Probabilistic Timing Analysis of Distributed Real-time Automotive Systems

Haibo Zeng

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2008-157
http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-157.html

December 13, 2008

Probabilistic Timing Analysis of Distributed Real-time Automotive Systems

by

Haibo Zeng

B.E. (Tsinghua University, Beijing, China) 1999
M.E. (Tsinghua University, Beijing, China) 2002
M.S. (University of California, Berkeley) 2008

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Engineering-Electrical Engineering and Computer Sciences

in the

GRADUATE DIVISION
of the
UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:
Professor Alberto L. Sangiovanni-Vincentelli, Chair
Professor Sanjit A. Seshia
Professor Philip M. Kaminsky

Fall 2008

The dissertation of Haibo Zeng is approved:

Chair Date

Date

Date

University of California, Berkeley

Fall 2008

Probabilistic Timing Analysis of Distributed Real-time Automotive Systems

Haibo Zeng

Abstract

Probabilistic Timing Analysis of Distributed Real-time Automotive Systems

by

Haibo Zeng

Doctor of Philosophy in Engineering-Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Alberto L. Sangiovanni-Vincentelli, Chair

Distributed architectures supporting the execution of real-time applications are common in automotive systems. Many applications, including most of those developed for active safety and chassis systems, do not impose hard real-time deadlines. Nevertheless, they are sensitive to the latencies of the end-to-end computations from sensors to actuators. We believe a characterization of the timing metrics that, not only provides the worst case bound, but assigns a probability to each possible latency value, is very desirable to estimate the quality of an architecture configuration. In this dissertation, we present stochastic analysis frameworks that calculate the probability distributions of response times for software tasks and messages, and end-to-end latencies in a Controller Area Network based system for the performance evaluation of automotive distributed architectures. Also, the regression technique is used to quickly characterize the message response time probability distribution, which is suitable when only part of the message set is known as in the early design stage. The applicability of the analysis frameworks is validated by either simulation, or trace data extracted

from experimental vehicles.

Professor Alberto L. Sangiovanni-Vincentelli
Dissertation Committee Chair

To my dear parents and wife

Contents

List of Figures

List of Tables

Acknowledgments

This research was supported by the Marco/Gigascale Systems Research Center (GSRC), Center for Hybrid and Embedded Software Systems (CHESS), and General Motors (GM). The inspiration of this research comes mainly from the discussion with Marco Di Natale from Computer Science and Computer Engineering Department at Scuola Superiore S. Anna, Pisa, and Paolo Giusto from GM Advanced Technology Silicon Valley office in Palo Alto, California. Their tremendous guidance, feedback, and support are critical for the long time collaboration, and are gratefully remembered by the author. More specifically, the work on stochastic analysis framework as in Chapters 2, 3 and 4 are summarized in [52], and the statistical analysis framework as in Chapter 5 is summarized in [51]. The author would like to thank many researchers and engineers from GM Warren technical center, more specifically, Katrina Schultz and Bob Kirchoff, for providing the test case, as well as a description of the system, the findings and the interactions with the node supplier; Ken Orlando and Tom Forest, for the valuable discussions, suggestions and help to understand the system and the tools; the management at GM, especially Thomas Fuhrman and Joe D'Ambrosio for supporting this effort; Sri Kanajan and Arkadeb Ghosal, for helping engineer this research work. The author would also like to appreciate Zile Wei, Wei Zheng and Qi Zhu from Department of Electrical Engineering and Computer Sciences at University of California, Berkeley respectively for their initial coding of the simulator for software task and message systems, discussion and pioneer work that help me understand the design of hard real-time automotive systems; Eelco Scholte from United Technologies research center at East Hartford for his guidance and discussion on related projects. Professor Edward Lee reviewed my master's report on stochastic analysis of controller area network message response times, which is part of this dissertation. Professors Robert Brayton, Philip

Kaminsky, Sanjit Seshia and Alberto Sangiovanni-Vincentelli, my advisor, served on the qualifying exam committee and provided invaluable feedback and guidance in the dissertation writing process.

Finally, I would like to thank my family and friends, especially my lovely wife, for their contributions to make my life in graduate school interesting, and sometimes challenging. I will always look back at it with sweet memories.

Chapter 1

Introduction

The complexity of automotive electronic systems is rapidly growing. A typical modern vehicle contains between 20 and about 100 built-in electronic control units (ECUs, the term in use for CPUs in the automotive industry), with several million lines of embedded software code [11]. These nodes in vehicles are networked over standard automotive communication buses, for example, up to 10 Controller Area Network (CAN) buses that represent the large majority of the communication links, and two or three lower-speed Local Interconnect Network (LIN) buses used for a relatively small number of local low speed data communications, and, optionally, some dedicated high-speed links such as Media Oriented System Transport (MOST) for information and entertainment [32]. The FlexRay standard is recently emerging for high speed communication, e.g., X-by-wire applications that need predictability and fault tolerance [32].

Furthermore, as many of the automotive electronic systems involve real-time control of mechanical parts, such as chassis control, powertrain, and active safety control, they are characterized by non-functional requirements, including timing, safety, cost, together with reusability,

flexibility, scalability and extensibility of the architecture.

As automotive systems increase in complexity while their time to market decreases, designers often face the challenge of analyzing requirements as early as possible in the design process to reduce the risk of late and expensive design changes. In particular, car electronic architectures need to be defined and selected years in advance, when the functions they will support are only partly defined. This stage, *architecture evaluation and selection*, is of enormous importance for its implications on cost, performance and extensibility. Typically models of the functions and of the possible physical architecture need to be defined and matched to evaluate the quality and select the best possible hardware platform with respect to cost, performance, and reliability. Currently, the vital stage of architecture selection and evaluation is driven by a *what-if* iterative process [43]. First a set of metrics and constraints is defined by the developers; then a few initial candidate architectures and their configurations are evaluated based on the results of quantitative analysis, and a solution is selected as the best choice; or if none of the initial candidates are satisfying, a new set of architectures is produced and the iterative process continues until a solution is finally obtained.

Automotive Architectures Priority-based scheduling is very popular in control applications due to resource efficiency and ultimately price concerns. Many automotive systems are designed based on run-time priority-based scheduling of tasks and messages. For example, the OSEK compliant operating system [7] and the CAN bus protocol [2] are automotive standards supporting this model. Priority-based scheduling fits well within the traditional design cycle, where worst case timing analysis is used to check correctness against hard real-time constraints. Moreover, timing predictability and thus composability of components are hard to achieve, as for example small changes in the timing parameters may result in a significant degradation of the response times of tasks and messages

[43].

Time-triggered design methodologies, like the Time-Triggered Architecture (TTA) and its network protocol TTP [22] have been proposed for distributed systems. OSEKTime, a time-triggered operating system [3], along with the recent FlexRay standard [8] for high speed communication in cars, is based on these concepts. Time-triggered scheduling forces context switches on the ECUs and the assignment of the communication bus at predefined points in time; thus provides much better time determinism than priority based scheduling does [43].

In this work, we focus on the analysis of probabilistic timing performance of distributed automotive architecture with priority based scheduling standards, i.e., OSEK and CAN. We define stochastic and statistical analysis methods that provide a characterization of the average case timing behavior of the application as shown in Figure 1.1.

The communication model considered in this research is the *periodic activation* model, where all tasks are activated periodically, with a minimum inter-arrival time, and communicate by means of asynchronous buffers preserving the latest value written on them. This model is supported by AUTOSAR, the open and standardized automotive software architecture [1]. Communication may happen at the interface between two abstraction layers (for example, the application software task and the middleware), and also at the interface between any two resource domains, such as from a task to a message. Similarly, message transmission is triggered periodically and each message contains the latest values of the signals that are mapped into it. The periodic activation model suffers from possible large latencies of end-to-end computations, as the consumer of the data may need to wait up to an entire period (*sampling delay*) to get the latest data stored in the communication buffer.

Research Motivation Distributed functions include time-critical controls, but most often, also functions that are characterized by requirements for average performance together with hard deadline constraints (as for most active-safety functions) and functions with soft real-time requirements (controls for enhanced driver comfort).

The definition of a new architecture framework for one or more car product families is an extremely important step: ECUs, networks and the topology of connections must be defined and frozen years in advance of production. Later, during the architecture lifespan, functions are placed on ECUs and communication scheduled on the bus.

The work presented in this dissertation is particularly motivated by the study of current and future *active safety functions*. These functions, e.g., Adaptive Cruise Control, typically gather a 360° view of the environment via several radars and cameras, and require several processing stages before the actuation signals are produced, including sensory fusion, object detection, controls and arbitration layers. Typical car architectures supporting these functions are heterogeneous on the processing side, yet mostly homogenous on the communication links as CAN protocols are mainly used to control bus communications among processing units (especially control-related) [13]. End-to-end computations that span over several ECUs and CAN buses are difficult to characterize by verification experts or feature developers in terms of hard deadline requirements, although a predictable worst case latency, and the knowledge of latency distributions is a very valuable design asset. The analysis of the end-to-end latencies of functions for a given architecture hypothesis is a very valuable design method. These latencies in the transmission of information from sensors to actuators are a function of task response times, sampling period delays, message response times, and task/message activation relative phases.

In the early evaluation and selection of distributed embedded architectures for next-generation automotive controls, the application performance depends on the end-to-end latencies of active-safety functions. Automobile architecture must be evaluated and selected having in mind that they will have a lifespan of 5 to 10 years and that during this lifespan the communication and computation load is partly unknown because new functions are still being decided on and have not been designed as yet. Hence, when verifying that the architecture is sufficiently robust with respect to constraints on latency and performance targets of present and future functionalities, loads can only be roughly estimated by looking at past trends or by exploiting early indications of designers.

In practice, hard deadline requirements are often defined after the design stage, when a correct implementation must be guaranteed by ensuring that no critical messages are lost or overwritten. For that purpose, worst case analysis on the shared resources of the system (CPUs, buses), based on schedulability theory, is a popular analytical method for computing the contribution of tasks and messages to the end-to-end latencies [13]. Worst-case latency analysis provides the architecture designer with a set of values (one for each end-to-end path) on which he/she can base his/her evaluation in a what-if analysis and synthesis flow. However, worst case evaluation may not be sufficient and needs to be complemented by probabilistic analysis for two main reasons:

- Many applications are not time-critical, but are nevertheless sensitive to delays. For other functions, satisfaction of the deadline constraints is required, but the performance and the quality of the controls depends also on the average response time, which needs to be controlled and minimized.

- In the periodic activation model, each time a message is transmitted or received, a task (message) may need to wait up to an entire period of *sampling delay* to read (forward) the latest

data stored in the middleware buffers. Adding worst case delays at each step allows to obtain the worst-case latency in end-to-end paths, but the probability of a worst-case event can be very small. So small in fact (in the architectures we considered it can be easily lower than 10^{-12}), to be smaller than the probability of failure of the HW components! In this case, designing for worst-case guaranteed performance can be quite wasteful.

To better understand the motivation of our approach, let us consider an example in which the worst case timing metrics of two different alternative implementations are equal. The top curve in Figure 1.1 represents an implementation with a better average timing behavior than that of the other curve. In fact, in the top curve, a given time value (r_0 in the figure) is rarely exceeded (only in 10% of the cases), as opposed to the second implementation (the bottom curve), where it is always exceeded.

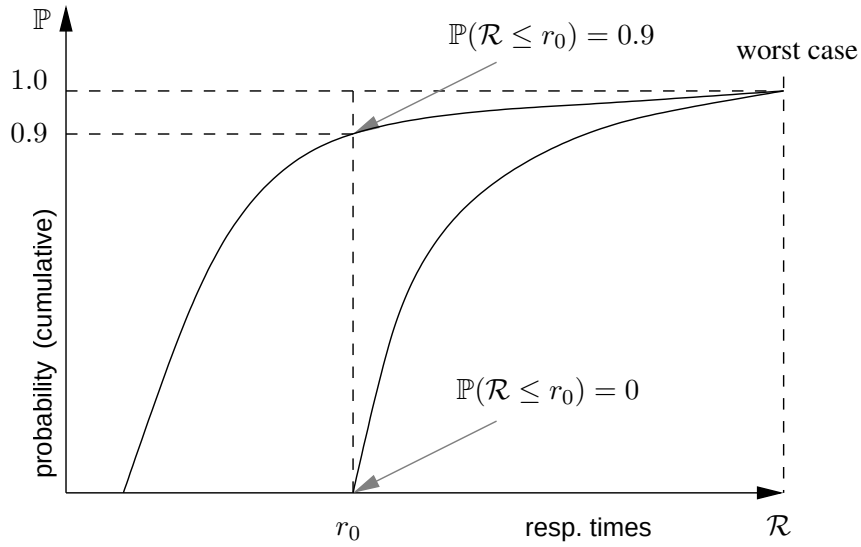

Figure 1.1: Characterization of timing metric cumulative probability distributions

The next question is then: how can such curves of probability functions for timing metrics

be derived? Several methods exist including trace analysis, statistical sampling and simulation. These methods, in general, are applicable but with limited coverage as the population size is huge for real applications. Thus more efficient methods are needed to derive probability distribution curves.

State of the Art The problem of probabilistic analysis of priority-scheduled systems has been addressed in the past. Gardner et al. [16] compute the probability of deadline misses for each job released in the busy interval, and choose the maximum of these probabilities as an upper bound on the probability of deadline misses for the corresponding task. Simulation-based analysis and stochastic methods exist [14, 21] for computing the probability density functions (*pdf*s or probability mass functions *pmf*s in the discrete case) of the response times of tasks scheduled on single or multi-processor platforms. Also, results exist for messages scheduled on a CAN bus. For example, in [31], Navet et al. introduce the concept of worst case deadline failure probability (WCDFP), the probability that too many errors occur such that a message can not meet its deadline. In their work, an error model including both error frequency and gravity is introduced - the occurrence of errors is assumed to be a generalized Poisson process. This method computes the error threshold under which the message deadlines are still met on a CAN network; however, probabilities WCDFP are computed with respect to the *critical instant*, that is, the worst case response time scenario. In [33] Nolte et al. extend the worst case response time analysis using random message transmission times that take into account the probability distribution of a given number of stuff bits due to the mechanism in CAN protocol that an additional trailing complementary bit is added whenever the message frame contains a sequence of five consecutive identical bits, but the response time analysis is still performed according to the worst case analysis ([46]).

Worst case analysis of task and message response times in priority-based scheduled systems has produced many results, including work on independent periodic tasks [25], and periodic tasks with offsets and jitter [35], and OSEK compliant software task systems [26]. In [13] Davis et al. discuss scheduling and response time analysis of CAN messages, where a flaw in the traditional analysis documented in [46] is reported and a new solution is provided. In [47] Tindell extensively discusses the limitations in the applicability of worst case analysis to real CAN systems.

Lehoczky defines a unique approach to the stochastic analysis of task response times, Real-Time Queueing Theory [23]. In this work, the task set is modeled as a Markovian process. Given a scheduling algorithm and the distribution of the deadlines, the distributions of the lead times (time remaining until the deadline) of jobs are computed. The space of the lead times is analyzed in heavy traffic conditions, when the behavior can be approximated by a diffusion process. This approach assumes that arrival and execution times are modeled by Poisson distributions, and the system has a very large set of tasks with very high utilization ($U \rightarrow 1$). In an independent but related work, Kim and Shin [20] model an application as a queuing network scheduled as first-in first-out (FIFO), and use exponentially distributed task execution times. The underlying mathematical model is a continuous-time Markov chain.

In the context of multi-processor systems, Manolache presents an analytical and exact method for obtaining the expected deadline-miss ratio that can be efficiently applied to single processor systems, and an approximate method that can be applied to multi-processor systems [27]. Gardner takes a different approach [15] where the results are based on statistical analysis.

The most relevant method to our approach was proposed by Díaz et al. [14, 21], which computes the *pmf*s of the response times of a set of independent periodic tasks dispatched on a single

processor by a preemptive priority-based scheduler. The activation times of all the task instances are known and the task execution times are defined by known *pmf*s. In [14] Díaz et al. introduce the concept of P-level backlog at time t to represent the sum of the *remaining* requested CPU times of all the task instances (jobs) of priority higher than or equal to P that have been released before t. The P-level backlog at the beginning of each hyperperiod is a random variable, and the stochastic process composed of the sequence of these backlogs is proven to be a Markov chain. When the maximum utilization is less than 1, a stationary distribution (*pmf*) of the backlog possible values is reached at the beginning of the second hyperperiod. Then, a stationary backlog at any time *within* the hyperperiod can be calculated using two operations called *convolution* and *shrinking*. The P-level backlog *pmf* right after the release of a job with priority higher than or equal to P is obtained by performing a convolution, at the job release time, of the backlog *pmf* with the job execution time *pmf*. Given the *pmf* of a backlog at time t, shrinking allows to compute the backlog at time $t' > t$ by shifting the *pmf* to the left by $t' - t$ units and by defining the probability of zero backlog be the sum of the probabilities defined for non-positive values. The time t' represents a time instant *right before the next release time of a task instance that contributes to the P-level backlog*. Indeed, shrinking is equivalent to simulating the progression of time. Finally, the stationary response time *pmf* of a job can be computed with a sequence of splitting, convolution, and merging operations starting with the backlog at the release time and the *pmf* of the job execution time itself, and considering the *pmf*s of the execution times of higher priority jobs that are released before the job terminates, as the current job can be preempted. This process continues until either the time instant corresponding to the job deadline is reached, in which a deadline violation is found, or until the job completes its execution before any other higher priority job is released. As there may be multiple jobs of the same task

released within the hyperperiod, the stationary task response time *pmf* is obtained by averaging the response time *pmf*s of all the jobs within the hyperperiod. For more details, we refer the interested readers to [14, 21] and Chapter 2.

Issues Related to Simulation-based Statistical Analysis *Fitting* is a common method for deriving known probability distribution functions from statistical data. The accuracy of the method relies on a sufficiently large simulation coverage. In an automotive distributed system of realistic size with unsynchronized ECUs, there are too many possible relative phases between tasks and messages. Thus a simulation with statistically significant coverage is not feasible. For example, assuming a system with periodic messages, with the *hyperperiod* H of the system defined as the least common multiple of the periods of all messages, i.e., $H = lcm(\bigcup_i T_i)$, if the number of ECUs (nodes) is n, and the time is discretized with granularity τ, then the total number of possible relative phase combinations is $(H/\tau)^{n-1}$ (one ECU is assumed as a reference with phase equal to 0). As an example, for a system with 10 ECUs and hyperperiod $100ms$, if the granularity is $10\mu s$, then there are 10^{36} possible phase configurations to simulate.

Summary of the Dissertation Our objective is to provide the theory for probabilistic analysis of the latency in the end-to-end propagation of information among periodically activated tasks and messages. A communication mechanism based on the preservation of the latest written value and the overwriting of old ones (shared variable buffer) is assumed. We target our analysis in the context of current automotive domain standards. In particular, priority based scheduling as in OSEK standard is assumed for the ECUs, all messages are exchanged on a CAN bus, where they are transmitted in order of their IDs (priorities), and finally all the messages transmitted by the same ECU are assumed

to be enqueued by the same middleware-level task.

Our experimental results show that our technique provides a good approximation of the latency distribution. We provide experimental results that simulate the system behaviors and from trace data of the real vehicles, and we compare them with the results obtained by our stochastic and statistical analysis techniques.

We assume the tasks are periodically activated with known phases on a single processor system, and their execution times are stochastic and defined by a known probability mass function. Because OSEK operating system standard is mixed preemptive, the scheduling policy is non-preemptive if the currently running job is non-preemptable. The concept of P-level backlog is first extended to consider the possible blocking time when the current executing job has a priority lower than P and is non-preemptable. Then the backlog update procedure considering the blocking time from lower priority non-preemptable jobs at each time tick is given. This work is described in Chapter 2.

In a distributed system with unsynchronized ECUs, the arrival and queuing times of messages are non-deterministic. From the standpoint of a given message on a given node, assumed as the *observer*, messages are queued on other nodes with random phases. Hence, we provide a characterization of the bus load at priority level P or higher, by introducing a single *characterization message* of level P for each remote node, with random transmission time and random queuing jitter. For each message in the system, we compute the probability distribution of its backlog using the characterization messages, then we compute the *pmf* of its response time. Furthermore, as the CAN protocol is non-preemptive, a message may be blocked by a lower priority message. Within our framework, we model and estimate the probability of blocking in the context of non-deterministic

message queuing times. This work is presented in Chapter 3.

In Chapter 4, we first describe the architecture of the automotive distributed systems; then, we define a formal model for it and we formalize the concept of end-to-end latency, describing the components contributing to the delay. The stochastic analysis of the computation and communication stages that compose the end-to-end latency are addressed.

Also, in Chapter 5, we describe the main statistics that can be extracted from simulation data for a given bus, where message response time distribution can be approximated with good accuracy by a possible fit of exponential distributions shifted with fitting offsets. We validate our hypothesis that the parameters of the distribution of a message response time can be computed from generic system-level design information, such that the method can be used as a predictor of the message response times for a given bus configuration.

Finally, we provide conclusions, and look forward to future work to perform statistical analysis of software task response times and later end-to-end latencies, and to use average end-to-end timing performance as the metric in the automatic mapping and configuration process, as in Chapter 6.

Chapter 2

Stochastic Analysis of OSEK Software Task Response Times

In this chapter, we present the work for stochastic analysis of software task response times in OSEK compliant real-time systems. First, we introduce the task management concepts in the automotive industry OSEK operating system standard, and the appropriate model for OSEK compliant software task systems. Then previous work on stochastic analysis frameworks for periodic preemptable tasks is described in detail, and is extended to systems with mixed preemptive scheduling policy. Finally, we provide some experimental results on a real automotive application.

2.1 System Model and Notation

2.1.1 OSEK Compliant Software Task Systems

OSEK/VDX, or simply OSEK (Offene Systeme und deren Schnittstellen für die Elektronik in Kraftfahrzeugen; English: "Open Systems and their Interfaces for the Electronics in Motor Vehicles"), is a standards body which sets up specifications for open-ended architecture in automotive industry [4]. Many of the major companies in automotive industry participate in the consortium which founded OSEK, and some parts of OSEK are standardized by ISO. The standard specifications include real-time operating systems [7], software interfaces and functions for communication [5], network management [6], etc.

An OSEK compliant operating system is specified as a single processor operating system meant for distributed embedded control units [7], which serves as a basis for real-time execution of application software. The application software tasks are assumed to be independent from each other. Two different types of tasks are provided by OSEK compliant operating systems: basic tasks and extended tasks. A basic task can only finish or be preempted by a higher priority task or interrupt service routine, while extended tasks can use events for synchronization.

There are four different OSEK compliant operating system features defined as four conformance classes: *BCC1, BCC2, ECC1 and ECC2*, which are determined by the following attributes: multiple requesting of task activation, task types, and number of tasks per priority. In conformance class BCC2, only basic tasks are supported, while there may be more than one task per priority, and multiple requests of activation from the same task are allowed. We assume BCC2 is used since it can meet the requirements of most real-time control systems in vehicles.

To enhance efficiency, an OSEK compliant operating system only supports static priority

management. Accordingly the user cannot change the task priority at run time. For flexibility and performance considerations, an OSEK compliant operating system regards preemptiveness as a task attribute: preemptable and non-preemptable tasks can be mixed in the same system. Thus the scheduling policy is mixed preemptive, which depends on the preemption property of the running task. If the running task is preemptable, then a preemptive scheduling algorithm is performed; otherwise, a non-preemptive scheduling algorithm is used such that task switching is performed after the current task terminates. This is quite different from the assumption of many time analysis methods, such as the stochastic analysis framework presented in [14, 21].

An OSEK compliant operating system also leaves the flexibility to combine aspects of preemptive and non-preemptive scheduling by defining groups of tasks. For tasks which have the same or lower priority as the highest priority within a group, the tasks within the group behave like non-preemptable tasks. This allows for a possible further improvement of response times of some tasks and stack space reuse [50]. To avoid the problems of priority inversion and deadlocks, an OSEK compliant operating system standardizes an implementation of the immediate priority ceiling protocol [44] for sharing resources with predictable worst case blocking time. These standards allow prediction of the worst case timing behavior of computations and communications under the assumptions that there is no fault and the worst case task execution times can be safely estimated [17, 46].

2.1.2 Model of OSEK Software Tasks

In this work, we consider the *periodic* activation model, as it is currently deployed in industry (such as General Motors E/E architectures) and supported by AUTOSAR (AUTomotive Open System ARchitecture), the open and standardized automotive software architecture [1].

Now we consider the model for a OSEK compliant software system that contains a set of independent periodic tasks. A task τ_i is modeled by $(T_i, O_i, \mathcal{E}_i, P_i, N_i)$, where T_i is its period, O_i its initial phase, $\mathcal{E}_i$ its execution time, P_i its priority, and N_i its preemption property. The task execution time $\mathcal{E}_i$ is a discrete random variable [1] with a known probability distribution $f_{\mathcal{E}_i}$. The *hyperperiod* H of the system is defined as the least common multiple (*lcm*) of task periods, i.e., $H = lcm(\bigcup_i T_i)$.

For each periodic activation, we consider a task instance, defined as a *job* of this task. We denote the j-th job of task τ_i as $\Gamma_{i,j}$, its *arrival time* as $A_{i,j} = O_i + (j-1) \times T_i$, which is the time $\Gamma_{i,j}$ is logically ready for execution. Because of the periodic activation, on its arrival, $\Gamma_{i,j}$ is *activated* or *released* immediately, thus its activation time (or release time, the time a job is dispatched and actually ready for execution) is $Q_{i,j} = A_{i,j}$. Also, the start time of $\Gamma_{i,j}$ is denoted as $\mathcal{S}_{i,j}$, and its finish time as $\mathcal{F}_{i,j}$. For all the jobs of task τ_i, their execution times are independent and identically distributed as $f_{\mathcal{E}_i}$, and are independent from the execution times of other tasks.

Each task τ_i is associated with a priority P_i, and a preemption property N_i. We assume that $P_i < P_j$ implies task τ_i has a higher priority than τ_j. The preemption property N_i is either preemptable, or non-preemptable. The scheduling policy is *mixed preemptive scheduling*, which depends on the preemption property of the running task. If the running task is non-preemptable, then non-preemptive scheduling is performed. If the running task is preemptable, then preemptive scheduling is performed.

The *response time* $\mathcal{R}_{i,j}$ of a job $\Gamma_{i,j}$ is defined as the time interval from its arrival to its finish, i.e., $\mathcal{R}_{i,j} = \mathcal{F}_{i,j} - A_{i,j}$, which is also a discrete random variable. The *pmf* $f_{\mathcal{R}_i}$ of the

[1] In the dissertation we use calligraphic letters to denote random variables. All the notations are listed alphabetically in Appendix A.

response time of task τ_i is obtained as the average of the response time *pmf*s of all the jobs $\Gamma_{i,j}$ in the hyperperiod provided that the distribution is stationary.

2.2 Previous Work on Stochastic Analysis of Periodic Preemptable Tasks

In [14, 21] Díaz et al. present a stochastic analysis framework for real-time systems. It can compute the task response times for fixed priority systems where tasks are periodic and preemptable, i.e., for each task τ_i, $N_i = Preemptable$, thus the scheduling policy is always preemptive. Please note that the work in [14, 21] also addresses dynamic priority systems, but we will focus on systems with fixed priority tasks as in OSEK compliant real-time systems.

2.2.1 Analysis Framework Overview

To calculate task response times, in [14] Díaz et al. introduce the concept of *P-level backlog at time t* (denoted as $\mathcal{W}_t^P$) as the sum of the remaining execution times of all the jobs that have priorities higher than or equal to P and are not completed till time t. The task release pattern in one hyperperiod is repeated for all the other hyperperiods, thus Díaz et al. focus on the P-level backlog observed at the beginning of each hyperperiod, denoted as $\mathcal{G}_k^P = \mathcal{W}_{((k-1)H)^-}^P$. Here t^- denotes the time instant arbitrarily smaller than t, i.e., *right before the release of a job at time t*. Díaz et al. prove that the stochastic process defined as the sequence of random variables $\{\mathcal{G}_1^P, \mathcal{G}_2^P, ..., \mathcal{G}_k^P, ...\}$ is a Markov chain. In addition, a stationary distribution $\mathcal{G}^P$ of the P-level backlog $\mathcal{G}_k^P$ exists as long as the stability condition that the average system utilization is less than one is met.

After the stationary distribution $\mathcal{G}^P$ is calculated, the stationary backlog at any time *within* the hyperperiod can be calculated by iteratively using two operations called *convolution* and *shrinking*. The P-level backlog *pmf* right after the release of a job with priority higher than or equal to P is obtained by performing a convolution, at the job release time, of the backlog *pmf* with the job execution time *pmf*. Given the *pmf* of the backlog at time t, shrinking allows to compute the backlog at time $t' > t$ by shifting the *pmf* to the left by $t' - t$ units and by defining the probability of zero backlog be the sum of the probabilities defined for non-positive values. Here time t' represents a time instant *right before the next release time of a task instance that contributes to the P-level backlog*. Indeed, shrinking is equivalent to simulating the progression of time.

Finally, the stationary response time *pmf* of a job can be computed with a sequence of splitting, convolution, and merging operations starting with the backlog at the release time and the execution time *pmf* of the job itself, and considering the *pmf*s of the execution times of higher priority jobs that are released before the job terminates, as the current job can be preempted. This process continues until either the time instant corresponding to the job deadline is reached, in which a deadline violation is found, or until the job completes its execution before any other higher priority job is released. As there may be multiple jobs of the same task released within the hyperperiod, the stationary task response time *pmf* is obtained by averaging the response time *pmf*s of all the jobs within the hyperperiod, i.e.,

$$\forall r \geq 0, f_{\mathcal{R}_i}(r) = \frac{1}{n_i} \sum_{j=1}^{n_i} f_{\mathcal{R}_{i,j}}(r) \tag{2.1}$$

where $n_i = H/T_i$.

The procedure **Calc_Stat_Task_Resp** to calculate the stationary task response time $\mathcal{R}_i$ of τ_i is summarized in Algorithm 1. First, in procedure **Calc_Stat_Backlog** the stationary distribution

Algorithm 1 Calc_Stat_Task_Resp(τ_i)

1: $\mathcal{G}^{P_i}$ = **Calc_Stat_Backlog**(P_i)

2: **for** each job $\Gamma_{i,j}$ in one hyperperiod H **do**

3: $\mathcal{W}^{P_i}_{Q_{i,j}}$ = **Update_Backlog**($0^-, \mathcal{G}^{P_i}, Q_{i,j}$)

4: $\mathcal{R}_{i,j}$ = **Calc_Job_Resp**($\Gamma_{i,j}$)

5: **end for**

6: $f_{\mathcal{R}_i} = \frac{T_i}{H} \sum_j f_{\mathcal{R}_{i,j}}$ *// Task response time is the average of job response times in one hyperperiod*

of P_i-level backlog at the beginning of the hyperperiod $\mathcal{G}^{P_i}$ is calculated; this is further described in Section 2.2.2 and Algorithm 2. Then for each job $\Gamma_{i,j}$ in one hyperperiod, the backlog $\mathcal{W}^{P_i}_{Q_{i,j}}$ at its job release time $Q_{i,j}$ is calculated assuming the backlog at the beginning of the hyperperiod (time 0) is $\mathcal{G}^{P_i}$; this is done by using the procedure **Update_Backlog** as explained in Section 2.2.3 and Algorithm 3. Finally the response time $\mathcal{R}_{i,j}$ of job $\Gamma_{i,j}$ is calculated in procedure **Calc_Job_Resp** (Section 2.2.4), and the task response time $\mathcal{R}_i$ is obtained by averaging the job response times in one hyperperiod.

2.2.2 Stationary Backlog at the Beginning of the Hyperperiod

To calculate the stationary backlog $\mathcal{G}^{P_i}$, i.e., the stationary P_i-level backlog observed at the beginning of the hyperperiod, one way is to obtain the transition probability matrix $\mathbf{P} = (p_{m,n})$ that gives the transition probability $p_{m,n} = \mathbb{P}(\mathcal{G}^{P_i}_{k+1} = n | \mathcal{G}^{P_i}_k = m)$ for any two possible backlog values m and n, then compute the exact stationary $\mathcal{G}^{P_i}$ by solving the equation $\mathcal{G}^{P_i} = \mathcal{G}^{P_i}\mathbf{P}$ (regarding $\mathcal{G}^{P_i}$ as a row vector). This is difficult since the dimension of the transition matrix $\mathbf{P}$ can be infinity if the maximum system utilization is larger than 1. To solve it, Díaz et al. explore the

regular structure in matrix $\mathbf{P}$, which allows us to provide a general method to obtain an analytical expression for the stationary distribution. However, this method is hard to extend to other systems where tasks may be non-preemptable, or periodic with jitter. For example, if some of the tasks are non-preemptable, then the backlog $\mathcal{G}_{k+1}^{P_i}$ not only depends on $\mathcal{G}_k^{P_i}$ and the higher priority jobs released in the k-th hyperperiod, but also on the lower priority non-preemptable jobs that are released but not start execution, since these lower priority jobs may start execution at any time and block jobs of τ_i and introduce additional delay to these jobs. This will be further explained in Section 2.3.

Algorithm 2 Calc_Stat_Backlog(τ_i)

1: Initialize $\mathcal{G}_1^{P_i}$ as $\mathbb{P}(\mathcal{G}_1^{P_i} = 0) = 1$

2: Initialize ϵ as desired analysis error

3: $k = 1$

4: **repeat**

5: $\quad \mathcal{G}_{k+1}^{P_i}$ = **Update_Backlog**$(((k-1)H)^-, \mathcal{G}_k^{P_i}, (kH)^-)$

6: $\quad k = k + 1$

7: **until** $|\mathcal{G}_k^{P_i} - \mathcal{G}_{k-1}^{P_i}| < \epsilon$

8: Return $\mathcal{G}_k^{P_i}$

One alternative of calculating $\mathcal{G}^{P_i}$ that does not require explicit derivation of the transition probability matrix $\mathbf{P}$ is to iteratively calculate the backlog at the end of each hyperperiod until backlogs at the end of two consecutive hyperperiods converge (within certain numerical error). This iterative approximation is described in Algorithm 2, which takes zero as the initial backlog and uses the procedure **Update_Backlog** as in Algorithm 3 to update backlog based on the one at the beginning of the previous hyperperiod. As pointed out in [14, 21], a stationary distribution of the

backlog exists as long as the stability condition that the average system utilization $\bar{U}$ is less than one is met; also, it is said that the closer the average utilization $\bar{U}$ is to 1, the slower the convergence is, but it is still unknown how many iterations are required in terms of $\bar{U}$.

2.2.3 Backlog Update

In this section, we describe the procedure **Update_Backlog**, which calculates backlog at time $t_2 > t_1$ given the backlog $\mathcal{W}_{t_1}^P$ at time t_1. This is done by iteratively using two operations *convolution* and *shrinking*. The P-level backlog *pmf* right after the release of a job with priority higher than or equal to P is obtained by performing a convolution operation, at the job release time, of the backlog *pmf* with the job execution time *pmf*. The convolution of two *pmf*s $f_{\mathcal{V}_1}$ and $f_{\mathcal{V}_2}$ is denoted as $f_{\mathcal{V}_1} \bigotimes f_{\mathcal{V}_2}$. Figure 2.1 gives an example of convolution operation. Given the *pmf* of a backlog at time t, shrinking computes the backlog at time $t'^{-} > t$ where t'^{-} is the time instant arbitrarily smaller than t', i.e., *right before the next release time of a job that contributes to the P-level backlog*. This is done by shifting the *pmf* to the left by $t' - t$ units and by defining the probability of zero backlog to be the sum of the probabilities defined for non-positive values. Please note that there is no job with priority $\leq P$ released in the interval (t, t'). Indeed, shrinking is equivalent to simulating the progression of time. An example of shrinking is given in Figure 2.2.

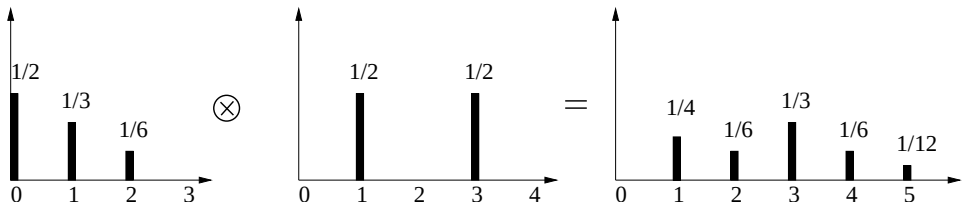

Figure 2.1: Convolution

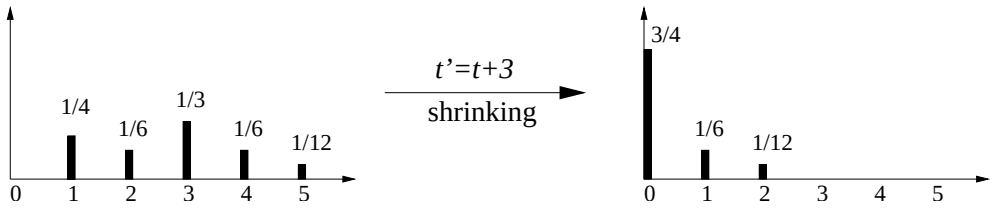

Figure 2.2: Shrinking

The procedure in Algorithm 3 is slightly different from the above, which advances time by only one time tick further. After adding (by convolution) the execution times of all the jobs released at time t that contribute to the backlog, the shrinking operation is performed from t to $(t+1)^-$, only one time tick ahead. The procedure will check the set of jobs to see whether there are higher priority jobs released at this time. Please note that the algorithm is not necessarily the most efficient in terms of implementation, but it is explained in this way for better understanding, especially when we consider the extensions to systems with non-preemptable tasks, and tasks with random release times.

Please note that all the tasks here are assumed to be preemptable, thus we only need to consider the release of jobs with priority higher than or equal to P when we update the P-level backlog, as in Algorithm 3. This is not enough in case some of the tasks are non-preemptable, which may introduce additional blocking delays to the backlog.

2.2.4 Stationary Job Response Time Calculation

As in [14], for each job $\Gamma_{i,j}$ in the hyperperiod, once the stationary backlog $\mathcal{W}_{Q_{i,j}}^{P_i}$ at its release time $Q_{i,j}$ is calculated, its stationary response time can be computed by taking into consideration possible future interferences. The set of jobs released after $Q_{i,j}$ that may preempt $\Gamma_{i,j}$

Algorithm 3 Update_Backlog_Preemptive($t_1, \mathcal{W}_{t_1}^P, t_2$) *// Given the backlog* $\mathcal{W}_{t_1}^P$ *at time* t_1*, calculate backlog at time* $t_2 > t_1$

1: **for** time instant t from t_1 to t_2 **do**
2: **for** each job $\Gamma_{i,j}$ released at time t **do**
3: **if** $P_i \leq P$ **then**
4: $f_{\mathcal{W}_t^P} = f_{\mathcal{W}_t^P} \bigotimes f_{\mathcal{E}_i}$
5: **end if**
6: **end for**
7: Shrink $f_{\mathcal{W}_t^P}$ from t to $(t+1)^-$
8: **end for**

is denoted as $hp(P_i)^{(Q_{i,j},\infty)} = \{\Gamma_{m,n} | P_m < P_i \text{ and } Q_{m,n} > Q_{i,j}\}$; moreover, the jobs in the set $hp(P_i)^{(Q_{i,j},\infty)}$ are ordered by their release times. Let Γ_k be the k-th job with relative release time $t_k = Q_k - Q_{i,j}$, and $f_{\mathcal{R}_{i,j}^{<k>}}$ denote the response time *pmf* of job $\Gamma_{i,j}$ after considering the possible interferences from $\Gamma_1, \Gamma_2, ..., \Gamma_k$. Its response time $\mathcal{R}_{i,j}$ is initialized as the sum of the P_i-level backlog right before its release time and its execution time, i.e., $f_{\mathcal{R}_{i,j}^{<0>}} = f_{\mathcal{W}_{Q_{i,j}^{P_i}}} \bigotimes f_{\mathcal{E}_i}$. Then for each job $\Gamma_k \in hp(P_i)^{(Q_{i,j},\infty)}$, the response time *pmf* $f_{\mathcal{R}_{i,j}^{<k-1>}}$ calculated at the previous step is split into two parts, the head part $f_{\mathcal{R}_{i,j}^{<k-1>}}^{[0,t_k]}$ and the tail part $f_{\mathcal{R}_{i,j}^{<k-1>}}^{(t_k,\infty)}$. An example of splitting operation is given in Figure 2.3. In the head part $\Gamma_{i,j}$ finishes execution before the release of Γ_k, while in the tail part Γ_k further interferes $\Gamma_{i,j}$ thus $\mathcal{E}_k$ needs to be added (by convolving these two *pmf*s). Then $f_{\mathcal{R}_{i,j}^{<k>}}$ is constructed by merging the head part and the updated tail part, as shown in Figure 2.4.

This iterative process stops when there is no further interference, i.e., at some step $n+1$ the tail part $f_{\mathcal{R}_{i,j}^{<n>}}^{(t_{n+1},\infty)}$ after splitting is null, which implies that job $\Gamma_{i,j}$ will have no chance to run till to the release time t_{n+1} of job Γ_{n+1}, or at the time instant where the deadline is reached.

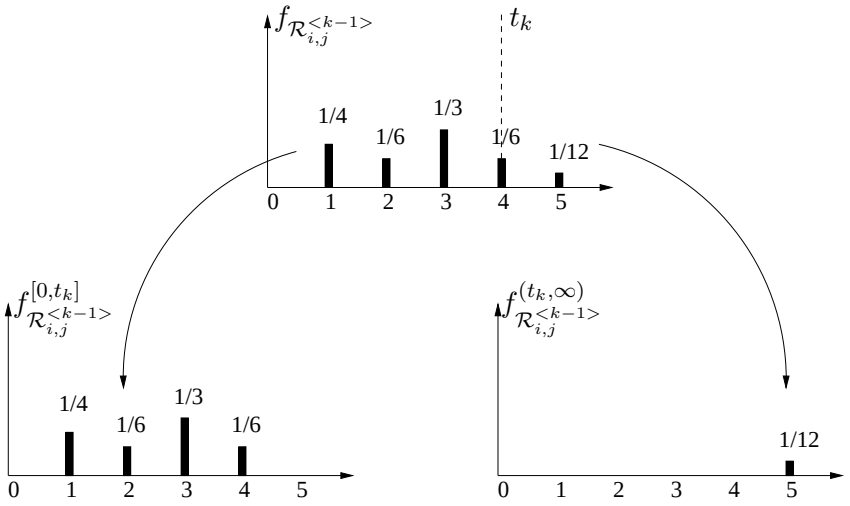

Figure 2.3: Splitting the response time *pmf* from step $k-1$ for a preemptable job

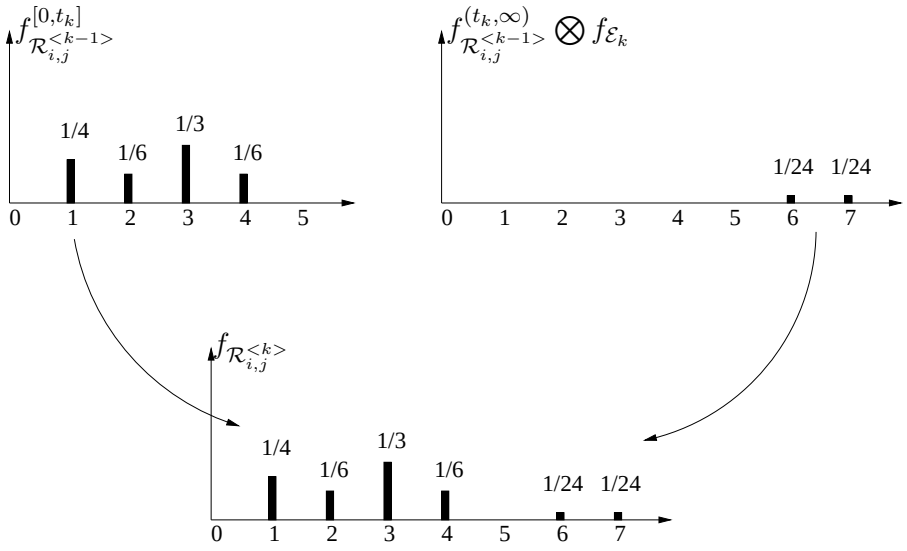

Figure 2.4: Merging the head part and updated tail part to get the response time *pmf* of a preemptable job at step k

2.3 Stochastic Analysis of Periodic Mixed Preemptable Tasks

In this section, we extend the result of [14, 21] to OSEK compliant real-time software systems. As in Section 2.1.1, the scheduling policy in OSEK compliant operating system standard [7] is mixed preemptive. This is the same as in Section 2.2 except that the scheduling policy is non-preemptive if the currently running job is non-preemptable, that is, once a non-preemptable job occupies the shared resource, it will not free the shared resource until it finishes its execution; during this time, all the other jobs will have to wait even if they may have higher priorities than the currently running one.

We now redefine backlog to take into consideration the blocking delay due to the fact that the currently running job has a lower priority and is non-preemptable. Formally, *P-level backlog* $\mathcal{W}_t^P$ observed at time t is defined as the sum of the remaining execution times of all the jobs that have priorities higher than or equal to P and are not completed up to the time t (which includes that the current running job has a priority $\leq P$), *and the possible blocking time when the current executing job has a priority lower than P and is non-preemptable*. Of course if all the tasks in the system are preemptable, this definition is consistent with that of [14].

The overall procedure to calculate the stationary response time of task τ_i as Algorithm 1 is also applicable to systems with mixed preemptive scheduling. First, the stationary distribution of P_i-level backlog at the beginning of the hyperperiod $\mathcal{G}^{P_i}$ is calculated. Then for each job $\Gamma_{i,j}$ in one hyperperiod, the backlog $\mathcal{W}_{Q_{i,j}}^{P_i}$ at its job release time $Q_{i,j}$ is calculated assuming the backlog at the beginning of the hyperperiod (time 0) is $\mathcal{G}^{P_i}$. Finally the response time $\mathcal{R}_{i,j}$ of job $\Gamma_{i,j}$ is calculated, and the task response time $\mathcal{R}_i$ is obtained by averaging the job response times in one hyperperiod.

Please note that in systems with mixed preemptive scheduling, given the P-level backlog

at t, it is much more difficult to calculate the P-level backlog at $t' > t$ because the non-preemptable jobs released between t and t' with lower priority than P may or may not block the job with priority P. This problem is addressed in the next section.

2.3.1 Backlog Update for the Start of a Lower Priority Non-preemptable Job

To calculate the P-level backlog, let us start from the assumption that we can calculate the *pmf* of the start time $\mathcal{S}_{k,l}$ for each job $\Gamma_{k,l}$ that is non-preemptable and has a priority $> P$. This is feasible since the calculation of P_k-level backlog can be calculated in advance based on the release pattern of jobs with priority higher than or equal to P_k and the start time *pmf*s of the jobs with priority lower than P_k, without any knowledge of the P-level backlog. In other words, at each priority level starting from the lowest priority level to the highest one, we can calculate its backlog and further the job start time at this priority level based on the release pattern of the jobs with priority higher or equal and the start time distribution of lower priority non-preemptable jobs.

Let t^+ be the time instant arbitrarily close and immediately after t, and $\mathcal{W}_{t^+}^P$ be the backlog after considering the blocking due to lower priority jobs starting execution at t. Please note that $\mathcal{W}_t^P$ denotes the backlog after considering the release of higher priority jobs at time t. Once the start time *pmf* $f_{\mathcal{S}_{k,l}}$ for each lower priority non-preemptable job $\Gamma_{k,l}$ is known, the intuition is that at each time instant t, we should update the P-level backlog $\mathcal{W}_t^P$ by adding the blocking delay due to these lower priority jobs. However, random variables $\mathcal{W}_t^P$ and $\mathcal{S}_{k,l}$ are not independent. In the following, Lemma 2.1 gives the relationship between two events $(\mathcal{W}_t^P = 0)$ and $(\mathcal{S}_{k,l} = t)$; based on that, Theorem 2.2 determines how to calculate $\mathcal{W}_{t^+}^P$ based on the values of $\mathcal{W}_t^P$ and $\mathcal{S}_{k,l}$.

Lemma 2.1. $\forall P_k > P$, if job $\Gamma_{k,l}$ with priority P_k starts execution at time t, i.e., event $(\mathcal{S}_{k,l} = t)$

happens, then $\mathcal{W}_t^P = 0$; or equivalently $(\mathcal{W}_t^P > 0) \rightarrow (\mathcal{S}_{k,l} \neq t)$.

In terms of probabilities of these random events, since event $(\mathcal{S}_{k,l} = t)$ implies $(\mathcal{W}_t^P = 0)$,

$$\mathbb{P}(\mathcal{S}_{k,l} = t, \mathcal{W}_t^P = 0) = \mathbb{P}(\mathcal{S}_{k,l} = t) \tag{2.2}$$

Also, because event $(\mathcal{S}_{k,l} = t)$ is mutually exclusive with $(\mathcal{W}_t^P = w)$ for any $w > 0$,

$$\forall w > 0, \mathbb{P}(\mathcal{S}_{k,l} = t, \mathcal{W}_t^P = w) = 0 \tag{2.3}$$

Proof. Consider the priority level one higher than P_k, say $P' = P_k - 1$. If job $\Gamma_{k,l}$ starts at time t, then the following two conditions must be satisfied:

- there is no remaining execution time from jobs with priority equal to P' or higher
- there is no job currently running

The P'-level backlog $\mathcal{W}_t^{P'}$ at time t, which is the sum of remaining execution times from jobs with priority $\leq P'$ and the possible blocking time when the current executing job has a priority lower than P' and is non-preemptable, must be equal to zero. Also, for priority level $P \leq P'$, since every contribution to P-level backlog also contributes to P'-level backlog, at any time t, $\mathcal{W}_t^P$ is no greater than $\mathcal{W}_t^{P'}$. Thus we have $\mathcal{W}_t^P \leq \mathcal{W}_t^{P'} = 0$, and furthermore $\mathcal{W}_t^P = 0$. □

Now given the set of lower priority non-preemptable jobs $lp(P) = \{\Gamma_{k,l} | P_k > P \text{ and } N_k = non - preemptable\}$, with their start time *pmf*s $f_{\mathcal{S}_{k,l}}$ and the backlog $\mathcal{W}_t^P$, we calculate the probability of $\mathcal{W}_{t^+}^P$ based on different implications when $\mathcal{W}_t^P$ takes different values: $\mathcal{W}_t^P = 0$ or $\mathcal{W}_t^P > 0$, as in Lemma 2.1. Theorem 2.2 formulates the backlog update procedure considering the blocking time from lower priority non-preemptable jobs at each time tick.

Theorem 2.2. Consider the set of lower priority non-preemptable jobs $lp(P)$, which may start execution at time t. $\mathcal{W}_t^P$ is the backlog at time t, and $\mathcal{W}_{t^+}^P$ the backlog right after t, i.e., immediately

after the possible starts of these lower priority non-preemptable jobs. The probability $\mathbb{P}(\mathcal{W}_{t^+}^P = w)$ is calculated with respect to two different cases of w:

$$\mathbb{P}(\mathcal{W}_{t^+}^P = w) = \begin{cases} \mathbb{P}(\mathcal{W}_t^P = 0) - \sum\limits_{\Gamma_{k,l} \in lp(P)} \mathbb{P}(\mathcal{S}_{k,l} = t) + \sum\limits_{\Gamma_{k,l} \in lp(P)} (\mathbb{P}(\mathcal{S}_{k,l} = t) \times \mathbb{P}(\mathcal{E}_k = 0)) & \\ & \text{if } w = 0 \\ \mathbb{P}(\mathcal{W}_t^P = w) + \sum\limits_{\Gamma_{k,l} \in lp(P)} (\mathbb{P}(\mathcal{S}_{k,l} = t) \times \mathbb{P}(\mathcal{E}_k = w)) & \text{if } w > 0 \end{cases} \tag{2.4}$$

In terms of probability mass functions, the computation of the *pmf* of the backlog $\mathcal{W}_{t^+}^P$ can be performed as follows ($A \times f_{\mathcal{V}}$ denotes the multiplication of the scalar A and the vector $f_{\mathcal{V}}$)

$$f_{\mathcal{W}_{t^+}^P}(w) = \begin{cases} f_{\mathcal{W}_t^P}(0) - \sum\limits_{\Gamma_{k,l} \in lp(P)} f_{\mathcal{S}_{k,l}}(t) + \sum\limits_{\Gamma_{k,l} \in lp(P)} (f_{\mathcal{S}_{k,l}}(t) \times f_{\mathcal{E}_k}(0)) & \text{if } w = 0 \\ f_{\mathcal{W}_t^P}(w) + \sum\limits_{\Gamma_{k,l} \in lp(P)} (f_{\mathcal{S}_{k,l}}(t) \times f_{\mathcal{E}_k}(w)) & \text{if } w > 0 \end{cases} \tag{2.5}$$

Proof. $\forall \Gamma_{k_1,l_1} \neq \Gamma_{k_2,l_2} \in lp(P)$, we consider the following two events: (a)Γ_{k_1,l_1} starts execution at time t; (b)Γ_{k_2,l_2} starts execution at t. Events (a) and (b) are mutually exclusive since at any time there is at most one job running, thus

$$\mathbb{P}(\bigcup_{\Gamma_{k,l} \in lp(P)} \mathcal{S}_{k,l} = t) = \sum_{\Gamma_{k,l} \in lp(P)} \mathbb{P}(\mathcal{S}_{k,l} = t) \tag{2.6}$$

By Lemma 2.1, the start of any job $\Gamma_{k,l} \in lp(P)$ at time t implies that $\mathcal{W}_t^P$ equals zero, thus

$$\begin{aligned} & \mathbb{P}(\bigcup_{\Gamma_{k,l} \in lp(P)} \mathcal{S}_{k,l} = t, \mathcal{W}_t^P = 0) \\ = \; & \mathbb{P}(\bigcup_{\Gamma_{k,l} \in lp(P)} \mathcal{S}_{k,l} = t) \\ = \; & \sum_{\Gamma_{k,l} \in lp(P)} \mathbb{P}(\mathcal{S}_{k,l} = t) \text{(by Equation 2.6)} \end{aligned} \tag{2.7}$$

Furthermore, since the execution time $\mathcal{E}_k$ is independent from random variables $\mathcal{W}_t^P$ and $\mathcal{S}_{k,l}$, we have

$$\begin{aligned}
&\forall e, \mathbb{P}(\bigcup_{\Gamma_{k,l} \in lp(P)} (\mathcal{W}_t^P = 0, \mathcal{S}_{k,l} = t, \mathcal{E}_k = e)) \\
&= \sum_{\Gamma_{k,l} \in lp(P)} \mathbb{P}(\mathcal{W}_t^P = 0, \mathcal{S}_{k,l} = t, \mathcal{E}_k = e) \\
&= \sum_{\Gamma_{k,l} \in lp(P)} (\mathbb{P}(\mathcal{W}_t^P = 0, \mathcal{S}_{k,l} = t) \times \mathbb{P}(\mathcal{E}_k = e)) \\
&= \sum_{\Gamma_{k,l} \in lp(P)} (\mathbb{P}(\mathcal{S}_{k,l} = t) \times \mathbb{P}(\mathcal{E}_k = e)) (\text{by Lemma 2.1})
\end{aligned} \tag{2.8}$$

By Bayes' theorem [38],

$$\mathbb{P}(\mathcal{W}_t^P = 0) = \mathbb{P}(\mathcal{W}_t^P = 0, \bigcup_{\Gamma_{k,l} \in lp(P)} \mathcal{S}_{k,l} = t) + \mathbb{P}(\mathcal{W}_t^P = 0, \overline{\bigcup_{\Gamma_{k,l} \in lp(P)} \mathcal{S}_{k,l} = t}) \tag{2.9}$$

thus the probability that the backlog $\mathcal{W}_t^P$ is zero and there is no job from $lp(P)$ starting execution is

$$\begin{aligned}
&\mathbb{P}(\mathcal{W}_t^P = 0, \overline{\bigcup_{\Gamma_{k,l} \in lp(P)} \mathcal{S}_{k,l} = t}) \\
= \ &\mathbb{P}(\mathcal{W}_t^P = 0) - \mathbb{P}(\mathcal{W}_t^P = 0, \bigcup_{\Gamma_{k,l} \in lp(P)} \mathcal{S}_{k,l} = t) \\
= \ &\mathbb{P}(\mathcal{W}_t^P = 0) - \sum_{\Gamma_{k,l} \in lp(P)} \mathbb{P}(\mathcal{S}_{k,l} = t) (\text{by Equation 2.7})
\end{aligned} \tag{2.10}$$

Now we consider the probability that $\mathcal{W}_{t^+}^P$, the backlog right after the possible starting execution of lower priority non-preemptable jobs at time t, equals zero. It contains two mutually exclusive possibilities:

- $\mathcal{W}_t^P = 0$ and there is no lower priority non-preemptable job starting execution at t;
- $\mathcal{W}_t^P = 0$ and a lower priority non-preemptable job $\Gamma_{k,l}$ starts execution at t, with its execution time $\mathcal{E}_k$ equal to zero.

Thus the probability of $\mathcal{W}_{t^+}^P = 0$ is

$$
\begin{aligned}
& \mathbb{P}(\mathcal{W}_{t^+}^P = 0) \\
= \; & \mathbb{P}(\mathcal{W}_t^P = 0, \overline{\bigcup_{\Gamma_{k,l} \in lp(P)} \mathcal{S}_{k,l} = t}) + \mathbb{P}(\bigcup_{\Gamma_{k,l} \in lp(P)} (\mathcal{W}_t^P = 0, \mathcal{S}_{k,l} = t, \mathcal{E}_k = 0)) \\
= \; & \mathbb{P}(\mathcal{W}_t^P = 0) - \sum_{\Gamma_{k,l} \in lp(P)} \mathbb{P}(\mathcal{S}_{k,l} = t) + \mathbb{P}(\bigcup_{\Gamma_{k,l} \in lp(P)} (\mathcal{W}_t^P = 0, \mathcal{S}_{k,l} = t, \mathcal{E}_k = 0)) \\
& \text{(by Equation 2.10)} \\
= \; & \mathbb{P}(\mathcal{W}_t^P = 0) - \sum_{\Gamma_{k,l} \in lp(P)} \mathbb{P}(\mathcal{S}_{k,l} = t) + \sum_{\Gamma_{k,l} \in lp(P)} (\mathbb{P}(\mathcal{S}_{k,l} = t) \times \mathbb{P}(\mathcal{E}_k = 0)) \text{(by Equation 2.8)}
\end{aligned}
\tag{2.11}
$$

Consider the probability of $\mathcal{W}_{t^+}^P = w, \forall w > 0$. To have the contribution from $\mathcal{W}_t^P$ and the blocking of a lower priority non-preemptable job $\Gamma_{k,l}$ adding up to w, there are three mutually exclusive possibilities as follows:

- Scenario (1): $\mathcal{W}_t^P = w > 0$, and there is no lower priority non-preemptable job starting execution at t;
- Scenario (2): a lower priority non-preemptable job starts execution at t, with execution time equal to $e < w$, and $\mathcal{W}_t^P = w - e > 0$;
- Scenario (3): $\mathcal{W}_t^P = 0$, and a lower priority non-preemptable job starts execution at t, with execution time equal to w.

By Lemma 2.1, $\mathcal{W}_t^P = w > 0$ implies that there is no lower priority non-preemptable job starting execution at t. The probability of scenario (1) can be calculated as

$$\begin{aligned}
\forall w > 0, \quad & \mathbb{P}(\mathcal{W}_t^P = w, \overline{\bigcup_{\Gamma_{k,l} \in lp(P)} \mathcal{S}_{k,l} = t}) \\
= \quad & \mathbb{P}(\mathcal{W}_t^P = w) - \mathbb{P}(\mathcal{W}_t^P = w, \bigcup_{\Gamma_{k,l} \in lp(P)} \mathcal{S}_{k,l} = t) \\
= \quad & \mathbb{P}(\mathcal{W}_t^P = w) - \sum_{\Gamma_{k,l} \in lp(P)} \mathbb{P}(\mathcal{W}_t^P = w, \mathcal{S}_{k,l} = t) \\
= \quad & \mathbb{P}(\mathcal{W}_t^P = w)
\end{aligned} \tag{2.12}$$

For scenario (2), by Lemma 2.1 again, $\mathcal{W}_t^P = w - e > 0$ implies that there is no lower priority non-preemptable job starting execution at t,

$$\forall w > e, \mathbb{P}(\bigcup_{\Gamma_{k,l} \in lp(P)} (\mathcal{S}_{k,l} = t, \mathcal{E}_k = e, \mathcal{W}_t^P = w - e)) = 0 \tag{2.13}$$

For scenario (3), its probability has been calculated as in Equation 2.8.

Combining all the above three cases, the probability of $\mathcal{W}_{t^+}^P = w, \forall w > 0$ is

$$\begin{aligned}
\forall w > 0, \quad & \mathbb{P}(\mathcal{W}_{t^+}^P = w) \\
= \quad & \mathbb{P}(\mathcal{W}_t^P = w, \overline{\bigcup_{\Gamma_{k,l} \in lp(P)} \mathcal{S}_{k,l} = t}) \\
+ \quad & \sum_{e=0}^{w-1} \mathbb{P}(\bigcup_{\Gamma_{k,l} \in lp(P)} (\mathcal{S}_{k,l} = t, \mathcal{E}_k = e, \mathcal{W}_t^P = w - e)) \\
+ \quad & \mathbb{P}(\bigcup_{\Gamma_{k,l} \in lp(P)} (\mathcal{S}_{k,l} = t, \mathcal{E}_k = w, \mathcal{W}_t^P = 0)) \\
= \quad & \mathbb{P}(\mathcal{W}_t^P = w) + 0 + \sum_{\Gamma_{k,l} \in lp(P)} (\mathbb{P}(\mathcal{S}_{k,l} = t) \times \mathbb{P}(\mathcal{E}_k = w)) \text{(by Equations } 2.12, 2.13, 2.8) \\
= \quad & \mathbb{P}(\mathcal{W}_t^P = w) + \sum_{\Gamma_{k,l} \in lp(P)} (\mathbb{P}(\mathcal{S}_{k,l} = t) \times \mathbb{P}(\mathcal{E}_k = w))
\end{aligned} \tag{2.14}$$

□

For each lower priority non-preemptable job $\Gamma_{k,l} \in lp(P)$, the update of the backlog $\mathcal{W}_{t^+}^P$ is performed by first subtracting the probability of $\mathcal{S}_{k,l} = t$ from $f_{\mathcal{W}_{t^+}^P}(0)$, then the *pmf* of the

execution time $\mathcal{E}_k$ is scaled by $\mathbb{P}(\mathcal{S}_{k,l} = t)$ and added to the *pmf* of $\mathcal{W}^P_{t^+}$. This operation is called *spreading* since the probability of $\mathcal{S}_{k,l} = t$ is taken away from $f_{\mathcal{W}^P_{t^+}}(0)$ and spread according to the shape of $f_{\mathcal{E}_k}$. Figure 2.5 and 2.6 give an example of how the P-level backlog should be updated from t to t^+ considering the probability of a lower priority non-preemptable job $\Gamma_{k,l}$ starting its execution at time t.

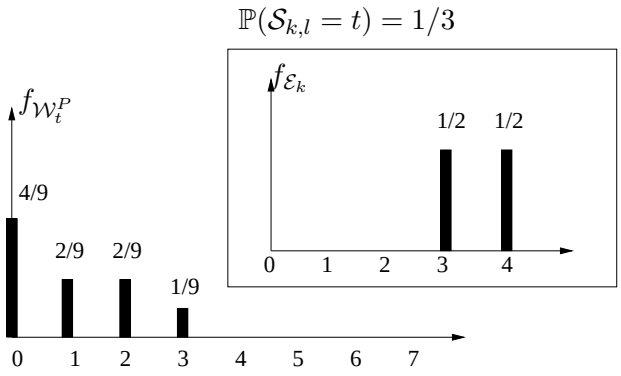

Figure 2.5: At time t, just before considering the start of the job with priority lower than P (the execution time *pmf* and the probability of the job starting at t are shown in the box)

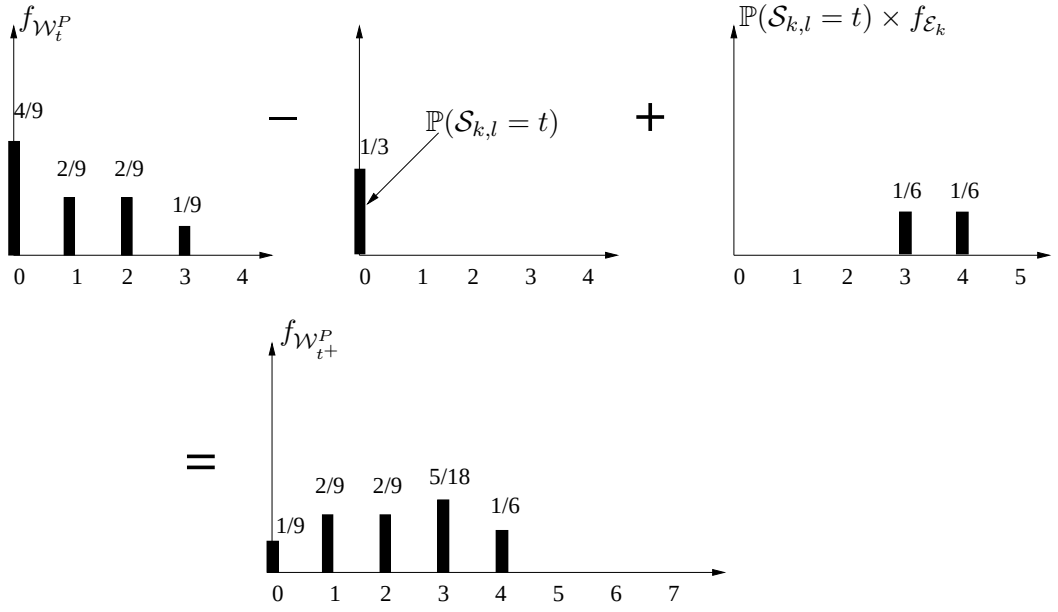

Figure 2.6: At time t^+, just after considering the start of the job: spreading the execution time *pmf* with probability $\mathbb{P}(\mathcal{S}_{k,l} = t)$

The backlog update procedure in Algorithm 3 is modified considering the blocking of lower priority non-preemptable tasks in the system. At each time tick, first the amount of interference is added by convolving the *pmf*s of the backlog and the execution times of the jobs with priority higher than or equal to P; then blocking from lower priority non-preemptable jobs is added by the operation of spreading; the shrinking operation is performed from t to $(t+1)^-$, the next time tick. The operations of convolution, spreading and shrinking are iteratively applied. Note that the backlog right before the time t is denoted as $\mathcal{W}_{t^-}^P$, the backlog after considering the interferences but without taking into account the blocking from lower-priority non-preemptable jobs is denoted as $\mathcal{W}_t^P$, and the backlog right after t, i.e., the start of these jobs introducing blocking delay to the backlog, is denoted as $\mathcal{W}_{t^+}^P$. This is to make clear that the backlog is updated at each time tick by first considering the interferences, then the possible blocking delays.

2.3.2 Stationary Response Time Calculation for Non-preemptable Jobs

As in [14], the computation of the response time of a job $\Gamma_{i,j}$ with priority P_i, requires, as a first step, the calculation of the backlog $\mathcal{W}_{Q_{i,j}}^{P_i}$ at its release time $Q_{i,j}$. Note that for a non-preemptable job, once it starts running, it will finish its execution, thus the possible future interferences will happen before its start time. Similar to Section 2.2.4, the set of jobs released after $Q_{i,j}$ that may preempt $\Gamma_{i,j}$ is denoted as $hp(P_i)^{(Q_{i,j},\infty)} = \{\Gamma_{m,n} | P_m < P_i \text{ and } Q_{m,n} > Q_{i,j}\}$. The jobs in the set $hp(P_i)^{(Q_{i,j},\infty)}$ are ordered by their release times. Let Γ_k be the k-th job with relative release time $t_k = Q_k - Q_{i,j}$, and $f_{\mathcal{S}_{i,j}^{<k>}}$ denote the start time *pmf* of job $\Gamma_{i,j}$ after considering the possible interferences from $\Gamma_1, \Gamma_2, ..., \Gamma_k$. $\mathcal{S}_{i,j}$ is initialized as the P_i-level backlog right before its release, i.e., $f_{\mathcal{S}_{i,j}^{<0>}} = f_{\mathcal{W}_{Q_{i,j}^{P_i}}}$. Then for each job $\Gamma_k \in hp(P_i)^{(Q_{i,j},\infty)}$, the start time *pmf*

Algorithm 4 Update_Backlog_MixedPreemptive($t_1, \mathcal{W}_{t_1}^P, t_2$) *// Given the backlog* $\mathcal{W}_{t_1}^P$ *at time* t_1*, calculate backlog at time* $t_2 > t_1$

1: **for** time instant t from t_1 to t_2 **do**
2: **for** each job $\Gamma_{i,j}$ released at time t **do**
3: **if** $P_i \leq P$ **then**
4: $f_{\mathcal{W}_t^P} = f_{\mathcal{W}_t^P} \bigotimes f_{\mathcal{E}_i}$
5: **end if**
6: **end for**
7: **for** each job $\Gamma_{i,j}$ starting execution at time t **do**
8: **if** $P_i > P$ and $N_i = non - preemptable$ **then**
9: Spread probability $\mathbb{P}(\mathcal{S}_{i,j} = t)$ by $f_{\mathcal{E}_i}$ to $f_{\mathcal{W}_t^P}$ as Theorem 2.2 and the example in Figure 2.5 and 2.6
10: **end if**
11: **end for**
12: Shrink $f_{\mathcal{W}_t^P}$ from t to $(t+1)^-$
13: **end for**

$f_{\mathcal{S}_{i,j}^{<k-1>}}$ calculated at the previous step is split into two parts, the head part $f_{\mathcal{S}_{i,j}^{<k-1>}}^{[0,t_k)}$ and the tail part $f_{\mathcal{S}_{i,j}^{<k-1>}}^{[t_k,\infty)}$. In the head part $\Gamma_{i,j}$ starts execution before the release of Γ_k, while in the tail part Γ_k will further interfere $\Gamma_{i,j}$ thus $\mathcal{E}_k$ needs to be added (by convolving these two *pmf*s). After this, $f_{\mathcal{S}_{i,j}^{<k>}}$ is constructed by merging the head part and the updated tail part.

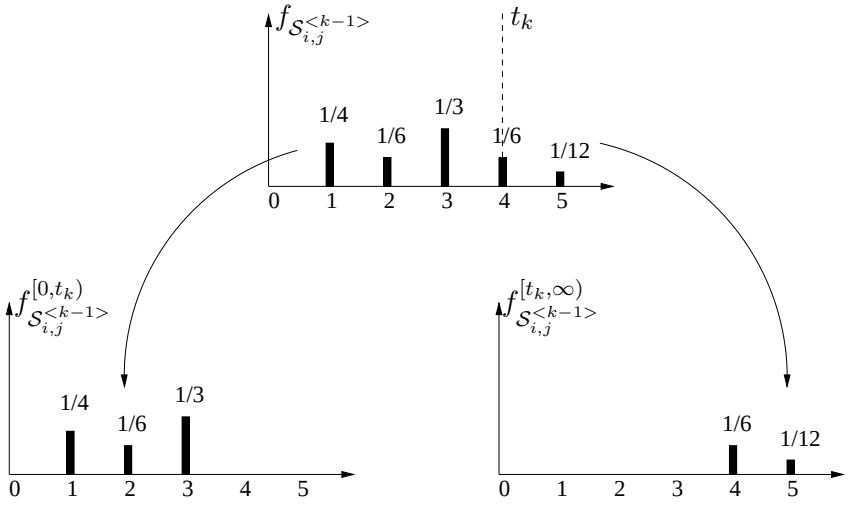

Figure 2.7: Splitting the start time *pmf* from step $k-1$ for a non-preemptable job

An example of the splitting and merging operation on the start time *pmf* is illustrated in Figure 2.7 and 2.8. Please note the differences of the calculation of future preemption between preemptable (Figure 2.3 and 2.4) and non-preemptable jobs (Figure 2.7 and 2.7). First, for a preemptable job, the calculation is using its response time, while for a non-preemptable job its start time is used. Second, for a preemptable job, the head part contains the point t_k, the relative release time of a higher priority job Γ_k, while for a non-preemptable job, it does not contain t_k since the future interference needs to be added if Γ_k is released at the same time when job $\Gamma_{i,j}$ with a lower priority tries to start.

Once the *pmf* of the time $\mathcal{S}_{i,j}$ at which $\Gamma_{i,j}$ starts its execution is known, the *pmf* of its

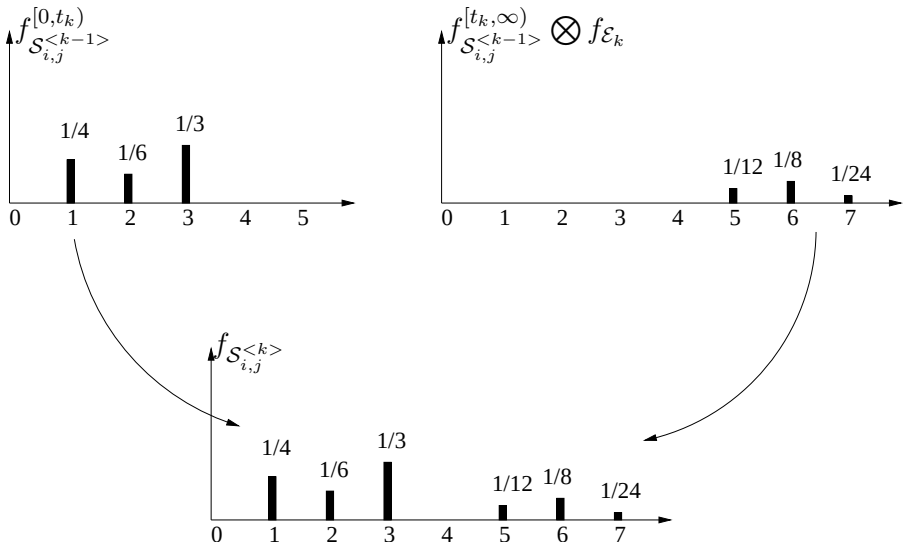

Figure 2.8: Merging the head part and updated tail part to get the start time *pmf* of a non-preemptable job at step k

finish time is obtained by simply adding its execution time $\mathcal{E}_i$, i.e., performing a convolution on the *pmf*s of $\mathcal{S}_{i,j}$ and $\mathcal{E}_i$. The response time is finally computed by a further left shift of $Q_{i,j}$ time units.

$$\begin{aligned} & f_{\mathcal{F}_{i,j}} = f_{\mathcal{S}_{i,j}} \bigotimes f_{\mathcal{E}_i} \\ & \forall t, \mathbb{P}(\mathcal{R}_{i,j} = t - Q_{i,j}) = \mathbb{P}(\mathcal{F}_{i,j} = t) \end{aligned} \tag{2.15}$$

2.4 Experimental Results

In this section, we present the experimental results on a set of application tasks from an automotive ECU. The test set consists of 16 software tasks as shown in Table 2.1, where for each task τ_i, parameters T_i, O_i, P_i and N_i [2] are the same as in Section 2.1.2. E_i^{min} and E_i^{max} are the minimum and maximum execution times, and the execution time is assumed to be uniformly distributed within $[E_i^{min}, E_i^{max}]$. The worst case system utilization is 145%, and the mean utilization of the system is

[2]The preemption property N_i is set to be non-preemptive for the first 8 highest priority tasks in the set.

72.8%.

τ_i	$T_i(\times 10\mu s)$	O_i	P_i	N_i	$E_i^{min}(\times 10\mu s)$	$E_i^{max}(\times 10\mu s)$
τ_1	1000	0	1	Non-Preemptive	1	133
τ_2	1000	0	2	Non-preemptive	1	371
τ_3	2000	0	3	Non-preemptive	1	440
τ_4	2000	0	4	Non-preemptive	1	345
τ_5	4000	0	5	Non-Preemptive	1	255
τ_6	4000	0	6	Non-Preemptive	1	309
τ_7	4000	0	7	Non-preemptive	1	248
τ_8	4000	0	8	Non-preemptive	1	248
τ_9	5000	0	9	Preemptive	1	212
τ_{10}	5000	0	10	Preemptive	1	212
τ_{11}	5000	0	11	Preemptive	1	402
τ_{12}	10000	0	12	Preemptive	1	283
τ_{13}	10000	0	13	Preemptive	1	108
τ_{14}	10000	0	14	Preemptive	1	260
τ_{15}	10000	0	15	Preemptive	1	255
τ_{16}	10000	0	16	Preemptive	1	343

Table 2.1: The list of tasks on an example automotive ECU

We check the quality of our stochastic analysis method by comparing its results with the results of simulations. The granularity of our analysis and simulation is set to $10\mu s$. The simulation is performed by randomly choosing execution times, and repeated for 8×10^8 hyperperiods. The stationary distribution of backlogs is obtained by iterative approximation, that is, by first applying the algorithm that computes the backlogs, then stopping when the *pmf*s of backlogs are close enough at the end of two consecutive hyperperiods, where the analysis error ϵ as in Alogrithm2 is assigned to be 10^{-9}. The stationary distribution of the task response time is calculated as the average of the instances queued in the next hyperperiod once the stationary distribution of backlogs is achieved.

The deadline of each task is assumed to be half of its period. Table 2.2 shows the results of calculated deadline miss probability for each task in the test set. The one obtained by the

τ_i	Deadline	Simulation	Analysis	τ_i	Deadline	Simulation	Analysis
τ_1	500	0.000	0.000	τ_9	2500	0.011	0.011
τ_2	500	0.023	0.023	τ_{10}	2500	0.026	0.026
τ_3	1000	0.000	0.000	τ_{11}	2500	0.083	0.083
τ_4	1000	0.037	0.037	τ_{12}	5000	0.001	0.001
τ_5	2000	0.000	0.000	τ_{13}	5000	0.002	0.002
τ_6	2000	0.000	0.000	τ_{14}	5000	0.005	0.005
τ_7	2000	0.003	0.003	τ_{15}	5000	0.013	0.013
τ_8	2000	0.018	0.018	τ_{16}	5000	0.039	0.038

Table 2.2: Deadline miss probability for each task in the test set

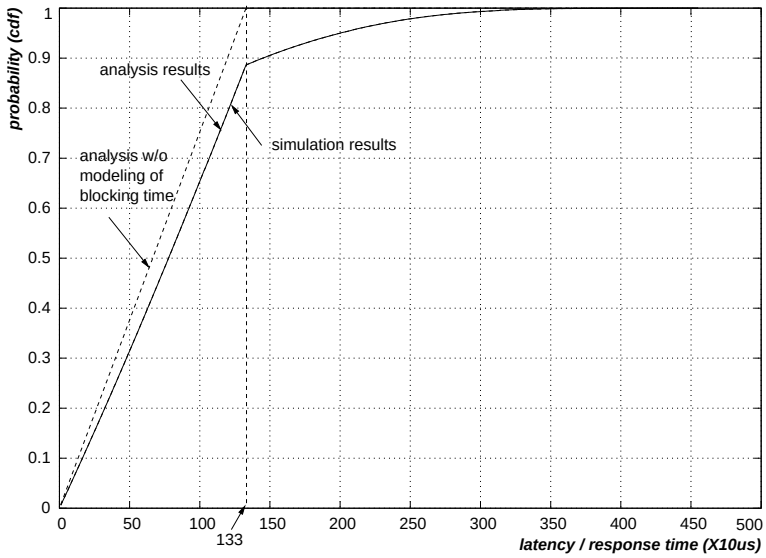

Figure 2.9: The response time *cdf* of task τ_1 in the test set

proposed analysis method is almost identical to the simulation result. Furthermore, we present the *cumulative distribution function* of the response time for τ_1, the highest priority task, as shown in Figure 2.9. Theoretically, if we run the simulation long enough, the only source of error between our analysis and simulation is limited computational precision. As expected, the results obtained from the analysis and the simulation are almost the same, as demonstrated in both Table 2.2 and Figure 2.9. In the figure, the analysis of the response time of τ_1 has been tentatively performed assuming that each task in the test set is preemptive, thus there is no blocking delay to τ_1 due to lower priority non-preemptable tasks. The result is quite optimistic with respect to the actual values and shows the accuracy in the modeling of the additional contribution to the backlog caused by the blocking factor.

Chapter 3

Stochastic Analysis of Controller Area Network Message Response Times

In this chapter, we present the stochastic analysis framework that approximates the response times for CAN messages. *None of the stochastic methods proposed so far*, including the one in [14, 21], *covers the analysis of CAN periodic messages transmitted by nodes without clock synchronization and without preemption*. The analysis provided in [14, 21], for example, assumes the knowledge of the (relative) activation times between the tasks, but it does not cover blocking delays due to CAN non-preemptive nature. Other stochastic approaches assume Poisson arrivals and/or are not applicable to medium or light load conditions. Previous stochastic analysis of CAN-based systems in [31] and [33] focus on critical instant conditions, *without considering the probability of occurrence of a critical instant*. Besides, in these methods there is no analysis of different phase configurations, which may result in different response time values from those computed in the critical instant scenario. To the best of our knowledge, our method is *the first attempt toward the evaluation*

of the probability functions of message response times in the scenarios of non-deterministic message phasing.

Next, we first define a model of the distributed CAN system, then we describe our characterization message abstraction. Next, we provide the formula for computing the backlog, and the *pmf*s of the message response times. Finally, we provide some experimental results on a subset of messages from the real automotive application.

3.1 System Model and Notation

3.1.1 CAN Bus Arbitration and Message Format

The CAN arbitration protocol is both priority-based and *non-preemptive*, that is, the transmission of a message can not be preempted by higher priority messages that are queued when the message is being transmitted. The CAN [2] bus is essentially a wired AND channel connecting all nodes. The media access protocol works by alternating contention and transmission phases. The time axis is divided into slots that must be larger than or equal to the time it takes for the signal to travel back and forth along the channel. The contention and transmission phases take place during the digital transmission of the frame bits. The CAN protocol has the message format of Figure 3.1, where the sizes of the fields are expressed in bits. Priorities are encoded in the message identifier field (11 bits wide for the standard format, 29 for the extended format). The identifier is used not only for bus arbitration, but also for describing the data content (identification of the message stream). The CAN protocol requires that all contending messages have a unique identifier (unique priority). The other fields are: the control field containing information on the type of message; the data field containing the actual data to be transmitted, up to a maximum of 8 bytes; the checksum

used to check the correctness of the message bits; the acknowledge (ACK) used to acknowledge the reception; the ending delimiter (ED) used to signal the end of the message and the idle space (IS) or inter-frame bits (IF) used to separate one frame from the following one.

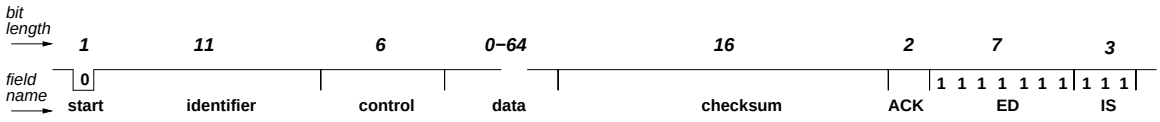

Figure 3.1: The CAN data frame format

If a node wishing to transmit finds the shared medium in an idle state, it waits for the next slot and starts an arbitration phase by issuing a start-of-frame bit. At this point, each node with a message to be transmitted (e.g., the message may be placed in a peripheral register called *TxObject*) can start racing to grant access of the shared medium, by serially transmitting the identifier (priority) bits of the message in the arbitration slots, one bit for each slot starting from the most significant one. Collisions among identifier bits are resolved by the logical AND semantics: if a node reads its priority bits on the medium without any change, it realizes it is the winner of the contention and it is granted access for transmitting the rest of the message, and the other nodes switch to a listening mode. In fact, if one of the bits is changed when reading it back from the medium, this means there is a higher priority (dominant bit) contending the medium and thus the message withdraws. Finally, as an additional protocol feature, whenever the message frame contains a sequence of five consecutive identical bits, an additional trailing complementary bit is added (bit stuffing). This is necessary to keep the clocks of the receiving nodes synchronized with the bit stream (transmitted with NRZ encoding). For more details on the protocol the interested reader can read the actual specification of the CAN protocol in [2].

3.1.2 Model of the CAN System

We consider a CAN system where messages are queued periodically. At each node, the bus adapter sends the queued messages in priority (ID) order. Its transmit registers (TxObjects) can be overwritten if a higher priority message becomes available in the sorted queue.

A message m_i is defined by $(T_i, \mathcal{O}_i, \mathcal{E}_i, P_i)$, where T_i is its period, $\mathcal{O}_i$ its *random* initial phase, $\mathcal{E}_i$ its transmission time, and P_i its priority. The message transmission time $\mathcal{E}_i$ is a random variable due to the effect of bit-stuffing. The *hyperperiod* H of the system is defined as the least common multiple of message periods, i.e., $H = lcm(\bigcup_i T_i)$.

For each periodic activation, we consider a *message instance*. On its arrival, a message instance is *queued* by a periodic middleware task, executing with a period equal to the greatest common divisor (*gcd*) of the message periods at the node. We denote the j-th instance of message m_i as $M_{i,j}$, its arrival time as $\mathcal{A}_{i,j} = \mathcal{O}_i + (j-1) \times T_i$ [1], its queuing time as $\mathcal{Q}_{i,j} = \mathcal{A}_{i,j}$, its start time as $\mathcal{S}_{i,j}$, and its finish time as $\mathcal{F}_{i,j}$. Note that the arrival times of the instances from the same message are not independent, i.e., given a value to the phase $\mathcal{O}_i$ of the message, the arrival times of all its message instances are determined as well. In other words, although the initial phase is random, we assume it is constant in the hyperperiod.

Each message m_i is associated with a unique ID P_i, which also represents its priority. According to the CAN protocol, the lower the message ID, the higher its priority, $P_i < P_j$ implies that m_i has a higher priority than m_j.

We assume there is no synchronization mechanism for the local clocks of the nodes connected to the CAN bus. From the standpoint of a message instance on node ECU_k, the initial phase

[1] The *pmf* of the sum of a random variable $\mathcal{V}_1$ and a regular variable V_2 is that $\forall v_1, \mathbb{P}(\mathcal{V}_1 + V_2 = v_1 + V_2) = \mathbb{P}(\mathcal{V}_1 = v_1)$.

of a message m_i from any other node $ECU_l (l \neq k)$ is $\mathcal{O}_i = \Psi_i + \mathcal{O}_{k,l}$, where Ψ_i is the local phase of message m_i, and $\mathcal{O}_{k,l}$ the relative phase between ECU_k and ECU_l. Obviously the phases and thus the arrivals of different messages on the same node are not independent. $\mathcal{O}_{k,l}$ is a continuous random variable, modeling the random message queuing times. We assume, without loss of generality, that the *pdf* of the relative phase $\mathcal{O}_{k,l}$ is uniformly distributed within the interval $(0, H)$.

The *response time* $\mathcal{R}_{i,j}$ of a message instance $M_{i,j}$ is defined as the time interval from its arrival to its finish, i.e., $\mathcal{R}_{i,j} = \mathcal{F}_{i,j} - \mathcal{A}_{i,j}$, which is a random variable that depends on the phases of the messages from other nodes, and the non-deterministic transmission times. The *pmf* $f_{\mathcal{R}_i}$ of the response time of message m_i is obtained as the average of the response time *pmf*s of all the message instances $M_{i,j}$ in the hyperperiod provided that the distribution is stationary.

3.1.3 A Modeling Abstraction for CAN Messages

3.1.3.1 The Characterization Message

Our abstraction is based on the observation that in most automotive systems message queuing is performed by a single middleware-level task, executing with a period equal to the *gcd* of the message periods at the node. Hence, every message on the same node has the initial phase and period as the integer multiple of the *gcd* of the periods. From the standpoint of a generic message m_j, the queuing of message instances from any other node is separated by at least $gcd(\bigcup_i T_i)$. If the worst case response time of m_j is no greater than the greatest common divider of the message periods, i.e.,

$$gcd(\bigcup_i T_i) \geq max(\mathcal{R}_j) \tag{3.1}$$

then it will suffer interferences from at most one set of message instances from any other node. At any time t, the backlog from message instances that are queued before or at t, can be assumed as due to one such message instance set for each node.

In general, of course, this condition will be true only for a (high priority) subset of all messages, but we claim that our method retains sufficient accuracy for the evaluation of message response times even when this assumption is violated, as our experiments show. Besides, we can provide an estimate of the error.

Within a hyperperiod, the queuing of instances from messages in the set $hp(P_j) = \{m_i | P_i < P_j\}$ (with priority higher than P_j), only happens at integer multiples of the $gcd(\bigcup_{i \in hp(P_j)} T_i)$. Thus we can use one *characterization message* per node to model the interference due to such messages. The characterization message transmission time, from the standpoint of a lower priority message on another node, is non-deterministic because of the random clock phases.

Figure 3.2 provides an example with three messages with periods and transmission times equal to (60,2),(10,1),(20,1), respectively. During the hyperperiod [0,60), the message instances are queued at the time instants $k \times 10, k = 0, 1, 2, 3, 4, 5$. At time 0, all messages are queued and the total transmission time is 4. Similarly, the transmission times that are requested at the other time instants $k \times 10$ are 1, 2, 1, 2, 1, respectively. From the standpoint of a message at another node, the messages are queued at these time instants with equal probabilities. Three time instants out of six contribute to a transmission time of 1 unit, two out of six to 2 units, and in one case out of six, 4 units. Thus the *pmf* of the characterization message transmission time is defined as $\mathbb{P}(1) = 1/2, \mathbb{P}(2) = 1/3, \mathbb{P}(3) = 0, \mathbb{P}(4) = 1/6$.

In CAN systems, message transmission times $\mathcal{E}_i$ are random because the number of stuff

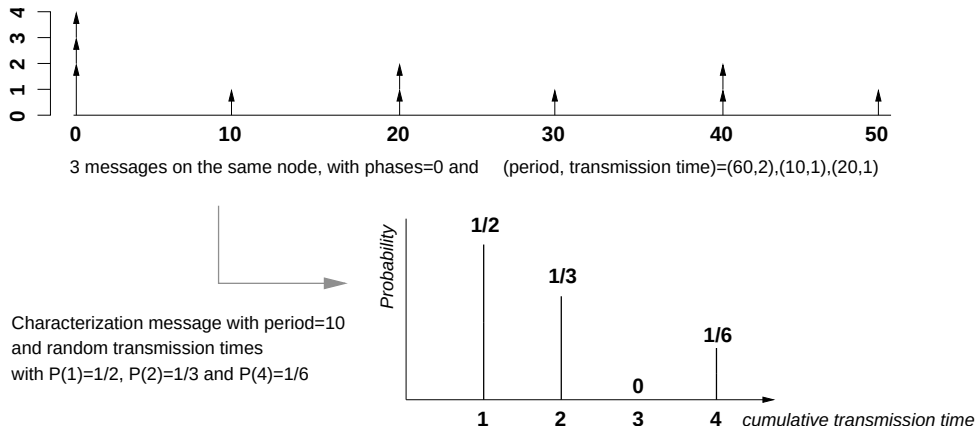

Figure 3.2: An example of characterization message transmission time

bits are non-deterministic, and are characterized by *pmf*s that account for bit stuffing. A simple model for bit stuffing can be obtained by assuming equal probability of having a zero or a one for each bit. Models for computing the probability of a given number of stuff bits can be found in [34].

Algorithm 5 computes the transmission time *pmf* of the P-level characterization message. First, the set of messages $hp(P)$ with priority higher than P is identified, where the lcm of their periods is the hyperperiod H, and the gcd of their periods is the time interval T_c that separates the queuing of higher priority messages. Then, for each possible multiple of T_c within one hyperperiod $[0, H)$, i.e., $t = kT_c, k = 0, 1, ..., H/T_c - 1$, the *pmf* of the total transmission time $\mathcal{E}[k]$ is calculated by accumulating (by convolution) the *pmf*s of the transmission times of the message instances queued at t. Finally, the transmission time *pmf* of the characterization message is obtained by averaging the *pmf*s of $\mathcal{E}[k]$ at these time instants.

To model the effect of randomized phases for the sets of remote messages, we define a random queuing jitter for the characterization message m_c, with deterministic arrival time. Its random queuing jitter is uniformly distributed within its period T_c. This second abstraction further

Algorithm 5 Calculate the transmission time *pmf* $f_{\mathcal{E}_c}$ of P-level characterization message m_c

$hp(P) = \{m_i | P_i < P\}$

$H = lcm(\bigcup_{i \in hp(P)} T_i), T_c = gcd(\bigcup_{i \in hp(P)} T_i)$

for $k = 0, 1, ..., H/T_c - 1$ **do**

  $\mathbb{P}(\mathcal{E}[k] = 0) = 1.0$

  $t = k \times T_c$

  for each $i \in hp(P)$ **do**

    if t mod T_i = 0 **then**

      $f_{\mathcal{E}[k]} = f_{\mathcal{E}[k]} \bigotimes f_{\mathcal{E}_i}$

    end if

  end for

end for

for each e **do**

  $\mathbb{P}(\mathcal{E}_c = e) = \frac{T_c}{H} \sum_{k=0}^{H/T_c - 1} \mathbb{P}(\mathcal{E}[k] = e)$

end for

introduces inaccuracy due to the non-periodic queuing patterns of the message instances.

We now provide the formal definition of the *P-level characterization message* m_c for $\{m_i\}$, as follows:

- Period $T_c = gcd(\bigcup_{i \in hp(P)} T_i)$, $hp(P) = \{m_i | P_i < P\}$
- Initial phase $O_c = -T_c/2$ [2]
- Queuing jitter $\mathcal{J}_c$ uniformly distributed in $[0, T_c)$
- Random transmission time $\mathcal{E}_c$, as shown in the example in Figure 3.2 and Algorithm 5
- Priority P_c higher than P

3.1.3.2 Approximate System Model

To analyze the response time of a message m_i, we model an approximate system consisting of all the messages from the same node with priority higher than or equal to P_i, including m_i itself. These messages have no queuing jitter. In addition, we consider one characterization message of level P_i for each of the other nodes, with random queuing jitter and transmission time. In the approximate system, a message m_i (including the characterization messages) is modeled by $(T_i, O_i, \mathcal{J}_i, \mathcal{E}_i, P_i)$. The attributes T_i, O_i and P_i have the same meaning as in Section 3.1.2, the transmission time $\mathcal{E}_i$ is the random transmission time with a *pmf* computed as in Algorithm 5, and $\mathcal{J}_i$ is the random queuing jitter. On its arrival, each message instance $M_{i,j}$ is *queued* after some time $\mathcal{J}_i$, the *queuing jitter*. $\mathcal{J}_i$ is a discrete random variable with a known uniform *pmf*. For the j-th instance $M_{i,j}$ of m_i, the deterministic arrival time is $A_{i,j} = O_i + (j-1) \times T_i$, and its queuing time

[2] We maximize the chance that at most only one characterization message instance will interfere with any real message because its queuing jitter spans between $[-T_c/2, +T_c/2)$ and its initial phase is the deterministic offset with respect to any other real message in the system.

$\mathcal{Q}_{i,j} = A_{i,j} + \mathcal{J}_i$ is assumed to be independent from other instances of the same message, and of any other message, because of the non-deterministic jitter contribution.

Another possible source of inaccuracy is that real messages have random phases, but request transmission after exactly one period. This is not true for the characterization message with a random release jitter, which does not ensure periodic requests transmission after the first arrival. This limitation has no effect if the worst case response time of any message is shorter than half of the greatest common divider of the periods of all messages. When this is not true, the analysis is approximate and is affected by an increasing error.

3.1.3.3 Extended Definition of Backlog

We now introduce an extension of the backlog defined in [14]. The *P-level backlog $\mathcal{W}_t^P$ at time t* is the sum of the remaining transmission times of the queued message instances that have priorities higher than or equal to P.

For characterization message instances with random queuing jitter, we define the event that a message instance $M_{i,j}$ is queued after t as $V_t^{i,j}$, that is, $V_t^{i,j} = (\mathcal{Q}_{i,j} > t)$, and its *complement* $\overline{V}_t^{i,j} = (\mathcal{Q}_{i,j} \leq t)$. Let us suppose that at time t there are n message instances $M_{i_1,j_1}, M_{i_2,j_2}, ..., M_{i_n,j_n}$ in the queue, and have priorities higher than or equal to P (In CAN, no two messages can have the same priority). The set of these n message instances is defined as the *P-level message instance set at time t*, denoted as $I(P,t)$.

For each such instance $M_{i_k,j_k} \in I(P,t)$, the two mutually exclusive events $V_t^{i_k,j_k}$ and $\overline{V}_t^{i_k,j_k}$ divide the event space into two subspaces. We define the *P-level event space* $S(P,t)$ at time t as $S(P,t) = \{(V_t^{i_1,j_1}, V_t^{i_2,j_2}, ..., V_t^{i_n,j_n}), (V_t^{i_1,j_1}, V_t^{i_2,j_2}, ..., \overline{V}_t^{i_n,j_n}),...,(\overline{V}_t^{i_1,j_1}, \overline{V}_t^{i_2,j_2}, ..., \overline{V}_t^{i_n,j_n})\}$, which contains 2^n event vectors, representing different combinations of the random queuing times

of n message instances with priority P or higher with respect to time t. We define the *queuing pattern vector* at time t for the possible queuing times of these message instances, denoted as $\mathcal{X}_t$. Therefore, $\mathcal{X}_t$ is a random variable defined over $S(P,t)$, the P-level event space, with a known probability mass function.

For a queuing pattern $\mathcal{X}_t$, if the queuing event at t corresponding to the instance $M_{i,j}$ is $V_t^{i,j}$, then $\mathcal{X}_t$ is called a *positive queuing pattern* of $V_t^{i,j}$; otherwise, if $\overline{V}_t^{i,j}$ applies, it is a *negative queuing pattern* of $V_t^{i,j}$. The *complementary queuing pattern* $\overline{\mathcal{X}}_{t(i,j)}$ *with respect to* $M_{i,j}$ is a queuing pattern defined by complementing the $V_t^{i,j}$ entry in $\mathcal{X}_t$.

We define a compound probability mass function of two random variables, $\mathcal{W}_t^P$, defined over the discretized R^+, and $\mathcal{X}_t$ defined over the 2^n dimension P-level event space, e.g., $f_{(\mathcal{W}_t^P,\mathcal{X}_t)}(w,(V_t^{i_1,j_1},V_t^{i_2,j_2},...,V_t^{i_n,j_n})) = \mathbb{P}(\mathcal{W}_t^P = w, \mathcal{X}_t = (V_t^{i_1,j_1},V_t^{i_2,j_2},...,V_t^{i_n,j_n}))$. Please note that the 2^n queuing patterns $\mathcal{X}_t$ are mutually exclusive and represent an event space with total probability 1 defined for each priority level P and at each time t. The total *pmf* can be reconstructed from the compound *pmf*s:

$$f_{\mathcal{W}_t^P}(\cdot) = \sum_{\mathcal{X}_t \in S(P,t)} f_{(\mathcal{W}_t^P,\mathcal{X}_t)}(\cdot,\cdot) \tag{3.2}$$

Finally, in case the response time of a message instance [3] is larger than its period, we assume that message requests are processed in FIFO order. From a message instance standpoint, we need to consider the backlog due to message instances of the same message type as the one we are considering the backlog of. In fact, in case the response time is greater than the period, any message instance will be potentially delayed by instances from the same message and still in the queue. We are interested in computing the backlog $\mathcal{W}_t^{M_{i,j}}$ of an instance $M_{i,j}$ of m_i, where t is any time instant after its queuing time ($t \geq \mathcal{Q}_{i,j}$) and m_i is one of the actual messages (not the

[3] This is a real message, not the characterization message.

characterization abstractions).

The interference set of $M_{i,j}$ at time t, denoted as $I(M_{i,j}, t)$, is defined as the set of message instances that are queued at t and have priorities higher than P_i, which do not include the instances of m_i queued after $M_{i,j}$. We define $\mathcal{W}_t^{M_{i,j}}$ as the sum of the remaining transmission times of the queued message instances in its interference set. In case the response time is always lower than the period, obviously $\mathcal{W}_t^{M_{i,j}} = \mathcal{W}_t^{P_i}$, meaning there is no interference from past instances of the same message still in the queue.

3.2 Stochastic Analysis of the Approximate System

We approximate the continuous time model with a discrete time model with granularity τ such that all response times and backlogs are expressed as *pmf*s of time values that are multiples of the given tick unit τ. In order to compute the *pmf*s of the message response times in the approximate system, we first compute the stationary distribution of the backlog at the beginning of the hyperperiod; then, we compute the *pmf* of the backlog at the queuing time of each message instance, and finally, the *pmf* of the response time of each *message instance* within the hyperperiod. The message response time *pmf* is obtained by averaging the response time *pmf*s of all the instances in the hyperperiod.

3.2.1 Stationary Backlog within the Hyperperiod

The example in Figure 3.3 illustrates the method used to update the backlog at priority level P or higher for all instances of a generic message m_k. The characterization message instances, which represent the load from remote ECUs (such as $M_{i,j}$ in the figure), are queued with random

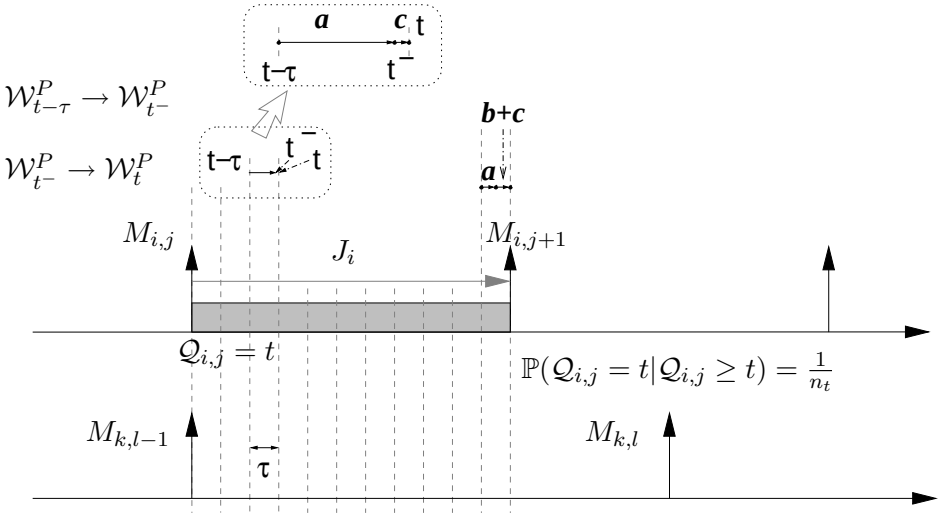

Figure 3.3: Updating the backlog in the discrete-time model

jitter in $[0, T_c)$ (in the figure $T_c = T_i$). Generally speaking, at any time t we compute the backlog knowing its value at $t - \tau$ and going through an intermediate step, which is an instant t^- arbitrarily close to t. Starting from the backlog at time $t - \tau$, we first update the backlog by shrinking (step **a** in the figure) to time t^-, right before the possible queuing time of message instance $M_{i,j} \in I(P, t)$. By shrinking, we advance the time, accumulating in the origin (backlog equal to zero) all the probabilities of non-positive backlogs. The probability that a message instance still to be activated is queued at t, $\mathbb{P}(\mathcal{Q}_{i,j} = t|\mathcal{Q}_{i,j} \geq t)$ can be easily computed (Figure 3.3) as $1/n_t$ where n_t is the number of ticks from t to the latest possible queuing time $q^{max}_{i,j}$ for $M_{i,j}$[4].

Finally, we provide the rules to compute the backlog from the time instant t^- to t in two possible scenarios, as a result of the possible (and non-deterministic) queuing of characterization messages from remote nodes at t. One possible scenario (step **c** in Figure 3.3) is when the set of message instances that can contribute to the backlog remains the same as that of the previous

[4]For all message instances $M_{i,j} \in I(P, t)$ thus $q^{max}_{i,j} \geq t$, $n_t > 0$

tick. The other case is when there is at least one new instance $M_{i,j+1}$ that must be considered in place of $M_{i,j}$. In this case, an additional step, labeled as **b** in Figure 3.3, must be performed after the shrinking and before considering the backlog update as the result of new messages being possibly queued. The following subsections describe the procedure for updating the backlog in the two cases. Section 3.2.1.1 describes the procedure for updating the backlog as a consequence of a possible queuing of a message instance, and Section 3.2.1.2 describes the update in case the set of message instances changes.

3.2.1.1 Updating the Backlog for a Possible Queuing of a Message Instance

We consider the compound *pmf*s of the P-level backlog and the possible queuing patterns $\overline{V}_t^{i,j}$ and $V_t^{i,j}$, that is, $f_{(\mathcal{W}_t^P, \overline{V}_t^{i,j})}$ and $f_{(\mathcal{W}_t^P, V_t^{i,j})}$.

First, we define $\mathcal{W}_t^P = \mathcal{W}_{t^-}^P + e = w$ where e is the possible contribution to the backlog by a message instance $M_{i,j}$. Then, we consider two scenarios:

- Scenario (1): $\mathcal{W}_t^P = w$ and $\overline{V}_t^{i,j} : (\mathcal{Q}_{i,j} \leq t)$, that is, the backlog at time t due to messages of priority higher or equal to P is w *and* the new message instance $M_{i,j}$ has been queued at time prior to or at t;

- Scenario (2): $\mathcal{W}_t^P = w$ and $V_t^{i,j} : (\mathcal{Q}_{i,j} > t)$, that is, the backlog at time t due to messages of priority higher or equal to P is w *and* the new message instance $M_{i,j}$ has been queued at any time greater than t, so $M_{i,j}$ has not contributed to the backlog at time t.

If (1) is true, then either $M_{i,j}$ contributed to the backlog at any time prior or equal to t^-, and in this case $\mathcal{W}_t^P = \mathcal{W}_{t^-}^P = w$ with $e = 0$, that is, $M_{i,j}$'s transmission time $\mathcal{E}_i = e$ has been already included in $\mathcal{W}_{t^-}^P$, or $M_{i,j}$ is queued at t, and the sum of its transmission time $\mathcal{E}_i = e$ and the

backlog at t^- must be equal to w. If (2) is true, then $M_{i,j}$ is queued later than t, and $\mathcal{E}_i$ is not part of the backlog at time t, therefore $\mathcal{W}_t^P = \mathcal{W}_{t^-}^P = w$.

The following theorem reformulates the backlog update at each time tick (τ).

Theorem 3.1. Consider a characterization message instance $M_{i,j}$, possibly queued at time t, with priority higher than P. $\mathcal{W}_{t^-}^P$ is the P-level backlog immediately before t, that is, before the possible queuing of $M_{i,j}$ at t. The probability that the backlog at time t equals a given value w can be computed with respect to the two mutually exclusive events $\overline{V}_t^{i,j}$ and $V_t^{i,j}$.

$$\begin{aligned}
& \mathbb{P}(\mathcal{W}_t^P = w, \overline{V}_t^{i,j}) \\
= \; & \mathbb{P}(\mathcal{W}_{t^-}^P = w, \overline{V}_{t^-}^{i,j}) + \sum_{e=0}^{w} \mathbb{P}(\mathcal{W}_{t^-}^P = w - e, V_{t^-}^{i,j}) \times \mathbb{P}(\mathcal{Q}_{i,j} = t | \mathcal{Q}_{i,j} \geq t) \times \mathbb{P}(\mathcal{E}_i = e) \\
& \mathbb{P}(\mathcal{W}_t^P = w, V_t^{i,j}) \\
= \; & \mathbb{P}(\mathcal{W}_{t^-}^P = w, V_{t^-}^{i,j}) \times \mathbb{P}(\mathcal{Q}_{i,j} > t | \mathcal{Q}_{i,j} \geq t) \\
= \; & \mathbb{P}(\mathcal{W}_{t^-}^P = w, V_{t^-}^{i,j}) \times (1 - \mathbb{P}(\mathcal{Q}_{i,j} = t | \mathcal{Q}_{i,j} \geq t))
\end{aligned} \tag{3.3}$$

In terms of probability mass functions, the computation of the compound *pmf*s of the P-level backlog with respect to the two events $\overline{V}_t^{i,j}$ and $V_t^{i,j}$ can be performed as follows ($f_{\mathcal{V}_1} \bigotimes f_{\mathcal{V}_2}$ denotes the convolution of $f_{\mathcal{V}_1}$ and $f_{\mathcal{V}_2}$):

$$\begin{aligned}
f_{(\mathcal{W}_t^P, \overline{V}_t^{i,j})} = \; & f_{(\mathcal{W}_{t^-}^P, \overline{V}_{t^-}^{i,j})} + \mathbb{P}(\mathcal{Q}_{i,j} = t | \mathcal{Q}_{i,j} \geq t) \times (f_{(\mathcal{W}_{t^-}^P, V_{t^-}^{i,j})} \bigotimes f_{\mathcal{E}_i}) \\
f_{(\mathcal{W}_t^P, V_t^{i,j})} = \; & (1 - \mathbb{P}(\mathcal{Q}_{i,j} = t | \mathcal{Q}_{i,j} \geq t)) \times f_{(\mathcal{W}_{t^-}^P, V_{t^-}^{i,j})}
\end{aligned} \tag{3.4}$$

Proof. Consider two events $(\mathcal{W}_{t^-}^P = w)$ and $(\mathcal{Q}_{i,j} = t')$ for any $t' \geq t$, which are independent from each other under the condition $\mathcal{Q}_{i,j} > t^-$, because $\mathcal{W}_{t^-}^P$ is independent from the queuing of any message instance after or at t.

Thus $\forall t' \geq t$, and $\forall w \geq 0$

$$\begin{aligned} & \mathbb{P}(\mathcal{W}_{t^-}^P = w, \mathcal{Q}_{i,j} = t') \\ = \ & \mathbb{P}(\mathcal{W}_{t^-}^P = w, \mathcal{Q}_{i,j} = t', \mathcal{Q}_{i,j} \geq t) \\ = \ & \mathbb{P}(\mathcal{W}_{t^-}^P = w, \mathcal{Q}_{i,j} = t' | \mathcal{Q}_{i,j} \geq t) \times \mathbb{P}(\mathcal{Q}_{i,j} \geq t) \\ = \ & \mathbb{P}(\mathcal{W}_{t^-}^P = w | \mathcal{Q}_{i,j} \geq t) \times \mathbb{P}(\mathcal{Q}_{i,j} = t' | \mathcal{Q}_{i,j} \geq t) \times \mathbb{P}(\mathcal{Q}_{i,j} \geq t) \\ = \ & \mathbb{P}(\mathcal{W}_{t^-}^P = w, \mathcal{Q}_{i,j} \geq t) \times \mathbb{P}(\mathcal{Q}_{i,j} = t' | \mathcal{Q}_{i,j} \geq t) \\ = \ & \mathbb{P}(\mathcal{W}_{t^-}^P = w, \mathcal{Q}_{i,j} > t^-) \times \mathbb{P}(\mathcal{Q}_{i,j} = t' | \mathcal{Q}_{i,j} \geq t) \end{aligned} \tag{3.5}$$

In particular, when $t' = t$,

$$\begin{aligned} & \mathbb{P}(\mathcal{W}_{t^-}^P = w, \mathcal{Q}_{i,j} = t) \\ = \ & \mathbb{P}(\mathcal{W}_{t^-}^P = w, \mathcal{Q}_{i,j} > t^-) \times \mathbb{P}(\mathcal{Q}_{i,j} = t | \mathcal{Q}_{i,j} \geq t) \end{aligned} \tag{3.6}$$

and

$$\begin{aligned} & \mathbb{P}(\mathcal{W}_{t^-}^P = w, \mathcal{Q}_{i,j} > t) \\ = \ & \mathbb{P}(\mathcal{W}_{t^-}^P = w, \mathcal{Q}_{i,j} \geq t) - \mathbb{P}(\mathcal{W}_{t^-}^P = w, \mathcal{Q}_{i,j} = t) \\ = \ & \mathbb{P}(\mathcal{W}_{t^-}^P = w, \mathcal{Q}_{i,j} \geq t) \times (1 - \mathbb{P}(\mathcal{Q}_{i,j} = t | \mathcal{Q}_{i,j} \geq t)) \\ = \ & \mathbb{P}(\mathcal{W}_{t^-}^P = w, \mathcal{Q}_{i,j} > t^-) \times (1 - \mathbb{P}(\mathcal{Q}_{i,j} = t | \mathcal{Q}_{i,j} \geq t)) \end{aligned} \tag{3.7}$$

Consider the probability of event ($\mathcal{W}_t^P = w, \mathcal{Q}_{i,j} \leq t$), there are two possibilities that are mutually exclusive:

- $M_{i,j}$ is queued before t, and $\mathcal{W}_{t^-}^P = w$;
- $M_{i,j}$ is queued at t, and $\mathcal{W}_{t^-}^P$ and the transmission time from $M_{i,j}$ $\mathcal{E}_i$ sum up to w.

Thus

$$
\begin{aligned}
& \mathbb{P}(\mathcal{W}_t^P = w, \mathcal{Q}_{i,j} \leq t) \\
= \;& \mathbb{P}(\mathcal{W}_{t^-}^P = w, \mathcal{Q}_{i,j} \leq t^-) + \sum_{e=0}^{w} \mathbb{P}(\mathcal{W}_{t^-}^P = w - e, \mathcal{Q}_{i,j} = t, \mathcal{E}_i = e) \\
= \;& \mathbb{P}(\mathcal{W}_{t^-}^P = w, \mathcal{Q}_{i,j} \leq t^-) + \sum_{e=0}^{w} \mathbb{P}(\mathcal{W}_{t^-}^P = w - e, \mathcal{Q}_{i,j} = t) \times \mathbb{P}(\mathcal{E}_i = e) \\
= \;& \mathbb{P}(\mathcal{W}_{t^-}^P = w, \mathcal{Q}_{i,j} \leq t^-) \\
+ \;& \sum_{e=0}^{w} \mathbb{P}(\mathcal{W}_{t^-}^P = w - e, \mathcal{Q}_{i,j} > t^-) \times \mathbb{P}(\mathcal{Q}_{i,j} = t | \mathcal{Q}_{i,j} \geq t) \times \mathbb{P}(\mathcal{E}_i = e) \text{ (by Equation 3.6)}
\end{aligned}
\tag{3.8}
$$

Consider the probability of event ($\mathcal{W}_t^P = w, \mathcal{Q}_{i,j} > t$), the only possibility is

- $\mathcal{W}_{t^-}^P = w$ and $M_{i,j}$ is queued after t.

Thus

$$
\begin{aligned}
& \mathbb{P}(\mathcal{W}_t^P = w, \mathcal{Q}_{i,j} > t) \\
= \;& \mathbb{P}(\mathcal{W}_{t^-}^P = w, \mathcal{Q}_{i,j} > t) \\
= \;& \mathbb{P}(\mathcal{W}_{t^-}^P = w, \mathcal{Q}_{i,j} > t^-) \times (1 - \mathbb{P}(\mathcal{Q}_{i,j} = t | \mathcal{Q}_{i,j} \geq t)) \text{ (by Equation 3.7)}
\end{aligned}
\tag{3.9}
$$

□

In general, there may be more than one message instance queued at time t, therefore we need to consider the compound *pmf*s of the backlog with all the possible queuing patterns $\mathcal{X}_t \subset S(P,t)$. Using Equation 3.3, we compute the compound probability of the P-level backlog at t together with each pattern $\mathcal{X}_t$, after the possible queuing of a message instance $M_{i,j}$ with priority higher than or equal to P at t.

In case $\mathcal{X}_t$ is a negative pattern of $V_t^{i,j}$, the *pmf* $f_{(\mathcal{W}_t^P, \mathcal{X}_t)}$ can be obtained by computing the convolution of the compound *pmf* defined with respect to the complementary queuing pattern

$\overline{\mathcal{X}}_{t^-(i,j)}$, $f_{(\mathcal{W}^P_{t^-},\overline{\mathcal{X}}_{t^-(i,j)})}$, with the *pmf* of the transmission time of $M_{i,j}$, $f_{\mathcal{E}_i}$, multiplied by $\mathbb{P}(\mathcal{Q}_{i,j} = t|\mathcal{Q}_{i,j} \geq t)$, and adding the result to the compound *pmf* $f_{(\mathcal{W}^P_{t^-},\mathcal{X}_{t^-})}$.

In case $\mathcal{X}_t$ is a positive pattern of $V_t^{i,j}$, the *pmf* $f_{(\mathcal{W}^P_t,\mathcal{X}_t)}$ is obtained by multiplying $f_{(\mathcal{W}^P_{t^-},\mathcal{X}_{t^-})}$ with the scalar $(1 - \mathbb{P}(\mathcal{Q}_{i,j} = t|\mathcal{Q}_{i,j} \geq t))$.

This operation is called *cross-convolution* since the update of the compound *pmf* $f_{(\mathcal{W}^P_t,\mathcal{X}_t)}$ for some negative pattern $\mathcal{X}_t$ of $V_t^{i,j}$ is performed by adding a term resulting from the convolution of the message transmission time *pmf* with the compound *pmf* of the backlog and the complementary queuing pattern $\overline{\mathcal{X}}_{t^-(i,j)}$ with respect to $M_{i,j}$. The procedure is formally described in Algorithm 6.

Algorithm 6 Cross-convolution: calculate P-level backlog after the possible queuing of $M_{i,j}$ at time t

1: **for** each **negative** queuing pattern $\overline{\mathcal{X}}_t \in S(P,t)$ of $V_t^{i,j}$ **do**

2: $\mathcal{X}_{t(i,j)}$ = complementary queuing pattern of $\overline{\mathcal{X}}_t$ with respect to $M_{i,j}$

3: $f_{(\mathcal{W}^P_t,\overline{\mathcal{X}}_t)} = f_{(\mathcal{W}^P_{t^-},\overline{\mathcal{X}}_{t^-})} + \mathbb{P}(\mathcal{Q}_{i,j} = t|\mathcal{Q}_{i,j} \geq t) \times (f_{(\mathcal{W}^P_{t^-},\mathcal{X}_{t^-(i,j)})} \bigotimes f_{\mathcal{E}_i})$

4: **end for**

5: **for** each **positive** queuing pattern $\mathcal{X}_t \in S(P,t)$ of $V_t^{i,j}$ **do**

6: $f_{(\mathcal{W}^P_t,\mathcal{X}_t)} = (1 - \mathbb{P}(\mathcal{Q}_{i,j} = t|\mathcal{Q}_{i,j} \geq t)) \times f_{(\mathcal{W}^P_{t^-},\mathcal{X}_{t^-})}$

7: **end for**

3.2.1.2 Queuing Pattern Update

After considering the possible queuing of message instances with priority higher than or equal to P, the P-level backlog *pmf* along with each queuing pattern $\mathcal{X}_t$, i.e., $f_{(\mathcal{W}^P_t,\mathcal{X}_t)}$, is computed, and the backlog at $t' = (t+\tau)^- < (t+\tau)$ is updated by *shrinking* the *pmf*. We keep applying shrinking and cross-convolution to update the backlog at each discrete instant (adding τ). This

iterative process is repeated at each time tick because of the non-deterministic random queuing of the remote messages.

A special case occurs at those time instants when the sets of P-level message instances change. When a characterization message instance $M_{i,j}$ is replaced by $M_{i,j+1}$ (Figure 3.4), $V_{t^-}^{i,j}$ and $\overline{V}_{t^-}^{i,j}$, included in every queuing pattern $\mathcal{X}_t$, must be replaced by two new queuing events $V_{t^-}^{i,j+1}$ and $\overline{V}_{t^-}^{i,j+1}$.

Suppose t is the time instant when $M_{i,j+1}$ replaces $M_{i,j}$. At t^-, it is clear that $\mathbb{P}(\overline{V}_{t^-}^{i,j}) = 1$ and $\mathbb{P}(V_{t^-}^{i,j}) = 0$. A new set of patterns $\mathcal{X}_{t^-}$ must be defined at time t^-, where $V_{t^-}^{i,j}$ and $\overline{V}_{t^-}^{i,j}$ are replaced by $V_{t^-}^{i,j+1}$ and $\overline{V}_{t^-}^{i,j+1}$. The compound *pmf*s $f_{(\mathcal{W}_{t^-}^P, \mathcal{X}_{t^-})}$ for the new patterns can be defined starting from the consideration that $\mathbb{P}(\overline{V}_{t^-}^{i,j+1}) = \mathbb{P}(V_{t^-}^{i,j}) = 0$ and $\mathbb{P}(V_{t^-}^{i,j+1}) = \mathbb{P}(\overline{V}_{t^-}^{i,j}) = 1$. Hence, the *pmf* $f_{(\mathcal{W}_{t^-}^P, \mathcal{X}'_{t^-})}$ for each queuing pattern $\mathcal{X}'_{t^-}$ that includes $\overline{V}_{t^-}^{i,j+1}$, that is, $\mathcal{X}'_{t^-} = \{\ldots, \overline{V}_{t^-}^{i,j+1}, \ldots\}$ can be defined as $\mathbb{P}(\mathcal{W}_{t^-}^P = w, \mathcal{X}'_{t^-}) = 0$ for any value $w \geq 0$. The values of $f_{(\mathcal{W}_{t^-}^P, \mathcal{X}''_{t^-})}$ for the other patterns that include $V_{t^-}^{i,j+1}$ can be obtained by setting $f_{(\mathcal{W}_{t^-}^P, \mathcal{X}''_{t^-})} = f_{(\mathcal{W}_{t^-}^P, \mathcal{X}^*_{t^-})}$ where all the elements of $\mathcal{X}^*_{t^-}$ match the corresponding elements of $\mathcal{X}''_{t^-}$, except that $\overline{V}_{t^-}^{i,j}$ is replaced in $\mathcal{X}''_{t^-}$ by $V_{t^-}^{i,j+1}$.

Thus, given the stationary P-level backlog *pmf* at the beginning of the hyperperiod, the stationary P-level backlog *pmf* at any time in the hyperperiod can be computed by iteratively using the operations of shrinking, queuing pattern update (possibly) and cross-convolution. Formally, there is no guarantee that there exists a stationary distribution for our approximate system with characterization messages, but the existence of such a stationary solution for a particular system configuration can be verified experimentally, by computing the conditional backlogs at each time tick and stopping when identical values are found after an interval of exactly one hyperperiod.

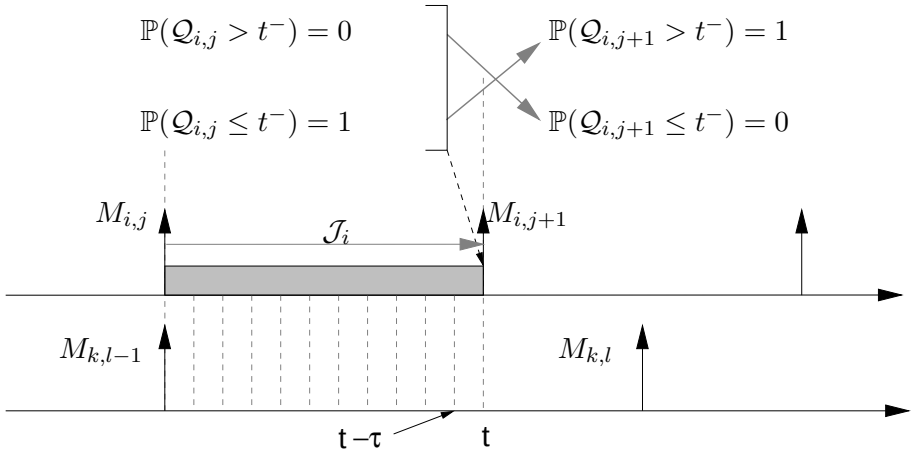

Figure 3.4: Updating the backlog when message instances change

For each of the CAN systems in our experiments, a stationary distribution is found after just one hyperperiod.

3.2.2 Initial Blocking Time

Besides the delay from higher priority messages, a message in general is also subject to blocking delay from lower priority messages due to the non-preemptive message transmission in CAN protocol. The computation of the possible initial blocking time is performed by using a very simple model. Given a message instance $M_{i,j}$, we consider the set of all the lower priority messages, denoted as $lp(P_i) = \{m_k | P_k > P_i\}$. Their message instances cause a blocking delay to $M_{i,j}$ if they are being transmitted at the time $M_{i,j}$ is queued. If we assume that these instances are evenly distributed in the hyperperiod, the probability that the message instance $M_{i,j}$ waits for a time $b > 0$ (because of a lower priority message m_k with transmission time $\mathcal{E}_k$ being transmitted on the bus at

least one time instant before the queuing time of $M_{i,j}$) is

$$\mathbb{P}(\mathcal{B}_{i,j} = b) = \sum_{m_k \in lp(P_i)} \frac{\mathbb{P}(\mathcal{E}_k > b)}{T_k/\tau} \tag{3.10}$$

Given these probabilities, the *pmf* $f_{\mathcal{B}_{i,j}}$ of the blocking time $\mathcal{B}_{i,j}$ can be easily computed. The initial blocking time must be added to the backlog at the time a message is queued by performing a convolution of the two *pmf*s.

3.2.3 Message Response Time Calculation

The computation of the response time of a message instance $M_{i,j}$ with priority P_i, requires, as a first step, the evaluation of its backlog at its queuing time.

When computing the response time of each instance $M_{i,j}$ of m_i, we assume it is queued periodically with no jitter. The other local messages are queued periodically, and the remote messages are represented by their P_i-level characterization messages queued periodically with jitter. Hence, the queuing time $\mathcal{Q}_{i,j}$ of $M_{i,j}$ is deterministically equal to the arrival time, say as $q_{i,j}$. The backlog can be computed by applying the methods for updating $f_{(\mathcal{W}_t^{M_{i,j}}, \mathcal{X}_t)}$ defined in the previous sections and then adding them (Equation 3.2).

In addition, the backlog at the queuing time of a message instance is updated in order to account for a possible initial blocking delay, as described in Section 3.2.2.

The earliest possible start time for $M_{i,j}$ is its queuing time, i.e., $\mathcal{S}_{i,j} \geq q_{i,j}$. Starting from $t = q_{i,j}$ and, considering the following times $t_k = t + k\tau$ ($k \in N^+$), we further decompose $f_{(\mathcal{W}_{t_k}^{M_{i,j}}, \mathcal{X}_{t_k})}$ into the compound *pmf*s $f_{(\mathcal{W}_{t_k}^{M_{i,j}}, \mathcal{S}_{i,j} < t_k, \mathcal{X}_{t_k})}$ and $f_{(\mathcal{W}_{t_k}^{M_{i,j}}, \mathcal{S}_{i,j} \geq t_k, \mathcal{X}_{t_k})}$, and the latter in $f_{(\mathcal{W}_{t_k}^{M_{i,j}}, \mathcal{S}_{i,j} = t_k, \mathcal{X}_{t_k})}$ and $f_{(\mathcal{W}_{t_k}^{M_{i,j}}, \mathcal{S}_{i,j} > t_k, \mathcal{X}_{t_k})}$.

For $k = 0$, at time $t_0 = t = q_{i,j}$, and for each queuing pattern $\mathcal{X}_t \in S(M_{i,j}, t)$, clearly

$f_{(\mathcal{W}_t^{M_{i,j}},\mathcal{S}_{i,j}\geq t,\mathcal{X}_t)} = f_{(\mathcal{W}_t^{M_{i,j}},\mathcal{X}_t)}$ (the start time is no earlier than the queuing time).

The probability that $M_{i,j}$ starts transmission immediately, that is, $\mathbb{P}(\mathcal{S}_{i,j} = q_{i,j} = t)$ equals the probability that the backlog of $M_{i,j}$ at time t is zero, i.e., $\mathbb{P}(\mathcal{W}_t^{M_{i,j}} = 0)$.

$$\mathbb{P}(\mathcal{S}_{i,j} = t) = \mathbb{P}(\mathcal{W}_t^{M_{i,j}} = 0) = \sum_{\mathcal{X}_t} \mathbb{P}(\mathcal{W}_t^{M_{i,j}} = 0, \mathcal{S}_{i,j} \geq t, \mathcal{X}_t) \tag{3.11}$$

Given that $\mathcal{W}_t^{M_{i,j}} = 0$ implies $\mathcal{S}_{i,j} = t$, and $\mathcal{W}_t^{M_{i,j}} = w$ with $w > 0$ implies $\mathcal{S}_{i,j} > t$, we have

$$\begin{cases} \mathbb{P}(\mathcal{W}_t^{M_{i,j}} = 0, \mathcal{S}_{i,j} > t) = 0 \\ \mathbb{P}(\mathcal{W}_t^{M_{i,j}} = w, \mathcal{S}_{i,j} > t) = \mathbb{P}(\mathcal{W}_t^{M_{i,j}} = w, \mathcal{S}_{i,j} \geq t), \forall w > 0 \end{cases} \tag{3.12}$$

which means that $f_{(\mathcal{W}_t^{M_{i,j}},\mathcal{S}_{i,j}>t,\mathcal{X}_t)}(0) = 0$, and, for all $w > 0$, it is $f_{(\mathcal{W}_t^{M_{i,j}},\mathcal{S}_{i,j}>t,\mathcal{X}_t)}(w) = f_{(\mathcal{W}_t^{M_{i,j}},\mathcal{S}_{i,j}\geq t,\mathcal{X}_t)}(w)$ for each $\mathcal{X}_t \in S(M_{i,j}, t)$. As the probability that the backlog is equal to zero when $\mathcal{S}_{i,j} > t$ is zero, then

$$\sum_{w} \mathbb{P}(\mathcal{W}_t^{M_{i,j}} = w, \mathcal{S}_{i,j} > t) = 1 - \mathbb{P}(\mathcal{S}_{i,j} = t) \tag{3.13}$$

The probability that $M_{i,j}$ starts transmission at the next time instants $t_k = t + k\tau$ can be computed by iteratively applying the following steps, starting from t.

Starting from $f_{(\mathcal{W}_t^{M_{i,j}},\mathcal{S}_{i,j}>t,\mathcal{X}_t)}$, the compound *pmf*s of the backlog can be updated to the next time point $t_1 = t + \tau$ by applying the methods defined in Section 3.2.1. The result is the backlog at t_1, $f_{(\mathcal{W}_{t_1}^{M_{i,j}},\mathcal{S}_{i,j}>t)}$. In our discrete system, $\mathcal{S}_{i,j} > t$ is the same as $\mathcal{S}_{i,j} \geq t_1$, hence, $f_{(\mathcal{W}_{t_1}^{M_{i,j}},\mathcal{S}_{i,j}\geq t_1)} = f_{(\mathcal{W}_{t_1}^{M_{i,j}},\mathcal{S}_{i,j}>t)}$. Since the operations of cross-convolution, shrinking and queuing pattern update do not change the total probability of the backlog, we have

$$\sum_{w} \mathbb{P}(\mathcal{W}_{t_1}^{M_{i,j}} = w, \mathcal{S}_{i,j} \geq t_1) = 1 - \mathbb{P}(\mathcal{S}_{i,j} = t) \tag{3.14}$$

The probability that $M_{i,j}$ starts transmission at time t_1 equals the probability that its backlog is zero at t_1

$$\mathbb{P}(\mathcal{S}_{i,j} = t_1) = \mathbb{P}(\mathcal{W}_{t_1}^{M_{i,j}} = 0) = \sum_{\mathcal{X}_{t_1}} \mathbb{P}(\mathcal{W}_{t_1}^{M_{i,j}} = 0, \mathcal{S}_{i,j} \geq t_1, \mathcal{X}_{t_1}) \tag{3.15}$$

Similar to time instant t, we compute the compound *pmf* for the case $\mathcal{S}_{i,j} > t_1$ as $f_{(\mathcal{W}_{t_1}^{M_{i,j}}, \mathcal{S}_{i,j} > t_1, \mathcal{X}_{t_1})}(0) = 0$, and $\forall w > 0, f_{(\mathcal{W}_{t_1}^{M_{i,j}}, \mathcal{S}_{i,j} > t_1, \mathcal{X}_{t_1})}(w) = f_{(\mathcal{W}_{t_1}^{M_{i,j}}, \mathcal{S}_{i,j} \geq t_1, \mathcal{X}_{t_1})}(w)$. Iterating Equation 3.13, the total probability of $f_{(\mathcal{W}_{t_1}^{M_{i,j}}, \mathcal{S}_{i,j} > t_1)}$ is

$$\sum_{w} \mathbb{P}(\mathcal{W}_{t_1}^{M_{i,j}} = w, \mathcal{S}_{i,j} > t_1) = 1 - \sum_{t_k \leq t_1} \mathbb{P}(\mathcal{S}_{i,j} = t_k) \tag{3.16}$$

Thus we iteratively apply the above two steps for the time instants $t_k = t + k\tau$ until we get to an index n such that the compound backlog is identically null.

$$\forall \mathcal{X}_n \in S(M_{i,j}, t_n), \forall w > 0, \mathbb{P}(\mathcal{W}_{t_n}^{M_{i,j}} = w, \mathcal{S}_{i,j} \geq t_n, \mathcal{X}_n) = 0 \tag{3.17}$$

This happens when, at time n, the total probability of the start time *pmf* accumulates to 1.

$$\sum_{k \leq n} \mathbb{P}(\mathcal{S}_{i,j} = t_k) = 1 \tag{3.18}$$

This procedure of calculating message start time *pmf* is formally described in Algorithm 7.

Algorithm 7 Calculate the start time *pmf* of $M_{i,j}$

$t = q_{i,j}$

for each queuing pattern $\mathcal{X}_t \in S(M_{i,j}, t)$ **do**

$f_{(\mathcal{W}_t^{M_{i,j}}, \mathcal{S}_{i,j} \geq t, \mathcal{X}_t)} = f_{(\mathcal{W}_{t^-}^{P_i}, \mathcal{X}_t)}$

end for

$\forall k \geq q_{i,j}, \mathbb{P}(\mathcal{S}_{i,j} = k) = 0$

while $\sum_{k=q_{i,j}}^{t} \mathbb{P}(\mathcal{S}_{i,j} = k) < 1$ **do**

  for each queuing pattern $\mathcal{X}_t \in S(M_{i,j}, t)$ **do**

    $\mathbb{P}(\mathcal{S}_{i,j} = t) += \mathbb{P}(\mathcal{W}_t^{M_{i,j}} = 0, \mathcal{S}_{i,j} \geq t, \mathcal{X}_t)$

    $\forall w > 0, \mathbb{P}(\mathcal{W}_t^{M_{i,j}} = w, \mathcal{S}_{i,j} > t, \mathcal{X}_t) = \mathbb{P}(\mathcal{W}_t^{M_{i,j}} = w, \mathcal{S}_{i,j} \geq t, \mathcal{X}_t)$

  end for

  Update $f_{(\mathcal{W}_t^{M_{i,j}}, \mathcal{S}_{i,j} > t)}$ to get $f_{(\mathcal{W}_{t+\tau}^{M_{i,j}}, \mathcal{S}_{i,j} > t)}$, i.e., $f_{(\mathcal{W}_{t+\tau}^{M_{i,j}}, \mathcal{S}_{i,j} \geq t+\tau)}$

  $t = t + \tau$

end while

Once the *pmf* of the time $\mathcal{S}_{i,j}$ at which $M_{i,j}$ starts its transmission is known, the *pmf* of its finish time (the time at which its transmission terminates) is obtained by simply adding its transmission time $\mathcal{E}_i$, i.e., performing a convolution on the *pmf*s of $\mathcal{S}_{i,j}$ and $\mathcal{E}_i$. The response time is finally computed by a further left shift of $q_{i,j}$ time units.

$$\begin{aligned} f_{\mathcal{F}_{i,j}} &= f_{\mathcal{S}_{i,j}} \bigotimes f_{\mathcal{E}_i} \\ \mathbb{P}(\mathcal{R}_{i,j} = t - q_{i,j}) &= \mathbb{P}(\mathcal{F}_{i,j} = t), \forall t \end{aligned} \tag{3.19}$$

Finally, the response time *pmf* $f_{\mathcal{R}_i}$ of message m_i is obtained as the arithmetic average of the *pmf*s $f_{\mathcal{R}_{i,j}}$ of its instances $M_{i,j}$ inside the hyperperiod.

3.2.4 Algorithm Complexity

This section analyzes the worst case complexity of the algorithm to compute the backlog of level i for message instance $M_{i,j}$ on node ECU_n. The algorithm refers to the computations in one hyperperiod and requires at each step the evaluation of cross convolution and shrinking. Let n be the number of nodes in the system. Within each hyperperiod, there are $(n-1)H/\tau + \sum_{k \in hp(i,n)} H/T_k$ possible queuing events of higher priority message instances (the number is $O((n-1)H/\tau)$ since $T_k \gg \tau$). A backlog update requires updating 2^{n-1} compound backlogs, 2^{n-2} of which along with positive queuing patterns. Each of these compound backlogs requires considering, at each step, each possible queuing of a message instances by performing a cross-convolution. The complexity of convolution depends on the algorithm used and on the worst case length of the distributions. In our analysis, the maximum execution time of each characterization message and the compound backlogs are less than R_i^{max}. To perform a convolution on two vectors f and g, with size n_f and n_g

respectively, the number of operations required is $O(n_f n_g)$ [5]. Thus the complexity of calculation within one hyperperiod is $O(2^{n-2} \times (R_i^{max}/\tau)^2 \times (n-1)H/\tau) = O(\frac{n2^n(R_i^{max})^2 H}{\tau^3})$. This is a worst case bound, the algorithm is typically much faster in the average case.

3.3 Experimental Results

In this section, we present some experimental results on a set of messages from experimental vehicles. The test set consists of 6 ECUs and 69 messages as shown in Table 3.1. The bus rate is $500kbps$, and the message transmission times are computed according to the bit length of the actual messages. Please note that, as opposed to the SAE and PSA benchmarks documented in [30] and [46] consisting of 17 and 12 messages respectively, this set is a more real example of an actual automotive message set. Furthermore, as opposed to the benchmarks in [30] and [46], a middleware task running at the *gcd* of the message periods is used to queue messages. This implementation is very real in actual automotive implementations. In Table 3.1, T_i is the period, P_i is the priority, and $\mathcal{E}_i$ is the transmission time, where both T_i and $\mathcal{E}_i$ are in milliseconds. The maximum bus utilization of the system is 60.25%. *For simplicity, in the experiment we assume worst case bit stuffing thus deterministic transmission times.*

We check the quality of our stochastic analysis method by comparing its results with the results of simulations when the size of the system still allows a good coverage for a simulation-based approach. To have a good trade-off between the number of possible phase combinations and the accuracy of the analysis and simulation, the granularity of our analysis, as defined by τ, is set to $0.01ms$, that is, the gcd of the message maximum transmission times. It takes less than 2 hours to

[5] The operation of convolution can be performed faster using Fast Fourier Transform algorithm.

Msg	ECU	T_i	$\mathcal{E}_i$	P_i	Msg	ECU	T_i	$\mathcal{E}_i$	P_i	Msg	ECU	T_i	$\mathcal{E}_i$	P_i
m_1	E_2	10	0.27	1	m_{24}	E_3	25	0.23	24	m_{47}	E_2	50	0.19	47
m_2	E_2	10	0.27	2	m_{25}	E_3	25	0.25	25	m_{48}	E_4	100	0.27	48
m_3	E_3	5	0.19	3	m_{26}	E_2	20	0.27	26	m_{49}	E_3	100	0.27	49
m_4	E_3	10	0.25	4	m_{27}	E_5	25	0.27	27	m_{50}	E_1	100	0.27	50
m_5	E_1	10	0.19	5	m_{28}	E_2	20	0.27	28	m_{51}	E_3	100	0.13	51
m_6	E_6	10	0.27	6	m_{29}	E_1	25	0.17	29	m_{52}	E_3	100	0.27	52
m_7	E_1	100	0.27	7	m_{30}	E_2	10	0.21	30	m_{53}	E_1	100	0.19	53
m_8	E_1	100	0.15	8	m_{31}	E_2	20	0.27	31	m_{54}	E_3	100	0.13	54
m_9	E_1	100	0.17	9	m_{32}	E_4	10	0.27	32	m_{55}	E_3	100	0.13	55
m_{10}	E_5	25	0.27	10	m_{33}	E_3	10	0.27	33	m_{56}	E_5	100	0.27	56
m_{11}	E_1	100	0.15	11	m_{34}	E_3	10	0.27	34	m_{57}	E_3	100	0.25	57
m_{12}	E_1	20	0.19	12	m_{35}	E_1	25	0.25	35	m_{58}	E_3	100	0.13	58
m_{13}	E_2	100	0.19	13	m_{36}	E_6	25	0.23	36	m_{59}	E_3	100	0.13	59
m_{14}	E_1	100	0.19	14	m_{37}	E_4	50	0.27	37	m_{60}	E_4	100	0.17	60
m_{15}	E_5	100	0.27	15	m_{38}	E_4	50	0.27	38	m_{61}	E_1	100	0.27	61
m_{16}	E_1	10	0.23	16	m_{39}	E_5	50	0.27	39	m_{62}	E_1	100	0.13	62
m_{17}	E_2	100	0.25	17	m_{40}	E_3	50	0.21	40	m_{63}	E_3	100	0.19	63
m_{18}	E_2	100	0.27	18	m_{41}	E_4	50	0.27	41	m_{64}	E_1	100	0.27	64
m_{19}	E_2	50	0.25	19	m_{42}	E_6	50	0.27	42	m_{65}	E_1	100	0.13	65
m_{20}	E_3	10	0.27	20	m_{43}	E_2	50	0.27	43	m_{66}	E_1	100	0.13	66
m_{21}	E_6	10	0.27	21	m_{44}	E_5	100	0.27	44	m_{67}	E_2	50	0.27	67
m_{22}	E_6	25	0.27	22	m_{45}	E_2	25	0.15	45	m_{68}	E_1	100	0.13	68
m_{23}	E_3	25	0.23	23	m_{46}	E_2	50	0.19	46	m_{69}	E_4	100	0.27	69

Table 3.1: An example automotive CAN system with 6 ECUs and 69 messages

analyze all the messages, with a maximum of 19 minutes for each. The simulation is performed with a granularity of $0.05ms$, leading to $2000^5 = 3.2 \times 10^{18}$ possible combinations of relative phases among nodes. This space cannot be explored exhaustively. Hence, the simulation is performed by randomly choosing relative phases, and repeated 2×10^{10} times, which takes about 100 hours.

The stationary distribution of backlogs is obtained by iterative approximation, that is, by first applying the algorithm that computes the backlogs, then stopping when the *pmf*s of backlogs are close enough at the end of two consecutive hyperperiods. The stationary distribution of the message response time is calculated as the average of the instances queued in the next hyperperiod once the stationary distribution of backlogs is achieved.

We show the results for four representative messages, one with relatively high priority, one with medium priority, one with medium low priority, and the other with low priority, and compare the results of the stochastic analysis, simulation, and the worst case analysis based upon [13].

For the first representative message, the worst case response time is shorter than half of the *gcd* of the periods ($2.5ms$). Hence, the analysis is very accurate as the actual backlog does not include any multiple message instances from any remote characterization message. As expected (Figure 3.5), the *cumulative distribution function*s (*cdf*s) obtained from the analysis and the simulation match very well.

In Figure 3.5, the analysis of the response time of the high priority message m_5 has been tentatively performed without the additional contribution to the backlog caused by the blocking factor. The result is quite optimistic with respect to the actual values and shows the accuracy in the modeling of this term.

For the second representative m_{25} where the worst case response time is slightly larger

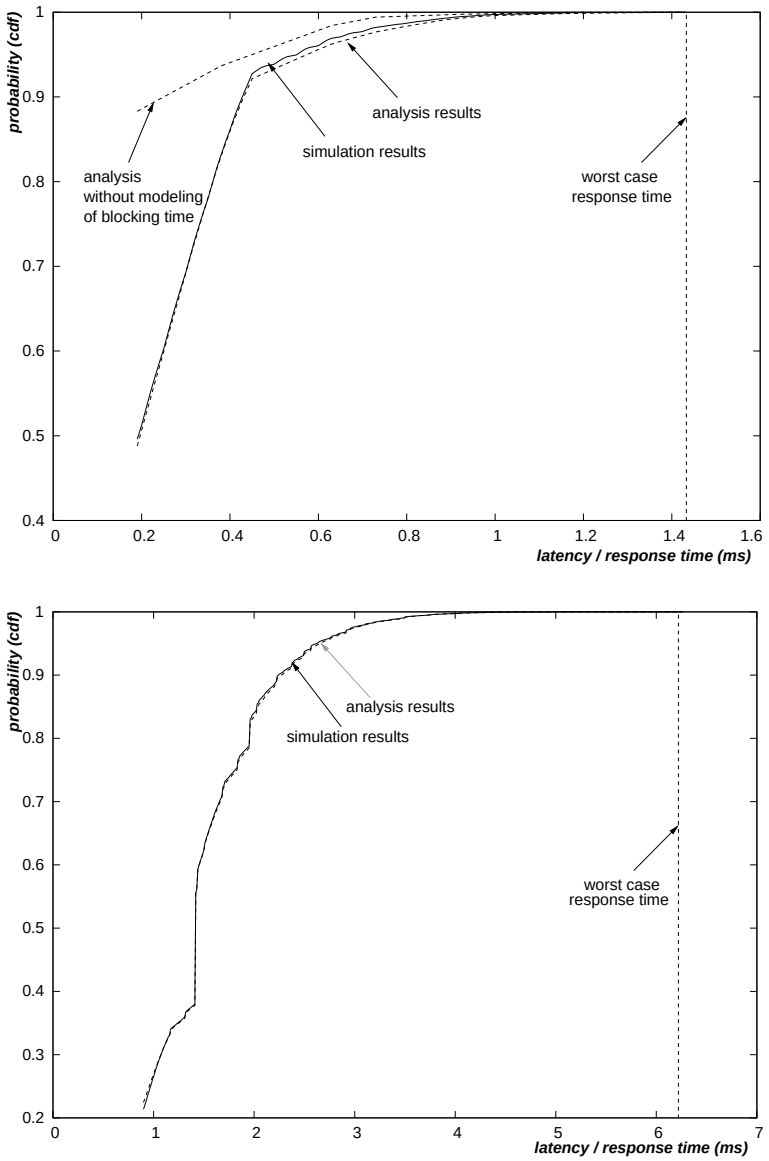

Figure 3.5: The response time *cdf*s of two high priority messages (m_5 and m_{25}) in the test set

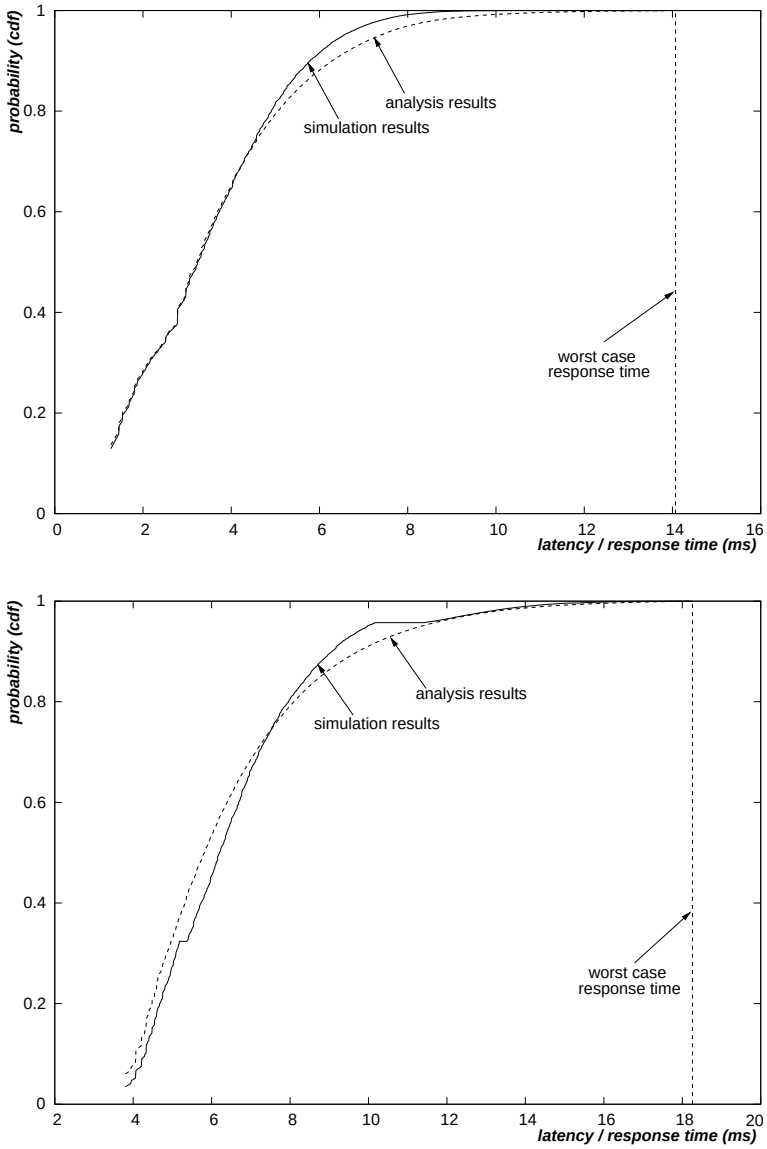

Figure 3.6: The response time *cdf*s of two low priority messages (m_{43} and m_{63}) in the test set

than the *gcd* of the periods ($5ms$), the analysis is still pretty accurate. But it starts to deviate from the simulation when the worst case response time grows longer. Many messages, including m_{43} and m_{63} have worst case response times significantly larger than the *gcd* of the periods. Hence, we expect a significantly reduced accuracy. However, the distribution obtained from the analysis is still a good match to that of the simulation. For low priority messages, mainly delayed by interferences, the probabilities computed by the stochastic analysis for long response times are larger than those computed by the simulation as the random queuing times of the characterization messages are pessimistic when considering the interferences due to higher priority messages (multiple instances of the same higher priority message contributing to the interference are possibly queued in two consecutive instances of the characterization message in the approximate system).

Furthermore, as we can see in Figure 3.6 for each of the representative messages, the cumulative probability approaches 100% probability at response time values significantly smaller than the worst case value. Please note that in most cases, the messages of greater importance for the response time analysis are the higher priority ones, for which the worst case response times are smaller and the analysis is more accurate.

Finally, Figure 3.7 shows the results of the message response time analysis of a complex message trace extracted from an automotive bus. The measured response time *cdf* of a low priority message (CAN id $>$ 0x500) is compared with the response time estimated by the stochastic analysis. The analysis is accurate despite the large worst case response time of the message and the non-idealities of the actual implementation (possible priority inversions at the network adapter and finite copy times between the message queues and the TxObjects).

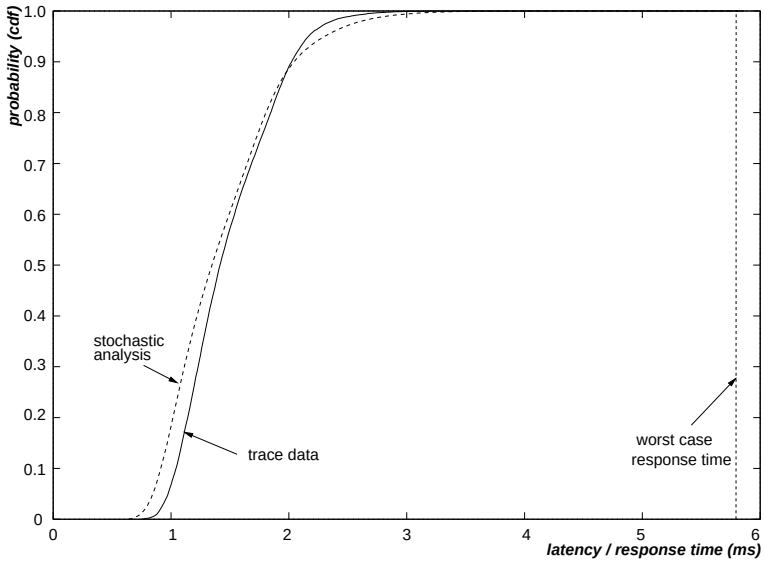

Figure 3.7: The response time *cdf* of a low priority message in a bus trace compared with analysis estimates

Chapter 4

Stochastic Analysis of End-to-end Latency of Periodic Tasks/Messages

In this chapter we provide the theory for stochastic analysis of the latency in the end-to-end propagation of information among periodically activated tasks and messages. A communication mechanism based on the preservation of the latest written value and the overwriting of old ones (shared variable buffer) is assumed. We target our analysis in the context of automotive domain standards. In particular, priority based, mixed preemptive scheduling policy is assumed for the ECUs, all messages are exchanged on a CAN bus, where they are transmitted in order of their IDs, and all the messages transmitted by the same ECU are assumed to be enqueued by the same middleware-level task.

We provide experimental results that simulate the system behavior, and then compare the simulation results with the analytical results obtained with our technique. The experimental results show that our technique provides a good approximation of the latency distribution.

In the rest of the chapter, we first describe the architecture of the typical automotive distributed system. Then we define a formal model for it and we formalize the concept of end-to-end latency, describing each contributing part, and the stochastic analysis of the computation and communication stages that compose the end-to-end computations are detailed in the following sections. Finally, we provide the results on an experimental vehicle architecture.

4.1 Architecture of Distributed Automotive Systems

A typical automotive system consists of several ECUs communicating through CAN buses. A software stack is implemented on each ECU to provide support for the computations and communication, including the application tasks, the middleware, the drivers and the bus peripherals. Figure 4.1 shows the detailed structure with all the layers of the stack. The end-to-end latency analysis must be performed considering the standards and conventions used at each of these levels.

4.1.1 Application Tasks

Application tasks are the implementation of a control or dataflow algorithm, developed following a model-based design flow or by manual coding. They are activated periodically as supported by the AUTOSAR standard [1], and executed under the control of an OSEK compliant operating system [7]. As such, they have a static priority that does not change at run time. Depending on their preemption properties, they can either be preempted by a higher priority task, or keep running until termination. A more detailed description of the application task model is presented in Section 2.1.

For communication purpose, tasks read input signals at the beginning of their execution

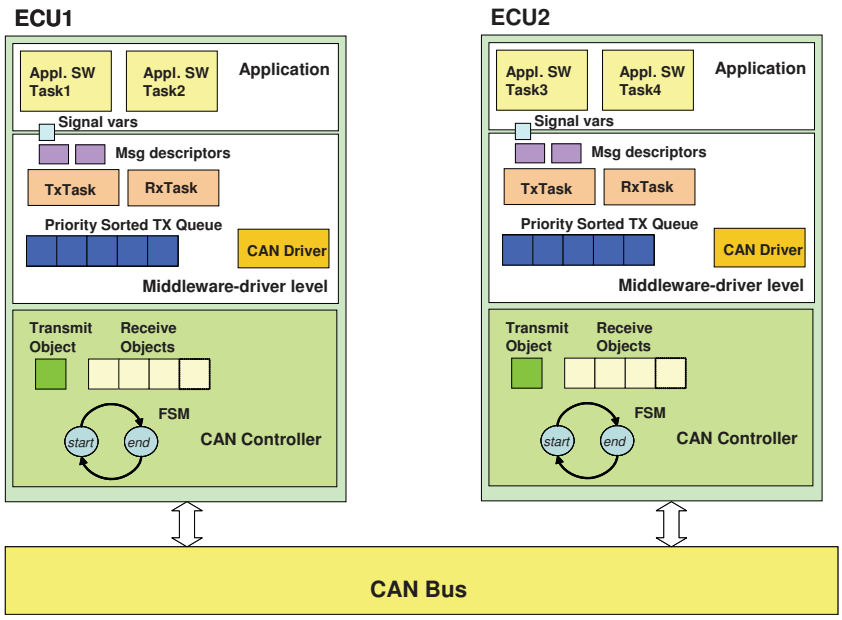

Figure 4.1: Structure of distributed automotive systems

and write their outputs in shared variables at the end. One special set of such shared variables is at the boundary between the application and the middleware. These variables, called *middleware buffers*, contain one signal each.

4.1.2 Middleware

The middleware acts as the first interface layer from the application to the CAN bus. Middleware buffers are shared variables including as many places as the signals that are required by application tasks. These buffers are shared using wait-free protocols, therefore reads and writes will not block the application software tasks.

Transmission The transmit task (*TxTask*) is a special middleware-level task that is activated periodically and has the responsibility of assembling the message from the signal values and then

enqueuing it for transmission on the bus. Each message m_i has an associated period T_i, expressed as an integer multiple of the TxTask period T_{Tx}, i.e., $T_i = k_i \times T_{Tx}$ for some integer k_i. Inside the middleware, a descriptor consisting of a binary moding variable, a transmission flag and a counter are associated to each message. The moding variable indicates whether the message must be sent for the currently active mode. When the TxTask is activated, it scans all the local message descriptors. For each of them, if the moding variable is set, it decrements the counter and, if the counter is zero, reads the values of all the signals that are mapped inside the message and assembles the message data packet. Whenever a message is assembled, the TxTask also checks the CAN controller for one empty transmit object. In case it is found, the message frame is immediately copied into the CAN controller transmit object. In case all transmit objects are full, the message is copied into a priority sorted *TxQueue* shared with the CAN driver, which is responsible to copy messages from TxQueue to transmit object (see Section 4.1.3). After the message is copied into the transmit object, either by the TxTask or by the CAN driver, its transmission flag is reset to zero.

The execution time of the TxTask, just like any other tasks, depends upon its implementation (e.g., from Vector CANtech, Inc. [1]) and the speed of the host processor. The priority of this task should be high to reduce release jitter. The TxTask can be possibly synchronized with the other tasks running on the same ECU. In practice, we assume that its execution time and response time are neglectable, thus the messages are queued periodically with no queuing jitter.

Reception On the reception side, a middleware level task called *RxTask* is periodically activated. The RxTask checks all the receive objects for a new message. When a message is found, it is disaggregated into signals that are copied into the middleware buffer. The RxTask period is typically

[1] see http://www.vector-cantech.com/

the same as that of the TxTask.

4.1.3 CAN Drivers and Peripherals

Transmission On the transmission side, the CAN driver is typically an interrupt service routine running on the ECU. Messages are queued by the CAN driver, which is triggered by the interrupt signal that informs the completion of the transmission of the message occupying one of the transmit objects. The interrupt handler selects the message on top of the queue (the highest priority message) as the one that has to be placed into the empty transmit object (TxObject). When executed, it copies the messages (i.e., the highest priority ones) in the priority sorted TxQueue into the transmit objects.

The CAN controller typically hosts a number of hardware registers, called transmit objects, acting as buffers for messages to be transmitted over the network. These registers are shared by all messages. We assume transmit objects to be preemptable, i.e., allowing aborting if the message currently in the register has a priority lower than one of the enqueued messages. Transmission priority is assigned to the transmit objects by the peripheral microcode according to the ID of the message contained in the register.

Reception On the reception side, the middleware RxTask periodically checks the CAN controller for new messages that have been received and are available in the receive objects. The RxTask copies the message from the peripheral receive register to the corresponding middleware buffers, which will be read by the application tasks.

4.1.4 CAN Bus

The CAN bus is a wired AND channel connecting all nodes. The scheduling policy in CAN protocol is priority-based and *non-preemptive*, that is, a message instance transmission can not be preempted by higher priority message instances queued during its transmission. Our analysis also assumes that the peripheral always sends the ready messages in priority (ID) order and that the transmit registers can be preempted (or the transmission aborted) if a new, higher priority message becomes available at the node. A more detailed description of the CAN protocol and the model for message management is presented in Section 3.1.

4.2 System Model and Notation

The model for the distributed real-time system considered in this chapter is the following: a periodic activation signal from a clock triggers the computation, some application task reads the input data from a sensor on some node, it computes intermediate results that are possibly sent over the network to other tasks and, finally, another task, possibly executing on a remote node, generates the outputs as the result of the computation.

A natural model for the description of these end-to-end computations is a *dataflow* of tasks. We restrict the dataflow to be acyclic, thus the system is a *Directed Acyclic Graph (DAG)*. This graph has inputs from the environment, which can also provide the external events that trigger execution. At the other end of the graph the outputs mark the end of the execution. In the middle the collection of vertices represent the *tasks* and *messages* in the systems, and the edges represent the *data signals* communicated among them. Formally, the system is a tuple (V, E, S), where V is the set of vertices, E the set of edges, and S the set of shared resources.

$V = \{o_1, \ldots, o_n\}$ is the set of objects implementing the computation and communication functions of the system. When an object o_i is a task we also use the notation τ_i. A task τ_i is characterized by $(\Upsilon_i, T_i, O_i, \mathcal{E}_i, P_i, N_i)$, where Υ_i is the CPU resource it needs to execute, and the other characteristics have the same meaning as in Section 2.1. Note that the task execution time $\mathcal{E}_i$ is a discrete random variables with a known probability mass function. A task may have one or more *input* ports and one or more *output* ports, which are used to exchange signal data. Each task will run an infinite sequence of *job*s, each job $\Gamma_{i,j}$ reads *its input at its release time* $Q_{i,j}$ and *writes its results at the finish time* $\mathcal{F}_{i,j}$ of its execution. The response time $\mathcal{R}_{i,j}$ of job $\Gamma_{i,j}$ is defined as the time interval from its arrival time $A_{i,j}$ to its finish time $\mathcal{F}_{i,j}$, which is also a random variable. The task response time $\mathcal{R}_i$ is defined as the average of the response times $\mathcal{R}_{i,j}$ of its jobs $\Gamma_{i,j}$.

A message object o_i, also denoted as m_i, is similarly defined by $(\Upsilon_i, T_i, \mathcal{O}_i, \mathcal{E}_i, P_i, N_i)$, where Υ_i is the bus resource needed for its transmission, and the preemption property N_i is always non-preemptable according to the CAN protocol. For each periodic activation, we consider a *message instance*. On its arrival, a message instance is *queued* by a periodic middleware task with zero response time. We assume there is no synchronization among the local clocks of the nodes connected to the CAN bus. Hence, even if the messages from the same node have the same phase and zero queuing jitter, globally messages may be enqueued at random relative times with a non-deterministic initial phase $\mathcal{O}_i$.

The *hyperperiod* H_k of resource Υ_k is defined as the least common multiple of the periods of all objects executing or transmitting on it, i.e., $H_k = lcm(\bigcup_i T_i)$ where $\Upsilon_i = \Upsilon_k$.

The edges $E = \{e_1, e_2..., e_m\}$ represent the input/output connections between objects. An edge $e_i = (o_h, o_k)$ connects the output port of object o_h to the input port of object o_k. e_i will

carry a data signal with a given bit width produced by o_h and immediately available at the input port of o_k.

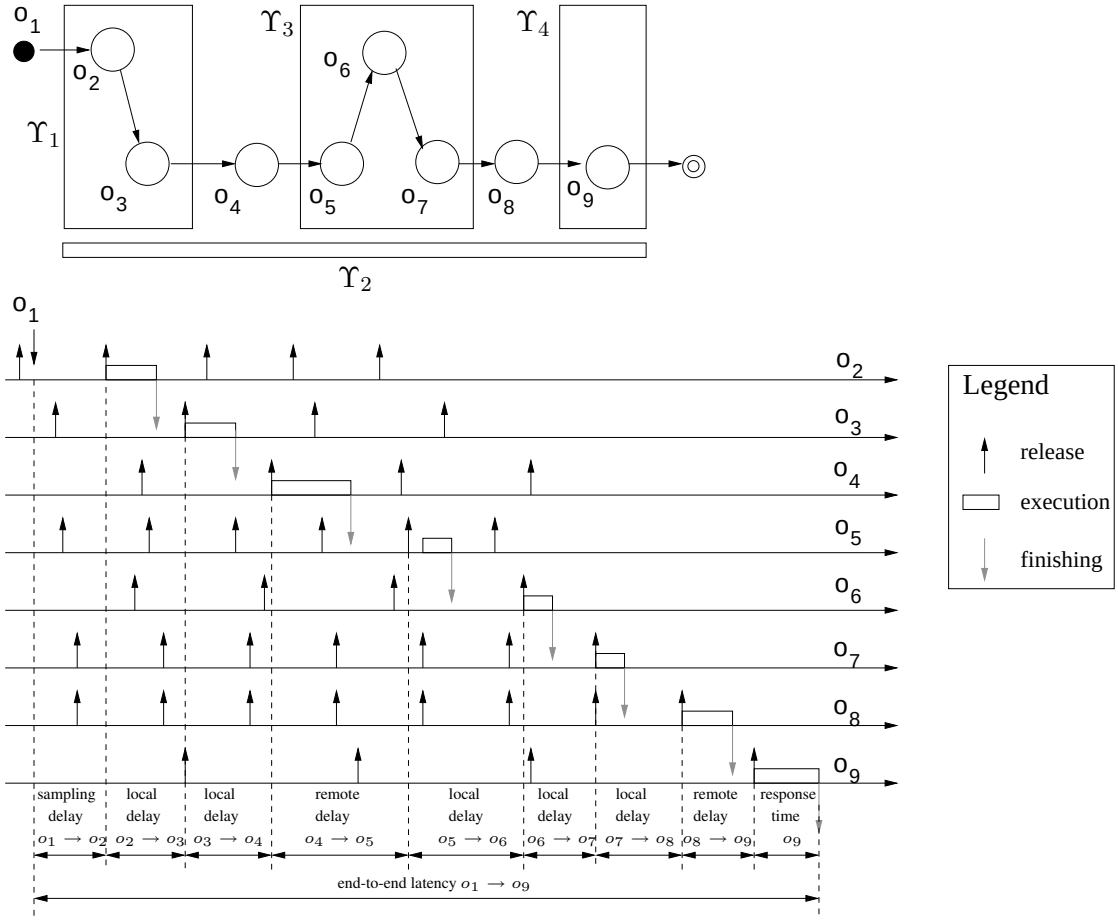

Figure 4.2: Model of an example distributed automotive architecture and its end-to-end latency

A *functional chain* or *path* from o_i to o_j, denoted as $\Pi_{i,j}$, is an ordered sequence $(o_i, \ldots, o_j)$ of objects such that there is an edge between any two consecutive objects. Figure 4.2 gives an example of a path between the objects o_1 and o_9, where the top diagram is the architecture of the resources and objects. Normally the source object (o_1) of the path whose activation represents the triggering of the external events, and the sink object (o_9) represents the actuation output. Resources

Υ_1, Υ_3 and Υ_4 are CPUs, thus the objects on them are actually software tasks. Resource Υ_2 is a CAN bus, and the objects o_4 and o_8 are messages transmitting on it.

Each path consists of one or more interactions among *local* objects. In the path of the figure, local interactions are between the software tasks o_2 and o_3, and in the chain $o_5 \rightarrow o_6 \rightarrow o_7$. Also, please note that each message is enqueued by a middleware task at the transmitting node. As such, the interaction between the transmitting task and the message is still considered to be part of the local chain. Thus the communication through the edges (o_3, o_4) and (o_7, o_8) is considered as local interaction.

Other interactions are between a message and the receiving task, such as the edges (o_4, o_5) and (o_8, o_9), where the input and output objects of the edge are activated according to two unsynchronized clocks. These interactions are thus considered as remote.

For an edge (o_h, o_k) connects the output port of object o_h to the input port of object o_k, the data sent by o_h may be overwritten before they are read thus never get propagated to the end, and they may be read by multiple instances of o_k. Consider the communication chain $o_2 \rightarrow o_3 \rightarrow o_4$ as in Figure 4.3. The data produced by the instance $\Gamma_{2,i}$ of o_2 is never propagated because the next instance $\Gamma_{2,i+1}$ finishes execution and overwrite it before the instance $\Gamma_{3,j}$ of the receiving task is activated. On the other hand, the data generated by $\Gamma_{3,j}$ can be read and propagated by two instances $M_{4,k}$ and $M_{4,k+1}$ of the following message o_4.

For the path $\Pi_{i,j} = (o_i, \ldots, o_j)$, we are interested in the time interval required for the change of the input at the source task o_i of the chain to be propagated to the last task o_j at the other end of the chain, whatever is the state of the tasks in the path. We consider the fact that some intermediate results may be overwritten before they are read. Also, if some data is read multiple

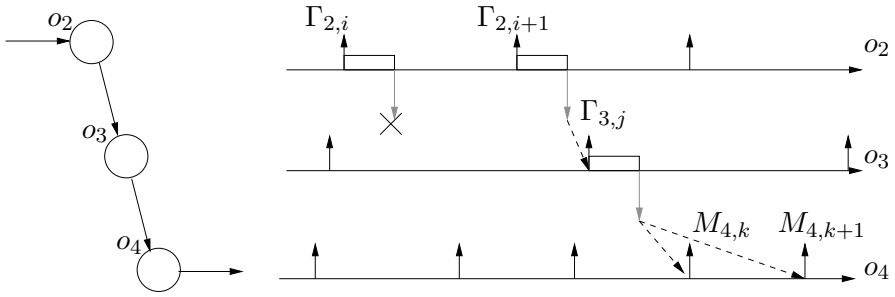

Figure 4.3: Data Loss and Duplication During Communication

times, we take into consideration all the different latency values associated to this data. The *end-to-end latency* $\mathcal{L}_{i,j}$ associated to the path $P_{i,j} = (o_i, \ldots, o_j)$ is defined as the time interval between the activation of one instance of o_i and the completion of the instance of o_j that produces a result dependent on the output of o_i.

4.3 End-to-end Latency Analysis

Given a path in the distributed architecture, its end-to-end latency analysis is divided into sections, according to different types of interaction between objects in the path. At this stage, we assume that the response time of each object in the path is already known: the response times of software tasks are analyzed as in Chapter 2, and messages in Chapter 3. With reference to the bottom part of Figure 4.2, the latency consists of an initial *sampling delay*, from the time of the occurrence of the external event (o_1 in Figure 4.2) to the activation of the task (o_2) that reads it. Following, there are local interactions and remote interactions, with the corresponding latencies, always measured between the activation times of the corresponding tasks and messages (for messages, the activation time of the middleware task that is responsible for their enqueuing is considered). Finally, the end-to-end latency is computed by the response time of the last task (o_9) in the chain.

4.3.1 Delays of Local Communication

In the case of local communication from tasks to tasks, and tasks to messages, the latency from the activation of the sender to the activation of the receiver that reads its output depends on their relative phase and the response time of the sender. For each end-to-end path we need to identify the local segment of the dataflow with each starting task. Each instance of the starting task defines an instance of the local communication path.

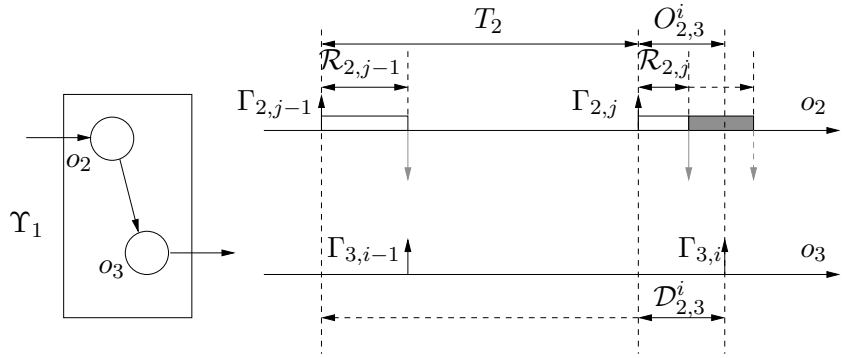

Figure 4.4: Latency of task to task communication: local interaction

Consider the scenario in Figure 4.4, which is part of the end-to-end latency diagram in Figure 4.2. The local communication is between tasks o_2 and o_3, or equivalently, τ_2 and τ_3. Suppose that there are n jobs of τ_3 in the hyperperiod, $\Gamma_{3,1}, \Gamma_{3,2}, ..., \Gamma_{3,n}$. Each time a job $\Gamma_{3,i}$ of τ_3 is activated and reads some data, we need to trace back and find out the job $\Gamma_{2,j}$ of τ_2 which produces this data and compute the latency between the activation events of $\Gamma_{2,j}$ and $\Gamma_{3,i}$. We use the notation $\mathcal{D}^i_{2,3}$ to denote the delay between the activation of $\Gamma_{3,i}$, the i-th job of τ_3 in the hyperperiod, and the activation of the job of τ_2 which produces the data consumed by $\Gamma_{3,i}$. Also, we denote the average delay associated to all the jobs of τ_3 in one hyperperiod as $\mathcal{D}_{2,3}$. Note that the data produced by a job of τ_2 may be overwritten by the finish of a new instance of τ_2 before they are consumed. In this

case, the value is not propagated and not considered in the distribution of the delay.

Since τ_2 and τ_3 are on the same node, the relative phase of each job $\Gamma_{3,i}$ with the previous job $\Gamma_{2,j}$ of τ_2 within the hyperperiod is known and denoted as $O^i_{2,3}$. In case the response time of $\Gamma_{2,j}$ is less than or equal to $O^i_{2,3}$ (Figure 4.4), the output of $\Gamma_{2,j}$ is consumed by the following job $\Gamma_{3,i}$ of τ_3 and the latency of this communication stage is $O^i_{2,3}$. Thus the probability that the delay $\mathcal{D}^i_{2,3}$ is equal to $O^i_{2,3}$ is

$$\mathbb{P}(\mathcal{D}^i_{2,3} = O^i_{2,3}) = \mathbb{P}(\mathcal{R}_{2,j} \leq O^i_{2,3}) \tag{4.1}$$

If the finish time of $\Gamma_{2,j}$ is larger than the activation time of $\Gamma_{3,i}$, the data consumed by $\Gamma_{3,i}$ is produced by $\Gamma_{2,j-1}$ provided that $\Gamma_{2,j-1}$ finishes execution before the activation of $\Gamma_{3,i}$.

$$\begin{aligned} & \mathbb{P}(\mathcal{D}^i_{2,3} = O^i_{2,3} + T_2) \\ = \;& \mathbb{P}(\mathcal{R}_{2,j} > O^i_{2,3}, \mathcal{R}_{2,j-1} \leq O^i_{2,3} + T_2) \end{aligned} \tag{4.2}$$

Also, we assume that the job response time is independent from other jobs of the same task, and from jobs of other tasks, thus the probability that the delay $\mathcal{D}^i_{2,3}$ is equal to $O^i_{2,3} + T_2$ is

$$\begin{aligned} & \mathbb{P}(\mathcal{D}^i_{2,3} = O^i_{2,3} + T_2) \\ = \;& \mathbb{P}(\mathcal{R}_{2,j} > O^i_{2,3}, \mathcal{R}_{2,j-1} \leq O^i_{2,3} + T_2) \\ = \;& \mathbb{P}(\mathcal{R}_{2,j} > O^i_{2,3}) \times \mathbb{P}(\mathcal{R}_{2,j-1} \leq O^i_{2,3} + T_2) \end{aligned} \tag{4.3}$$

In general, the data read by $\Gamma_{3,i}$ can be produced by any of the previous k instances from τ_2, until the probability for one such instance of τ_2 is zero. We need to try all the $k = 0, 1, ..., k_i$, where k_i is the first integer that satisfies $r^{\max}_{2,j-k_i} \leq O^i_{2,3} + k_i \times T_2$ and $r^{\max}_{2,j-k_i}$ is the worst case response time of job $\Gamma_{2,j-k_i}$. In another word, $\Gamma_{2,j-k_i}$ is the first job to guarantee finishing execution before the activation of $\Gamma_{3,i}$ such that for any previous job $\Gamma_{2,j-k}$ where $k > k_i$, its output data is overwritten by $\Gamma_{2,j-k_i}$ and can not be read by $\Gamma_{3,i}$.

$$
\begin{aligned}
& \forall k = 0, 1, ..., k_i \text{ where } r^{\max}_{2,j-k_i} \leq O^i_{2,3} + k_i \times T_2, \\
& \mathbb{P}(\mathcal{D}^i_{2,3} = O^i_{2,3} + kT_2) \\
= \;& \mathbb{P}(\mathcal{R}_{2,j-k} \leq O^i_{2,3} + kT_2, \mathcal{R}_{2,j-k+1} > O^i_{2,3} + (k-1)T_2, ..., \mathcal{R}_{2,j} > O^i_{2,3}) \\
= \;& \mathbb{P}(\mathcal{R}_{2,j-k} \leq O^i_{2,3} + kT_2) \times \mathbb{P}(\mathcal{R}_{2,j-k+1} > O^i_{2,3} + (k-1)T_2) \times ... \times \mathbb{P}(\mathcal{R}_{2,j} > O^i_{2,3}) \\
= \;& \mathbb{P}(\mathcal{R}_{2,j-k} \leq O^i_{2,3} + kT_2) \times \prod_{m=1}^{k} \mathbb{P}(\mathcal{R}_{2,j-k+m} > O^i_{2,3} + (k-m)T_2)
\end{aligned}
\tag{4.4}
$$

The distribution of $\mathcal{D}_{2,3}$ is the average of the distributions of $\mathcal{D}^i_{2,3}$ for all the jobs $\Gamma_{3,i}$ of τ_3 within the hyperperiod, i.e.,

$$
\forall d, \mathbb{P}(\mathcal{D}_{2,3} = d) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{P}(\mathcal{D}^i_{2,3} = d) \tag{4.5}
$$

Similarly, the communication between a task and the middleware task that is responsible for the formatting and the enqueuing of a message is also local, and the delay between them is calculated in a similar equation as Equation 4.5.

4.3.2 Delays of Remote Communication

In the case of remote communication, i.e., from a message to its receiving task, the relative phase of the next task on the path is independent from the message response time. Hence, we assume that for each instance of the message, its response time distribution is the same as the message response time.

As an example, we consider the scenario in Figure 4.5, which is part of the end-to-end latency in Figure 4.2. Here the communication is between objects o_4 and o_5, i.e., a message m_4 and a task τ_5, which are activated from different ECUs. The relative phase $\mathcal{O}_{4,5}$ between τ_5 and

the previously activated instance of m_4 is assumed to be uniformly distributed within $[0, T_4)$ and is independent from the response time of m_4.

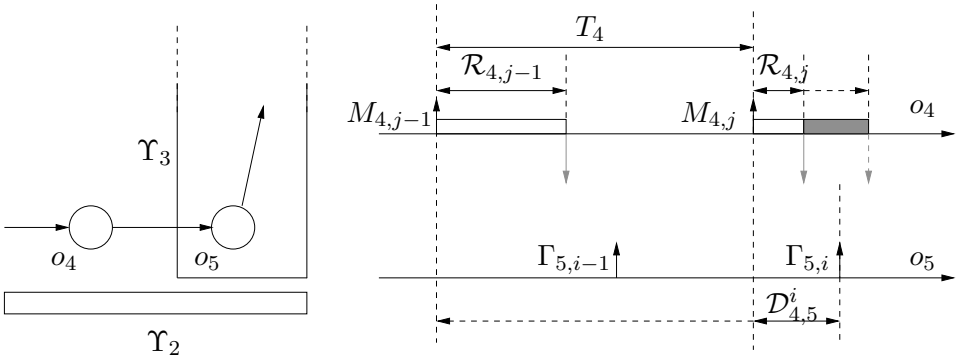

Figure 4.5: Latency of message to task communication: remote interaction

Similar to the case in local communication, we denote $\mathcal{D}^i_{4,5}$ to be the delay between the activation of a job $\Gamma_{5,i}$ of τ_5 and the activation of the instance of m_4 which produces the data consumed by $\Gamma_{5,i}$. If the relative phase $\mathcal{O}^i_{4,5}$ a job $\Gamma_{5,i}$ of τ_5 and the previous message instance $M_{4,j}$ of m_4 is known to be d, we can follow the same reasoning as in Section 4.3.1 and find an equation to calculate the probability function of $\mathcal{D}^i_{4,5}$ which is similar to Equation 4.4

$$
\begin{aligned}
& \forall k = 0, 1, ..., k_i \text{ where } r^{\max}_{4,j-k_i} \leq d + k_i \times T_4, \\
& \mathbb{P}(\mathcal{D}^i_{4,5} = d + kT_4 | \mathcal{O}^i_{4,5} = d) \\
= \;& \mathbb{P}(\mathcal{R}_{4,j-k} \leq d + kT_4, \mathcal{R}_{4,j-k+1} > d + (k-1)T_4, ..., \mathcal{R}_{4,j} > d | \mathcal{O}^i_{4,5} = d) \\
= \;& \mathbb{P}(\mathcal{R}_{4,j-k} \leq d + kT_4 | \mathcal{O}^i_{4,5} = d) \times \mathbb{P}(\mathcal{R}_{4,j-k+1} > d + (k-1)T_4 | \mathcal{O}^i_{4,5} = d) \\
& \times ... \times \mathbb{P}(\mathcal{R}_{4,j} > d | \mathcal{O}^i_{4,5} = d) \\
= \;& \mathbb{P}(\mathcal{R}_{4,j-k} \leq d + kT_4 | \mathcal{O}^i_{4,5} = d) \times \prod_{m=1}^{k} \mathbb{P}(\mathcal{R}_{4,j-k+m} > d + (k-m)T_4 | \mathcal{O}^i_{4,5} = d)
\end{aligned}
\tag{4.6}
$$

Provided that the relative phase $\mathcal{O}^i_{4,5}$ is independent from the response time of any instance

of m_4,

$$\begin{aligned}
&\mathbb{P}(\mathcal{R}_{4,j-k} \leq d + kT_4 | \mathcal{O}^i_{4,5} = d) = \mathbb{P}(\mathcal{R}_{4,j-k} \leq d + kT_4) \\
&\mathbb{P}(\mathcal{R}_{4,j-k+m} > d + (k-m)T_4 | \mathcal{O}^i_{4,5} = d) = \mathbb{P}(\mathcal{R}_{4,j-k+m} > d + (k-m)T_4)
\end{aligned} \tag{4.7}$$

Equation 4.6 can be simplified as

$$\begin{aligned}
& \forall k = 0, 1, ..., k_i \text{ where } r^{\max}_{4,j-k_i} \leq d + k_i \times T_4, \\
& \mathbb{P}(\mathcal{D}^i_{4,5} = d + kT_4 | \mathcal{O}^i_{4,5} = d) \\
= \quad & \mathbb{P}(\mathcal{R}_{4,j-k} \leq d + kT_4) \times \prod_{m=1}^{k} \mathbb{P}(\mathcal{R}_{4,j-k+m} > d + (k-m)T_4)
\end{aligned} \tag{4.8}$$

We assume that for each instance $M_{4,j}$ of m_4, the *pmf* of its response time $\mathcal{R}_{4,j}$ is the same as the task response time $\mathcal{R}_4$ of m_4; the distribution of $\mathcal{O}^i_{4,5}$ is uniformly distributed within $[0, T_4)$, thus the probability of each time instant is τ/T_4, assuming the granularity of the discretized system is τ. We now calculate the probability function of $\mathcal{D}^i_{4,5}$ as in Equation 4.9

$$\begin{aligned}
& \forall 0 \leq d < T_4, \forall k = 0, 1, ..., k_i \text{ where } r^{\max}_{4,j-k_i} = r^{\max}_4 \leq d + k_i \times T_4, \\
& \mathbb{P}(\mathcal{D}^i_{4,5} = d + kT_4) \\
= \quad & \mathbb{P}(\mathcal{D}^i_{4,5} = d + kT_4 | \mathcal{O}^i_{4,5} = d) \times \mathbb{P}(\mathcal{O}^i_{4,5} = d) \\
= \quad & \mathbb{P}(\mathcal{R}_{4,j-k} \leq d + kT_4) \times \prod_{m=1}^{k} \mathbb{P}(\mathcal{R}_{4,j-k+m} > d + (k-m)T_4) \times \frac{\tau}{T_4} \\
= \quad & \frac{\tau}{T_4} \times \mathbb{P}(\mathcal{R}_4 \leq d + kT_4) \times \prod_{m=1}^{k} \mathbb{P}(\mathcal{R}_4 > d + (k-m)T_4)
\end{aligned} \tag{4.9}$$

As in Equation 4.9 the calculation of the $\mathcal{D}^i_{4,5}$ is independent from the instance index i, thus the right hand side of it remains the same for all the other instance of τ_5. The distribution of $\mathcal{D}_{4,5}$, which is the average of $\mathcal{D}^i_{4,5}$ for all the instances $\Gamma_{5,i}$ of τ_5 in the hyperperiod, is

$$\begin{aligned}
& \forall 0 \leq d < T_4, \forall k = 0, 1, ..., k^{\max} \text{ where } r^{\max}_4 \leq d + k^{\max} \times T_4, \\
& \mathbb{P}(\mathcal{D}_{4,5} = d + kT_4) \\
= \quad & \frac{\tau}{T_4} \times \mathbb{P}(\mathcal{R}_4 \leq d + kT_4) \times \prod_{m=1}^{k} \mathbb{P}(\mathcal{R}_4 > d + (k-m)T_4)
\end{aligned} \tag{4.10}$$

4.3.3 End-to-end Latency

We now define the method for computing the latency in a path $\Pi_{1,n}$ containing a sequence of n objects o_i with an edge between (o_{i-1}, o_i) for each $i = 2, ..., n$. The *end-to-end latency* $\mathcal{L}_{1,n}$ associated to the path $\Pi_{1,n}$ is the time interval between the change of the input data of o_1 and the completion of task o_n where the data change is reflected to the output of o_n. Within the path from o_1 to o_n, the first piece of latency comes from the sampling delay $\mathcal{Z}_{1,2}$ between o_1 and o_2, i.e., from the activation of the source object o_1 of the path, which represents the triggering of the external events, to the activation of the following object o_2. $\mathcal{Z}_{1,2}$ can be assumed as a uniform random variable defined on $[0, T_2)$. Following that, $\mathcal{D}_{i-1,i}, \forall i = 2, ..., n$ is the delay from the activation of o_{i-1} thus the time o_{i-1} consumes the data from its predecessor, to the following activation time o_i after the finish of o_{i-1} thus the consumption of the data from o_{i-1}. Thus the delay of data change at the time o_1 is activated to the time the related data are consumed by o_n at its activation time is the sum of $\mathcal{Z}_{1,2}$ and $\mathcal{D}_{i-1,i}$. The end-to-end latency is added by another additional delay from the consumption of the data by o_n at its activation time, to its finish time when the data change to the output are reflected, which is the task response time of o_n. Thus,

$$\mathcal{L}_{1,n} = \mathcal{Z}_{1,2} + \sum_{i=2}^{n} \mathcal{D}_{i-1,i} + \mathcal{R}_n \tag{4.11}$$

We assume that $\mathcal{D}_{i-1,i}$, $\mathcal{Z}_{1,2}$ and $\mathcal{R}_n$ are independent from each other, so the end-to-end latency *pmf* is the convolution of the *pmf*s of all these contributions.

$$f_{\mathcal{L}_{1,n}} = f_{\mathcal{Z}_{1,2}} \bigotimes_{i=2}^{n} f_{\mathcal{D}_{i-1,i}} \bigotimes f_{\mathcal{R}_n} \tag{4.12}$$

4.4 Experimental Results

We demonstrated the applicability and the possible benefits of our approach with a case study derived from the analysis of a bus subsystem of an experimental vehicle that incorporates advanced active safety functions. The vehicle supports advanced distributed functions with end-to-end computations collecting data from 360° sensors to the actuators, consisting of the throttle, brake and steering subsystems and of advanced Human-Machine Interface (HMI) devices. The analysis focuses on the subset of tasks and signals that are part of paths with timing constraints.

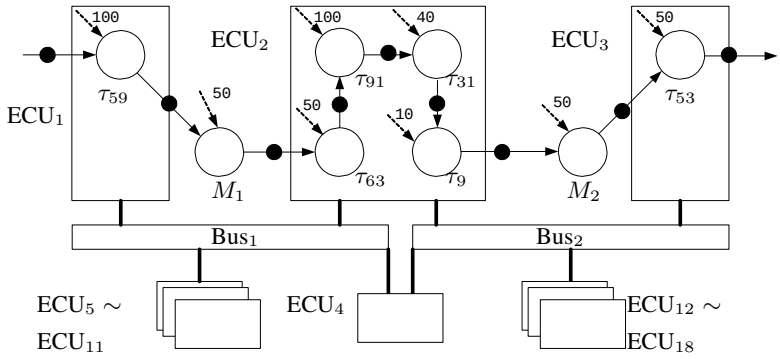

Figure 4.6: A path of periodic tasks/messages in an example automotive architecture

The subsystem of the architecture platform involved with the selected end-to-end path consists of 18 ECUs connected with two CAN buses at speed 500kb/s. A total of 71 tasks is executed on the ECU nodes, and 136 CAN messages exchanged among the tasks by the CAN buses. Worst-case execution time estimates have been obtained for all tasks, and the bit length of the signals is between 1 (for binary information) and 64 (full CAN message). The task execution time is assumed to be uniformly distributed within its minimum value (as 1 unit) and the profiled worst case value. The selected path is composed of 6 software tasks sitting on 3 different ECUs, and

2 CAN messages. Figure 4.6 gives the path in the example architecture platform, where the periods of tasks and messages are shown at the dashed arrows.

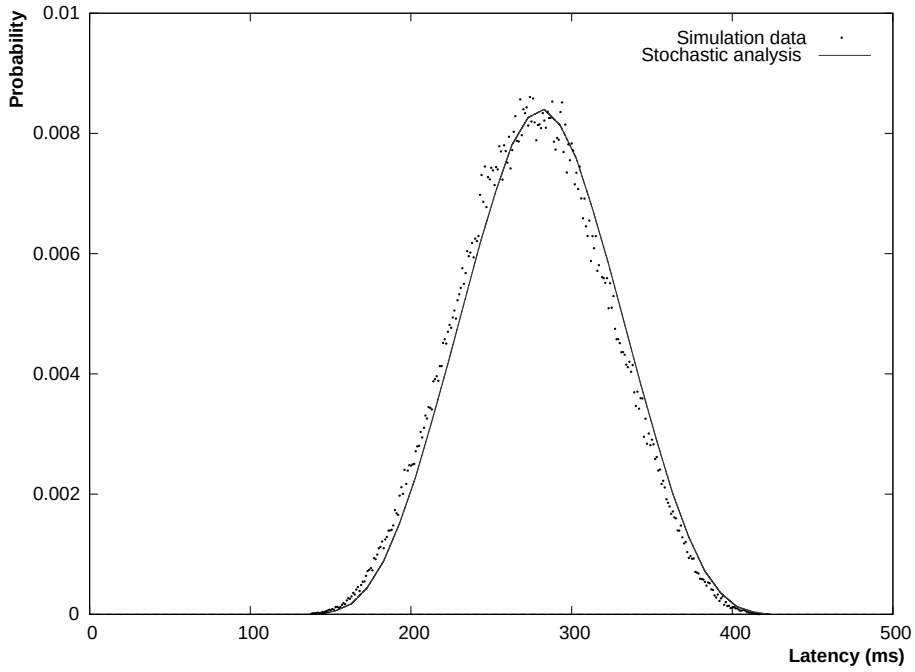

Figure 4.7: The end-to-end latency *pmf* of the path in the example automotive architecture

The experiments have been performed by comparing the results obtained with the stochastic analysis presented in this dissertation and simulation results.

4.4.1 Simulation Setup

The simulation data have been obtained using a system level simulator purposely developed in the context of this research. The simulator consists of a number of modules in C++ that receive as input a set of files with the description of the functional architecture of tasks exchanging signal information. Additional files provide a description of the physical computation architectures consisting of the ECUs, the CAN buses, the bus topology and the CAN adapter configuration. Fi-

nally another set of files defines the mapping of the tasks into the ECUs, with the priority assigned to them and the mapping of signals into CAN messages, with the CAN identifier associated with each message. The simulator supports the CAN bus arbitration policy and fixed priority scheduling policies with preemption on the CPUs. It also simulates the flow of information associated with the signals and tags each signal with the time it is produced, read, written, sent inside a message or received. The middleware and driver model used by the simulator follows the description presented in the earlier sections of this paper. The simulation study has been performed on the automotive case study simulating 4.5 seconds of steady state system execution (3 hyperperiods) for each phase configuration. Approximately 50 millions phase configurations have been tried (50 million runs of the program). The total running time of the simulations has been of approximately 20 hours.

4.4.2 Validation of Stochastic Analysis

For the case study, it takes 7.8 seconds to perform the stochastic analysis implemented in Java on a laptop with 1.6GHz Cpu and 1.5G RAM. Figure 4.7 shows the results of the end-to-end latency *pmf* of the example path in the architecture. The simulated latency *pmf* is compared with the response time estimated by the stochastic analysis. As in the figure, these two curves are very close to each other. The result demonstrates the accuracy on analysis of task/message response times, and the models on the communication semantics and sampling delays (2 remote communication links and 5 local links) in our analysis framework.

Chapter 5

Statistical Analysis of Controller Area Network Message Response Times

As in the early stage of the automotive design, since designers only have knowledge about possible average traffic load a node generates on the bus, we plan to extend our analysis to a method equivalent to the so called *rest-bus* simulation, where only one node is refined to the message level interface (including rates) and the rest of the network is modeled by a set of nodes providing an average (or statistical) load. Our results also show that it is possible to support designers in the early phases of the development process when tactical decisions, such as message priority assignment, which have a big impact on the following phases, are to be made in the presence of incomplete and estimated information.

In this chapter, we explore the possibility to support designers in the early phases of the development process when tactical decisions, such as message priority assignment, which have a big impact on the following phases, are to be made in the presence of incomplete and estimated

information. Our objective now is to analyze the main statistics of the message response time distribution on a CAN bus, trying to extract significant trends and verifying if the use of regression techniques allows us to predict the response times of messages on a given bus when only incomplete information (such as the higher priority and lower priority bus utilization for each message) is available. Our approach is as follows. Based on simulation data on a reference system configuration, we test the conjecture that the probability distributions of the message response times can be approximated by a mix of distribution models with degenerate and gamma distributions. Then, we find, for each message, the correlation between the parameters of the fitted model and a set of parameters that include its attributes (size, priority) and system-level attributes (bus utilization and the fraction of utilization at higher priority). We show that this correlation exists and can be used not only to approximate the probability distribution of the response times for messages belonging to the bus on which the study is performed, but also for other buses. The main contribution of this work is the set of formulas that are found by regression analysis and can be used to approximate, with good accuracy, the probability distribution of the response times of messages on buses for which partial and incomplete information is available, thereby acting as a predictor. We are also able to discuss the existing trade-offs between statistical and stochastic methods, in terms of speed of the analyses vs. availability of data and accuracy of results. Specifically, statistical analysis proves to be very fast and mostly suitable when the complete set of messages is unknown. On the other hand, stochastic analysis is more accurate when the complete message set is available. However, it might be significantly slower as the probability distribution of the message response time must be computed at each discretized time instant. Based upon our preliminary experiments we have noticed that statistical analysis is roughly 7 orders of magnitude faster than stochastic analysis.

5.1 Fitting Exponential Distributions to Message Response Times

In this work, we assume the same model for messages as in the stochastic analysis framework (Section 3.1.2), in which a message m_i is periodically activated and is characterized by $(T_i, \mathcal{O}_i, \mathcal{E}_i, P_i)$, where T_i is its period, $\mathcal{O}_i$ its initial phase, $\mathcal{E}_i$ its transmission time, and P_i its priority. We start from a *reference* system configuration consisting of a CAN bus with messages as in Table 3.1. Messages with their periods and transmission times, originating from 6 nodes, are identified by their priorities, which are also their indices in the table. The CAN bus rate is 500kb/s and its utilization is 60.25%. A simulator is used to extract the distribution of the response time values for each message. The granularity of our simulation is set to $0.05ms$. *For simplicity, in this experiment we assume worst case bit stuffing thus deterministic transmission times.* In our system, as in most real systems, messages are not enqueued without synchronization. There is a single middleware task at each node, running at the greatest common divider of the messages' periods that is responsible for message enqueuing. This means that a low priority message is typically always enqueued together with higher priority messages having a period that is an exact divider of its period.

Note that our experiment has been focused on the current model and simulation data. Of course it is possible to extend our statistical methods when some of the assumptions are relaxed, but the model will have a big impact on the statistics of message response times. For example, if messages are event-driven thus are queued periodically with jitter, their response time distributions may be different from the periodically activated messages. Our hope is that the main statistics extracted from the example message set are also applicable to other messages with the same model, which is validated by the results of messages on other buses.

5.1.1 Common Characteristics of Message Response Time *cdf*s

After performing the simulation, we found that the distributions of message response times are of different types, but with some common characteristics. The type of distribution depends on the message priority, on the local load, on the number of higher priority messages on the local node, and possibly on other factors. For example, the cumulative distribution function (*cdf*) of message m_{25} (priority 25 out of 69) is shown in Figure 5.1. m_{25} is a medium priority message on its node E_3, which has five higher priority messages than m_{25}. It is always enqueued together with three higher priority messages, $\{m_3, m_{23}, m_{24}\}$, while other higher priority messages $\{m_4, m_{20}\}$, with period $10ms$, are only enqueued every two instances of m_{25}. We label the set of messages that are always enqueued together with a generic message m_i on node N_j as its *first local harmonic higher priority set* or *the first harmonic set of level P_i for node j*. Similarly, the set of messages that are enqueued together with every two instances of m_i is labeled as its *second local harmonic higher priority set*. In the example of m_{25}, its first local harmonic higher priority set is $\{m_3, m_{23}, m_{24}\}$, and the messages in the first harmonic set plus $\{m_4, m_{20}\}$ is its second set.

For message m_i, the smallest possible response time value is the time that it takes to transmit all the messages in its first local harmonic higher priority set plus itself, without any interference or blocking from remote load. This minimum response time value is labeled as the first *X offset* in Figure 5.1 and has a finite probability, resulting in a step of the *cdf* (the first *Y offset* in the figure). Furthermore, the message *cdf* may have other discontinuities for other higher priority harmonic sets. These discontinuities are characterized by other pairs of X and Y offsets. For example, message m_{25} has 3 messages in its first harmonic higher priority set. The probability that these three messages and m_{25} are transmitted without interference is approximately 0.2. m_{25} has 2 other messages

in the second harmonic higher priority set, resulting in a second pair of X and Y offsets.

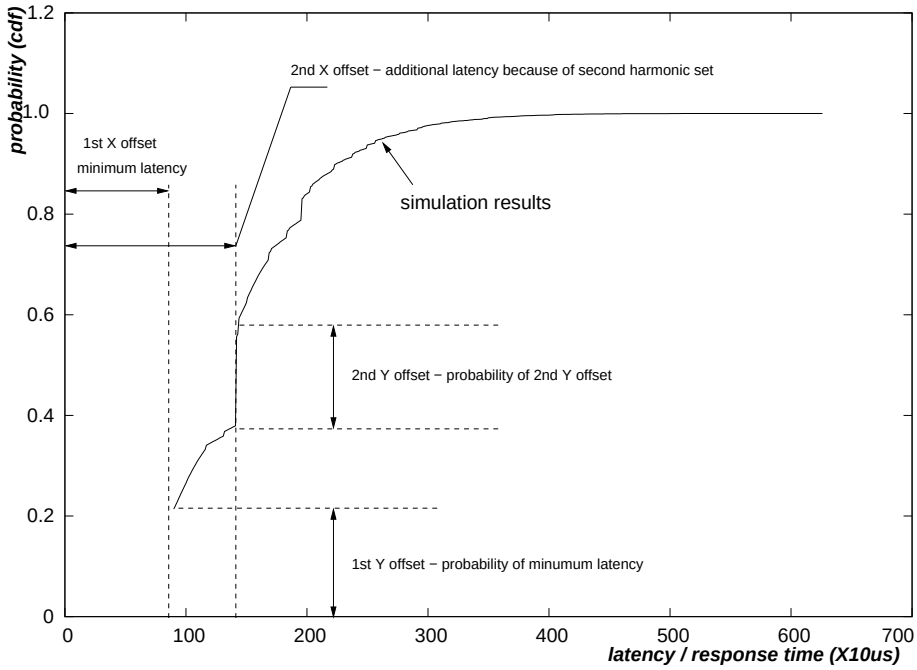

Figure 5.1: Response time *cdf* of an example message m_{25} with more than one higher priority harmonic set

Other messages with different priority may exhibit different characteristics of the *cdf*. Figures 5.2 shows the *cdf* for a high priority messages m_5. m_5 is the highest priority message on its node. Its X offset is very small (equal to its transmission time) and has a high probability (almost 0.5). Furthermore, the *cdf* curve is rapidly increasing. However, for relatively low priority message, e.g., m_{69} in Figure 5.3, the X offset is quite large with a very small probability, and the *cdf* is slowly increasing as shown in the figure. The slopes of these two curves can be compared considering that response times are expressed in Figure 5.3 with a scale ten times larger than the one of Figure 5.2.

Our conjecture is that the message response time *cdf*, normalized by shifting left and down along the axes until the X and Y offsets are zero, can be approximated by a gamma distribution

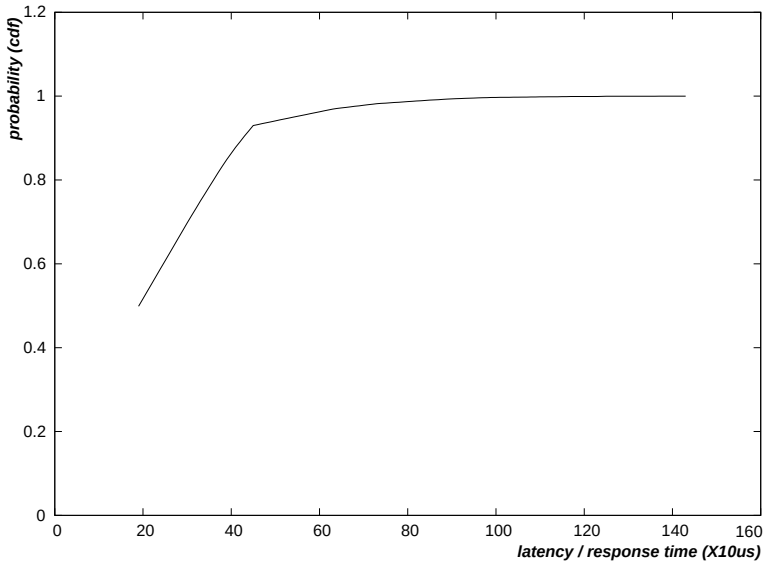

Figure 5.2: Response time *cdf* of a high priority message m_5

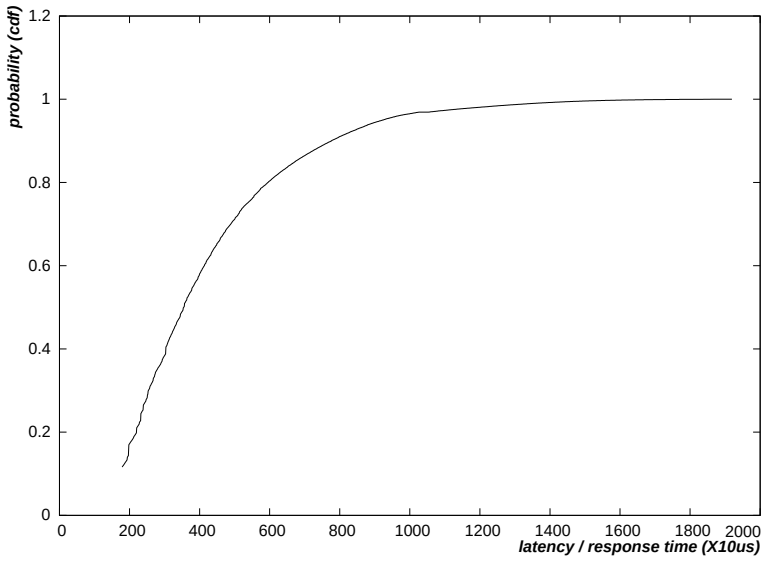

Figure 5.3: Response time *cdf* of a low priority message m_{69}

whose shape parameter a and scale parameter b can be expressed as functions of simple parameters of the message set in the system. In another word, we use the probability mixture model of degenerate distribution and gamma distribution to fit the message response times.

5.1.2 Fitting the Message Response Times

The *pmf* of a degenerate distribution, which is the probability distribution of a discrete random variable whose support consists of only one value x^{off}, is given by:

$$f^D(x, x^{\text{off}}) = \begin{cases} 0 & \text{if } x \neq x^{\text{off}} \\ 1 & \text{if } x = x^{\text{off}} \end{cases} \tag{5.1}$$

its *cdf* is

$$F^D(x, x^{\text{off}}) = \begin{cases} 0 & \text{if } x < x^{\text{off}} \\ 1 & \text{if } x \geq x^{\text{off}} \end{cases} \tag{5.2}$$

The *pdf* of a gamma distribution has the form

$$f^\Gamma(x, a, b) = \begin{cases} 0 & \text{if } x \leq 0 \\ x^{a-1} \dfrac{e^{-x/b}}{b^a \Gamma(a)} & \text{if } x > 0 \end{cases} \tag{5.3}$$

where b is the scale parameter, a is the shape parameter, and $\Gamma(\cdot)$ is the gamma function $\Gamma(a) = \int_0^\infty t^{a-1} e^{-t} dt$.

Its *cdf* is expressed in terms of the incomplete gamma function $\gamma(a, x) = \int_0^x t^{a-1} e^{-t} dt$,

$$F^\Gamma(x, a, b) = \begin{cases} 0 & \text{if } x < 0 \\ \dfrac{\gamma(a, x/b)}{\Gamma(a)} & \text{if } x \geq 0 \end{cases} \tag{5.4}$$

If we shift the gamma distribution to the right by x^{off}, the *pdf* of the *offsetted gamma distribution* becomes

$$f^{\Gamma}(x, x^{\text{off}}, a, b) = \begin{cases} 0 & \text{if } x \leq x^{\text{off}} \\ (x - x^{\text{off}})^{a-1} \dfrac{e^{-(x-x^{\text{off}})/b}}{b^a \Gamma(a)} & \text{if } x > x^{\text{off}} \end{cases} \tag{5.5}$$

and *cdf*

$$F^{\Gamma}(x, x^{\text{off}}, a, b) = \begin{cases} 0 & \text{if } x < x^{\text{off}} \\ \dfrac{\gamma(a, (x - x^{\text{off}})/b)}{\Gamma(a)} & \text{if } x \geq x^{\text{off}} \end{cases} \tag{5.6}$$

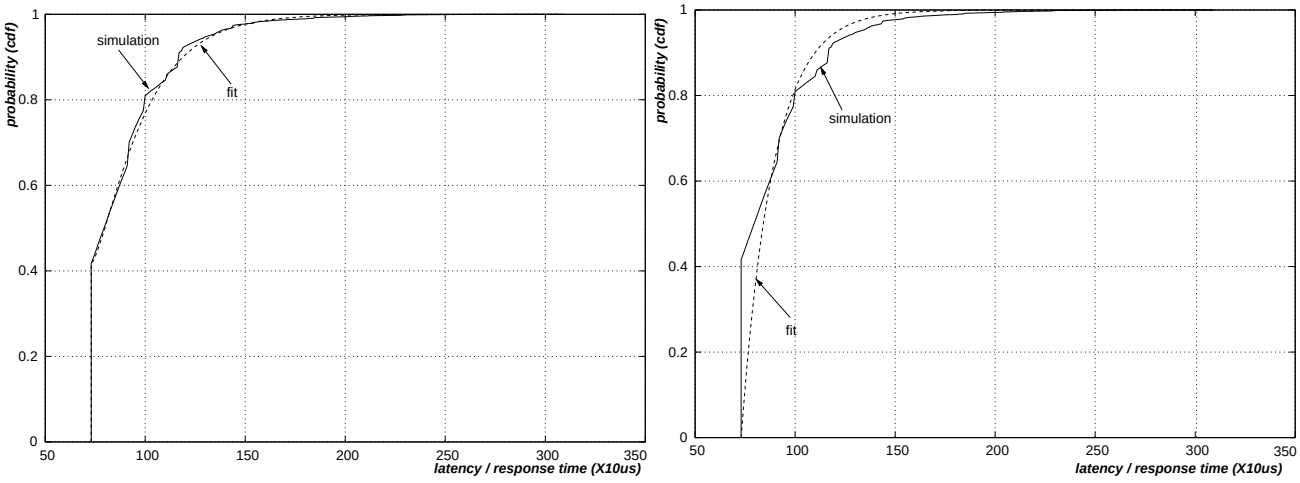

Figure 5.4: Fitting mixture model distributions with and without Y offset to response time of message m_{13}

The response time of message m_i is assumed to be a random variable whose probability distribution is a probability mixture model of degenerate and offsetted gamma distributions, which is the mixture of a discrete and a continuous distributions, thus we use *cdf* to present it, as below

$$
\begin{aligned}
F_i(x, x_i^{\text{off}}, a_i, b_i, y_i^D, y_i^\Gamma) = \quad & y_i^D F^D(x, x_i^{\text{off}}) + y_i^\Gamma F^\Gamma(x, x_i^{\text{off}}, a_i, b_i) \\
= \quad & \begin{cases} 0 & \text{if } x < x_i^{\text{off}} \\ y_i^D + y_i^\Gamma \dfrac{\gamma(a_i, (x - x_i^{\text{off}})/b_i)}{\Gamma(a_i)} & \text{if } x \geq x_i^{\text{off}} \end{cases} \\
\text{w.r.t.} \quad & y_i^D + y_i^\Gamma = 1
\end{aligned}
\tag{5.7}
$$

where x_i^{off}, the minimum message response time, is calculated as the sum of the message transmission time and the queuing delay from local higher priority messages, and the parameters $(a_i, b_i, y_i^D, y_i^\Gamma)$ are estimated using expectation-maximization (EM) algorithm. The result of the fit to response time of message m_{13} in the set is shown in Figure 5.4, which has only one local higher priority harmonic set. The left hand side of the figure shows the best fit to the response time *cdf* of m_{13} without compensating the initial Y offset, i.e., using only the offsetted gamma distribution. The right hand side of the figure shows the fit when we consider the mixture model of degenerate and offsetted gamma distributions. As in Figure 5.4, the error of the fitted distribution of mixture model is much lower and the approximation is sufficiently accurate for the purpose of an early estimation.

For messages with more than one local higher priority harmonic set, the response time *cdf* is approximated using multiple pairs of degenerate and offsetted gamma distributions, each of which with its own X offset. Consider a message m_i on node N_j with multiple local higher priority harmonic sets, m_i suffers different amount of local queuing delays from different higher priority harmonic sets. When we only consider the local messages on N_j, the possible response times and their corresponding probability can be calculated as $x_{i,k}^{\text{off}}$ and $y_{i,k}$, where $\sum_k y_{i,k} = 1$. After taking into consideration the interference and blocking from remote nodes, the probability that its response time is equal to $x_{i,k}^{\text{off}}$ is dropped from $y_{i,k}$ to $y_{i,k}^D$. Furthermore, as in the case where messages have only one higher priority harmonic set, we assume that the shape of the probability distribution due

to further delays from remote nodes can be approximated by an offsetted gamma distribution with its shape parameter a_k and scale parameter $b_{i,k}$. In summary, the response time *cdf* of m_i can be reconstructed as the sum of multiple pairs of degenerate and offsetted gamma distributions, each with five characteristic parameters $(x^{\text{off}}_{i,k}, a_{i,k}, b_{i,k}, y^D_{i,k}, y^\Gamma_{i,k})$, as below

$$
\begin{aligned}
F_i(x, \textstyle\bigcup_k (x^{\text{off}}_{i,k}, a_{i,k}, b_{i,k}, y^D_{i,k}, y^\Gamma_{i,k})) = &\quad \textstyle\sum_k F_{i,k}(x, x^{\text{off}}_{i,k}, a_{i,k}, b_{i,k}, y^D_{i,k}, y^\Gamma_{i,k}) \\
\text{where } F_{i,k}(x, x^{\text{off}}_{i,k}, a_{i,k}, b_{i,k}, y^D_{i,k}, y^\Gamma_{i,k}) = &\quad y^D_{i,k} F^D(x, x^{\text{off}}_{i,k}) + y^\Gamma_{i,k} F^\Gamma(x, x^{\text{off}}_{i,k}, a_{i,k}, b_{i,k}) \\
= &\quad \begin{cases} 0 & \text{if } x < x^{\text{off}}_{i,k} \\ y^D_{i,k} + y^\Gamma_{i,k} \dfrac{\gamma(a_{i,k}, (x - x^{\text{off}}_{i,k})/b_{i,k})}{\Gamma(a_{i,k})} & \text{if } x \geq x^{\text{off}}_{i,k} \end{cases} \\
\text{w.r.t.} \quad & y^D_{i,k} + y^\Gamma_{i,k} = y_{i,k}, \forall k
\end{aligned}
\tag{5.8}
$$

Note that the X coordinate of the offset $x^{\text{off}}_{i,k}$ is the total transmission time of the local higher priority harmonic set plus itself, and $y_{i,k}$ is its probability considering only the local message set. These two parameters can be estimated by analyzing the local message set only. Of course, if the exact local set is not known because of incomplete information, we can approximate it with the expected number of messages in the harmonic set, assuming that every message has maximum size, or considering an average message size.

We use the Quantile-quantile plot (Q-Q plot) to graphically compare the probability distributions from simulation and the fitted mixture model. 1000 random samples from each are taken, and as some examples, the results for message m_5, m_{25} and m_{69} are shown in Figure 5.5, 5.6 and 5.7 respectively.

The Q-Q plots for these messages have obvious deviation from linearity, but are approximately close to be linear, especially for m_{69}. Remember that we are using mixture model of

degenerate distribution and offsetted gamma distribution to approximate message response time, not claiming the mixture model is the actual distribution.

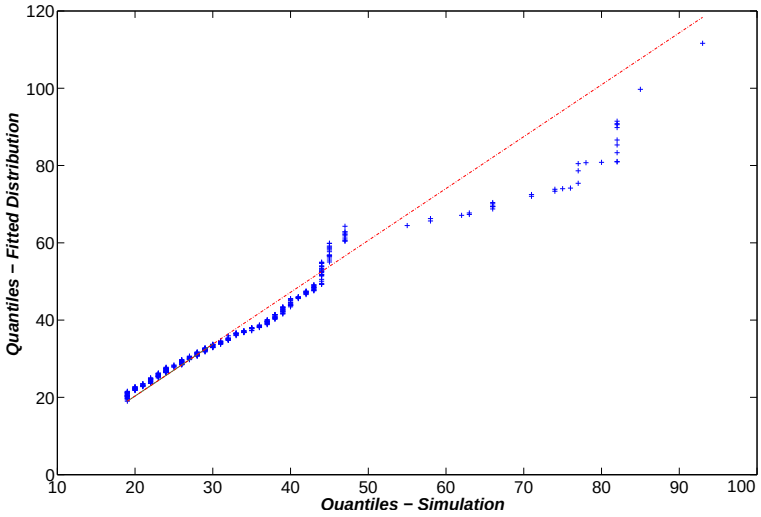

Figure 5.5: Quantile-quantile plot of samples from simulation and fitted distribution for message m_5

Also, to quantitatively evaluate our hypothesis that the message response time *cdf* can be approximated by one or multiple pairs of degenerate and offsetted gamma distributions, we present several metrics to assess the accuracy of the fitted distribution using the EM algorithm compared to the simulation data: the root mean squared error (RMSE), the coefficient of determination R^2, and the Kolmogorov-Smirnov statistic (K-S statistic, column "K-S", the maximum vertical deviation between the two *cdf* curves). The results for the messages of the reference bus are shown in Table 5.1. The maximum RMSE is 0.034, the minimum coefficient of determination is 0.980, and the maximum K-S statistic is 0.11, all indicating that our hypothesis is accurate for the purpose of an early estimation.

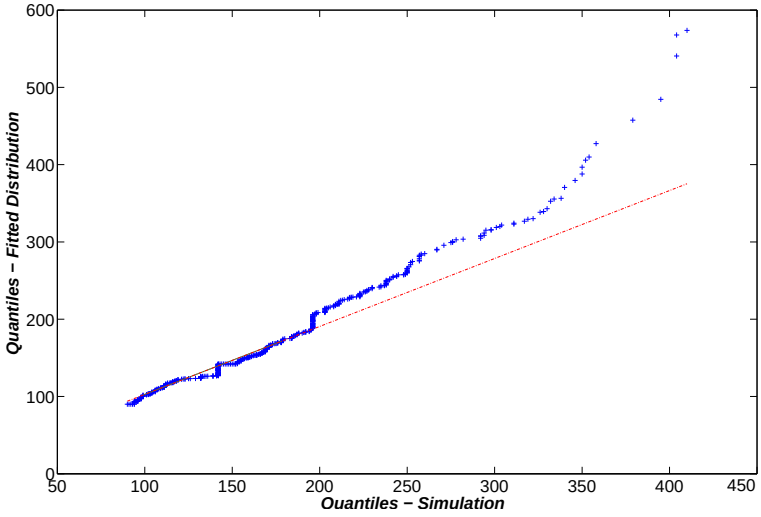

Figure 5.6: Quantile-quantile plot of samples from simulation and fitted distribution for message m_{25}

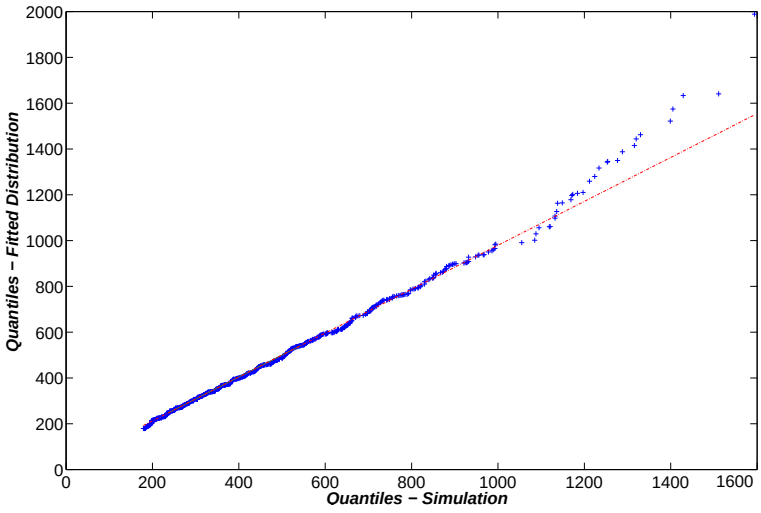

Figure 5.7: Quantile-quantile plot of samples from simulation and fitted distribution for message m_{69}

ID	RMSE	R^2	K-S	ID	RMSE	R^2	K-S	ID	RMSE	R^2	K-S
1	0.017	0.985	0.03	24	0.015	0.994	0.07	47	0.031	0.988	0.07
2	0.017	0.985	0.03	25	0.016	0.994	0.08	48	0.008	0.998	0.02
3	0.009	0.994	0.03	26	0.012	0.994	0.04	49	0.007	0.999	0.03
4	0.012	0.989	0.03	27	0.003	1.000	0.02	50	0.014	0.997	0.04
5	0.009	0.995	0.04	28	0.013	0.995	0.04	51	0.008	0.999	0.03
6	0.008	0.995	0.03	29	0.020	0.989	0.06	52	0.010	0.998	0.03
7	0.009	0.995	0.04	30	0.011	0.998	0.04	53	0.015	0.997	0.04
8	0.010	0.996	0.04	31	0.014	0.996	0.04	54	0.011	0.998	0.04
9	0.010	0.996	0.04	32	0.008	0.993	0.04	55	0.013	0.997	0.04
10	0.009	0.994	0.04	33	0.006	0.999	0.03	56	0.008	0.998	0.02
11	0.010	0.997	0.04	34	0.007	0.999	0.03	57	0.014	0.997	0.05
12	0.021	0.983	0.06	35	0.018	0.993	0.05	58	0.016	0.997	0.05
13	0.008	0.996	0.04	36	0.008	0.998	0.04	59	0.017	0.996	0.05
14	0.011	0.997	0.05	37	0.005	0.999	0.02	60	0.006	0.999	0.02
15	0.010	0.994	0.05	38	0.007	0.998	0.02	61	0.012	0.998	0.03
16	0.015	0.993	0.05	39	0.006	0.999	0.01	62	0.013	0.998	0.03
17	0.008	0.997	0.05	40	0.006	0.999	0.02	63	0.015	0.997	0.05
18	0.009	0.997	0.04	41	0.009	0.998	0.02	64	0.012	0.998	0.03
19	0.034	0.981	0.11	42	0.009	0.998	0.03	65	0.013	0.998	0.03
20	0.007	0.994	0.05	43	0.030	0.986	0.09	66	0.012	0.998	0.03
21	0.010	0.991	0.06	44	0.010	0.998	0.03	67	0.018	0.995	0.05
22	0.017	0.992	0.08	45	0.029	0.980	0.07	68	0.012	0.999	0.03
23	0.014	0.994	0.07	46	0.030	0.988	0.08	69	0.003	1.000	0.02

Table 5.1: Statistics of fitted mixture model distributions for messages on the reference bus

5.2 Estimate Parameters x^{off} and y

In order to reconstruct the response time probability distribution of a message m_i for the mixture model in Equation 5.8 , we need to estimate the parameters $(x^{\text{off}}_{i,k}, y_{i,k}, a_{i,k}, b_{i,k}, y^D_{i,k}, y^\Gamma_{i,k})$. Parameters $x^{\text{off}}_{i,k}$ and $y_{i,k}$ can be calculated by considering the interference from the local higher priority message set. Consider, for example, message m_{39} with period 5000 and transmission time 27 on node E_5. There are three higher priority messages on the same node, m_{10}, m_{15}, and m_{27}, with period and transmission time as $(27, 2500)$, $(27, 10000)$, and $(27, 2500)$, respectively. During the hyperperiod [0,10000), message m_{39} is queued at time 0 and 5000. At time 0, all three higher priority messages m_{10}, m_{15}, and m_{27} are queued, thus the minimum response time of the message instance from m_{39} is 81. At time 5000, m_{10} and m_{27} are queued together with m_{39}, the minimum response time of the message instance from m_{39} is 54. Due to the different minimum response times of these two instances from m_{39}, there are two pairs of $x^{\text{off}}_{i,k}$ and $y_{i,k}$ parameters: $(x^{\text{off}}_{39,1} = 54$, $y_{39,1} = 0.5)$ and $(x^{\text{off}}_{39,2} = 81$, $y_{39,2} = 0.5)$.

5.3 Estimate Parameters y^D and y^Γ

Since in our mixture model as in Equation 5.8, the constraint $y^D_{i,k} + y^\Gamma_{i,k} = y_{i,k}$ needs to be satisfied, we can get $y^\Gamma_{i,k} = y_{i,k} - y^D_{i,k}$ after estimating $y_{i,k}$ and $y^D_{i,k}$. In the following, we will focus on the estimate of parameter y^D.

Estimating the y^D values, however, is challenging. Figure 5.8 plots the y^D values for each of the messages. The y^D values clearly depend on the message priorities, but in a non-linear way, at least if good accuracy is required. Remember that the $y^D_{i,k}$ of message m_i is the probability that all the messages in the corresponding higher priority harmonic set and m_i itself are transmitted

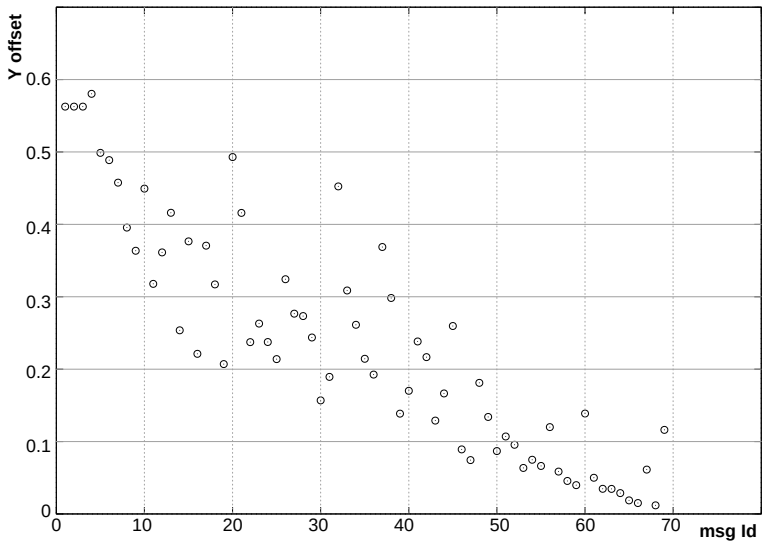

Figure 5.8: y^D values of messages on the reference bus

immediately upon enqueuing, considering the possible interference and blocking from remote load. This means that the first message in the queue finds the network idle at queuing time (no blocking nor interference), and all the others do not have to wait because of a higher priority message, as described in Figure 5.9. Finding the probability of such an event is very difficult. However, we can try to build up a parameterized model based on the underlying meaning of y^D, then use regression technique to estimate the model parameters.

The first step is to approximate the stochastic process with a binomial (Bernoulli) process with replacement. For each of the j-th local higher priority harmonic set of message m_i, we define $n_{i,j}$ as the length (number of messages) of the message queue, including the messages in its j-th higher priority harmonic set and m_i itself. Each message transmission consists of a Bernoulli trial. We assume that all transmissions take the same time e, while $q_{i,j}$ is the time instant at which m_i is queued (together with all the other messages in its j-th harmonic set). Also, we define $\mathbb{P}(idle)_t$

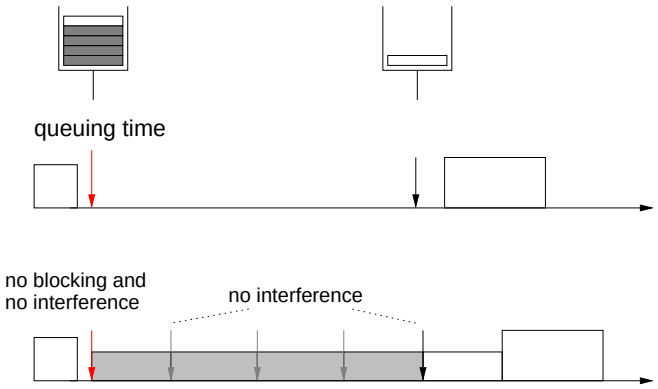

Figure 5.9: Probability of finding sufficient idle time for transmission of messages in the queue

as the probability of the bus being idle at time t, and $\mathbb{P}(idle)_{[t_1,t_2]}$ as the probability that the bus is idle in the interval $[t_1, t_2]$. Bus idle of priority level P_i at time t, denoted as $\mathbb{P}(idle, P_i)_t$, is the probability that no message with priority higher than or equal to P_i is transmitted on the bus or ready for transmission at time t. The probability that the highest local priority message in the j-th harmonic set is sent at time $q_{i,j}$ is equal to the probability that it finds the network idle which can be approximated with

$$\mathbb{P}(idle)_{q_{i,j}} = 1 - U_i^r \tag{5.9}$$

where U_i^r is the network utilization from remote nodes for m_i. The intuition is that U_i^r is the fraction of time in which the network is utilized and busy.

After the transmission of the message on top of the queue, which has transmission time equal to e, $q_{i,j} + e$ is the time at which the second message in the queue attempts transmission. In order to compute the probability that the following messages in the queue are transmitted immediately, the utilization due to higher priority messages transmitted by other nodes, U_i^{hr}, must be

used in place of U_i^r. This is because the following messages cannot experience blocking due to low priority messages if the first message is transmitted immediately. Hence, the second message is transmitted with probability

$$\mathbb{P}(idle, P_i)_{q_{i,j}+e} = 1 - U_i^{hr} \tag{5.10}$$

The same is true for all the other messages in the queue, assuming that the probability of finding the network idle at the following time instants is independent. The probability of finding an idle time for the first k messages in the queue is therefore equal to the probability that the bus is idle from $q_{i,j}$ to $q_{i,j} + (k-1)e$ and then becomes busy at the time the $k+1$ transmission is attempted, i.e.,

$$\mathbb{P}(idle)_{[q_{i,j}, q_{i,j}+(k-1)e]} = (1 - U_i^r)(1 - U_i^{hr})^{k-1} U_i^{hr} \tag{5.11}$$

The probability of the message m_i being transmitted with response time $x_{i,j}^{\text{off}}$, where $x_{i,j}^{\text{off}}$ is the minimum response time when m_i is queued with the j-th harmonic higher priority set, is the probability that there is an idle time of length sufficient for the transmission of the minimum queue or larger, i.e., the first $k \geq n_{i,j}$ trials of the Bernoulli process result in a condition of bus idle. The queue length $n_{i,j}$ is given by the number of queued higher priority local messages that are transmitted before m_i plus one (for m_i itself), i.e.,

$$\begin{aligned} y_{i,j}^D = \mathbb{P}(\mathcal{R}_i = x_{i,j}^{\text{off}}) = & \sum_{k=n_{i,j}}^{\infty} (1 - U_i^r)(1 - U_i^{hr})^{k-1} U_i^{hr} \\ = & (1 - U_i^r) U_i^{hr} \sum_{k=n_{i,j}}^{\infty} (1 - U_i^{hr})^{k-1} \end{aligned} \tag{5.12}$$

The sum of this geometric series converges and can be computed as

$$y_{i,j}^{D} = \mathbb{P}(\mathcal{R}_i = x_{i,j}^{\text{off}}) = (1 - U_i^r)(1 - U_i^{hr})^{n_{i,j}-1} \tag{5.13}$$

However, this estimate needs to be corrected in many ways. The first consideration is that queuing events that result in the queuing length as $n_{i,j}$ only occur as a fraction of all the queuing events for a given message m_i. The probability of a queuing event with the j-th harmonic set is y_j which can be evaluated by simply considering all the queuing events in the hyperperiod. Therefore, the probability that a message instance is transmitted with response time $x_{i,j}^{\text{off}}$ can be adjusted as

$$y_{i,j}^{D} = \mathbb{P}(\mathcal{R}_i = x_{i,j}^{\text{off}}) = y_{i,j}(1 - U_i^r)(1 - U_i^{hr})^{n_{i,j}-1} \tag{5.14}$$

The second consideration is that Equation 5.14 is based on a number of simplified assumptions. The first is the assignment of idle/busy status to the slots for the set of remote messages or the higher priority subset is defined with replacement, while the number of idle slots in the hyperperiod is fixed. More importantly, the assumption that the status of the bus for a transmission attempt is independent from the status at previous attempts leads to an underestimate of the actual probability of long idle times. This is because the local synchronization of the transmission time operated by the TxTask results in bursts of transmissions followed by idle intervals. Therefore, if at some point the bus is busy due to higher priority load, there is a higher priority that it will be busy also at the next bus contention attempt. Similarly, no higher priority transmission at some time means higher probability that there will be no higher priority message ready also at the next transmission of a local message. Hence, the above method provides a good estimate for short queue lengths and an underestimation of the actual probability for longer queues. In order to compensate for this correlation, we use the following heuristics: (1) use local queuing delay $Q_{i,j} = x_j^{\text{off}} - E_i$ in

place of $n_{i,j}$ in equation 5.14, since it is a more accurate estimate of the local transmission attempt. (2) when $Q_{i,j} = 0$, the $y^D_{i,j}$ value is estimated by $y_{i,j}(1 - U^r_i)$; (3) when $Q_{i,j} > 0$, we use a parameterized model of U^{hr}_i and $Q_{i,j}$ to estimate $y^D_{i,j}$. The best model in terms of accuracy we can find is as follows:

$$y^D_{i,j} = \mathbb{P}(\mathcal{R}_i = x^{\text{off}}_{i,j}) = \begin{cases} y_{i,j}(1 - U^r_i) & \text{if } Q_{i,j} = 0 \\ y_{i,j}(1 - U^r_i)e^{-\beta_1(U^{hr}_i+\beta_2)(Q_{i,j}+\beta_3)+\beta_4} & \text{if } Q_{i,j} > 0 \end{cases} \tag{5.15}$$

Figure 5.10 shows the absolute errors in the evaluation of the probability y^D of the X offset for all the messages in the set. The regression function as in Equation 5.15 results in a sufficiently accurate estimate, always within ± 0.03 of the actual probability value, and the RMSE of the estimate is less than 0.01.

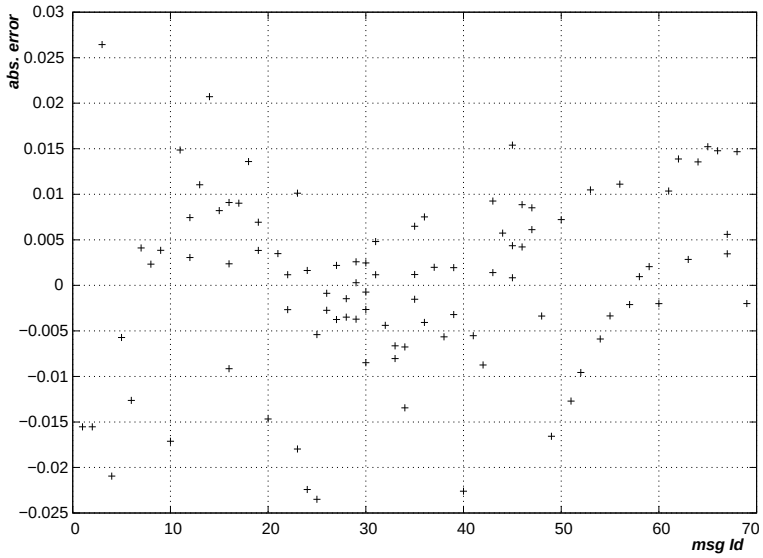

Figure 5.10: Absolute errors in the estimation of the y^D values for messages

5.3.1 Dependency on Average Message Size

Unfortunately, if the system configuration or the message set changes, the y^D values differ for the same values of the predictor parameters in Equation 5.15, meaning that these predictor parameters are not the only ones that control the y^D values. We started evaluating other parameters than can influence the y^D values, such as number of nodes in the system, number of messages for each node, average number of harmonic sets for each node, and finally, average size of messages. Of all these, the last one is the most promising.

As in CAN protocol, each standard message must be between 0 and 8 bytes. Table 5.2 shows the variation of coefficients $\beta_1 - \beta_4$ in Equation 5.15 for different average message sizes, where $Q_{i,j}$ is expressed in unit of $10\mu s$.

Avg msg size	**6.971**	**6.478**	**5.797**	**5.101**	**4.377**	**3.623**	**3.000**	**2.406**
$\beta_\mathbf{1}$	1.031	1.022	1.026	1.022	1.019	1.027	1.027	1.029
$\beta_\mathbf{2}$	9.22E-3	9.50E-3	9.69E-3	1.01E-2	1.09E-2	1.15E-2	1.21E-2	1.16E-2
$\beta_\mathbf{3}$	0.4526	0.4255	0.4056	0.3876	0.3424	0.3073	0.3034	0.3194
$\beta_\mathbf{4}$	-1.102	-2.508	-1.870	-2.409	-2.881	-1.235	-1.388	-0.359

Table 5.2: Coefficients $\beta_1 - \beta_4$ of the parameterized model of y^D for messages on the reference bus

5.4 Estimate Parameters a and b of Gamma Distributions

The parameters a and b of the gamma distribution in the fitted mixture model is computed for all the messages on the reference bus listed in Table 3.1. To find a parameterized model for a and b is very difficult, since there is no clear physical meaning for them. Instead, $\mu_{i,k} = a_{i,k} \times b_{i,k}$, the mean of a gamma distribution with parameters $a_{i,k}$ and $b_{i,k}$, is the average additional interference delay from remote nodes of message m_i given that its local queueing delay is $Q_{i,k}$, the one

corresponds to the k-th X offset $x_{i,k}^{\text{off}}$. Intuitively, among the available system design variables, $\mu_{i,k}$ must depend on the local queueing delay $Q_{i,k}$, since the longer $Q_{i,k}$ is, the larger the chance that m_i will be interfered; it also should be dependent on the higher priority utilization U_i^{hr} from remote notes, which gives the rough information of the percentage of the bus being busy because of remote higher priority messages. Note that $Q_{i,k}$ and U_i^{hr} are two independent design parameters, one from local, the other about remote.

In the following, we try to find the factors that affect the values of $\mu_{i,k}$, then build up the parameterized model and estimate the fitting coefficients. It turns out the same model can be used for $b_{i,k}$.

5.4.1 μ and b v.s. Local Queueing Delay Q

To identify the relation between μ and Q, we group messages that approximately have the same U^{hr}, but different Q values, in the hope of finding a statistical model of μ and Q. Figure 5.11 plots three groups of data with U^{hr} value of approximately 0.15, 0.38, and 0.5 respectively. It is clear from the graph that μ is linearly dependent on Q, as all the points are closely distributed along the respective linear regression lines. Similarly Figure 5.11 confirms the existence of a linear relation between b and Q.

5.4.2 μ and b v.s. Remote Higher Priority Utilization U^{hr}

Similarly, to find the relation between μ, b and the remote higher priority utilization U^{hr}, we plot three groups of data with Q values around 24, 49, 143, respectively, as in Figure 5.13 and 5.14. From the figures, the logarithm of μ and b are both approximately linearly dependent on U^{hr}.

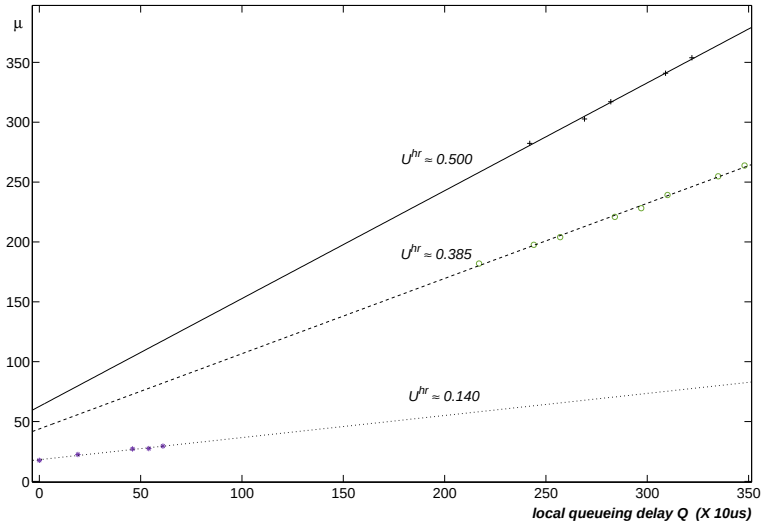

Figure 5.11: The linear relation of μ and local queueing delay Q for messages on the reference bus

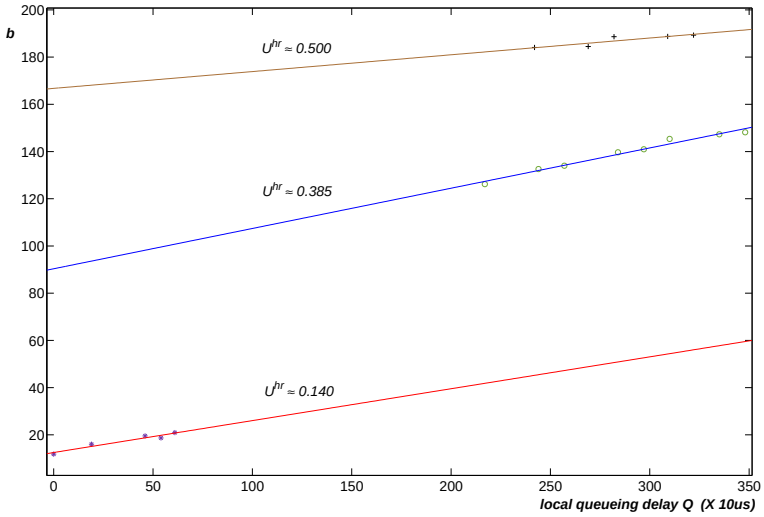

Figure 5.12: The linear relation of b and local queueing delay Q for messages on the reference bus

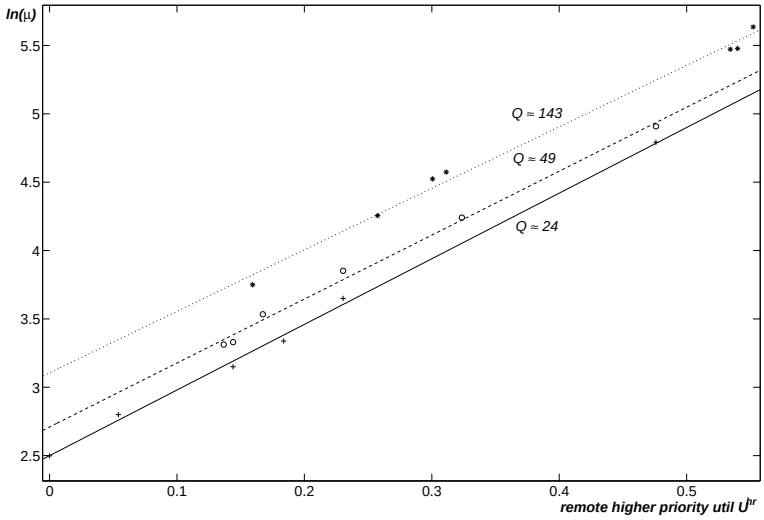

Figure 5.13: The linear relation of μ and remote higher priority utilization U^{hr} for messages on the reference bus

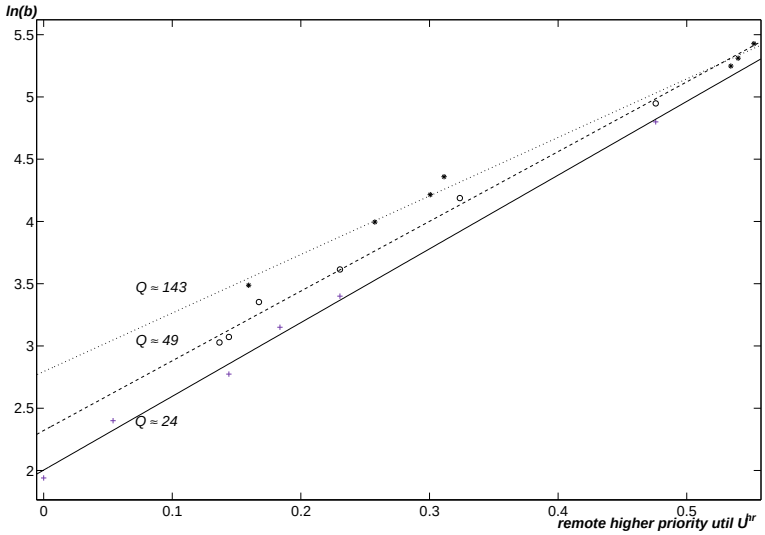

Figure 5.14: The linear relation of b and remote higher priority utilization U^{hr} for messages on the reference bus

5.4.3 Parameterized Model with Q and U^{hr}

Now we have the relation between μ and the predictor parameters Q and U^{hr}, as μ is linear to Q when U^{hr} is constant, then the following equation should be satisfied:

$$\mu_{i,k} = f_1(U_i^{hr}) \times Q_{i,k} + f_2(U_i^{hr}) \tag{5.16}$$

If the logarithm of μ is linear to U^{hr} when Q is constant, then

$$\mu_{i,k} = f_3(Q_{i,k}) \times e^{f_4(Q_{i,k})U_i^{hr}} \tag{5.17}$$

Compare Equations 5.16 and 5.17, it is clear that $f_4(Q_{i,k})$ must equals a constant coefficient, and $f_3(Q_{i,k})$ is a linear function of $Q_{i,k}$. Thus, the parameterized model for μ should be the following:

$$\mu_{i,k} = (Q_{i,k} + \beta_5)e^{\beta_6 + \beta_7 U_i^{hr}} \tag{5.18}$$

where $Q_{i,k}$ and U_i^{hr} are the predictor parameters, and $\beta_5 - \beta_7$ are the constant coefficients to be determined.

Following the same reasoning, the parameterized model for b is

$$b_{i,k} = (Q_{i,k} + \beta_8)e^{\beta_9 + \beta_{10} U_i^{hr}} \tag{5.19}$$

where $\beta_8 - \beta_{10}$ are some constant coefficients.

Figure 5.15 and 5.16 show the absolute errors in the evaluation of the μ and b values for all the messages in the set. The regression functions result in a sufficiently accurate estimate, always

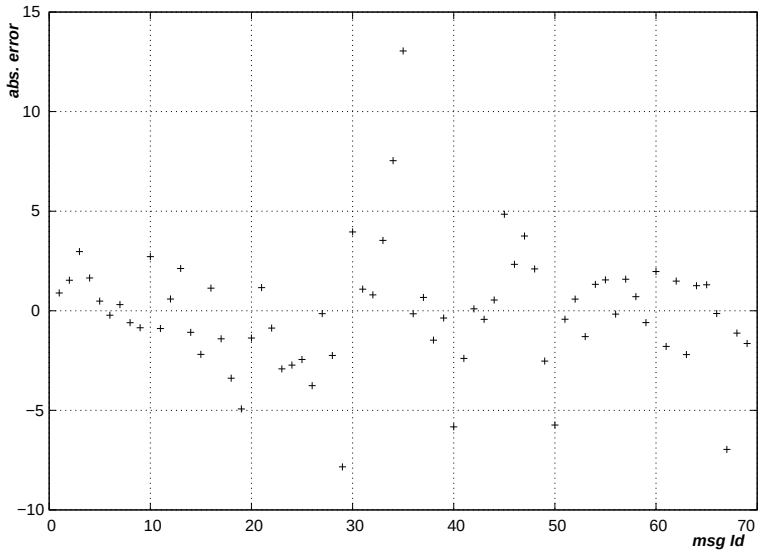

Figure 5.15: Absolute errors in the estimation of the μ values for messages on the reference bus

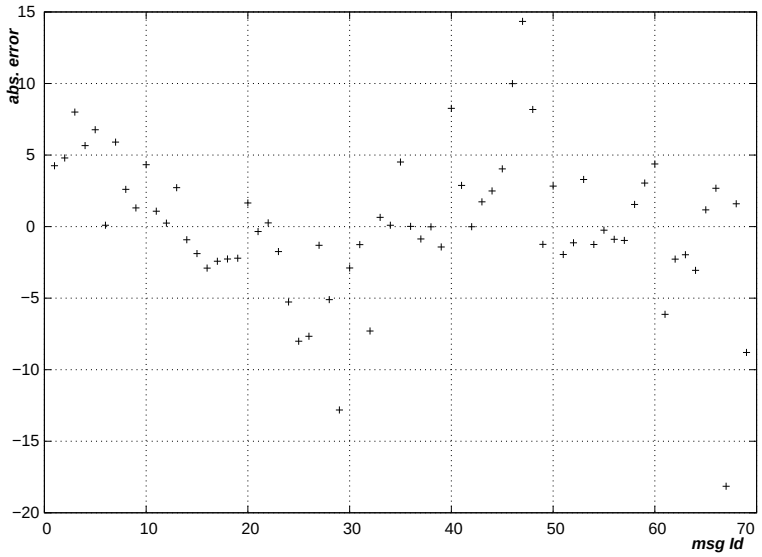

Figure 5.16: Absolute errors in the estimation of the b values for messages on the reference bus

within ± 20 ($\pm 0.2ms$) of the actual values, and the RMSE of the estimate of μ and b are 3.69 and 5.67 respectively.

5.4.4 Dependency on Average Message Size

Similar to the case of y^D, unfortunately coefficients $\beta_5 - \beta_{10}$ are different for different system configurations. Table 5.3 shows the variation of coefficients $\beta_5 - \beta_{10}$ in Equation 5.18 and 5.19 for different message sizes, where $\mu_{i,k}$, $b_{i,k}$ and $Q_{i,k}$ are expressed in unit of $10\mu s$.

Avg msg size	**6.971**	**6.478**	**5.797**	**5.101**	**4.377**	**3.623**	**3.000**	**2.406**
$\beta_{\mathbf{5}}$	105.2	113.1	123.7	132.4	143.6	146.1	158.2	160.7
$\beta_{\mathbf{6}}$	-2.248	-2.273	-2.319	-2.353	-2.467	-2.497	-2.6489	-2.730
$\beta_{\mathbf{7}}$	4.165	4.153	4.156	4.184	4.430	4.514	4.950	5.251
$\beta_{\mathbf{8}}$	506.2	498.3	462.8	473.5	452.1	411.0	428.4	389.7
$\beta_{\mathbf{9}}$	-3.506	-3.506	-3.440	-3.547	-3.609	-3.619	-3.728	-3.774
$\beta_{\mathbf{10}}$	4.244	4.272	4.220	4.440	4.784	5.027	5.424	5.913

Table 5.3: Coefficients $\beta_5 - \beta_{10}$ of the parameterized model of μ and b for messages on the reference bus

5.5 Prediction of Message Response Times

The goal of our analysis is to verify if the methods for estimating the X offsets, the y^D and y values, and the a and b parameters of the gamma distributions assumed as an approximation of the actual *cdf* of the message response time can be used to predict the response time distribution of a given message, either on the reference bus, or on other buses. The results are shown in the following figures, where the curve "simulation" is the message response time *cdf* from simulation data, "fit" is the curve using the X offsets and y values calculated from Section 5.2 combined with the y^D, a and b parameter from fitting the mixture model distribution to simulation data, "prediction" is the

one using the X offsets and y values from Section 5.2 combined with the y^D parameter from the regression function in Section 5.3 and μ, a, b parameters from the regression functions as in Section 5.4.

5.5.1 Prediction of Response Time *cdf*s for Messages on the Reference Bus

In the first experiment, we verify the capability of predicting the response time of a message on the reference bus.

The first pair, in Figure 5.17, shows the results for high priority messages. The predicted μ is close to the μ of the fitted exponential distribution, which in turn results in a quite accurate approximation of the actual *cdf*. The second pair, in Figure 5.18, is a representative of low priority messages. In these cases, the prediction is very close to the simulated data. In both cases, the error is acceptable for an early design estimation. From these figures, the quality of the approximation does not necessarily depend on the priority of the message but rather on the shape of the *cdf*. Finally, the method also allows to approximate messages with more than one higher priority harmonic set. In this case (Figure 5.19), the approximation is much better if multiple exponential distribution functions are used (one for each higher priority harmonic set) instead of only one.

The root mean squared errors, the coefficients of determination R^2, and the K-S statistics for messages on the reference bus are shown in Table 5.4. Other statistics are also accurate enough for an early stage estimation. For example, the relative error in the evaluation of the average response time for all the messages is always below 10%.

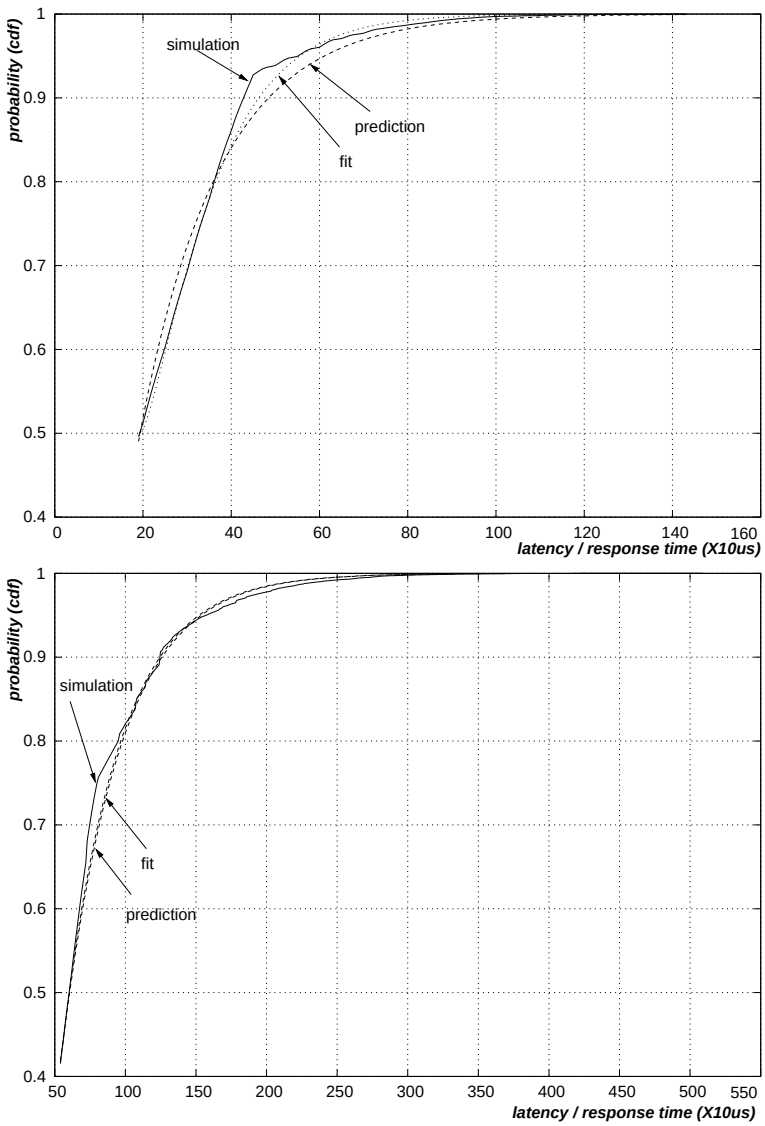

Figure 5.17: Prediction of response time *cdf*s for high priority messages m_5 and m_{21} on the reference bus

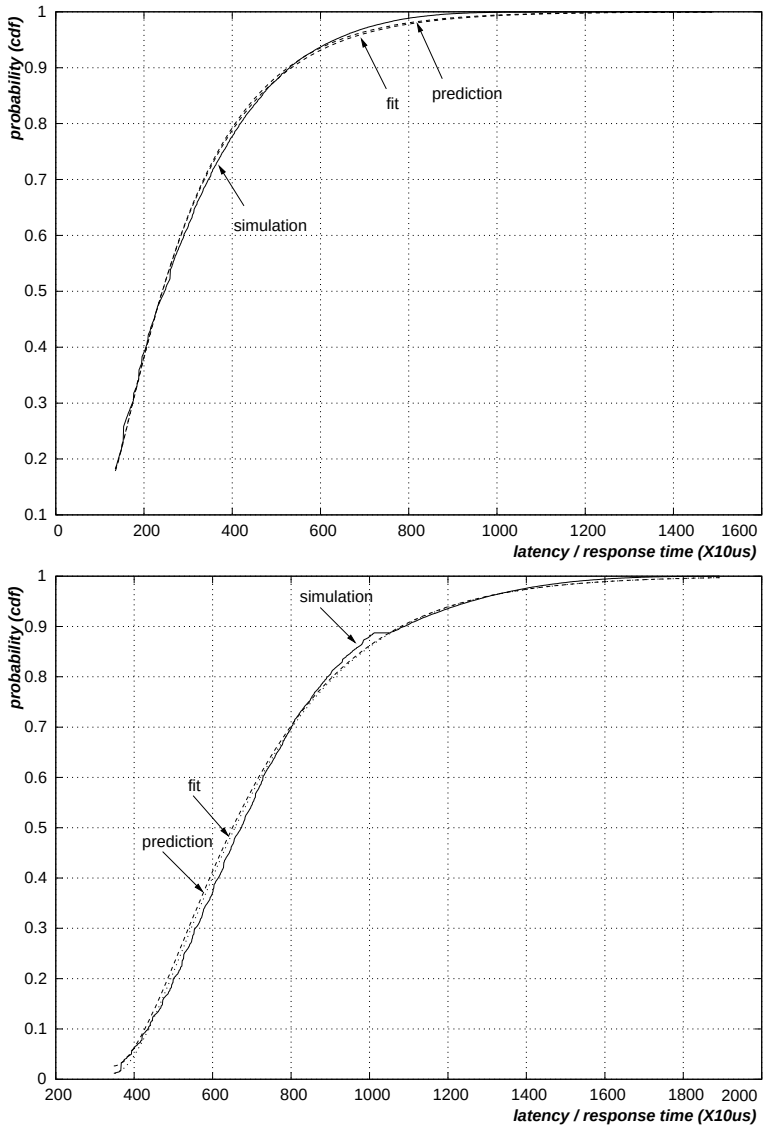

Figure 5.18: Prediction of response time *cdf*s for low priority messages m_{48} and m_{68} on the reference bus

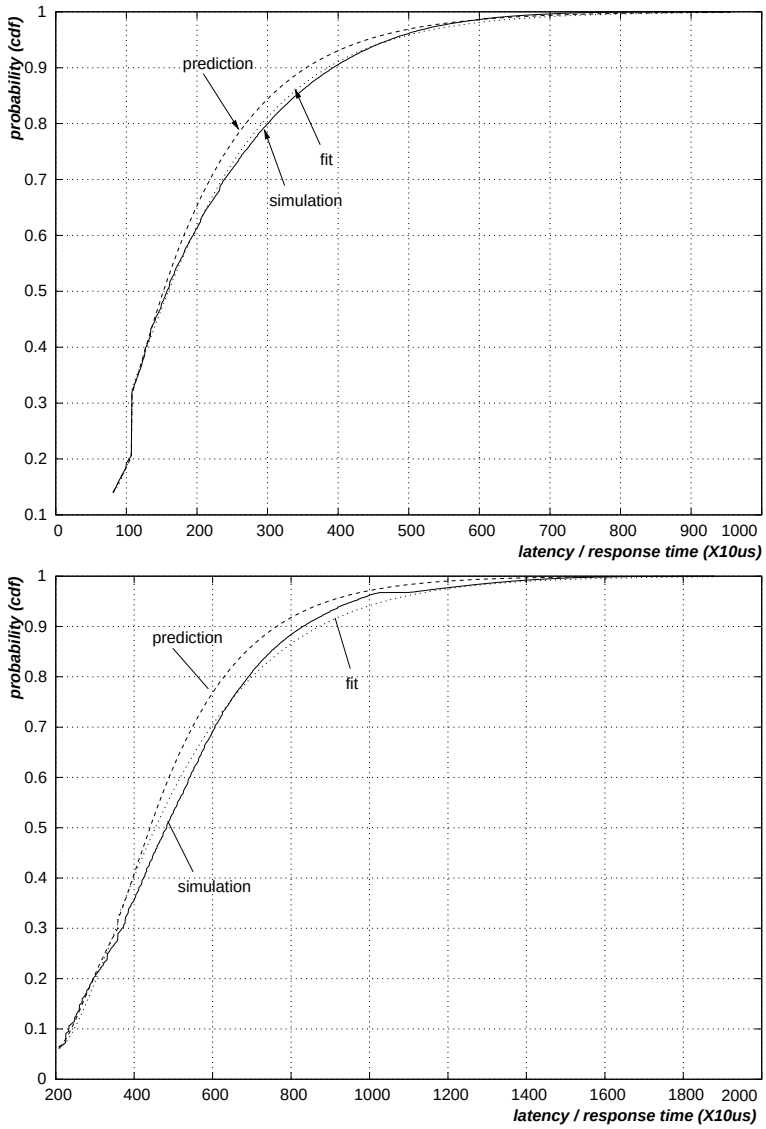

Figure 5.19: Prediction of response time *cdf*s for messages m_{39} and m_{67} with more than one harmonic set

ID	RMSE	R^2	K-S	ID	RMSE	R^2	K-S	ID	RMSE	R^2	K-S
1	0.024	0.968	0.05	24	0.007	0.999	0.04	47	0.038	0.982	0.10
2	0.028	0.959	0.06	25	0.009	0.998	0.05	48	0.008	0.999	0.02
3	0.025	0.955	0.06	26	0.015	0.991	0.05	49	0.009	0.998	0.03
4	0.021	0.965	0.06	27	0.014	0.991	0.06	50	0.019	0.994	0.05
5	0.016	0.983	0.05	28	0.019	0.989	0.05	51	0.010	0.998	0.04
6	0.009	0.994	0.03	29	0.015	0.994	0.03	52	0.010	0.998	0.04
7	0.012	0.992	0.04	30	0.016	0.994	0.04	53	0.018	0.995	0.05
8	0.009	0.997	0.03	31	0.029	0.982	0.07	54	0.012	0.998	0.03
9	0.010	0.996	0.04	32	0.012	0.985	0.06	55	0.012	0.998	0.04
10	0.020	0.966	0.08	33	0.012	0.995	0.03	56	0.010	0.998	0.02
11	0.013	0.995	0.05	34	0.012	0.996	0.03	57	0.014	0.997	0.04
12	0.008	0.997	0.03	35	0.030	0.978	0.06	58	0.016	0.997	0.05
13	0.013	0.991	0.05	36	0.013	0.996	0.05	59	0.017	0.996	0.06
14	0.016	0.994	0.06	37	0.005	0.999	0.02	60	0.006	0.999	0.02
15	0.008	0.995	0.03	38	0.008	0.998	0.03	61	0.013	0.998	0.04
16	0.004	0.999	0.02	39	0.022	0.989	0.05	62	0.014	0.998	0.04
17	0.007	0.997	0.04	40	0.010	0.998	0.03	63	0.015	0.997	0.05
18	0.013	0.994	0.06	41	0.010	0.997	0.03	64	0.013	0.998	0.04
19	0.012	0.998	0.05	42	0.009	0.998	0.04	65	0.015	0.997	0.04
20	0.006	0.996	0.04	43	0.029	0.987	0.08	66	0.016	0.997	0.04
21	0.011	0.988	0.06	44	0.010	0.997	0.03	67	0.036	0.982	0.09
22	0.011	0.997	0.05	45	0.019	0.991	0.05	68	0.016	0.997	0.04
23	0.005	0.999	0.03	46	0.035	0.984	0.09	69	0.005	0.999	0.02

Table 5.4: Error of the predicted distribution for messages on the reference bus

5.5.2 Prediction of Response Time *cdf*s for Messages on Other Buses

The $\beta_1 - \beta_{10}$ regression coefficients are estimated on the reference bus and then used to predict the response time *cdf* of messages on other buses. We used the messages on *bus2*, normalized for a bus speed of *500 kbps* to verify the quality of this assertion. Figures 5.20 and 5.21 shows some sample results for four messages with different priorities and different quality of the prediction. Figure 5.20 shows two messages (priority rank 9 and 73 out of 131) with low quality results. The curve representing the *cdf* prediction in this case is clearly separated from the simulation results and the best fitted exponential distribution. However, even in these low quality cases, the prediction retains sufficient accuracy for an early assessment. Please note that in the case of high priority messages, although the relative error is higher, the absolute error is quite low, being a message with very short worst case response time. Figure 5.21 shows two sample messages (priority rank 40 and 112 out of 131) with good quality results. The method can be quite accurate not only for messages with one higher priority harmonic set (message priority 112), but also for messages with more than one higher priority harmonic set (message priority 40). Although the accuracy typically decreases with lower priority messages, our experiments show that very good quality results can be retained for most messages in the set.

In general, the errors of the prediction from statistical analysis for the reference bus and other buses are in the same magnitude. In Figure 5.22 and 5.23, the comparison of K-S statistics and root mean squared error for each message on the reference bus and *bus2* is given, where the X axis is the total utilization of higher priority messages. Also, the curves of 9-central moving average [38] of the errors are plotted. The average (0.0145) and maximum (0.0381) RMSE for messages on *bus2* are roughly half of the ones on the reference bus (0.0271 and 0.0827, respectively), which

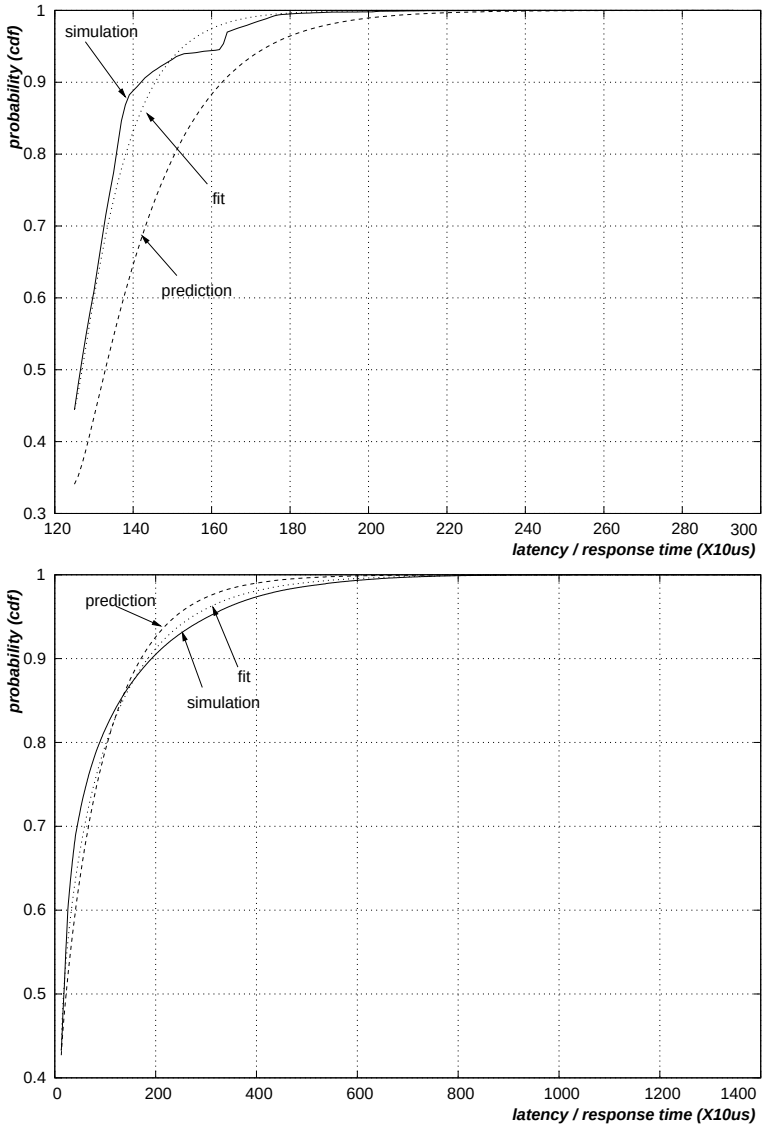

Figure 5.20: Prediction of response time *cdf*s for messages on a different bus: low quality results

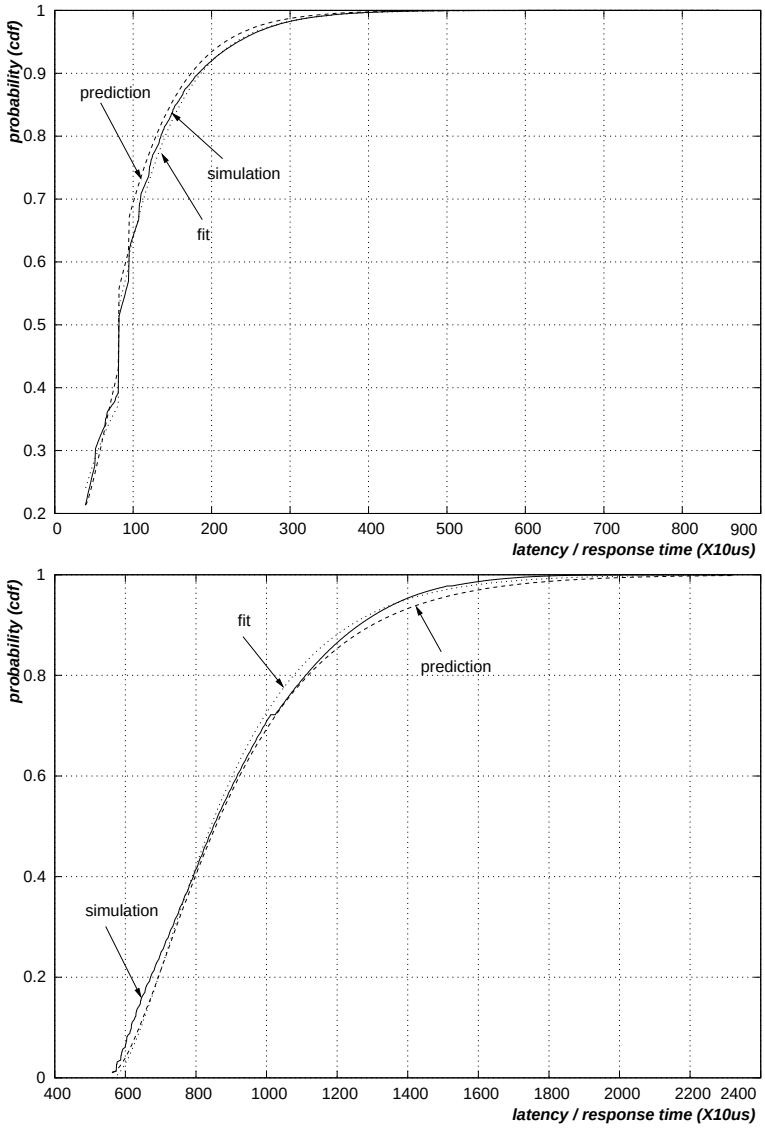

Figure 5.21: Prediction of response time *cdf*s for messages on a different bus: high quality results

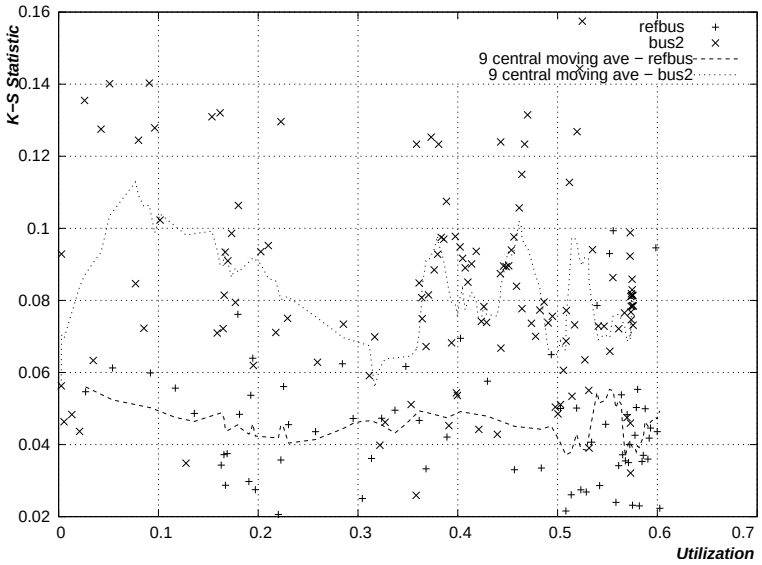

Figure 5.22: K-S statistics of statistical analysis: reference bus and *bus2*

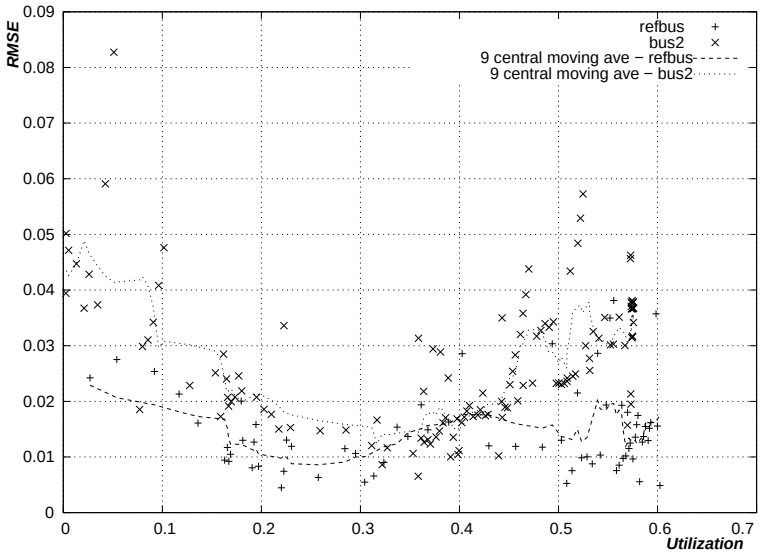

Figure 5.23: RMSE of statistical analysis: reference bus and *bus2*

makes it promising to use the $\beta_1 - \beta_{10}$ regression coefficients estimated from the reference bus to predict the response time *cdf* of messages on other buses.

5.6 Comparison of Stochastic and Statistical Analyses

In this section, we provide the trade-offs between using statistical and stochastic methods, in terms of analysis speed vs. availability of data and accuracy of results.

Input information Stochastic analysis, requires all the message characteristics in the system, including message ID (priority), its source node, period, phase, and data length. Statistical analysis, on the other hand, is suitable when only part of the message set is known as in the case of early design stage, e.g., new car architecture design when many features of the vehicle are to be added later in the design process. The required information for statistical analysis includes the local message set, the estimation of system utilization and the utilizatin from higher priority messages, and the number of nodes in the system. Of course if the exact local message set is unknown because of incomplete information, we can approximate it with the expected number of messages in the harmonic set, assuming that every message has maximum size, or considering an average message size.

Analysis error Stochastic analysis is more accurate when the complete message set is available. Figure 5.24 gives the comparison of the root mean squared error and its 9-central moving average for messages in the reference bus. As in the figure, the error of stochastic analysis is almost always half of the one of statistical analysis, no matter what the message priority is. It is clear that for high and medium priority messages with higher priority utilization less than 50%, the error is quite small

(average K-S statistic 0.0357 for stochastic analysis, 0.0452 for statistical analysis), while the error increases significantly for low priority messages from stochastic analysis. This is because for a low priority message its worst case response times is much larger than the *gcd* of message periods, i.e., the period of the TxTask, thus it has a very large possibility to suffer multiple bursts of messages queued by different activations of the TxTask. The correlation between these message queueings is not captured by stochastic analysis, which is the reason why the approximate system is different from the original model, but statistical analysis is more robust to these scenarios.

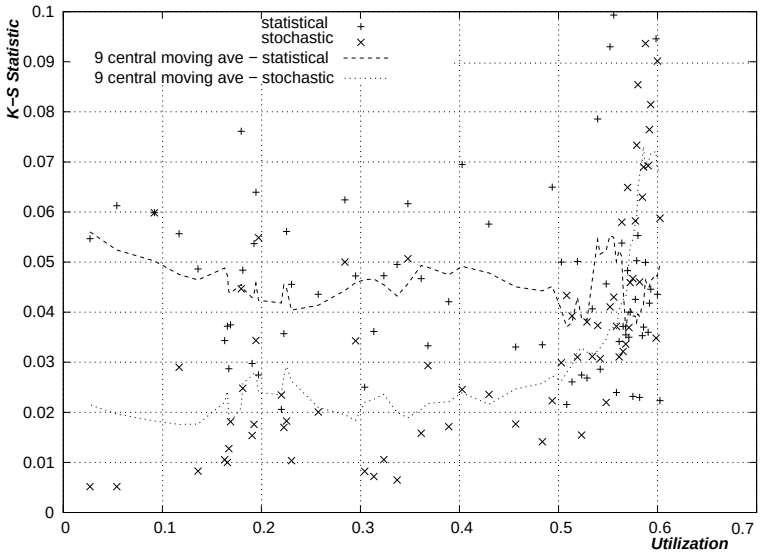

Figure 5.24: Comparison of stochastic and statistical analyses: K-S statistics

Analysis speed Stochastic analysis is more accurate when the complete message set is available. However, it is significantly slower, as shown in Table 5.5. The complexity of stochastic analysis is very sensitive to many system parameters. For example, it is worse than exponential to the number of nodes in the system, and increases cubically as the analysis granularity gets smaller. However,

	Stochastic	Statistical
# Nodes	$O(n2^n)$	$O(1)$
# Messages	$O(n)$	$O(n)$
Hyperperiod	$O(n)$	$O(1)$
Worst case response time	$O(n^2)$	$O(1)$
Analysis granularity	$O(n^{-3})$	$O(1)$
Runtime on the example system in Table 3.1	< 19 min each; 2 hours total	< 0.01 ms each; 0.24 ms total

Table 5.5: Comparison of stochastic and statistical analyses: analysis complexity

statistical analysis is independent from almost all the system parameters, since it involves only the evaluation of several closed form formulas. The runtime information[1] on the example system as in Table 3.1 is also compared, which indicates that statistical analysis is more than 10^7 faster than stochastic analysis.

[1] Java implementation on laptop with 1.6GHz CPU and 1.5G RAM

Chapter 6

Conclusions and Future Work

In this dissertation, we presented a stochastic analysis framework for the end-to-end latency of distributed real-time automotive systems. The previous work on periodic tasks with deterministic activation times on a single priority based preemptive CPU is extended to accommodate mixed preemptive scheduling in OSEK operating system standard. We proposed a stochastic analysis framework for computing the *pmf*s of CAN message response times in systems with unsynchronized ECUs and periodic messages. We provided a characterization of message interferences using a single message per node, defined by a random queuing jitter and transmission time. We compose the response times of tasks and messages with the sampling delays caused by information passing by shared variables among periodically activated tasks and messages. The experimental section with data extracted from an experimental vehicle proves the applicability of the analysis.

Our claim on statistical analysis for CAN message response times is two folds. First, it is possible to predict the response time probability distribution of a CAN message (for a user-defined priority) response time when the only information available is the bus utilization on the set

of higher priority messages. Second, it is possible to make the same prediction when the utilization is known for a message set on a different bus, using the different bus as an estimator of the traffic load, although with obviously less accuracy. We believe these results are of crucial importance while designing new car architectures when only partial information is available to the designers (for instance the existing bus utilization or the maximum queue length on the bus adapter side). In fact, designers can explore different priority assignments for a new message and evaluate different probability distributions of the message response times for the different priority assignments. In return, the distributions, as they represent the timing behavior of a message for a given priority, can be used in an automatic design process and synthesis flow supporting message priority assignment.

As for future work, the first piece is to perform statistical analysis of software task response times and later end-to-end latencies. This requires a good set of test benches that is the representative for automotive applications, by which we can characterize the typical task execution time distributions. Based on that, we can analyze the main statistics of the distributions of task response times. Also, it is interesting to extend the analysis framework to tasks and messages whose requests are produced by a sender (data-driven).

Second, as the automotive electronics continue to grow, the current manual analysis and design process is becoming increasingly impractical and error-prone [36, 40]. As advocated in the platform-based design methodology [10, 18, 39, 42], a much better approach is to automatically map the set of tasks onto the platform guaranteeing the correct functionality and timing with optimal resource utilization. Many of the recent researches are targeted to take the design description at the pure functional level with performance and other constraints along with the architecture of the platform, and produce correct configurations for the architecture layers including middleware

and communication network. Racu et al. [37] discuss algorithms based on binary search techniques to optimize priority and period assignments with respect to a number of constraints, including end-to-end deadlines, utilization on resources, and output jitter. Davare et al. [12] automatically assign task and message periods for distributed automotive systems based on geometric programming to minimize the total worst case response times over all objects in the system, or other metrics related to extensibility of the solution. Zheng et al. [29, 53] propose a synthesis procedure based on approximate timing analysis to optimize the selection of purely periodic and the data driven activation models in the functional network with respect to the latency constraints. The objective function can be either the number of event buffers, or the slack between the worst case latencies and the deadlines. In another work, Zheng et al. [54] optimize the task placement and the signal to message mapping and automate the assignment of priorities to tasks and messages in order to meet end-to-end deadline constraints, while minimizing the total worst case end-to-end latencies for all the paths.

However, none of the previous work, to the best of our knowledge, has used average end-to-end timing performance as the metric to compare different architecture and mapping choices. Regardless, the probability of the worst case end-to-end latency value is very small. As we showed in previous chapters, this probability is usually much smaller than 10^{-12}, which is hardly the representative of the typical timing performance of the system. Once the probability distribution of the end-to-end latency is quickly characterized, it would be very promising to see how it fits in the automatic mapping and configuration design process.

Bibliography

[1] Autosar consortium web page. *http://www.autosar.org*.

[2] Iso 11898-1. road vehicles - interchange of digital information - controller area network (can) for high-speed communication. *ISO Standard-11898, International Standards Organisation (ISO)*, November 1993.

[3] Osek/vdx time-triggered operating system specification version 1.0. *http://www.osek-vdx.org*, July 2001.

[4] Osek/vdx binding specification version 1.4.2. *http://www.osek-vdx.org*, July 2004.

[5] Osek/vdx communication specification version 3.0.3. *http://www.osek-vdx.org*, July 2004.

[6] Osek/vdx network management: Concept and application programming interface version 2.5.3. *http://www.osek-vdx.org*, July 2004.

[7] Osek/vdx operating system specification version 2.2.3. *http://www.osek-vdx.org*, February 2005.

[8] Flexray. protocol specification v2.1 rev. a. *http://www.flexray.com*, 2006.

[9] Ian Broster, Alan Burns, and Guillermo Rodríguez-Navas. Probabilistic analysis of can with faults. In *Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS'02)*, pages 269–278, 2002.

[10] Henry Chang, Larry Cooke, Merrill Hunt, Grant Martin, Andrew J. McNelly, and Lee Todd. *Surviving The SOC Revolution: A Guide To Platform-based Design*. Kluwer Academic Publishers, Norwell, MA, USA, 1999.

[11] Jeffrey A. Cook, Ilya V. Kolmanovsky, David McNamara, Edward C. Nelson, and K. Venkatesh Prasad. Control, computing and communications: Technologies for the twenty-first century model t. *Proceedings of the IEEE, Special Issue on Automotive Power Electronics and Motor Drives*, 95(2):334–355, 2007.

[12] Abhijit Davare, Qi Zhu, Marco Di Natale, Claudio Pinello, Sri Kanajan, and Alberto Sangiovanni-Vincentelli. Period optimization for hard real-time distributed automotive systems. In *Proceedings of the 44th annual conference on Design automation (DAC '07)*, pages 278–283, June 2007.

[13] Robert I. Davis, Alan Burns, Reinder J. Bril, and Johan J. Lukkien. Controller area network (can) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*, 35(3):239–272, 2007.

[14] José Luis Díaz, Daniel F. García, Kanghee Kim, Chang-Gun Lee, Lucia Lo Bello, José María López, Sang Lyul Min, and Orazio Mirabella. Stochastic analysis of periodic real-time systems. In *Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS'02)*, pages 289–302, 2002.

[15] Mark K. Gardner. Probabilistic analysis and scheduling of critical soft real-time systems. *UIUC, Phd Thesis, Computer Science, University of Illinois at Urbana-Champaign*, 1999.

[16] Mark K. Gardner and Jane W.-S. Liu. Analyzing stochastic fixed-priority real-time systems. In *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS '99)*, pages 44–58, 1999.

[17] M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal (British Computer Society)*, 29(5):390–395, October.

[18] K. Keutzer, S. Malik, R. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli. System level design: Orthogonalization of concerns and platform-based design. *IEEE Transactions on Computer-Aided Design*, 19(12), December 2000.

[19] Bart Kienhuis, Ed F. Deprettere, Pieter van der Wolf, and Kees A. Vissers. A methodology to design programmable embedded systems - the y-chart approach. In *Embedded Processor Design Challenges: Systems, Architectures, Modeling, and Simulation - SAMOS*, pages 18–37. Springer-Verlag, 2002.

[20] Jong Kim and Kang G. Shin. Execution time analysis of communicating tasks in distributed systems. *IEEE Transactions on Computers*, 45(5):572–579, May 1996.

[21] Kanghee Kim, José Luis Díaz, Lucia Lo Bello, José María López, Chang-Gun Lee, and Sang Lyul Min. An exact stochastic analysis of priority-driven periodic real-time systems and its approximations. *IEEE Transactions on Computers*, 54(11):1460–1466, 2005.

[22] Hermann Kopetz, Andreas Damm, Christian Koza, Marco Mulazzani, Wolfgang Schwabl,

Christoph Senft, and Ralph Zainlinger. Distributed fault-tolerant real-time systems: The mars approach. *IEEE Micro*, 9(1):25–40, 1989.

[23] John P. Lehoczky. Real-time queueing theory. In *Proceedings of the 17th IEEE Real-Time Systems Symposium (RTSS'96)*, pages 186–195, December 1996.

[24] John P. Lehoczky. Real-time queueing network theory. In *Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS'97)*, pages 58–67, December 1997.

[25] John P. Lehoczky, Lui Sha, and Ye Ding. The rate-monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings of the 10th IEEE Real-Time Systems Symposium (RTSS'89)*, pages 166–171, Santa Monica, CA USA, December 1989.

[26] Wang Lei, Zhaohui Wu, and Mingde Zhao. Worst-case response time analysis for osek/vdx compliant real-time distributed control systems. In *Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC'04)*, pages 148–153, 2004.

[27] Sorin Manolache. Schedulability analysis of real-time systems with stochastic task execution times. *PhD Thesis, Department of Computer and Information Science, IDA, Linkoping University*, 2002.

[28] Sorin Manolache, Petru Eles, and Zebo Peng. Schedulability analysis of multi-processor real-time applications with stochastic task execution times. In *Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design (ICCAD '02)*, pages 699–706, New York, NY, USA, 2002. ACM.

[29] Marco Di Natale, Wei Zheng, Claudio Pinello, Paolo Giusto, and Alberto Sangiovanni-Vincentelli. Optimizing end-to-end latencies by adaptation of the activation events in distributed automotive systems. In *Proceedings of the 13th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS '07)*, pages 293–302, April 2007.

[30] N. Navet and Y.-Q. Song. Valuation de performances de la messagerie can du vhicule prototype psa - action 1 du contrat psa-crin. In *Technical report, CRIN*, Dresden, Germany, 1996.

[31] 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*, 46(7):607–617, 2000.

[32] Nicolas Navet, Yeqiong Song, Francoise Simonot-Lion, and Cédric Wilwert. Trends in automotive communication systems. *Proceedings of the IEEE*, 93(6):1204–1223, 2005.

[33] Thomas Nolte, Hans Hansson, and Christer Norström. Probabilistic worst-case response-time analysis for the controller area network. In *Proceedings of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 200–207, Washington, DC, USA, May 2003.

[34] Thomas Nolte, Hans Hansson, Christer Norström, and Sasikumar Punnekkat. Using bit-stuffing distributions in can analysis. In *Proceedings of the IEEE/IEE Real-Time Embedded Systems Workshop (RTES)*, London, UK, December 2001.

[35] J.C. Palencia and M. Gonzáles Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *Proceedings of the 19th IEEE Real-Time Systems Symposium (RTSS'98)*, pages 26–37, December 1998.

[36] Razvan Racu, Arne Hamann, Rolf Ernst, and Kai Richter. Automotive software integration. In *Proceedings of the 44th Annual Conference on Design Automation (DAC '07)*, pages 545–550, 2007.

[37] Razvan Racu, Marek Jersak, and Rolf Ernst. Applying sensitivity analysis in real-time distributed systems. In *Proceedings of the 11th IEEE Real Time on Embedded Technology and Applications Symposium (RTAS '05)*, pages 160–169, 2005.

[38] Sheldon M. Ross. *Introduction to Probability Models, Ninth Edition*. Academic Press, Inc., Orlando, FL, USA, 2006.

[39] Alberto Sangiovanni-Vincentelli. Defining platform-based design. *EEDesign of EETimes, http://www.eedesign.com/story/OEG20020204S0062*, February 2002.

[40] Alberto Sangiovanni-Vincentelli. Integrated electronics in the car and the design chain evolution or revolution? In *Proceedings of the conference on Design, Automation and Test in Europe (DATE '05)*, pages 532–533, 2005.

[41] Alberto Sangiovanni-Vincentelli. Quo vadis, sld? reasoning about the trends and challenges of system level design. *Proceedings of the IEEE*, 95(3):467–506, 2007.

[42] Alberto Sangiovanni-Vincentelli, Luca Carloni, Fernando De Bernardinis, and Marco Sgroi. Benefits and challenges for platform-based design. In *Proceedings of the 44th Annual Conference on Design Automation (DAC '04)*, pages 409–414, 2004.

[43] Alberto Sangiovanni-Vincentelli and Marco Di Natale. Embedded system design for automotive applications. *Computer*, 40(10):42–51, 2007.

[44] Lui Sha, Ragunathan Rajkumar, and John P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE transactions on Computers*, 39(9):1175–1185, September 1990.

[45] Richard P. Stanley and Jim Pitman. A polytope related to empirical distributions, plane trees, parking functions, and the associahedron. *Discrete and Computational Geometry*, 27:603–634, 2002.

[46] K. Tindell, A. Burns, and A. J. Wellings. Calculating controller area network (can) message response times. *Control Engineering Practice*, 3(8):1163–1169, 1995.

[47] Ken Tindell. Real time systems and fixed priority scheduling. *Technical Report Tech. rept. DoCS 95/notes.Uppsala University. Uppsala, Sweden*, March 1995.

[48] Amy J. C. Trappey and David W. Hsiao. Applying collaborative design and modularized assembly for automotive odm supply chain integration. *Computers in Industry*, 59(2-3):277–287, 2008.

[49] Guoqiang Gerald Wang, Marco Di Natale, and Alberto Sangiovanni-Vincentelli. An osek/vdx implementation of synchronous reactive semantics preserving communication protocols. In *Workshop on Operating Systems Platforms for Embedded Real-Time applications*, pages 58–67, July 2007.

[50] Yun Wang and Manas Saksena. Scheduling fixed-priority tasks with preemption threshold. In *Proceedings of the Sixth International Conference on Real-Time Computing Systems and Applications (RTCSA '99)*, pages 328–337, 1999.

[51] Haibo Zeng, Marco Di Natale, Paolo Giusto, and Alberto Sangiovanni-Vincentelli. Statistical analysis of controller area network message response times. In *Submitted to 15th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS '09)*.

[52] Haibo Zeng, Marco Di Natale, Paolo Giusto, and Alberto Sangiovanni-Vincentelli. Stochastic analysis of distributed real-time automotive systems. *Submitted to IEEE Transactions on Industrial Informatics, Special Section on: "Real-Time and (Networked) Embedded Systems"*.

[53] Wei Zheng, Marco Di Natale, Claudio Pinello, Paolo Giusto, and Alberto Sangiovanni-Vincentelli. Synthesis of task and message activation models in real-time distributed automotive systems. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '07)*, pages 93–98, April 2007.

[54] Wei Zheng, Qi Zhu, Marco Di Natale, and Alberto Sangiovanni-Vincentelli. Definition of task allocation and priority assignment in hard real-time distributed systems. In *Proceedings of the 28th IEEE International Real-Time Systems Symposium (RTSS '07)*, pages 161–170, December 2007.

Appendix A

Alphabetic Notations

Throughout the dissertation, we use calligraphic typeface to denote random variables. For example, O_i denotes a normal variable, while $\mathcal{O}_i$ is a random variable.

- τ: granularity
- τ_i: the i-th task
- $\Gamma_{i,j}$: the j-th job of task τ_i
- $\Pi_{i,j}$: path from object o_i to o_j
- Υ_i: shared resource i
- $A_{i,j}$ ($\mathcal{A}_{i,j}$): arrival time of job $\Gamma_{i,j}$, or message instance $M_{i,j}$
- $\mathcal{B}_{i,j}$: blocking time of job $\Gamma_{i,j}$, or message instance $M_{i,j}$
- $D_{i,j}$ ($\mathcal{D}_{i,j}$): delay between object o_i and o_j

- $D_{i,j}^k$ ($\mathcal{D}_{i,j}^k$): delay of the data consumed by object instance $o_{j,k}$ and the instance of o_i who produces this data
- E_i ($\mathcal{E}_i$): execution time of task τ_i, or transmission time of message m_i
- $\mathcal{F}_{i,j}$: finish time of job $\Gamma_{i,j}$, or message instance $M_{i,j}$
- G_k^P ($\mathcal{G}_k^P$): P-level backlog at the beginning of the k-th hyperperiod, i.e. $\mathcal{W}_{((k-1)H)^-}^P$
- H: hyperperiod
- $hp(P)$: $\{\tau_i(m_i)|P_i < P\}$, i.e. the set of tasks (messages) with priority higher than P
- $I(P,t)$: P-level jobs or message instances at time t
- J_i ($\mathcal{J}_i$): release jitter of task τ_i, or queuing jitter of message m_i
- $\mathcal{L}_{i,j}$: end-to-end latency associated with path $\Omega_{i,j}$
- $lp(P)$: $\{\tau_i(m_i)|P_i > P$ and $N_i = non - preemptable\}$, i.e. the set of lower priority non-preemptable tasks (messages)
- m_i: the i-th message
- $M_{i,j}$: the j-th message instance of m_i
- N_i: preemption property of task τ_i, or message m_i
- O_i ($\mathcal{O}_i$): initial phase of task τ_i, or message m_i
- $O_{i,j}^k$ ($\mathcal{O}_{i,j}^k$): relative phase of $o_{j,k}$ and the previous instance of o_i
- o_i: the i-th object, i.e. task τ_i or message m_i

- P_i: priority of task τ_i, or message m_i
- $Q_{i,j}$ ($\mathcal{Q}_{i,j}$): release time of job $\Gamma_{i,j}$, or queuing time of message instance $M_{i,j}$
- $R_{i,j}$ ($\mathcal{R}_{i,j}$): response time of job $\Gamma_{i,j}$, or message instance $M_{i,j}$
- $S(P,t)$: P-level event space at time t
- T_i: period of task τ_i, or message m_i
- U: utilization, U^{max}: maximum utilization
- $V_t^{i,j}$: event $\mathcal{Q}_{i,j} > t$
- $\overline{V}_t^{i,j}$: the complement event of $V_t^{i,j}$, i.e. event $\mathcal{Q}_{i,j} \leq t$
- $\mathcal{W}_t^P$: P-level backlog at time t
- $\mathcal{W}_t^{\Gamma_{i,j}}$: backlog of job $\Gamma_{i,j}$ at time t
- $\mathcal{W}_t^{M_{i,j}}$: backlog of message instance $M_{i,j}$ at time t
- $\mathcal{X}_t$: queuing pattern at time t
- $\overline{\mathcal{X}}_{t(i,j)}$: the complement queuing pattern of $\mathcal{X}_t$ with respect to $\Gamma_{i,j}$ or $M_{i,j}$