errors) were injected to the nodes forming the triple-modular redundant unit. The outputs of the three nodes were voted using the VTV strategy. All node outputs were sent to a *perfect observer* module together with the voter outputs and the disagreement signals from the voters, in order to determine the *false positive rates* and the *false negative rates*. The drift in the local clocks

| $Task$ | $R^{min}$ | $R$ | $\overline{R}^{min}$ ($\delta = 2$) | $\overline{R}$ ($\delta = 2$) | $VJ$ |
|---|---|---|---|---|---|
| $A_1$ | 2 | 19 | 1 | 20 | 24 |
| $B_1$ | 12 | 16 | 11 | 17 | 8 |
| $C$ | 1 | 2 | 0 | 3 | - |
| $A_2$ | 2 | 24 | 1 | 25 | 19 |
| $B_2$ | 14 | 18 | 13 | 19 | 4 |
| $D$ | 2 | 2 | 1 | 3 | - |
| $A_3$ | 2 | 19 | 1 | 20 | 24 |
| $B_3$ | 14 | 16 | 13 | 17 | 6 |
| $E$ | 2 | 2 | 1 | 3 | - |

Table 5.2: WCRT of tasks before voting and the worst-case voting jitter

| $Task$ | $\alpha$ | $R^{voter}$ | $R^{voter}(cascading)$ |
|---|---|---|---|
| $A$ | 1 | 26 | 71 (UNSCH) |
| $B$ | | 20 | 46 (UNSCH) |
| $A$ | 0.5 | 26 | 61.5 |
| $B$ | | 20 | 42 |

Table 5.3: Voter response times

from the real-time was simulated by allowing local periods slightly longer or shorter than the task periods. The execution requirement was also scaled up or down based on the local period. Whenever the accumulated drift from the global clock, i.e. the accumulated sum of the difference between the local period and the real period, reaches the maximum admissible deviation from the real-time ($\delta_{clock}/2 = 1ms$), the local clock is synchronized with the global clock. This is realized by running a synchronization period shorter or longer than the real period with a value equal to the accumulated difference.

Critical tasks' true positive, false positive and false negative probabilities for $\alpha = 1$ and $\alpha = 0.5$ are found as shown in Table 5.5 after running the simulations for 100 simulation seconds for each task. In this table $n$ is the index of the interval length $I^n$ and simulated by changing the error probability which depends on the length of $I^n$ for constant $\lambda$.

| $Task$ | $n$ | $Pr_{i,agreement}(n)$ |
|---|---|---|
| $A$ | 0 ($\alpha = 1$) | 0.9996420 |
| | 1 ($\alpha = 0.5$) | 0.9999966 |
| $B$ | 0 ($\alpha = 1$) | 0.9997826 |
| | 1 ($\alpha = 0.5$) | 0.9999984 |

Table 5.4: Probabilities of reaching an agreement after $n$ re-executions (ideal voter)

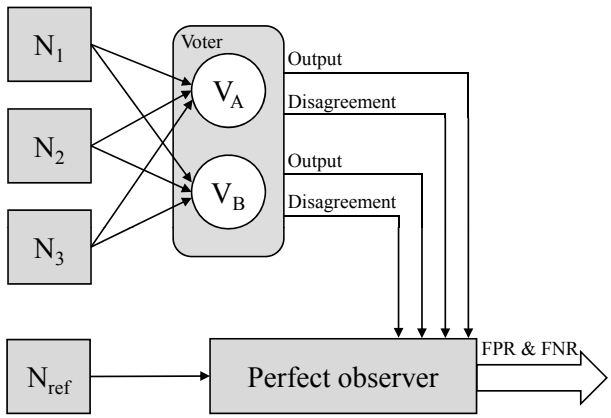


Figure 5.5: Experiment setup

By using an $\alpha = 0.5$ for critical tasks A and B, re-execution chances are given to both while still guaranteeing schedulability assuming that the second round of voting ends up in an agreement. By doing so, the probability of failure has been decreased by more than an *order of magnitude* for both tasks as shown in Table 5.6 (calculated by equations 5.12 and 5.13), though task B has benefited from the cascading redundancy approach more than twice as much.

On the other hand, by reducing $\alpha$ to 0.5, the probability of performing

| $Task$ | $\alpha$ | $n$ | $Pr_{i,TP}(n,\alpha)$ | $Pr_{i,FP}(n,\alpha)$ | $Pr_{i,FN}(n,\alpha)$ |
|---|---|---|---|---|---|
| $A$ | 1 | 0 | 0.0003 | 0.0022 | 0.00019 |
| | 0.5 | 0 | 0.00073 | 0.00604 | 0.00012 |
| | 0.5 | 1 | 0.00104 | 0.00987 | 0.00014 |
| $B$ | 1 | 0 | 0.00021 | 0.00113 | 0.00003 |
| | 0.5 | 0 | 0.00046 | 0.00586 | 0.00001 |
| | 0.5 | 1 | 0.00051 | 0.00612 | 0.00001 |

Table 5.5: Critical tasks' true positive, false positive and false negative probabilities for $\alpha = 1$ and $\alpha = 0.5$

| $Task$ | $n$ | $Pr_{i,agreement}(n)$ |
|---|---|---|
| $A$ | 0 ($\alpha = 1$) | 0.99731 |
| | 1 ($\alpha = 0.5$) | 0.99981 |
| $B$ | 0 ($\alpha = 1$) | 0.99863 |
| | 1 ($\alpha = 0.5$) | 0.99995 |

Table 5.6: Probabilities of reaching an agreement after $n$ feasible re-executions (simulated voter)

a feasible re-execution for task $B$ is increased by:

$$\frac{Pr_{B,TP}(0,0.5) + Pr_{B,FP}(0,0.5)}{Pr_{B,TP}(0,1) + Pr_{B,FP}(0,1)} = \frac{0.00632}{0.00134} \cong 4.72$$

while this increase for task $A$ is

$$\frac{Pr_{A,TP}(0,0.5) + Pr_{A,FP}(0,0.5)}{Pr_{A,TP}(0,1) + Pr_{A,FP}(0,1)} = \frac{0.00677}{0.0025} \cong 2.71$$

These results would assist the system designer to decide which $\alpha$ value to choose for each critical task, based on (i) the individual task reliability requirements and (ii) the probability of performing unnecessary re-executions due to false positives.

## 5.6 Summary

This chapter presents a fault-tolerance strategy where error recovery is performed by either masking up to a specified number of errors with the use of spatial redundancy, or, in a cascading manner, by first detecting a disagreement in the spatial redundancy stage and then performing temporal redundancy. The RTA for the voting process has been revised to account for the effect of local clocks' drift from the real-time, the voting jitter and the selected $\alpha$ value. Using a stochastic error model, a method for performing reliability analysis has been proposed for both ideal voters and real voters in case information regarding the real-world performance is available for the voter. This analysis shows the benefits and drawbacks of using different $\alpha$ values and allows performing a trade-off analysis during the design phase for choosing an $\alpha$ value suitable for the application.

# Chapter 6

# Summary

Dependability is a fundamental property of safety and mission critical systems due to the potential hazards they can cause to people and the environment. Provision of fault-tolerance in the real-time embedded systems domain is an essential requirement for achieving dependability, but a delicate task which not only includes predictable assurance of functional correctness, but also timeliness in an effective and resource efficient manner.

Safety and mission critical systems have been mainly utilizing fault-tolerance strategies in order to attain dependability where accurate detection of and effective recovery from errors are crucial. However, in many cases, conventional fault-tolerance strategies are needed to be redesigned or adapted to be applicable in the real-time embedded systems domain, in order to address many important aspects, such as the inclusion of error detection in the time domain. Moreover, computational resources for providing fault-tolerance are often limited in the embedded systems domain due to the underlying constraints related to, e.g., space, weight and cost. Considering these constraints, the design process of developing dependable embedded systems involves the selection of appropriate fault-tolerance strategies to be used in critical parts of the system. This requires appropriate design and development tools where the designer can perform various trade-off analyses in order to make favourable design choices.

A fundamental task towards predicting systems' dependability is a realistic and a detailed model of faults and errors. Particularly, the

behaviour of errors caused by transient and intermittent faults can be very complex, due to the random behaviour of these errors with often changing rates during the operational life of the system. One major challenge in predicting the dependability of real-time systems is how to combine the techniques used for providing timing guarantees with those used for providing reliability guarantees. Real-time scheduling research is based on worst-case guarantees, i.e. it tries to answer the question if a system is schedulable even if the absolute worst pattern of events take place together with the absolute longest execution times, whereas, in case of error events, there may not be a worst-case bound, as these events are random by their nature. The existing fault-tolerant real-time research most often assumes worst-case error scenarios (which hold for a certain probability), and develops timing analysis techniques based on these assumptions. As a result, in many cases, these analysis techniques do not permit tuning these assumptions at a later stage, in order to adapt to a new environment or changing system properties, thus, limiting its applicability.

Hence, there is a research need for accurately modeling the randomly occurring complex error scenarios, development of fault-tolerance strategies targeting the real-time embedded systems domain, and techniques to combine timing and reliability analysis.

## 6.1 Results

This thesis presents methods for

- stochastically modeling errors,
- providing fault-tolerance in task and message scheduling in embedded real-time systems,
- analyzing the reliability of these systems.

Firstly, a spatial redundancy approach with a majority voting strategy is presented, which performs voting in both the time and the value domains, applicable for loosely synchronized dependable real-time systems. Its performance is evaluated against the existing strategies targeting similar types of applications, which shows that the presented strategy outperforms the existing ones.

Secondly, two methods for providing fault-tolerance using the temporal redundancy approach is presented, targeting task and message scheduling respectively, for task and message sets with mixed criticality levels.

Subsequently, a set of probabilistic schedulability analysis techniques is presented focusing on the mixed criticality levels of task sets, as well as considering the burst errors in task and message scheduling respectively.

Finally, a cascading redundancy approach that combines the spatial and temporal redundancy approaches is presented, including a joint timing and reliability analysis for the proposed approach. This work aims to demonstrate how the presented concepts can be used in a fault-tolerant design and analysis framework.

## 6.2 Conclusions

The fault-tolerant design and analysis framework proposed in this thesis provides a more accurate modeling of real error scenarios compared to the fault-tolerance techniques targeting real-time systems using worst-case error scenarios.

The proposed fault-tolerance techniques focus on addressing the inclusion of various essential properties of dependable embedded real-time systems lacking in many of the existing approaches, such as resource awareness and consideration of the errors in the time domain.

The proposed probabilistic real-time analysis techniques allow the system designers to analyze if the dependability measures taken in the form of fault-tolerance are adequate to satisfy the dependability requirements.

## 6.3 Future Work

This thesis can form a basis for various possible future work in the area of dependable real-time embedded systems design, including the following:

- *Development of Probabilistic Real-Time Analysis (PRTA) techniques that utilize the proposed stochastic error model to its full extent.*

  Although each of the proposed probabilistic timing analysis techniques show how to relax various assumptions traditionally made in dependable real-time research, such as assuming only singleton

errors, maximum one error burst occurrence per any message instance and error bursts that consist of continuous errors, none of them do relax *all* of these assumptions and including them in a single framework remains as a future challenge.

- *Consideration of the mixed criticality of tasks/messages in each of the proposed PRTA techniques.*

  The PRTA techniques proposed for task and message scheduling under burst errors assume that all tasks/messages are critical. A future challenge is to extend it to address the mixed criticality nature of safety/mission critical embedded systems.

- *Implementation of a fault-tolerant design and analysis framework*

  Various fault-tolerance strategies and analysis techniques are presented in this thesis including a demonstration of a possible instantiation of a fault-tolerant design and analysis framework. A potential future work is to extend this framework by including other fault-tolerance and analysis techniques used in the embedded real-time system domain, thereby providing a more complete framework that additionally could be integrated into a development environment.

# Appendix A

# List of Acronyms

| | |
|---|---|
| *BCET* | Best-Case Execution Time |
| *CAN* | Controller Area Network |
| *CMV* | Compare Majority Voting |
| *CRC* | Cyclic Redundancy Check |
| *DM* | Deadline Monotonic |
| *DPS* | Dynamic Priority Scheduling |
| *ECU* | Electronic Control Unit |
| *EDF* | Earliest Deadline First |
| *EMC* | Electro Magnetic Compatibility |
| *EMI* | Electro Magnetic Interference |
| *FA* | Fault-Aware |
| *FMEA* | Failure Mode and Effects Analysis |
| *FNR* | False Negative Rate |
| *FPR* | False Positive Rate |
| *FPS* | Fixed Priority Scheduling |
| *FPTA* | Failure Propagation and Transformation Analysis |
| *FPTC* | Failure Propagation and Transformation Calculus |
| *FPTN* | Fault Propagation and Transformation Notation |
| *FT* | Fault-Tolerant |

| | |
|---|---|
| $FTA$ | Fault Tree Analysis |
| $ILP$ | Integer Linear Programming |
| $LCM$ | Least Common Multiple |
| $NMR$ | N-Modular Redundancy |
| $PRTA$ | Probabilistic Real-Time Analysis |
| $QMV$ | Quorum Majority Voting |
| $QoS$ | Quality of Service |
| $RM$ | Rate Monotonic |
| $RTA$ | Response Time Analysis |
| $TMR$ | Triple-Modular Redundancy |
| $TNR$ | True Negative Rate |
| $TPR$ | True Positive Rate |
| $VTV$ | Voting on Time and Value |
| $WCET$ | Worst-Case Execution Time |
| $WCRT$ | Worst-Case Response Time |

# Appendix B

# List of Notations

| | |
|---|---|
| $\alpha$ | detector coefficient |
| $\Gamma$ | task/message set |
| $\Gamma_c$ | subset of critical tasks/messages in $\Gamma$ |
| $\overline{\Gamma_c}$ | subset of critical task/message alternates in $\Gamma$ |
| $\Gamma_{nc}$ | subset of non-critical tasks/messages in $\Gamma$ |
| $\Gamma_{nc}^{FA}$ | subset of non-critical tasks/messages with valid FA deadlines in $\Gamma_{nc}$ |
| $\Gamma_{nc}^{non-FA}$ | subset of non-critical tasks/messages without valid FA deadlines in $\Gamma_{nc}$ |
| $\Gamma_{t_k}$ | subset of tasks/messages in $\Gamma$ that are released at time $t_k$ |
| $\delta$ | maximum deviation in the time domain between any two replica outputs, in an error-free scenario, as perceived by the voter |
| $\Delta$ | maximum admissible deviation between any two voter outputs in the time domain as per the real-time and dependability specifications |
| $\epsilon$ | an arbitrarily small real number |

| | |
|---|---|
| $\eta$ | number of data bytes in a CAN message frame |
| $\lambda$ | fault rate |
| $\lambda^{error}$ | error occurrence rate within a fault |
| $\sigma$ | maximum admissible deviation in the value domain between any two replica outputs |
| $\tau_{bit}$ | bit time |
| $\tau_i$ | task $i$ |
| $\tau_h$ | task with the highest priority |
| $\tau_i^j$ | $j^{th}$ instance of task $i$ |
| $\tau_{ij}$ | replica of task $i$ on node $j$ |
| $\overline{\tau_i}$ | alternate task $i$ |
| $B_i$ | non-preemptive transmission of a lower priority message frame, or the non-preemptive transmission of a message frame belonging to the previous instance of the message $i$ |
| $C_i$ | worst-case transmission/execution time of task/message $i$ |
| $\overline{C_i}$ | worst-case execution time of alternate task $i$ |
| $C_i^{min}$ | best-case transmission/execution time of task/message $i$ |
| $C_i^{voter}$ | worst-case execution time of the voter algorithm for task $i$ |
| $D_i$ | relative deadline of task/message $i$ |
| $\overline{D_i}$ | relative deadline of alternate task/message $i$ |
| $D_i^j$ | relative deadline of the $j^{th}$ instance of task/message $i$ |
| $D_i^j(FA)$ | relative fault-aware deadline of the $j^{th}$ instance of non-critical task/message $i$ |
| $D_i^j(FT)$ | relative fault-tolerant deadline of the $j^{th}$ instance of critical task/message $i$ |
| $\overline{D_i^j}(FT)$ | relative fault-tolerant deadline of the $j^{th}$ instance of alternate task/message $i$ |
| $e$ | size of a transmitted CAN error frame |
| $e^{max}$ | size of the largest CAN error frame |

| | |
|---|---|
| $est(\tau_i)$ | earliest start time of task $i$ |
| $E$ | total number of erroneous outputs |
| $E_t$ | number of outputs that are erroneous in the time domain |
| $E_v$ | number of outputs that are erroneous in the value domain |
| $E_{vt}$ | number of outputs that are erroneous in both the time and the value domains |
| $E_t^e$ | number of outputs that are erroneous in the time domain with an early timing error |
| $E_t^l$ | number of outputs that are erroneous in the time domain with a late timing error |
| $E_{vt}^e$ | number of outputs that are erroneous in both the time and the value domains with an early timing error |
| $E_{vt}^l$ | number of outputs that are erroneous in both the time and the value domains with a late timing error |
| $f$ | size of a transmitted CAN message frame |
| $f^{max}$ | size of the largest CAN message frame |
| $F_i$ | fault $i$ |
| $G$ | goal function |
| $hep(i)$ | set of tasks/messages with priorities higher than or equal to that of task/message $i$ |
| $hepc(i)$ | set of critical tasks/messages with priorities higher than or equal to that of task/message $i$ |
| $hp(i)$ | set of tasks/messages with priorities higher than that of task/message $i$ |
| $hpc(i)$ | set of critical tasks/messages with priorities higher than that of task/message $i$ |
| $I_i$ | total worst-case interference from higher priority tasks/messages or errors on task/message $i$ |

| | |
|---|---|
| $I_i^{err}$ | worst-case error interference on task/message $i$ |
| $I_i^{err_j}$ | worst-case error interference on task/message $i$ caused by fault $j$ |
| $I_i^{hp}$ | worst-case interference from higher priority tasks/messages on task/message $i$ |
| $I_i^n$ | the largest interval, for the $n^{th}$ re-execution of task $i$'s replicas, which spans from the start to the completion of any of its replicas assuming an error free scenario |
| $J_i$ | worst-case queuing jitter of message $i$ |
| $J_i^{min}$ | best-case queuing jitter of message $i$ |
| $k_i$ | number of instances of task $i$ |
| $l_i$ | duration of fault $i$ |
| $L$ | mission time |
| $m_i$ | number of frames forming message $i$ |
| $M_i$ | message $i$ |
| $M_h$ | message with the highest priority |
| $M_i^j$ | $j^{th}$ instance of message $i$ |
| $M_t$ | minimum number of replicas required to form a consensus in the time domain |
| $M_v$ | minimum number of replicas required to form a consensus in the value domain |
| $n$ | number of tasks/messages |
| $n_i$ | number of instances of task $i$ over LCM |
| $n^c$ | number of critical tasks |
| $N$ | the number of processing nodes in the system / modules in an NMR configuration |
| $N_i$ | node $i$ |
| $O_i$ | offset of task/message $i$ |
| $P_i$ | priority of task/message $i$ |
| $P_i^j$ | priority of the artifact task/message (originally the $j^{th}$ instance of task/message $i$) |
| $\overline{P_i^j}$ | priority of the artifact task/message alternate (originally the $j^{th}$ instance of alternate task/message $i$) |

| | |
|---|---|
| $Pr^{lb}$ | a lower bound probability |
| $Pr^{ub}$ | an upper bound probability |
| $Pr(S)$ | probability of schedulability |
| $Pr(U)$ | probability of unschedulability |
| $Pr_{single\_burst(l)}(C)$ | probability within a single burst caused by a fault with duration $l$ |
| $Pr_{single\_burst(random)}$ | probability within a single burst caused by a fault with random duration |
| $Pr_{i,agreement}(n)$ | probability that the voter reaches an agreement after $n$ re-executions of task $i$ |
| $Pr_{i,agreement}(n,\alpha)$ | probability that the voter reaches an agreement after $n$ re-executions of task $i$ for the selected $\alpha$ |
| $Pr_{i,FN}(n,\alpha)$ | probability of a false negative for the selected $\alpha$ during an interval equal to $I^n$ for task $i$ |
| $Pr_{i,FP}(n,\alpha)$ | probability of a false positive for the selected $\alpha$ during an interval equal to $I^n$ for task $i$ |
| $Pr_{i,TP}(n,\alpha)$ | probability of a true positive for the selected $\alpha$ during an interval equal to $I^n$ for task $i$ |
| $q_i$ | worst-case queuing delay of message $i$ |
| $r_i$ | the number of frames that are required to be guaranteed for re-transmission for message $i$ |
| $R(i)$ | reliability requirement of task $i$ |
| $R_i$ | worst-case latency/response time of message/task $i$ |
| $\overline{R}_i$ | worst-case latency/response time of message/task $i$ on a loosely synchronized node |
| $R_i^{min}$ | best-case latency/response time of message/task $i$ |
| $R_{ij}$ | worst-case latency/response time of task $i$'s replica on node $j$ |
| $R_{ij}^{min}$ | best-case latency/response time of task $i$'s replica on node $j$ |

| | |
|---|---|
| $R_i^{voter}$ | worst-case voter response time for task $i$ |
| $R_i^{voter}(cascading)$ | worst-case voter response time for task $i$ after re-execution of its replicas |
| $T_E$ | minimum error inter-arrival time within a fault |
| $T_F$ | minimum fault inter-arrival time |
| $T_{F_i}$ | minimum fault inter-arrival time specified for task $i$ |
| $T_i$ | period/minimum inter-arrival time of message/task $i$ |
| $t^*$ | the correct task/message output delivery time, as seen by a perfect observer |
| $t_i$ | the time point that node/task/message $i$ output is delivered |
| $v^*$ | the correct task/message output value, as seen by a perfect observer |
| $v_i$ | the output value of node/task/message $i$ |
| $VJ_{ij}$ | worst-case voting jitter of task $i$ on node $j$ |
| $W_F$ | actual minimum fault inter-arrival time |
| $W_E$ | actual minimum error inter-arrival time |

# Bibliography


[1] CAN in Automation, CAN Specifications. `http://www.can-cia.org`.

[2] IEC 60812. Functional safety of electrical/electronical/programmable electronic safety/related systems, analysis techniques for system reliability - procedure for failure mode and effect analysis (FMEA), 1991. International Electrotechnical Commission IEC.

[3] L. Abeni and G. Buttazzo. Stochastic Analysis of a Reservation Based System. *15th International Parallel and Distributed Processing Symposium*, pages 946–952, April 2001.

[4] B. Andersson and E. Tovar. The Utilization Bound of Non-Preemptive Rate-Monotonic Scheduling in Controller Area Networks is 25 percent. *IEEE Symposium on Industrial Embedded Systems*, July 2009.

[5] A. K. Atlas and A. Bestavros. Statistical Rate Monotonic Scheduling. *19th IEEE Real-Time Systems Symposium*, pages 123–132, December 1998.

[6] N. C. Audsley. Deadline Monotonic Scheduling. Technical Report YCS 146, Department of Computer Science, University of York, October 1990.

[7] N. C. Audsley, A. Burns, R. Davis, K. Tindell, and A. J. Wellings. Fixed Priority Preemptive Scheduling: A Historical Perspective. *Real Time Systems*, 8(3):173–198, March/May 1995.

[8] N. C. Audsley, A. Burns, M. F. Richardson, K. Tindell, and A. J. Wellings. Applying New Scheduling Theory to Static Priority Preemptive Scheduling. *Software Engineering Journal*, 8(5):284–292, September 1993.

[9] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings. Hard Real-Time Scheduling: The Deadline-Monotonic Approach. *8th IEEE Workshop on Real-Time Operating Systems*, pages 133–137, May 1991.

[10] A. Avizienis, J. Laprie, and B. Randell. Fundamental Concepts of Dependability. Technical Report Research Report N01145, LAAS-CNRS, April 2001.

[11] H. Aysan, R. Dobrin, and S. Punnekkat. A Cascading Redundancy Approach for Dependable Real-Time Systems. *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 467–476, August 2009.

[12] H. Aysan, R. Dobrin, and S. Punnekkat. Efficient Fault Tolerant Scheduling on Control Area Network (CAN). *International Conference on Emerging Technologies and Factory Automation*, pages 1–8, September 2010.

[13] H. Aysan, S. Punnekkat, and R. Dobrin. Error Modeling in Dependable Component-based Systems. *IEEE International Workshop on Component-Based Design of Resource-Constrained Systems*, pages 1309–1314, July 2008.

[14] H. Aysan, S. Punnekkat, and R. Dobrin. VTV - A Voting Strategy for Real-Time Systems. *IEEE Pacific Rim International Symposium on Dependable Computing*, pages 56–63, December 2008.

[15] D. M. Blough and G. F. Sullivan. A Comparison of Voting Strategies for Fault-Tolerant Distributed Systems. *Symposium on Reliable Distributed Systems*, pages 136–145, October 1990.

[16] A. Bondavalli and L. Simoncini. Failure Classification with respect to Detection. *IEEE Workshop on Future Trends in Distributed Computing Systems*, pages 47–53, September/October 1990.

[17] R. J. Bril, E. F. M. Steffens, and W. F. J. Verhaegh. Best-Case Response Times of Real-Time Tasks. *Philips Workshop on Scheduling and Resource Management*, pages 19–27, June 2001.

[18] S. S. Brilliant, J. C. Knight, and N. G. Leveson. The Consistent Comparison Problem in N-Version Software. *IEEE Transactions on Software Engineering*, 15(11):1481–1484, November 1989.

[19] I. Broster. Flexibility in Dependable Real-time Communication. *Department of Computer Science, University of York*, August 2003.

[20] I. Broster, A. Burns, and G. Rodrguez-Navas. Probabilistic Analysis of CAN with Faults. *IEEE Real-Time Systems Symposium*, pages 269–278, December 2002.

[21] I. Broster, A. Burns, and G. Rodrguez-Navas. Timing Analysis of Real-Time Communication under Electromagnetic Interference. *Real-Time Systems*, 30(1-2):55–81, May 2005.

[22] A. Burns, R. I. Davis, and S. Punnekkat. Feasibility Analysis of Fault-Tolerant Real-Time Task Sets. *Euromicro Workshop on Real-Time Systems*, pages 29–33, June 1996.

[23] A. Burns, S. Punnekkat, L. Strigini, and D. R. Wright. Probabilistic Scheduling Guarantees for Fault-Tolerant Real-Time Systems. *7th IFIP International Working Conference on Dependable Computing for Critical Applications*, pages 361–378, January 1999.

[24] J. Charzinski. Performance of the Error Detection Mechanisms in CAN. *International CAN Conference*, pages 20–29, September 1994.

[25] L. Chen and A. Avižienis. N-Version Programming: A Fault Tolerance Approach to Reliability of Software Operation. *Annual International Symposium on Fault Tolerant Computing*, pages 3–9, June 1978.

[26] H. Chetto and M. Chetto. Some Results of the Earliest Deadline Scheduling Algorithm. *IEEE Transactions on Software Engineering*, 15(10):1261–1269, October 1989.

[27] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien. Controller Area Network (CAN) Schedulability Analysis: Refuted, Revisited and Revised. *Real-Time Systems*, 35(3):239–272, April 2007.

[28] M. L. Dertouzos. Control Robotics: the Procedural Control of Physical Processes. *Information Processing*, August 1974.

[29] M. Di Natale. Scheduling the CAN Bus with Earliest Deadline Techniques. *IEEE Real-Time Systems Symposium*, pages 259–268, November 2000.

[30] R. Dobrin, H. Aysan, and S. Punnekkat. Maximizing the Fault Tolerance Capability of Fixed Priority Schedules. *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, August 2008.

[31] S. Edgar and A. Burns. Statistical Analysis of WCET for Scheduling. *IEEE Real-Time Systems Symposium*, December 2001.

[32] P. Ezhilchelvan, J.-M. Helary, and M. Raynal. Building Responsive TMR-Based Servers in Presence of Timing Constraints. *IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, May 2005.

[33] P. D. Ezhilchelvan and S. K. Shrivastava. A Characterization of Faults in Systems. Technical report, Computing Laboratory, University of Newcastle upon Tyne, September 1985.

[34] P. D. Ezhilchelvan and S. K. Shrivastava. A Classification of Faults in Systems. Technical report, Computing Laboratory, University of Newcastle upon Tyne, 1989.

[35] P. Fenelon and J. A. Mcdermid. Integrated Techniques For Software Safety Analysis. *IEE Colloquium on Hazard Analysis*, November 1992.

[36] J. Ferreira. An Experiment to Assess Bit Error Rate in CAN. *International Workshop of Real-Time Networks*, pages 15–18, June 2004.

[37] V. De Florio, G. Deconinck, and R. Lauwereins. The EFTOS Voting Farm: A Software Tool for Fault Masking in Message Passing

Parallel Environments. *Euromicro Conference*, 1:379–386, August 1998.

[38] M. K. Gardner and J. W. S. Liu. Analyzing Stochastic Fixed-Priority Real-Time Systems. *International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Joint European Conferences on Theory and Practice of Software*, pages 44–58, March 1999.

[39] X. Ge, R. F. Paige, and J. A. Mcdermid. Probabilistic Failure Propagation and Transformation Analysis. *International Conference on Computer Safety, Reliability, and Security*, pages 215–228, September 2009.

[40] S. Ghosh, R. Melhem, and D. Mosse. Enhancing Real-time Schedules to Tolerate Transient Faults. *IEEE Real-Time Systems Symposium*, December 1995.

[41] F. Di Giandomenico and L. Strigini. Adjudicators for Diverse-Redundant Components. *Symposium on Reliable Distributed Systems*, pages 114–123, October 1990.

[42] A. Grnarov, J. Arlat, and A. Avizienis. Modeling of Software Fault-Tolerance Strategies. *Annual Pittsburgh Modeling and Simulation Conference*, pages 571–578, May 1980.

[43] M. Grottke and K. S. Trivedi. Fighting Bugs: Remove, Retry, Replicate, and Rejuvenate. *Computer*, 40:107–109, February 2007.

[44] C.-C. Han, K. G. Shin, and J. Wu. A Fault-Tolerant Scheduling Algorithm for Real-Time Periodic Tasks with Possible Software Faults. *IEEE Transactions Computers*, 52(3):362–372, March 2003.

[45] A. L. Hopkins, T. B. Smith, and J. H. Lala. FTMP: A Highly Reliable Fault-Tolerant Multiprocessor for Aircraft. *Proceedings of the IEEE*, 66(10):1221–1239, October 1978.

[46] J. J. Horning, H. C. Lauer, P. M. Melliar-Smith, and B. Randell. A Program Structure for Error Detection and Recovery. *Lecture Notes in Computer Science*, 16:177–193, April 1974.

[47] ISO-11898. Road Vehicles - Interchange of Digital Information - Controller Area Network (CAN) for High Speed Communication, 1993. International Standards Organisation.

[48] V. Izosimov. Scheduling and Optimization of Fault-Tolerant Distributed Embedded Systems. *Department of Computer and Information Science, Linkping University*, November 2009.

[49] M. Joseph and P. Pandya. Finding Response Times in a Real-Time System. *The Computer Journal - British Computer Society*, 29(5):390–395, October 1986.

[50] K. Kim, J. L. Diaz, L. Lo Bello, J. M. Lopez, C.-G. Lee, and S. L. Min. An Exact Stochastic Analysis of Priority-Driven Periodic Real-Time Systems and Its Approximations. *IEEE Transactions on Computers*, 54(11):1460–1466, November 2005.

[51] H. Kopetz. Fault Containment and Error Detection in the Time-Triggered Architecture. *International Symposium on Autonomous Decentralized Systems*, April 2003.

[52] H. Kopetz. From a Federated to an Integrated Architecture for Dependable Embedded Real-Time Systems. Technical Report 22, Technische Universitat Wien, January 2004.

[53] H. Kopetz. On the Fault Hypothesis for a Safety-Critical Real-Time System. *Workshop on Future Generation Software Architectures in the Automotive Domain*, January 2004.

[54] C. M. Krishna and K. G. Shin. On Scheduling Tasks with a Quick Recovery from Failure. *IEEE Transactions on Computers*, 35(5):448–455, May 1986.

[55] J. Lala, L. Alger, S. Friend, G. Greeley, S. Sacco, and S. Adams. An Analysis of Redundancy Management Algorithms for Asynchronous Fault Tolerant Control Systems. Technical Report Research Report NASA-TM-100007, NASA, September 1987.

[56] J. H. Lala and R. E. Harper. Architectural Principles for Safety-Critical Real-Time Applications. *Proceedings of the IEEE*, 82(1):25–40, January 1994.

[57] J-C. Laprie. Dependable Computing and Fault-Tolerance: Concepts and Terminology. *International Symposium on Fault-Tolerant Computing, ' Highlights from Twenty-Five Years'*, June 1995.

[58] G. Latif-Shabgahi and and S. Bennett J. M. Bass. A Taxonomy for Software Voting Algorithms Used in Safety-Critical Systems. *IEEE Transactions on Reliability*, 53(3):319–328, September 2004.

[59] J. P. Lehoczky, L. Sha, and Y. Ding. The Rate Monotonic Scheduling Algorithm - Exact Characterization and Average Case Behaviour. *11th IEEE Real-time Systems Symposium*, pages 166–171, December 1989.

[60] A. Leulseged and N. Nissanke. Probabilistic Analysis of Multiprocessor Scheduling of Tasks with Uncertain Parameters. *Lecture Notes in Computer Science*, 2968:103–122, April 2004.

[61] J. Leung and J. Whitehead. On the Complexity of Fixed-Priority Scheduling of Periodic Real-Time Tasks. *Performance Evaluation*, 2(4):237–250, December 1982.

[62] J. Y.-T. Leung and M. L. Merrill. A Note on Preemptive Scheduling of Periodic, Real-Time Tasks. *Information Processing Letters*, 11(3):115–118, November 1980.

[63] N. G. Leveson. Software Safety: Why, What and How? *ACM Computing Surveys*, 18(2):125–163, June 1986.

[64] A. L. Liestman and R. H. Campbell. A Fault-Tolerant Scheduling Problem. *IEEE Transactions on Software Engineering*, 12(11):1089–95, November 1986.

[65] K. Lingasubramanian and S. Bhanja. An Error Model to Study the Behavior of Transient Errors in Sequential Circuits. pages 485–490, January 2009.

[66] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *Journal of the ACM*, 20(1):40–61, January 1973.

[67] P. R. Lorczak, A. K. Caglayan, and D. E. Eckhardt. A Theoretical Investigation of Generalized Voters for Redundant Systems. *International Symposium on Fault-Tolerant Computing, Digest of Papers*, pages 444–451, June 1989.

[68] R. E. Lyons and W. Vanderkulk. The Use of Triple-Modular Redundancy to Improve Computer Reliability. *Journal of Research and Development*, 6:200–209, April 1962.

[69] F. Many and D. Doose. Scheduling Analysis under Fault Bursts. *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 113–122, April 2011.

[70] M. Matsumoto and T. Nishimura. Mersenne Twister: a 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30, January 1998.

[71] A. K. Mok. Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment. *Massachusetts Institute of Technology*, May 1983.

[72] N. Navet. Controller Area Networks: CAN's use within Automobiles. *IEEE Potentials*, pages 12–14, October/November 1998.

[73] N. Navet, Y-Q. Song, and F. Simonot. Worst-Case Deadline Failure Probability in Real-Time Applications Distributed over Controller Area Network. *Journal of Systems Architecture*, pages 607–617, April 2000.

[74] J. Von Neuman. Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components. *Automata Studies*, pages 43–98, 1956.

[75] I. E. Noble. EMC and the Automotive Industry. *IEE Electronics and Communication Engineering Journal*, pages 263–271, October 1992.

[76] R. Obermaisser and P. Peti. A Fault Hypothesis for Integrated Architectures. *Workshop on Intelligent Solutions in Embedded Systems*, pages 1–18, June 2006.

[77] M. Pandya and M. Malek. Minimum Achievable Utilization for Fault-Tolerant Processing of Periodic Tasks. *IEEE Transactions on Computers*, 47(10):1102–1112, October 1998.

[78] T. S. Perry and L. Geppert. Do portable electronics endanger flight? the evidence mounts. *Spectrum, IEEE*, 33(9):26–33, September 1996.

[79] M. Pizza, L. Strigini, A. Bondavalli, and F. Di Giandomenico. Optimal Discrimination between Transient and Permanent Faults. *IEEE International Symposium on High-Assurance Systems Engineering*, pages 214–223, November 1998.

[80] D. Powell. Failure Mode Assumptions and Assumption Coverage. *International Symposium on Fault-Tolerant Computing*, pages 386–395, July 1992.

[81] D. Powell, J. Arlat, L. Beus-Dukic, A. Bondavalli, P. Coppola, A. Fantechi, E. Jenn, C. Rabejac, and A. Wellings. GUARDS: A generic upgradable architecture for real-time dependable systems. *IEEE Transactions on Parallel and Distributed Systems*, 10(6):580–599, June 1999.

[82] S. Punnekkat, H. Hansson, and C. Norström. Response Time Analysis under Errors for CAN. *IEEE Real-Time Technology and Applications Symposium*, May-June 2000.

[83] S. Ramos-Thuel and J. K. Strosnider. The Transient Server Approach to Scheduling Time-Critical Recovery Operations. *IEEE Real-Time Systems Symposium*, pages 286–295, December 1991.

[84] K. Ravindran, K. A. Kwiat, and A. Sabbir. Adapting Distributed Voting Algorithms for Secure Real-Time Embedded Systems. *International Conference on Distributed Computing Systems*, pages 347–353, March 2004.

[85] K. Ravindran, K. A. Kwiat, A. Sabbir, and B. Cao. Replica Voting: a Distributed Middleware Service for Real-time Dependable Systems. *International Conference on Communication System Software and Middleware*, January 2006.

[86] J. Rufino, P. Verissimo, G. Arroz, C. Almeida, and L. Rodrigues. Fault-Tolerant Broadcasts in CAN. *Annual International Symposium on Fault-Tolerant Computing, Digest of Papers*, pages 150–159, June 1998.

[87] J. Rushby. Formal Methods and the Certification of Critical Systems. Technical Report Technical Report CSL-93-7, Computer Science Laboratory, SRI International, December 1993.

[88] L. Sha, T. F. Abdelzaher, K.-E. Årzén, A. Cervin, T. P. Baker, A. Burns, G. C. Buttazzo, M. Caccamo, J. P. Lehoczky, and A. K. Mok. Real Time Scheduling Theory: A Historical Perspective. *Real-Time Systems*, 28(2-3):101–155, November/December 2004.

[89] K. G. Shin and J. W. Dolter. Alternative Majority-Voting Methods for Real-Time Computing Systems. *IEEE Transactions on Reliability*, 38(1):58–64, April 1989.

[90] J. Sosnowski. Transient Fault Tolerance in Digital Systems. *IEEE Micro*, 14(1):24–35, February 1994.

[91] J. A. Stankovic. Misconceptions about Real-Time Computing. *Computer*, pages 10–19, October 1988.

[92] J. A. Stankovic and K. Ramamritham, editors. *Hard Real-Time Systems Tutorial*. IEEE-Computer Society Press, 1988.

[93] B. Strauss, M. G. Morgan, J. Apt, and D. D. Stancil. Unsafe at Any Airspeed? *IEEESpectrum*, 43(3):44–49, March 2006.

[94] K. Tindell and A. Burns. Guaranteed Message Latencies for Distributed Safety-Critical Hard Real-Time Control Networks. Technical Report YCS 229, Dept. of Computer Science, University of York, May 1994.

[95] K. Tindell, A. Burns, and A. Wellings. Analysis of Hard Real-Time Communications. *Real-Time Systems*, 9(2):147–171, September 1995.

[96] K. Tindell, A. Burns, and A. Wellings. Calculating Controller Area Network (CAN) Message Response Times. *Control Engineering Practice*, 3:1163–1169, August 1995.

[97] K. W. Tindell, A. H. Hansson, and A. J. Wellings. Analysing Real-Time Communications: Controller Area Network (CAN). *IEEE Real-Time Systems Symposium*, pages 259–265, December 1994.

[98] M. Wallace. Modular Architectural Representation and Analysis of Fault Propagation and Transformation. *Formal Foundations of Embedded Systems and Component-based Software Architectures*, pages 53–71, April 2005.

[99] C. J. Walter and N. Suri. The Customizable Fault/Error Model for Dependable Distributed Systems. *Theoretical Computer Science*, 290(2):1223–1251, January 2003.

[100] F. Wang, K. Ramamritham, and J. A. Stankovic. Determining Redundancy Levels for Fault Tolerant Real-Time Systems. *IEEE Transactions on Computers - Special Issue on Fault-Tolerant Computing Archive*, 44(2):292–301, February 1995.

[101] H. A. Watson, editor. *Launch Control Safety Study.* Bell Labs, Murray Hill, NJ, USA, 1961.

[102] J. H. Wensley, L. Lamport, J. Goldberg, M. W. Green, K. N. Levitt, P. M. Milliar-Smith, R. E. Shostak, and C. B. Weinstock. SIFT: Design and Analysis of a Fault-Tolerant Computer for Aircraft Control. *Proceedings of the IEEE*, 66(10):1240–1255, October 1978.

[103] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and Per Stenström. The Worst-Case Execution Time Problem Overview of Methods and Survey of Tools. *ACM Transactions on Embedded Computing Systems*, 7(3), April 2008.