Czech Technical University in Prague, Czech Republic
Faculty of Electrical Engineering
Department of Control Engineering

Resource Reservation and Analysis in Heterogeneous and Distributed Real-Time Systems

Doctoral Thesis

by

Michal Sojka

Prague, August 2010

Ph.D. Programme: *Electrical Engineering and Information Technology*
Branch of study: *Control Engineering and Robotics*

Advisor: *Doc. Dr. Ing. Zdeněk Hanzálek*
Department of Control Engineering
Czech Technical University in Prague

To my wife Lucie.

Acknowledgements

First of all, I would like to express my thanks to my thesis advisor, Zdeněk Hanzálek, who supported me both personally and financially during the work on this thesis. Another person who deserves thanks is my colleague Pavel Píša. Technical discussion with him was always a great source of inspiration for me and it definitely led to higher quality of this thesis. And last, but not least, I would like to thank my students who worked with me on various real-time related projects. The experience from these mostly practical projects had also positive influence on the work in this thesis.

Research leading to the results in this thesis has been supported by the European Commission under grant agreement n.FP6/2005/IST/5-034026, in the context of the FRESCOR Project and by the Ministry of Education of the Czech Republic under project 1M0567 (CAK).

Czech Technical University in Prague *Michal Sojka*
August 2010

Abstract

This thesis describes the design, implementation and evaluation of a software framework that facilitates development of real-time, possibly distributed, applications. The basic idea of the framework is to let the application developer specify the temporal (and resource) requirements of his/her application and the framework guarantees keeping of these requirements, provided that there are enough resources in the system. In the case of insufficient resources, the framework does not let the application run. Application requirements are specified in the so called *service contract* that the application negotiates with the framework. A successfully negotiated contract results in creation of a *virtual resource*, which represents "a part" of the real resource reserved for the use by the application. To not over-reserve the available resources, the framework employs on-line admission tests that are based on state-of-the-art schedulability analysis. One of the main strengths of presented framework is its modularity with respect to support of additional resources, which is shown by integration of six different resources (CPU, network, etc.) into the framework. The prototype implementation of the framework was developed under Linux operating system and it was extensively evaluated on both synthetic tests and real-world multimedia application.

Keywords: real-time, middleware, schedulability analysis

Goals and Objectives

The main goals of this work have been set as follows.

1. Design and implement a modular software framework supporting resource reservations on heterogeneous resources for distributed real-time applications. The framework should be easily extensible with support for new resources and should allow task migration between resources.
2. Evaluate the framework on a real multimedia application.
3. Develop and evaluate an admission test for wireless network (Wi-Fi) to be used in the framework.
4. Formulate schedulability analysis for tasks with offsets as an integer linear programming problem and evaluate the performance on analyzing multiprocessor and distributed systems.

Contents

List of Figures

List of Tables

List of Acronyms

AC Access Category
ACK Acknowledge
AIFS Arbitration Interframe Space
AIFSN Arbitration Interframe Space Number
AP Access Point
API Application Programming Interface
BFQ Budget Fair Queuing
CAN Controller Area Network
CBS Constant Bandwidth Server
CORBA Common Object Request Broker Architecture
CPU Central Processing Unit
CSMA/CA Carrier Sense Multiple Access/Collision Avoidance
CTS Clear to Send
CW Contention Window
DIFS Distributed (Coordination Function) Interframe Space
DM Deadline-Monotonic
E-ASAP Extended Adaptive Slot Assignment Protocol
EDCA Enhanced Distributed Channel Access
EDF Earliest Deadline First
ERP Extended Rate PHYs
FCB FRSH Contract Broker
FORB FRSH Object Request Broker
FOSA FRSH Operating-System Abstraction
FPGA Field Programmable Gate Array
FPS Fixed Priority Scheduling
fps frames per second
FRA FRSH Resource Allocator
FRM FRSH Resource Manager
FWP FRSH WLAN Protocol
GSP Generic Scheduler Patch
ILP Integer Linear Programming
IP Internet Protocol
ITEM Integrated TDMA and E-ASAP
JTAG Joint Test Action Group, common name for IEEE 1149.1: "Standard Test Access Port and Boundary-Scan Architecture"
LAN Local Area Network
LLC Logical Link Control
LP Linear Programming
MAC Medium Access Control

MDE	Model Driven Engeneering
MP	Mathematical Programming
MTU	Maximum Transmission Unit
OOP	Object Oriented Programming
OS	Operating System
PCP	Priority Ceiling Protocol
PHY	physical (layer)
PIP	Priority Inheritance Protocol
PLCP	Physical Layer Convergence Protocol
POSIX	Portable Operating System Interface [for Unix]
QoS	Quality of Service
RM	Rate-Monotonic
RMA	Rate-Monotonic Analysis
RT	Real-Time
RTOS	Real-Time Operating System
RTP	Real-Time Transport Protocol
RTS	Request to Send
SIFS	Short Inter-Frame Space
SMP	Symmetrical Multiprocessor
STA	Station in Wi-Fi network
TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access
TOS	Type of Service
UDP	User Datagram Protocol
VRES	Virtual Resource
WCET	Worst-Case Execution Time
WLAN	Wireless LAN
WSN	Wireless Sensor Network

1 Introduction

Approximately 90% of all microprocessors is now used in embedded systems and their number is still rapidly increasing. Many of embedded systems are real-time systems, which means that they must react to various events within precise temporal constraints. As the embedded system platforms grow and becomes more and more complex, it is more difficult to develop software satisfying the required temporal constraints. Furthermore, industry is seeking for cost effective development process and short time-to-market, which is traditionally achieved by allowing the designers to work at higher levels at which the design is abstracted from unnecessary details. The details are encapsulated in independently developed components, which provide the desired functionality and the goal is to reuse as most components as possible among projects.

In the context of real-time systems, the major concern is how to ensure satisfaction of temporal constraints. When the system is simple, there exist many well known methods for checking that the system satisfies the required temporal constraints. However, todays systems are not simple, they are often distributed, composed from heterogeneous hardware, the requirements on those system dynamically change in the course of system run-time, the software which is run on these systems is complex and so on. For these reasons, ensuring temporal correctness of such systems is a very complex and expensive process. Component middleware platforms used in other areas of software industry usually deal only with functional properties and lack the support for temporal (also called non-functional) properties. When such a middleware platform is used to develop a real-time application, the violation of temporal constraints is often detected at the very last development stage. It is clear that such a finding prolongs the development schedule and increases development costs.

Although the real-time system research community developed many techniques for analyzing temporal properties of complex real-time systems, these techniques are not widely used in industry for several reasons. The use of the methods is often too

complicated, the methods expect unrealistic conditions such as task independence, or the methods are tailored to only one part of the real-time system, leaving other parts unanalyzed. Another big problem of real-time analysis techniques is that they expect the *Worst-Case Execution Time* (WCET) to be known. With modern hardware, this is almost impossible [McGuire et al., 2009] and overestimating WCET leads to pessimism in the results of analysis and increases the cost of the system.

As can be seen from the previous paragraphs, the current development methodologies targeting real-time systems have many limitations. Therefore, the following is a set of challenges for future design methodologies.

Cost-effective development of real-time systems. It is agreed that a way to lowering the development costs is to facilitate software reuse and to decrease the need for extensive testing at the last stages of the development process. The use of upcoming "resource-aware" component middleware platforms providing temporal isolation of components and the use of model-driven engineering approaches [Schmidt, 2006] will allow reaching these goals.

Dynamically changing resource requirements and availability. In todays real-time systems, the resource demands often change dynamically over time and are not known a priori. Also the resource availability may change over time, e.g. because of the need to save power. For these reasons real-time applications need a capability to adapt to changing conditions in a way that does not violate their temporal requirements in an uncontrollable manner.

Integration of design optimization and schedulability analysis into development process. In dynamic systems described in the previous paragraphs, it is quite complicated to assure the optimal use of available resources on one side and satisfying the temporal constraints of such systems on the other side. The future real-time systems will utilize on-line admission tests and optimization procedures to achieve both these properties.

1.1 Contribution

This thesis presents a framework which addresses certain aspects of the above mentioned challenges. In its essence, the framework provides temporal isolation of tasks running on various resources and facilitates the resource management in dynamic and distributed real-time applications. Due to these properties, it could be used as a run-time platform of a component-based middleware. Depending on the properties of the underlying platform, the framework could support both hard and soft real-time applications, but this thesis focuses more on soft real-time applications as the underlying platform is Linux.

In particular, the contributions of this thesis are the following.

Highly modular resource reservation framework supporting heterogeneous resources. The set of resources which are utilized in real-time systems

(CPUs, networks, etc.) differs from project to project. The framework presented in this thesis enables easy extension of the resources it supports. This is demonstrated by integration of a wide range of resources used in real-time systems into the framework. Currently integrated resources are: CPU (various schedulers), hard disk, various wired and wireless networks and *Field Programmable Gate Arrays* (FPGAs).

Evaluation of the framework on the real multimedia application. The framework was used to develop a distributed multimedia application resembling a video surveillance system. Based on the experience with this application we report on the practical usability strengths and weaknesses of the framework

Wireless network support in resource reservation framework. Wireless networks represent a challenge for real-time systems since the temporal guarantees provided by the underlying network layers are very limited. In this thesis, we describe how wireless *Local Area Networks* (LANs) based on IEEE 802.11e (QoS enabled Wi-Fi) standard can be integrated into a resource reservation framework and we evaluate the presented approach. We also describe the integration of wireless sensor networks into the framework.

Integer programming-based conditions for schedulability of fixed-priority tasks with offsets. Schedulability analysis approaches based on mathematical programming are gaining popularity among real-time researches because they allow direct integration of schedulability analysis into general design optimization processes. In this thesis we derive conditions for schedulability analysis of tasks with offsets and show their applicability to the schedulability analysis of multiprocessor and distributed systems. This kind of advanced schedulability analysis could be possibly integrated into the resource reservation framework to make it easily applicable in the industry.

1.2 Structure of the Thesis

This thesis is structured as follows: Chapter 2 introduces the basic terms and concepts that are used throughout this thesis. In Section 2.1 we mention the basics of real-time computing i.e. common scheduling algorithms, analysis techniques etc. Then we discuss distributed real-time systems and their challenges in Section 2.2 and conclude with Section 2.3 covering high-level approaches to real-time application development such as component-based development and model-driven engineering.

Chapter 3 describes the architecture and general principles of the developed resource reservation framework. In Section 3.1 we give our motivation for designing our framework. Then, the basic architecture of the framework is presented in Section 3.2. We follow with Section 3.3 describing the advanced concepts and important internal details of the framework. This chapter is concluded by the mathematical formalism used to formulate optimization problems to be solved by the framework.

Chapter 4 describes the resources supported by the framework. We start in Section 4.1 by CPU resource which is supported by integration of AQuoSA architecture [Palopoli et al., 2009] and continue with description of disk resource in Section 4.2. Both these resources were contributed by researches from Scuola Superiore Sant'Anna, Italy. In Section 4.3 we describe how wireless LANs are supported and the experiments, which lead to the current design of this support. Integration of wireless sensor networks is shown in Section 4.5 and we close this chapter by describing the support for coprocessors in FPGA.

The evaluation of the performance, properties and usability of the framework is provided in Chapter 5 . First, the overhead of the negotiation process is evaluated in Section 5.1, then we evaluate the Wi-Fi resource support in Section 5.2. This section is completed by Section 5.3 where we present an integrated case-study comprising of a multimedia application utilizing three different resources simultaneously.

Chapter 6 deals with *Integer Linear Programming* (ILP) formulation of schedulability analysis for tasks with offsets. Section 6.1 introduces the computational model and Section 6.2 recapitulate existing formulation published in [Palencia and González Harbour, 1998]. The ILP problem is formulated Section 6.3 and its performance is evaluated on experiments in Section 6.4.

Finally, the thesis ends with a conclusion in Chapter 7 summarizing the main contributions of the presented work.

2

Basic Concepts and State-of-the-Art

This chapter introduces the concepts on which the work in this thesis is based. The main topic in this thesis is real-time systems so this chapter starts with a brief overview of real-time systems theory in Section 2.1. Since the framework presented in this thesis is intended to be used in distributed real-time systems, Section 2.2 describes the advantages of distributed real-time systems as well as the challenges it imposes on real-time software. Finally, Section 2.3 provides an overview of model-driven engineering and component based development methodologies, which are promising approaches to development of real-time systems that reduce the development complexity and cost on one side and increase the reliability of the resulting systems on the other side.

2.1 Real-Time Computing

Real-Time (RT) systems are computing systems that must react within precise time constraints (deadlines) to events in the environment. As a consequence, the correct behavior of these systems depends not only on the results of the computations but also on the time at which the results are produced [Stankovic and Ramamritham, 1989]. Examples of applications that require real-time computing include:

- chemical and nuclear plant control,
- automotive applications,
- flight control systems,
- multimedia and virtual-reality systems,
- telecommunication systems,

– robotics and

– industrial automation.

There are two basic categories of real-time systems – *hard real-time systems* and *soft real-time systems*. Deadline miss in a hard real-time system has catastrophic consequences so such systems must be designed to always meet their deadlines. In soft real-time systems a certain amount of deadline misses can be tolerated although they are not desirable. Typical example of a soft real-time are multimedia applications.

Since real-time systems are often used in critical applications, where a failure of the system is very dangerous, it is necessary to verify the system before it is run for the first time in the target environment. For the real-time systems it is important to analyze whether the scheduling of activities in the system satisfies all required timing constraints. The analyzed system has to be modelled and then an *schedulability analysis* technique is used to analyze the properties of the model. Section 2.1.1 deals with models of real-time systems and some approaches to schedulability analysis are covered in Section 2.1.2.

2.1.1 A Model of Real-Time System

The definition of a real-time system model can be very complicated if we want the model to describe all possible real-time systems. Such a model can be found in [Liu, 2000]. For the purpose of this chapter I will introduce a basic model covering the most typical real-time systems and refine the model in the subsequent chapters where it is necessary.

In general, the model of a real-time system consists of tasks, resources and algorithms that determine how the resources are managed. Resources can be of two major types: *active resources* (processors) and *passive resources*.

Active Resources

Active Resources can execute tasks and each task, in order to be executed, needs at least one processor. Typical examples or processors are CPUs, networks, disks or databases. Some systems consist of only one processor (mono-processor systems) whereas others contain more processors, possibly of different type. If all the processors are of the same type, the system is called *Symmetrical Multiprocessor* (SMP). An example of a system with different types of processors is a distributed control system where there are CPUs executing computation tasks and network(s) "executing" communications tasks.

Passive Resources

Passive resources are additional resources in the system that cannot directly execute tasks, but a task may require such resource in addition to the processor in order to make progress. Typical examples of passive resources are memory, shared data accessed in mutually-exclusive manner or sequence numbers (in networks).

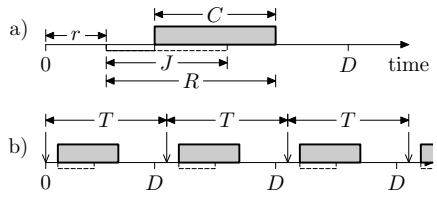

Figure 2.1: Parameters of a task; a) a non-periodic task, b) a periodic task

Tasks

Tasks in the system represent the workload that needs to be done by the system in order to perform its desired functionality. Various tasks will be denoted by Greek letter τ. Temporal properties of tasks are characterized by various parameters. The most typically used parameters are:

Release time r is the time at which the task enters the system (is activated) and is ready to be executed. Release time can be fixed or varying. In the latter case it is usually given by lower and upper bounds and the difference between these two values is called *release-time jitter*.

Execution time C (also know as *computation time*) is the amount of time required to complete the task when it executes alone and has all the resources it needs. In this thesis, it is assumed that this parameter has the meaning of the *Worst-Case Execution Time* (WCET), i.e. the actual execution time can be less than the value of this parameter.

Deadline D of a task is an instant in time by which the execution of the task is required complete. The deadline can be relative or absolute. *Relative deadline* is defined as the difference between the absolute deadline and some other point in time (usually release time).

Response time R is the length of time interval from release of the task to the instant when it completes.

Graphical representation of task parameters is shown in Figure 2.1a. The task in the figure has release time r and release jitter J so the actual release time occurs always between r and $r + J$. C is the execution (computation) time and R is the response time. If the task completes its execution before its deadline D (i.e. $R \leq D$) it is said to be *schedulable*. The whole system is schedulable if and only if all tasks in the system are schedulable.

For the purpose of this thesis the concept of a *periodic task* is very important. Periodic tasks are tasks activated periodically with a fixed period T as depicted in Figure 2.1b. Each activation of the task releases the execution of one instance of that task, which is called a *job*. All parameters of jobs are the same as of its associated task but some of them such as release time or deadline are treated as being relative to the beginning of the period.

Algorithms

Algorithms in the context of the real-time system can be divided into two classes.

The first class comprises *scheduling algorithms*, whose purpose is to assign tasks to active resources for execution. These algorithms are typically executed *on-line*, while the system is running. This is different from so called *off-line* (clock-driven or time-triggered) scheduling where the schedule is computed in advance and during run-time the tasks are executed according to the pre-computed schedule . Scheduling algorithms can be either preemptive or non-preemptive. *Preemptive* means that the currently executing task on a processor can be preempted in the course of its execution if the scheduler decides to execute another task on that processor. Tasks scheduled in non-preemptive manner cannot be preempted; as long as a task is being executed it always completes before another task has a chance to run. The most common scheduling algorithms are described in Section 2.1.1.

Algorithms in the other class control how the tasks access the passive resources. This class include algorithms like memory allocators or concurrency control algorithms. While memory allocators are out of the scope of this thesis, some of the most common concurrency control algorithms are described in Section 2.1.1.

The selection of algorithms for use in a real-time system has significant influence to temporal properties of the system.

Scheduling Algorithms

The most common scheduling algorithm for CPU in todays *Real-Time Operating Systems* (RTOSs) is *Fixed Priority Scheduling* (FPS) algorithm. Under this algorithm each task is assigned a unique fixed priority. The scheduler always chooses to execute the task with the highest priority among all tasks that are ready to be run.

In the system scheduled by a fixed priority scheduler it is very important how are the priorities assigned to the tasks. There exist several priority assignment algorithms which are optimal in some sense:

The *Rate-Monotonic* (RM) priority assignment algorithm assigns priorities to the tasks according to their periods: the shorter period, the higher priority. For a set of independent tasks with deadlines equal to their respective periods, where task priorities are assigned in rate-monotonic manner, the fixed priority scheduler produces optimal schedule in the sense that if the system is schedulable under some priority assignment, it is also schedulable under rate-monotonic priority assignment [Liu, 2000].

The *Deadline-Monotonic* (DM) priority assignment assigns priorities to the tasks according to their deadlines: the shorter deadline, the higher priority. This priority assignment is optimal in the same sense as the RM assignment even if deadlines are shorter the respective periods.

When the FPS algorithm is used together with RM priority assignment, it is sometimes referred to as RM algorithm for short. In the same way, FPS together with DM assignment is referred to as DM algorithm.

Another, quite often used, on-line real-time scheduling algorithm is *Earliest Deadline First* (EDF). This algorithm is sometimes referred to as one of the *dynamic*

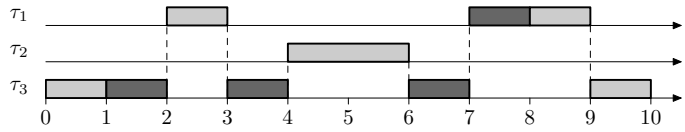

Figure 2.2: An example of priority inversion; dark color means that the task is in the critical section.

priority algorithms because it can be implemented on top of the fixed (static) priority scheduler by changing the priorities dynamically at run-time. This algorithm chooses the task to be executed as the one with the earliest deadline.

EDF has many advantages over FPS [Buttazzo, 2005] but as opposed to FPS, it is not widely used in industrial real-time operating system.

Concurrency Control Algorithms

This section deals with algorithms that determine how the concurrently tasks can access passive resources in a mutually exclusive way. In the case of shared data the access is typically guarded by a semaphore (mutex), which ensures mutual exclusion. When a task wants to access the shared data guarded by a semaphore, it must first lock the semaphore. Then it enters the *critical section* of the code where it can access the shared data. The critical section ends when the semaphore is unlocked. Concurrency control algorithms are typically defined as a *resource access-control protocol*, which is a set of rules that govern (1) when and under which conditions each request for resource is granted and (2) how tasks requiring these resources are scheduled. The reason for having such protocols is that these protocols make the execution of concurrently running tasks more deterministic by e.g. preventing priority inversion. Some of these protocols are capable of preventing deadlocks [Liu, 2000].

Priority inversion occurs when two tasks share a mutually exclusive accessed resource. In the example from Figure 2.2 a low priority task τ_3 starts accessing a resource at time 1 (dark rectangle). At time 2, it is preempted by high priority task τ_1. That task needs to access the resource at time 3 but as the resource is now accessed by τ_3, τ_1 is blocked and τ_3 continues execution. Unfortunately, at time 4, τ_3 is interrupted by middle priority task τ_2, which now blocks not only the lower priority task τ_3 but also the high priority task τ_1, because it is blocked by τ_3. Task τ_2 finishes its execution at time 6 and task τ_3 continues and releases the resource at time 7. Then the high priority task τ_1 can finally continue its execution by acquiring the access to the resource. The problem with priority inversion is that the high priority task can suffer unbounded blocking by lower priority tasks even if those tasks do not access the shared resource.

A basic protocol for avoiding the priority inversion problem is called *Priority Inheritance Protocol* (PIP). The protocol is described by a simple rule: If a lower priority task τ_l is blocking a higher priority task τ_h, then τ_l is executed with the priority of τ_h. It is said that τ_l "inherits" the priority of τ_h. The priority inheritance protocol prevents unbounded blocking.

Another protocol is called *Priority Ceiling Protocol* (PCP) and offers some advantages with respect to PIP. It prevents deadlocks and the worst-case blocking time is smaller then for PIP. One disadvantage is that a system designer has to assign every resource a ceiling value (see bellow). The PCP rules are as follows:

- Each task has a static default priority assigned (perhaps by the rate monotonic algorithm).
- Each resource has a static ceiling value defined. This is the maximum priority of the tasks that use it.
- A task has dynamic priority that is the maximum of its own static priority and any it inherits due to its blocking higher-priority tasks.
- A task can only lock a resource if its dynamic priority is higher than the ceiling of any currently locked resource (excluding any that it has already locked itself).

The important property of these resource access protocols is that the maximum time a task is blocked due to accessing a passive resource is bounded and can be simply calculated. This is different from the case when no resource access protocol is used, in which case the maximum blocking is potentially infinite.

The maximum time a task τ_i is blocked by another task due to accessing a shared resource is called *blocking term* B_i. For PIP the blocking term can be calculated according to

$$B_i = \sum_{r \in \text{used}(i)} C(r), \tag{2.1}$$

where used(i) is the set of shared resources accessed by task τ_i and $C(r)$ is the worst-case execution time of the critical section of resource r.

For PCP, the blocking term is calculated as follows:

$$B_i = \max_{r \in \text{used}(i)} C(r). \tag{2.2}$$

As can be seen from comparison of (2.1) and (2.2) PCP can offer smaller blocking term.

2.1.2 Schedulability Analysis Techniques

The purpose of schedulability analysis is to determine whether the model of a real-time system is schedulable i.e. whether the temporal constraints are always satisfied.

One of the simplest schedulability analysis techniques is *utilization-based analysis*. Utilization of a periodic task is the number $u = \frac{C}{T}$ (the ratio of computation time and period) and *system utilization* is the sum of the utilizations of all tasks in the system. If the system of independent tasks with deadlines equal to periods is

scheduled by rate-monotonic algorithm and satisfies inequality (2.3), then the system is schedulable [Liu, 2000].

$$U = \sum_{i=1}^{N} \frac{C_i}{T_i} \leq N(2^{1/N} - 1) \tag{2.3}$$

In this equation N is the total number of tasks in the system and C_i and T_i are computation times and periods of individual tasks. The right side of the inequality is called *utilization bound* and for $N \to \infty$ it is approximately equal to 0.693. Condition (2.3) is sufficient but not necessary so if the utilization of the model is greater than this bound, the system might or might not be schedulable.

This limitation of this method can be overcame by response-time analysis, which computes worst-case response times of tasks in the system and compare them with task deadlines. If all deadlines are met, the system is schedulable. Various response time analysis methods differ in the complexity of the model they are able to analyze. Often the model of the system is not precise enough and response-time analysis of the model gives too pessimistic results. In that case when, according to this method, the model is not schedulable the real system may or may not be schedulable and another technique must be used to obtain a less pessimistic answer.

One of the response-time analysis techniques is described in the following subsection and another – more advanced – is described in chapter 6.

Rate-Monotonic Analysis

As was written above, the *Rate-Monotonic Analysis* (RMA) technique calculates task worst-case response time. It assumes a system of independent tasks scheduled by RM algorithm.

Definition 1 A task τ *busy period* is an interval during which the processor is busy processing task τ or higher priority tasks.

If there is another job of either task τ or another higher priority task activated at the same time as the busy period would have ended if that job had not been there, then the busy period ends at the completion time of the previous job and another busy period begins at that same time.

Worst-Case busy period is the longest possible busy period. □

Definition 2 *Critical instant* of a task τ is an instant at which the worst-case busy period of task τ starts. □

Theorem 1 (from [Liu, 2000]) *In a fixed-priority system where every job completes before the next job of the same task is released, a critical instant of any task τ occurs when a job of this task is released at the same time as jobs of all higher-priority tasks.* □

When the critical instant of task τ_i is known the worst-case response time of that task R_i can be computed as

$$R_i = C_i + I_i, \tag{2.4}$$

where I_i is the interference from all higher priority tasks. To determine the value of I_i the number of activations of each of higher priority tasks during the task τ_i busy period have to be calculated. The number of activations of higher priority task τ_h is calculated as

$$n_{i,h} = \left\lceil \frac{R_i}{T_h} \right\rceil \tag{2.5}$$

where half square brackets represent *ceiling* operation. The total interference from that task is

$$I_{i,h} = n_{i,h} C_h = \left\lceil \frac{R_i}{T_h} \right\rceil C_h \tag{2.6}$$

Substituting (2.6) in (2.4) we get the following equation for response time R_i:

$$R_i = C_i + \sum_{h \in \mathrm{hp}(i)} \left\lceil \frac{R_i}{T_h} \right\rceil C_h, \tag{2.7}$$

where $\mathrm{hp}(i)$ is the set of tasks with priority higher than task τ_i.

This equation has the unknown variable R_i at booth sides and the right-hand side is a non-linear expression thanks to the ceiling operation. The equation can be solved by using the following recurrence formula:

$$w_i^{n+1} = C_i + \sum_{h \in \mathrm{hp}(i)} \left\lceil \frac{w_i^n}{T_h} \right\rceil C_h \tag{2.8}$$

The sequence $w_i^0, w_i^1, \ldots, w_i^n, \ldots$ is monotonically non-decreasing. When $w_i^{n+1} = w_i^n$ the solution to equation (2.7) has been found: $R_i = w_i^n$ [Burns and Wellings, 2001].

Response Time and Blocking

When tasks in the system are not independent and need to synchronize their access to shared passive resources in mutually exclusive manner, equation (2.7) no longer holds because tasks may suffer blocking from a lower-priority tasks as described in Section 2.1.1. In that case it is possible to calculate blocking term B_i, which represents the longest possible blocking time of the task and the equation (2.7) turns into

$$R_i = C_i + B_i + \sum_{h \in \mathrm{hp}(i)} \left\lceil \frac{R_i}{T_h} \right\rceil C_h \tag{2.9}$$

and can be solved the same way as equation (2.7).

Timed Automata-Based Response-Time Analysis

Another approach to response-time analysis is the approach based on timed automata. *Time automaton* is an extension to deterministic finite automaton, which

can evolve not only depending on its inputs but also on the time [Alur and Dill, 1994]. Using timed automata, it is possible to model the scheduler of an operating system, task executed in it as well as the environment [Waszinowski and Hanzálek, 2003]. As the time automaton can model the internal structure of the task, e.g. conditional branches, the response-time analysis based on the theory of timed automata can be very exact. The drawback is that the analysis suffer from combinatorial explosion and thus can only be used to analyze small systems.

2.1.3 Server-Based Scheduling

For real-time systems composed only of periodic activities, it is usually sufficient to use the scheduling algorithms described in Section 2.1.1. However, many system must deal with soft real-time activities whose temporal parameters are not known in advance. It might be that the rate of invocation changes between invocations (aperiodic/sporadic tasks) or the *Worst-Case Execution Time* (WCET) is not known exactly. One possibility of dealing with such activities is to execute them as tasks with low priority (under FPS) or with infinite deadline (under EDF) so that these activities cannot "steal" the processing resource from the hard real-time activities. However, if some guarantees are required even for such tasks (e.g. "when the aperiodic task arrives not faster than once per second, it will be completed by deadline of 10 milliseconds"), such approach may not work and one has to use something different. And this is what server-based scheduling offers.

Scheduling servers protect the processing resources needed by hard real-time tasks, but otherwise allow aperiodic/sporadic tasks to run as soon as possible. There exist many types of servers. All servers limit the capacity of the resource available to the tasks and differ in a way how this capacity is consumed and replenished.

Scheduling server are used in the framework described in the next chapter as a mean for providing temporal isolation between tasks. A buggy task running under the server cannot influence timing properties of another tasks in an uncontrollable way.

Deferrable Server

Deferrable server [Lehoczky et al., 1987] is designed for fixed-priority systems and is defined by two parameters: period T_s and execution budget B_s.

Consumption rule: The budget is consumed at the rate of one unit time whenever the server executes.

Replenishment rule: The execution budget is set to B_s at time instants kT_s for $k = 1, 2, \ldots$.

From the rules, it can be seen that if there was a non-zero budget before replenishment, that budget is lost, i.e. the unused budget doesn't cumulate from period to period.

From the schedulability analysis point of view, response-time analysis of systems using deferrable servers can accomplished similarly as shown in Section 2.1.2. The response time equation (2.4) has to be extended to count with the interference I_s

caused by the server:

$$R_i = C_i + I_i + I_s. \tag{2.10}$$

When the server is executed at the highest priority,

$$I_s = B_s + \left\lceil \frac{R_i - B_s}{T_s} \right\rceil B_s. \tag{2.11}$$

By comparing this expression with (2.6), it can be seen that the interference caused by a deferrable server is higher then the interference caused by a periodic task with execution time B_s and period T_s. This complicates the use of deferrable servers in cases when another schedulability analysis developed for periodic tasks needs to be employed.

Sporadic Server

Sporadic server [Sprunt et al., 1989] was designed with the goal of having the same properties as a periodic task with parameters corresponding to the server parameters, which are execution budget B_s and period T_s. The rules of sporadic server allow the budget to be consumed and replenished in chunks rather than as a whole as in the case of deferrable server. The sporadic server is defined by the following rules [Liu, 2000]:

Notation. The rules bellow use this notation: t_r denotes the latest (actual) replenishment time. t_f denotes the first instant after t_r at which the server begins to execute. t_e denotes the latest effective replenishment time. At any time t, BEGIN is the beginning instant of the earliest busy interval among the latest contiguous sequence of busy intervals of the tasks with higher-priority than the server that started before t. (Two busy intervals are contiguous if the later one begins immediately after the earlier one ends.) END is the end of the latest busy interval in the above defined sequence if this interval ends before t and equal to infinity if the interval ends after t.

Breaking of execution budget into chunks:

1. Initially, the budget $= B_s$ and $t_r = 0$.
2. Whenever the server is suspended, the last chunk of budget being consumed just before suspension, if not exhausted, is broken up into two chunks: The rst chunk is the portion that was consumed during the last server busy interval, and the second chunk is the remaining portion. The rst chunk inherits the next replenishment time of the original chunk. The second chunk inherits the last replenishment time of the original chunk.

Consumption rules:

1. The server consumes the chunks of budget in order of their last replenishment times.

2. The server consumes its budget only when it executes.

Replenishment rules:

1. At time t_f, if END $= t_f$, $t_e = \max(t_r, \text{BEGIN})$. If END $< t_f$, $t_e = t_f$. The next replenishment time is set at $t_e + T_s$.
2. The next replenishment occurs at the next replenishment time, except under the following conditions. Under these conditions, replenishment is done at times stated below.
   (a) If the next replenishment time $t_e + T_s$ is earlier than t_f, the budget is replenished as soon as it is exhausted.
   (b) If the system becomes idle before the next replenishment time $t_e + T_s$ and becomes busy again at t_b, the budget is replenished at $\min(t_e + T_s, t_b)$.
3. The chunks are consolidated into one whenever they are replenished at the same time.

The sporadic server scheduling policy is included in *Portable Operating System Interface [for Unix]* (POSIX) standard under the name SCHED_SPORADIC. The rules of POSIX sporadic server were modified to lower the algorithmic complexity of the implementation, however, recently it turned out, that the POSIX sporadic server does not have the always the same effect as a simple period task [Stanovich et al., 2010].

Constant Bandwidth Server

It is possible to implement sporadic server even in systems scheduled by EDF, but for such systems there exist servers which are much simpler. A very popular server is the *Constant Bandwidth Server* (CBS) [Abeni and Buttazzo, 1998], which guarantees that the server does not contribute to the resource utilization more than by a defined fraction, even in the case that the real WCET of jobs executed by the server is greater than declared. The authors of CBS define it by the following rules:

1. A CBS is characterized by a budget c_s and an ordered pair (Q_s, T_s), where Q_s is the maximum budget and T_s is the period of the server. The ratio $U_s = Q_s/T_s$ is denoted as the server bandwidth. At each instant, a fixed deadline $d_{s,k}$ is associated with the server. At the beginning $d_{s,k} = 0$.
2. Each served job $J_{i,j}$ is assigned a dynamic deadline $d_{i,j}$ equal to the current server deadline $d_{s,k}$.
3. Whenever a served job executes, the budget c_s is decreased by the same amount.
4. When the budget $c_s = 0$, the server budget is recharged to the maximum value Q_s and a new server deadline is generated as $d_{s,k+1} = d_{s,k} + T_s$.
5. When a job $J_{i,j}$ arrives and the server is busy processing another jobs, the request is enqueued in a queue of pending jobs.

6. When a job $J_{i,j}$ arrives and the server is not busy, if $c_s \geq (d_{s,k} - r_{i,j})U_s$ the server generates a new deadline $d_{s,k+1} = r_{i,j} + T_s$ and c_s is replenished to the maximum value Q_s, otherwise the job is served with the last server deadline $d_{s,k}$ using the current budget.

7. When a job finishes, the next pending job, if any, is served using the current budget and deadline.

8. At any instant, a job is assigned the last deadline generated by the server.

2.2 Distributed Real-Time Systems

Modern real-time systems are often distributed across multiple processing nodes which are interconnected by a network. The reasons in favor of distribution of real-time systems include the following [Burns and Wellings, 2001]:

- improved performance through the exploitation of parallelism,
- increased availability and reliability through the exploitation of redundancy,
- dispersion of computing power to the locations in which it is used.
- the facility for incremental growth through the addition or enhancement of processors and communication links.

One of the main issues, when it comes to distributed real-time systems, is how to ensure meeting of timing constraints in activities executed across multiple resources. Such activities are called *transactions*. An example of a transaction in a distributed system is a distributed feedback controller. A sensor node measures data from the plant and sends them across the network to the node running the control algorithm. Based on the measured data, the control algorithm calculates an action, which is sent to the actuator node. The actuator (e.g. motor) performs the desired action with the physical plant. In this kind of system, designers are usually interested in ensuring the properties of the system as a whole (*end-to-end properties*) rather than in the properties within the scope of a single resource (e.g. CPU of a single node). Continuing with the example of the distributed controller, an end-to-end deadline is the time from the measurement of the sensor by which the action must be applied. The difficulty of meeting such a deadline lies in the fact that the end-to-end response time time is influenced by scheduling policies of all involved resources, i.e. all CPUs and networks.

2.3 Component-Based Development of Real-Time Systems

Over the last few years, system design complexity increased to levels that cannot be managed by traditional software design methodologies, especially when there is demand for reduced development cost and short time-to-market. Big effort is

therefore put into methodologies which enable software reuse in multiple projects. In the component-based development methodology the applications are developed by combining appropriately predesigned and preverified components [Pinto et al., 2006]. Typically, a component is a software package that encapsulates certain functionality and has associated metadata, which allows automatic generation of the glue code between components.

The key problem of using component-based development methodologies in real-time system designs is the fact, that temporal properties (also called non-functional properties), that are verified for the individual components may not hold after multiple components are integrated in an application where they share system resources.

For that reason it is desired that run-time environment for component-based real-time systems provides *temporal isolation* of individual components. This means that the temporal properties of one component cannot be jeopardized by other components. One way of providing temporal isolation is the use of server-based scheduling described in Section 2.1.3.

2.3.1 Model-Driven Engineering

Component-Based development methodologies are often combined with *Model Driven Engeneering* (MDE). As the name suggests the MDE methodology focuses on the models as the primary means for software construction. In MDE, designers use a number of distinct model spaces which allow them to perform software specification at multiple levels of abstraction. In addition to plain component-based design methodologies, MDE introduces a new level of abstraction, called metamodel, which is used to specify the form of model elements and the relationships that can exist between them [Cancila et al., 2010]. The metamodel is typically composed from elements such as classes, interfaces and components. The MDE technologies employ transformation engines and generators that analyze certain aspects of models (e.g. temporal aspects) and synthesize various types of artifacts such as source code, simulation inputs etc. This automated transformation process is often referred to as "correct-by-construction" as opposed to conventional "construct-by-correction" development process [Schmidt, 2006].

2.3.2 Real-Time Component-Based Middleware Platforms

Given the facts above, component-based real-time middleware is a hot research topic of the last decade. Up to now, there is no universally accepted solution used by big industry players. There are, however, several efforts that head towards that goal.

[Bordin et al., 2008] discuss how to combine model-driven engineering with automated schedulability analysis. Their RCM modeling infrastructure allows for modeling of real-time system in a graphical way similar to UML2, all information needed for schedulability analysis is extracted from the model and it is automatically fed into analysis tools. Further, the source code is automatically generated from the model. The computational model of their run-time environment is equivalent to Ada Ravenscar Profile [Burns et al., 2003] and offers spacial and temporal isolation

of running activities. Temporal isolation is provided by employing hierarchical scheduling and priority band architecture together with enforcement of minimum inter-arrival times of sporadic tasks and monitoring of overruns against WCET budget specified in the model. In [Cancila et al., 2010] it is described how the methodology "correct by design" is used to address temporal correctness of the developed software. Although the RCM infrastructure targets distributed systems, in schedulability analysis the authors focus primarily on CPU and leave other resources like networks, disks, etc. unmanaged during runtime. Since the scheduling of these devices influences timing of CPU tasks, an integrated view of all resources, which is presented in this thesis can increase the predictability of system behavior.

[Deng et al., 2008] work on QoS-enabled component middleware called CIAO (*Component-Integrated ACE ORB*) and DAnCE (*Deployment And Configuration Engine*). CIAO is built on top of Real-Time CORBA [Object Management Group, 2008] implementation called TAO (*The ACE ORB*). In CIAO, components can declaratively specify their desired quality-of-service (QoS) such as rates of invocations. In deployment phase, DAnCE is used to deploy the components onto platforms, that are capable of supporting the specified real-time requirements. There is no direct support for schedulability analysis as this middleware targets soft real-time systems. DAnCE configures the underlying resources to enforce real-time QoS requirements by preparing thread pools and setting thread priorities, intra-process mutexes, a global scheduling service, communication protocol properties and memory buffers for requests. Besides run-time configuration, DAnCE supports also static (off-line) configuration to support systems with less resources. To further simplify development of real-time component based applications, MDE (Model Driven Engendering) tool chain called CoSMIC can be used to support deployment, configuration and validation of such applications. The main difference between these technologies and the framework presented in this thesis is that our framework provides schedulability analysis techniques that fit exactly the underlying resources and their schedulers. This tight coupling between analysis and run-time support can leads to less pessimistic analysis of the system.

The SPEEDS project [Döhmen et al., 2008] aims at defining a new methodology for model-driven systems engineering targeting embedded systems. It defines Heterogeneous Rich Components (HRC), which are characterized by formal contracts allowing various analysis techniques to validate a design already in the early design stages. Unlike the contracts described later in this thesis, HRC contracts define not only temporal properties but also functional and safety ones and represent assumption-promise pairs. The framework presented in this thesis could be used as an underlying run-time environment guaranteeing assumptions on the resources such as CPUs, networks etc.

The most similar work to the framework presented in this thesis is the FRSH (pronounced as fresh) framework [Telleria de Esteban, 2008], which was developed within FRESCOR project and therefore tries to solve very similar problems as our framework. It is based on the previous work in FIRST project, which supported only CPU resource. Since then, the FRSH framework was largely reworked and enhanced by support for networks etc. There are many similarities in architectures of FRSH framework and this thesis, however we have tried to design our architecture as a

superset of FRSH architecture with the aim to better abstract various resources to provide higher modularity of the framework, as detailed in the next chapter.

3

Contract-Based Resource Reservation Framework

This chapter describes the design and implementation of contract-based resource reservation software framework called FRSH/FORB.

The basic idea of the framework is to let the application developer specify the temporal (and resource) requirements of his/her application and if there is enough resources in the system to satisfy them, the framework reserves the resources for the use by the application. In the case of insufficient resources, the framework does not let the application run. Application requirements are specified in the so called *service contract* (contracts in short) that the application negotiates with the framework. A successfully negotiated contract results in creation of a *virtual resource*, which represents "a part" of the real resource reserved for the use by the application. To not over-reserve the available resources, the framework employs on-line admission tests that are based on state-of-the-art schedulability analysis.

In general, the contracts allow interchanging of arbitrary information between applications and resource management entities in the underlying, possibly distributed, system. This information is used by the system to manage the resources in an efficient way and allows the applications to get the guarantee on the services provided to them by the system. An example of the efficient use of the resources is the framework's ability to distribute spare resource capacity among applications that specified in the contract they can make use of it.

The FRSH/FORB framework can be used as a run-time environment for a component-based real-time middleware. It is suitable especially for dynamic application in open environments. Dynamic applications are those that change their resource requirements over the course of their run-time, and open environment means that it is not known in advance which application will run in the system and what will be their resource requirements.

The development of the framework started in FRESCOR project [FRESCOR,

2009] and it then continued as a stand-alone open-source project [FRSH/FORB, 2010].

The rest of this chapter is structured as follows: The motivation for development of FRSH/FORB framework is given in Section 3.1. Then, Section 3.2 describes the general framework architecture and the subsequent Section 3.3 provide details of the individual framework modules and some used algorithms. Finally, Section 3.4 introduces mathematical formalism used to describe the framework and its applications.

3.1 Motivation

The most important reason for designing a new framework instead of using an existing one (e.g. FRSH described in Section 2.3.2) was the requirement to support *Wireless Sensor Networks* (WSNs) and FPGAs. These resources are quite different from the resources typically considered by similar frameworks (CPUs and wired networks) that a more general approach was necessary for these resources to be supported by the framework. The following goals were set for the design of the FRSH/FORB framework:

High modularity. Adding the support for a new resource in original FRSH framework was a complicated process which required significant amount of development effort. If was necessary to duplicate common functionality for every resource. The FRSH/FORB framework eliminates duplicated functionality whenever possible by modularizing the framework in a way that the functionality of the individual modules may be reused by other of that is required.

Resources with varying capacity. Wireless LAN is a resource which gives very little guarantees. It is typically operated in environments where administrators cannot control the level of electromagnetic disturbances and therefore the Wi-Fi hardware and drivers are designed to adapt to changing environment by changing transmission rate. For some devices, it would be possible to disable this adaptive mechanisms but it would lead to performance degradation either because of low throughput or because of high packet loss. Instead it is better to let applications adapt to the actual environment conditions. FRSH already supported adaptive applications by means of *spare capacity* module (see Section 3.3.3) but the adaptation could be triggered only by negotiation requests from other applications and not by the resources themselves, for example when Wi-Fi transmission bit-rate changes.

Task migration between resources. FPGAs are typically used as coprocessors to the main CPU. When FRSH/FORB is used to manage FPGA resource, its goal is to decide whether a software only variant of a task is to be run on the CPU or whether the FPGA is used to accelerate the task and therefore the budget for the CPU task can be decreased. For this to be supported, the applications need some way to specify, that there are two variants of one task and that each variant uses different resource (CPU and FPGA). See Section 4.5 for details.

Consistent allocation of spare capacity in transactions. If contracts in the transactions specify application's ability to use available spare capacity, depending on the application it might or might not have sense to allocate spare capacity independently for the resources in the transaction. As it is shown in Section 4.5, for transactions comprising FPGA resource it is necessary to ensure that the spare capacity is allocated consistently across all resources participating in the transaction.

Wireless sensor networks are a special kind of resource in the sense that the FRSH/FORB is not running on every node of ZigBee network but the network is attached to only one node of the distributed FRSH/FORB system. Therefore this resource can be treated as any other non-network resource (e.g. CPU) which is local to the node attached to the ZigBee network. The only complication is that multiple applications may want to receive different data from the network and the implementation must provide a mean for distributing data received from the network to the applications which requested the data. This could be achieved even in the FRSH implementation, but in FRSH/FORB we efficiently reuse FORB middleware to distribute the data between multiple applications.

3.2 Architecture

This section describes the internal software architecture of the FRSH/FORB framework. The framework can be divided into three levels (see Figure 3.1) – *Application Programming Interface* (API), resource-independent level and resource-specific level.

The API is used by the applications to interact with the framework. The FRSH API was developed in the context of the FRESCOR project as a portable and generic interface to provide resource reservation services to hard and soft real-time applications. The main service provided by the API is contract negotiation: the application specifies its resource requirements in the form of a contract, and submits it for negotiation with the framework. If the negotiation succeeds, the framework provides the application with a so called *virtual resource* (VRES), which is a generic name for resource reservation. The application then uses FRSH API services to *bind* its entities (threads, communication endpoints) to the VRESes in order to use the reserved resources.

The *resource-independent level* is represented by the *contract broker* and is intended to implement algorithms for spare capacity distribution, multi-resource transactions and global *Quality of Service* (QoS) optimization. The contract broker interacts with the resource managers through abstract interfaces. The framework is implemented on top of a lightweight CORBA-like communication middleware called *FRSH Object Request Broker* (FORB) [Sojka et al., 2008]. This middleware hides the complexity and different nature of inter-process and inter-node communication and provides method-call semantics for remote application objects.

The *resource-specific level* consists of modules called *resource managers* and *resource allocators*, which are in charge of managing the individual resources (e.g.

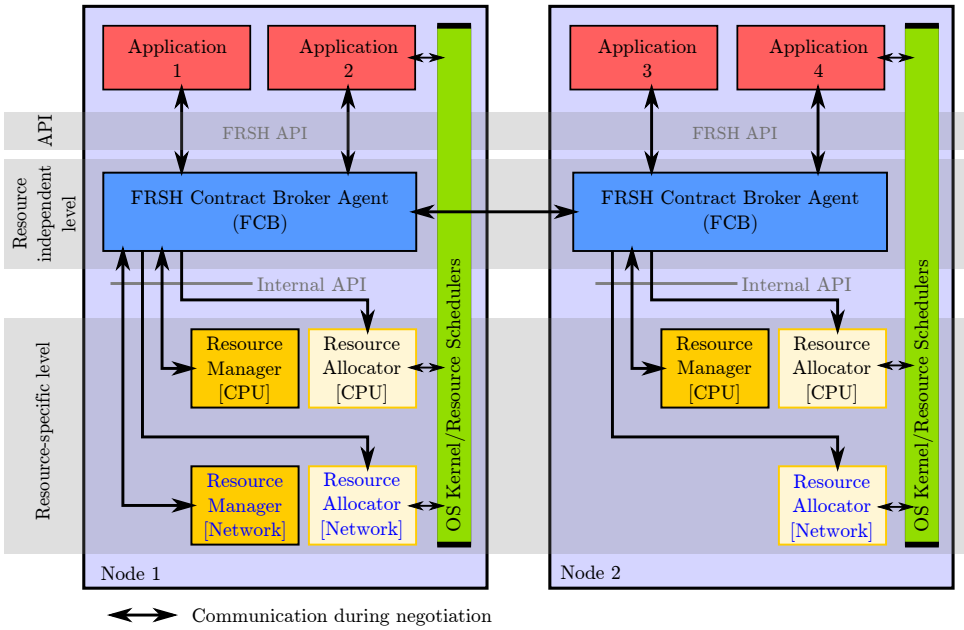

Figure 3.1: Block diagram of FRSH/FORB framework.

CPU, network, disk). Their goal is to implement resource reservation policies and management supporting real-time execution and temporal isolation for tasks running on the associated resource. One key feature of our architecture is that the modules in this level can be easily plugged in and out, allowing the framework to be used on various platforms which exploit different resource reservation mechanisms.

The whole framework is built on top of *operating system abstraction* libraries, which facilitates portability across multiple hardware/software platforms. It consists of the *FRSH Operating-System Abstraction* (FOSA) layer [Gonzáles Harbour and Tellería de Esteban, 2006], which implements the FOSA API. This is a cross-platform API designed within the FRESCOR project for the purpose of abstracting *Operating System* (OS) services related to the management of time, posting of timers, management of threads and synchronization primitives (e.g., signals and mutexes). The use of FOSA simplifies porting of the FRSH/FORB framework on different operating systems. Indeed, FOSA has been implemented in a straightforward way on Linux by means of the POSIX API for the just mentioned services, with a few Linux-specific extensions which have been leveraged for performance reasons. Also, FOSA has been implemented on MarteOS[1] [Rivas and Harbour, 2000, Rivas and Harbour, 2001], Partikle[2] [Peiro et al., 2007] and Enea's OSE[3]. Note that the implementation of FRSH/FORB requires usually extensions at the kernel resource scheduling level. Such extensions are forcibly OS specific and each has to be

[1] `http://marte.unican.es/`
[2] `http://www.e-rtl.org/partikle/`
[3] `http://www.enea.com/Templates/Product____27035.aspx`

interfaced separately in the resource-specific level. However, the presence of FOSA allows for the straightforward porting of all the contract management part of the framework.

The following sections describe the individual modules in more detail as well as their interactions.

3.2.1 Application Model and API

The model of FRSH/FORB applications was defined within the FRESCOR project in [Gonzáles Harbour and Tellería de Esteban, 2008]. The document defines a so called FRSH *Application Programming Interface* (API) for applications to interact with the framework. The API aims at providing resource reservation services independently of the underlying schedulers and operating systems and is divided into several modules offering different functionality (e.g. core, shared objects, networks and distribution, energy management etc.). Each module defines attributes that can be set in the contract. Whenever possible, the attributes are defined in resource independent way. The example of an attribute (defined by core module) is *resource*, *budget*, *period* or *deadline* (see section 3.3.1 for more details).

To summarize the API, the provided services (functions) can be divided into several categories:

Contract manipulation services There are functions to manipulate contract data structures. As an example function *frsh_contract_set_timing_reqs()* can be used to specify time-related requirements (e.g. deadline) requested by application.

Negotiation services When a contract is prepared with all attributes filled in a negotiation function like *frsh_contract_negotiate()* is called to negotiate the contract with the framework. Successful negotiation results in creation of a *Virtual Resource* (VRES), which is the runtime representation of the contract. It is also possible to cancel and renegotiate existing contracts.

VRES binding services Applications can use functions like *frsh_thread_bind()* to bind their entities (threads, communication endpoints, etc.) to the virtual resource to make use of the guaranteed service.

VRES manipulation services These functions can be used to manipulate VRES-es at run-time. An example is *frsh_vres_get_remaining_budget()* function, which returns the currently remaining budget available reserved to the application.

The key term in the presented application model is the VRES, which represents a particular resource reservation requested via contract negotiation. Every VRES should, independently of resource, provide the following two basic properties:

Service guarantee The application that use the physical resource through the VRES and FRSH API has a guarantee of resource availability as specified in the contract, i.e. deadlines will be met etc.

Overrun protection/detection In order to guarantee the service of other VRESes, the VRES implementation typically detects the attempts to overrun the budget and prevents the use of the resource beyond what was negotiated.

Although the API was carefully designed and serves very well for its purpose, during the development of the FRSH/FORB framework several deficiencies were identified. They are summarized in Appendix A and should serve as guidelines for future development of resource reservation frameworks.

3.2.2 Resource Managers

There are two kinds of modules in the resource-specific part of the framework. The first of them is *FRSH Resource Manager* (FRM), whose role is to provide an *admission test* for the given resource. The test is usually based on some kind of schedulability analysis, and its objectives are the following:

1. To check whether the new contract(s) representing application resource requirements can be accepted without violating the already negotiated contracts.

2. In the case of a mode-change [Real and Crespo, 2004], i.e. when an application changes its operating mode, and needs to renegotiate its contracts), the module is also able to test the feasibility of the mode change, because the manages has access to contracts of both old and new mode. Note that the work in this thesis does not utilize this possibility.

3. Based on the analysis, the resource manager may add a piece of information to the contract, which can be later utilized by the allocator or scheduler.

   A simple example might be a resource with a fixed-priority scheduler, which schedules tasks according to the priority calculated by deadline-monotonic algorithm. In the contract, the application specifies only the deadline and the manager calculates the priorities using the deadline-monotonic algorithm. The resulting priority is added to the contract by FRM and the used later by the scheduler.

3.2.3 Resource Allocators

The second module is *FRSH Resource Allocator* (FRA) which always accompanies the corresponding resource manager. There can be multiple allocators for a single resource, e.g. in case of a network, there is typically one allocator for every network node. The purpose of the resource allocator is:

1. to interact with the resource scheduler, i.e. to create, change or cancel *virtual resources* according to the "instructions" from the resource manager and contract broker;

2. to implement a generic API for VRES binding and manipulation (see Section 3.2.1).

3.2.4 Contract Broker

The *FRSH Contract Broker* (FCB) acts as a mediator between applications and individual resources. Contract broker is a distributed application with an agent running in every node. Agents collaborate on distribution of information about resources and contracts in the whole distributed system. In the simplest case, the FCB agent only resends the contracts received in negotiation requests to the appropriate resource manager and then, if the admission test succeeds, to the resource allocator and back to the application. FCB is also responsible for high-level tasks described in Sections 3.3.3 and 3.3.4.

3.2.5 Examples

Figure 3.1 shows two nodes running the FRSH/FORB framework and connected by a network. Every node runs two (arbitrary) FRSH/FORB applications and a contract broker agent. Furthermore, node 1 runs two resource managers: one for the local CPU and one for the network. Node 2 runs only the resource manager for its local CPU. The network resource uses a centralized manager, which means that the manager runs only in one node. The figure also contains blocks representing the allocators. Note that the network resource has an allocator in every node even if the manager is located in a single node. The reason is that the virtual resource implementation must enforce the application not to use the network bandwidth beyond what was negotiated and for most networks this can be only implemented at sending side.

To illustrate the interaction of these modules we present two example scenarios of the contract negotiation (a more detailed description is provided in section 3.3.2).

Example 1. Consider the case in which application 1 wants to use the local CPU for a periodic task, and requires a guarantee for meeting all deadlines. It prepares the contract with appropriate attributes (period, budget, deadline and resource). Then it sends the contract to the local contract broker agent. The agent finds out that the contract refers to the local CPU resource and resends the contract to the local CPU resource manager. The manager executes an admission test and returns the result (accepted/rejected) to the broker. If the contract is accepted the broker asks the resource allocator to create the virtual CPU resource according to the attributes specified in the contract.

Example 2. Application 3 wants to periodically communicate over the network with a guarantee of meeting all deadlines. Negotiation will be accomplished as follows: The application prepares a contract and sends it to the contract broker agent in node 2. The FCB agent issues a reservation request to the network resource manager running in node 1. If the contract is accepted, the FCB agent in node 2 requests the local network resource allocator to create the network virtual resource.

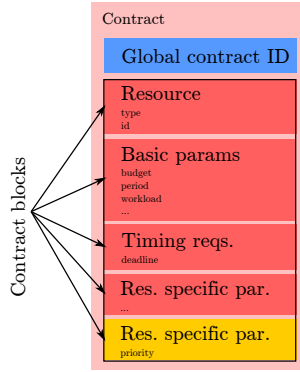

Figure 3.2: A contract and its attributes.

3.3 Advanced Concepts and Internals

3.3.1 Representation of Contracts and Virtual Resources

In order for the framework to be modular enough to support different resources, a dynamic data structure is used to represent contracts. We use the term "dynamic" to express that the number of attributes stored in the contract and their type can vary depending on the resource and the state of the negotiation process. The graphical representation of the contract data structure is depicted in Figure 3.2. Every contract is identified by a dynamically generated ID which is unique in the whole distributed system.

The contract attributes are grouped into so-called *blocks* and every contract contains one or more these blocks. The block is a set of attributes which are used together. Typically, most resources define an additional block with resource specific attributes.

The most common contract attributes are *budget*, *period*, *deadline* and *workload type*. Budget corresponds to the amount of service (execution time resp. the size of the data) which the application needs to execute resp. process in every period. If the contract specifies the deadline, the framework guarantees that the service is completed within the deadline. The workload type attribute describes the application workload model, which can be either *bounded* or *indeterminate*. Bounded workload means that the application has a bounded amount of work (called job) that to do during each virtual resource period, and it notifies the framework whenever the job is done. As a consequence, the framework can notify the application about overrunning its budget or about a deadline miss. Indeterminate workload model is used when there is no concept of jobs in the application.

The attributes of a contract can be set and modified not only by applications, but also by the contract broker and by resource managers. This makes the framework very flexible – for example, an application can specify only platform independent attributes in the contract. The contract broker may exploit knowledge of the underlying platform to add the platform-dependent attributes, and finally the

resource manager may also add additional "instructions" for the allocator/scheduler.

To simplify the explanation of the negotiation process in FRSH/FORB, we define three different forms of contracts. Every form is represented by the same data type but the difference is in the contained information. The three forms are the following (see Figure 3.3 for the example of different contract forms and their role in negotiation process):

User contract contains attributes requested by a user (application). This form of the contract can represent many possible reservations (variants) through the use of the *spare capacity* block.

Reservation contract contains specific attributes needed for resource reservation. Contract broker produces it from the user contract by selecting one concrete reservation (variant) from spare capacity block if that one is present.

The spare capacity block is kept in the contract as it can be used by the resource scheduler to support dynamic reclamation [Gonzáles Harbour and Tellería de Esteban, 2008]. Manager can also use this information to check that it can always satisfy minimal application requirements (see section 3.6.2 in [Sojka et al., 2008]).

Schedulable contract is a reservation contract extended by data needed for the allocator to create the VRES (e.g. priority for a fixed priority scheduler). This form is produced by resource managers.

Virtual Resource

A virtual resource is represented in an application by a data structure containing the negotiated schedulable contract together with any data needed by a particular resource allocator implementation to manipulate the reservation and to communicate with the scheduler.

3.3.2 Contract Negotiation Process

This section contains a detailed description of the negotiation process. For simplicity, the algorithm for distribution of spare capacity is not described here but in separate section 3.3.3. A collaboration diagram of the negotiation process is depicted in Figure 3.3. The description follows and the numbers corresponds to the edges in the figure:

1. The negotiation starts in an application by preparation of a contract and filling its attributes. Then the application calls a negotiation service such as *frsh_contract_negotiate()*.

2. This function uses FORB to call *negotiate_contract()* method of the local contract broker agent and to pass it the user contract.

3. Contract broker agent carries out the following operations:

   (a) Assigns the contract a global ID (see above).

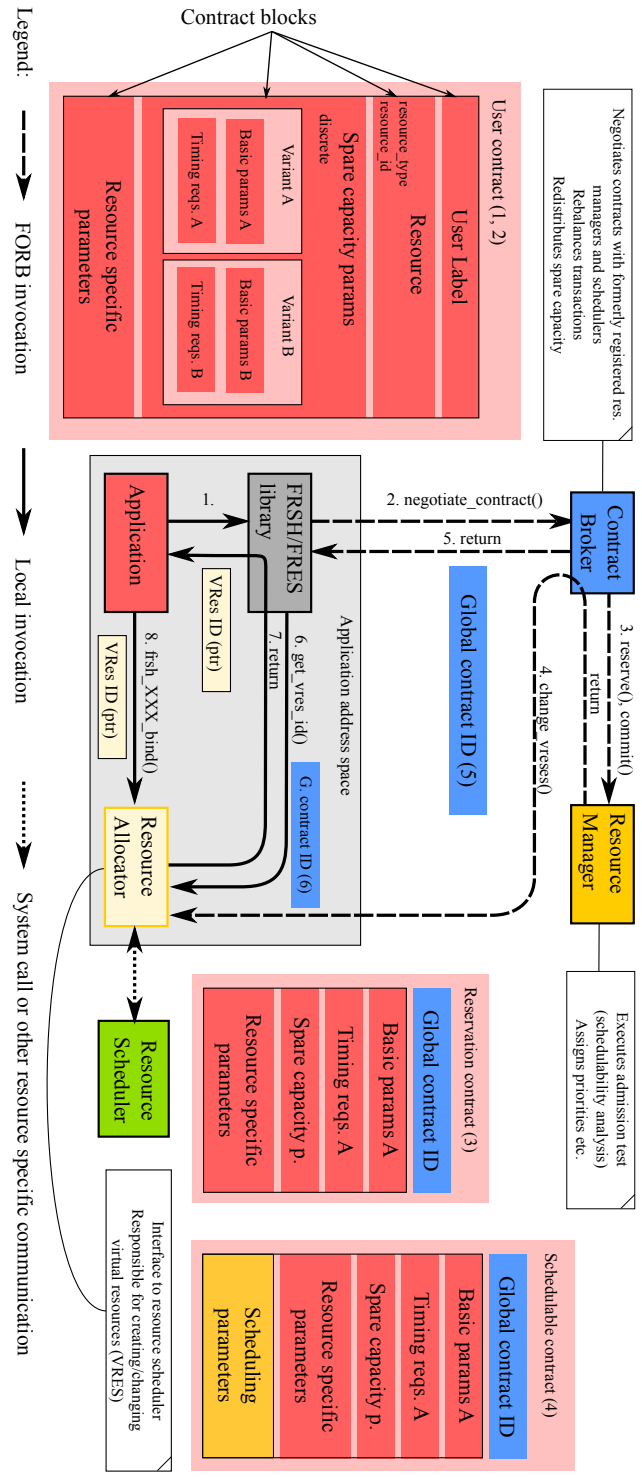

Figure 3.3: Collaboration diagram of FRSH/FORB modules during contract negotiation.

(b) If the contract contains spare capacity block, some variant is selected as determined by the spare capacity distribution algorithm and a reservation contract is created by putting all blocks from the selected variant directly to the contract.

(c) Then, resource block in the contract is consulted to find the appropriate resource manager.

(d) Finally, the reservation contract is passed as a parameter to *reserve_contracts()* method of the resource manager. The resource manager executes an admission test. Should the contract not be accepted, the resource manager reverts its actual state to the one before reservation and the agent reports *reject* to the application.

In the case of accepted contract, the agent invokes manager's *commit_contracts()* method and the manager returns one or more schedulable contracts.

For certain resources, it might be possible, that a negotiation of a single contract causes changes of several previously accepted contracts. An example can be again a fixed priority scheduler with DM priority assignment. Consider existing tasks τ_a and τ_b with deadlines $D_a = 1$ and $D_b = 3$. The corresponding DM priorities are $P_a = 1$ and $P_b = 2$. When a new task τ_c with deadline $D_c = 2$ arrives, the priority P_b needs to be changed from 2 to 3 and $P_c = 2$.

4. The contract broker agent takes the returned schedulable contracts and calls *change_vreses()* method of the resource allocator. The allocator applies the changes requested by the broker, i.e. it creates virtual resources or changes their parameters. This operation is accomplished by some interaction with the *resource scheduler*, which is typically done by using a system call interface to the kernel, but in principle, it might be implemented by any other type of communication.

5. If the changes to VRESes were successfully applied, the broker agent returns to the application the ID of the negotiated contract.

6. Application can use the returned ID to access the allocated VRES. Since applications usually need to interact with the VRES quickly (e.g. anytime algorithms need to repeatedly determine the remaining budget), instead of using the contract ID to identify the VRES and searching the VRES "registry" every time the access to the VRES is needed, this search is performed only once by calling *fra_get_vres()*.

7. This function returns a pointer to the VRES data structure.

8. Finally, the application may start using the VRES by binding its entity with the VRES. For example the *Central Processing Unit* (CPU) VRES is bound by calling *frsh_thread_bind()* function.

3.3.3 Distribution of Spare Capacity

An application can specify in the contract that it is able to make use of additional (spare) resource capacity if that is available. When the contract broker is requested to negotiate such a contract, it tries to reserve the maximum capacity requested. If that is not possible, the contract broker finds an optimal distribution of spare capacity among applications and reallocates the resources according to the result.

From the application perspective it means, that the application utilizing spare capacity must be written in such a way that enables it to adapt to the changes in VRES allocation. The changes may happen asynchronously to application execution and there exists an API allowing the application to determine the current allocation.

As it was mentioned in Section 3.1, FRSH/FORB framework allows that the reallocation of the spare capacity is triggered not only by applications but also by individual resources, when the resource manager or allocator decide. For example, when wireless network allocator, which is also responsible for monitoring transmission bitrate, determines that the bitrate was changed, it sends a request to the contract broker to run spare capacity reallocation algorithm.

As the contracts are represented by the dynamic data structure, applications have great flexibility in specifying all possible uses of spare capacity. For example, an application can specify two different budgets in the contract and the contract broker ensures that the highest possible budget is reserved/allocated at all times. Note that resource managers and allocators always receive a simple contract, representing a single reservation. This way the problem of spare capacity is only dealt with at the resource-independent level in the contract broker and the low-level resource support is not aware of it. This makes the support for new resources easier to develop.

3.3.4 Negotiation of Multi-Resource Transactions

Many applications operate on multiple resources. The typical example is a distributed systems with several computing nodes interconnected with a network where some part of the application is responsible for gathering data and sending them via the network to other nodes for processing by another part of the application. A part of an application consisting of activities on multiple resources and synchronizing these activities by some means is called *multi-resource transaction*. In the contract framework, the application needs to negotiate the contracts for all the activities in the transaction. In many cases it has no sense to negotiate a contract for one resource if the negotiation of another contract for another resource fails because the transaction could not run.

In the context of the FRSH/FORB framework, the transaction is simply a set of contracts with the following properties:

Atomicity – either all contract in the transaction are successfully negotiated and the respective virtual resources allocated or no resource is reserved and no virtual resources is allocated.

Consistency – when the transactions runs, it is required to be always in a consistent state. This means, for example, that all VRESes run with the same period.

The consistency is important especially when the individual contracts specify the use of the spare capacity.

Transactions are negotiated similarly to what is described in section 3.3.2, except that contract broker completes resource reservations (calls to resource managers) for all resources in the transaction before any resource is allocated by its resource allocator. The reservation phase of the transaction negotiation utilizes so called "two-phase commit protocol" to achieve the atomicity property.

Currently, only contracts without spare capacity can be negotiated in transactions. In future, we plan to remove this limitation by using global optimization techniques to find optimal distribution of spare capacity across multiple resources. A formal description of this goal is provided in Section 3.4.

3.3.5 Transaction API

Support for transactions in the original FRSH framework was implemented as an extension to the framework in a module called distributed transaction manager. The API was unnecessarily complicated and tied closely to the implementation. For that reason we developed a new API for FRSH/FORB framework, which is described in this section.

Transaction Object Manipulation

The following services are provided for manipulating of the transaction object:

frsh_transaction_init – initializes the transaction object in the application.

frsh_transaction_destroy – deallocates memory associated to the transaction object. Note that this service does not change any VRES.

frsh_transaction_add_contract – adds a contract to the transaction.

Transaction Negotiation

frsh_transaction_negotiate – reserves the resources for the transaction. Either all resources are reserved or none of them, depending on the results of admission tests in resource managers.

frsh_transaction_cancel – cancels (deallocates) all VRESes allocated for the given transaction.

frsh_transaction_wait_for_name – waits for a transaction with a given name to be reserved. Typically, only one part of the application negotiates the transaction and the other parts of the application wait for it by exploiting this service and use the VRESes reserved by the negotiation.

frsh_transaction_alloc_vres – Allocates the VRES reserved previously by transaction negotiation and makes the VRES available for use by the application calling this service.

3.4 Mathematical Model of the Framework

This section defines notation used to model the contracts and transactions of the framework in order to describe various analysis and optimization methods used in the framework. In particular, this notation is used later in Section 6.2.3 to show how one particular schedulability analysis method for distributed systems can be applied to the framework.

We model the our framework as a tuple $\mathcal{S}$ of the set $\mathbb{T}$ of n multi-resource transactions Γ and the set $\mathbb{R}$ of r resources ρ:

$$\mathcal{S} = (\mathbb{T}, \mathbb{R})$$
$$\mathbb{T} = \{\Gamma_1, \ldots, \Gamma_n\}$$
$$\mathbb{R} = \{\rho_1, \ldots, \rho_r\}$$

Transaction Γ_i is a tuple of m_i contracts c

$$\Gamma_i = (c_{i1}, c_{i2}, \ldots, c_{im_i}).$$

At run-time, every contract c_{ij} has its corresponding task τ_{ij} associated through the *bind* operation, but since the task is only allowed use the resource in a way described in the contract, the parameters of the tasks are not important and it is sufficient to deal with contracts in the following.

Contract c_{ij} has n_i different variants c_{ij}^{v} which represent different possible allocations of spare capacity:

$$c_{ij} \in \{c_{ij}^1, c_{ij}^2, \ldots, c_{ij}^{n_i}\}.$$

At any time instant, only one variant $c_{ij}^{v_i}$ can be *reserved*. All contracts in the transaction must reserve the same numbered variant and hence we denote the reserved variant of the transaction Γ_i as $\Gamma_i^{v_i} = (c_{i1}^{v_i}, c_{i2}^{v_i}, \ldots, c_{im_i}^{v_i})$, $v_i = 1, \ldots, n_i$. Every variant $c_{ij}^{v_i}$ has its weight $w(c_{ij}^{v_i})$, period $T(c_{ij}^{v_i})$, worst-case execution time $C(c_{ij}^{v_i})$ and deadline $D(c_{ij}^{v_i})$. All contracts in the transaction are supposed to have the same period $T(\Gamma_i^{v_i}) = T(c_{ij}^{v_i})$, $j = 1, \ldots, m_i$, $v_i = 1, \ldots, n_i$. Further, for every contract variant $c_{ij}^{v_i}$, there is a contracted resource $\rho(c_{ij}^{v_i})$. The set of resources contracted by a single contract it is denoted as $\rho(c_{ij})$. Note that for most contracts, this set has only one element. Only when task migrations are considered, as in the case of FPGAs, there is more resources in this set. The set of resources reserved by transaction Γ_i is denoted as $\rho(\Gamma_i) = \bigcup_{j=1}^{m_i} \rho(c_{ij})$.

The set of contracts on resource ρ_k is $c(\rho_k) = \{c_{ij} : \exists v_i : \rho(c_{ij}^{v_i}) = \rho_k\}$. The set of reserved contracts on resource ρ_k is $c^v(\rho_k) = \{c_{ij}^{v_i} : \rho(c_{ij}^{v_i}) = \rho_k\}$, where $v = (v_1, \ldots, v_{n_i})$ is a tuple of reserved variants for every transaction. The set of transactions on resource ρ_k is $\Gamma(\rho_k) = \{\Gamma_i : \rho_k \in \rho(\Gamma_i)\}$.

The problem of optimal distribution of spare capacity across multiple resources is defined as

$$\text{maximize} \quad \sum_{i=1}^{n} \sum_{j=1}^{m_i} w(c_{ij}^{v_i}), \tag{3.1}$$

$$\text{subject to} \quad c^v(\rho_k) \text{ is schedulable}, \quad k = 1, \ldots, r \tag{3.2}$$

$$R(\Gamma_i) \leq D(\Gamma_i), \quad i = 1, \ldots n_i \tag{3.3}$$

where $R(\Gamma_i)$ and $D(\Gamma_i)$ are end-to-end response time and end-to-end deadline of transaction Γ_i respectively. The term "is schedulable" means that the resource manager for the particular resource evaluates the set of contracts $c^v(\rho_k)$ as schedulable.

Example 1 Consider a distributed system consisting of three resources: two CPUs ρ_{cpu1} and ρ_{cpu2} and a network ρ_n. There are two applications in the system. One sends data from CPU1 to CPU2 so it negotiates transaction $\Gamma_1 = \{c_{11}, c_{12}, c_{13}\}$. There is one contract for every resource in the transaction: $\rho(c_{11}) = \{\rho_{\text{cpu1}}\}$, $\rho(c_{12}) = \{\rho_n\}$, $\rho(c_{13}) = \{\rho_{\text{cpu2}}\}$ i.e. $\rho(\Gamma_i) = \{\rho_{\text{cpu1}}, \rho_n, \rho_{\text{cpu2}}\}$. And all contracts have only one variant, i.e. $c_{11} = (c_{11}^1)$, etc.

The second application utilizes only CPU1 so its transaction reduces to single contract $\Gamma_2 = \{c_{21}\}$. However, this application is designed to use spare capacity and its contact has two variants: $c_{21} = (c_{21}^1, c_{21}^2)$. These variants differ in the period of the associated task, i.e. $T(c_{21}^1) = 20\,\text{ms}$ and $T(c_{21}^2) = 100\,\text{ms}$. The first variant is preferred so $w(c_{21}^1) = 2$ and $w(c_{21}^2) = 1$.

Optimal distribution of spare capacity is in this example defined as:

$$
\begin{aligned}
\text{maximize} \quad & w(c_{21}^{v_2}), \\
\text{subject to} \quad & c_{11}^1 \text{ and } c_{21}^{v_2} \text{ is schedulable on } \rho_{\text{cpu1}}, \\
& c_{12}^1 \text{ is schedulable on } \rho_n, \\
& c_{13}^1 \text{ is schedulable on } \rho_{\text{cpu2}}, \\
& R(\Gamma_1) \leq D(\Gamma_1).
\end{aligned}
$$

□

4

Resources Supported by the Framework

The following sections describe how are specific resources supported by the framework. The support was either developed from scratch or a 3rd party technology was integrated into the framework. Namely, the resources described in sections 4.1 (CPU) and 4.2 (Disk) was integrated into the framework by our partners and represent the work of co-authors of [Sojka et al., 2010]. Section 4.3 describes the support for wireless LANs developed mostly by the author of this thesis and sections 4.4 and 4.5 describe the resources developed by other co-workers and were integrated into the framework by the author. All the 3rd party resources are mentioned in this thesis to emphasize the modularity and universality of the described framework. Furthermore, the framework evaluation in Chapter 5 uses the 3rd party resources and hence the basic information needed for understanding that chapter is provided here.

4.1 CPU

Currently, the framework supports two resource reservation mechanisms for CPU – AQuoSA and Linux Cgroups. A brief description of AQuoSA from [Sojka et al., 2010] is provided in this section. Cgroups are not detailed here. The difference between Cgroups and AQuoSA is that the former is available in the vanilla Linux kernel and supports multiprocessor system while the latter operates only on a single-processor machines but has the advantage that the applications can specify arbitrary periods in their contracts.

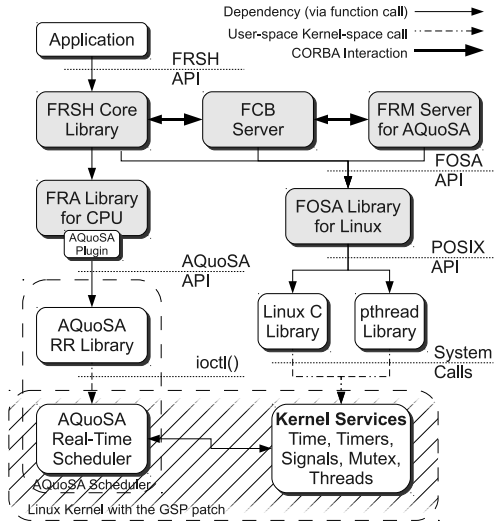

Figure 4.1: Integration of the AQuoSA scheduler within the FRSH/FORB architecture. Source: [Sojka et al., 2010].

4.1.1 The AQuoSA Architecture

The Adaptive Quality of Service Architecture for the Linux kernel (AQuoSA) is an open-source architecture enriching Linux with soft real-time capabilities, comprising: EDF-based scheduling, temporal encapsulation and enforcement of timing constraints, limited support for hierarchical scheduling, admission control, controllable and secure exposure of real-time capabilities to unprivileged processes, and feedback-based scheduling.

The components of AQuoSA which are relevant for this thesis are the following (the reader is referred to [Palopoli et al., 2009] for a more comprehensive description):

– the *Generic Scheduler Patch* (GSP), a small patch to the kernel which allows to extend the Linux scheduler by intercepting scheduling events and executing external code in a kernel module;

– the AQuoSA real-time scheduler, a dynamically loadable kernel module which, exploiting the GSP patch, enhances the Linux CPU scheduling with an EDF-based scheduling policy, and precisely a hard-reservation version of the CBS algorithm (see Section 2.1.3 and [Abeni and Buttazzo, 1998]).

– the AQuoSA Resource Reservation Library, which allows applications to request real-time scheduling services through a properly designed API, and forwards requests to the real-time scheduler via `ioctl()` system calls operated on a special virtual device.

4.1.2 Integration of AQuoSA in FRSH/FORB

Figure 4.1 shows how the AQuoSA scheduler is plugged within the FRSH/FORB framework, where the grayed blocks identify software components implementing the CPU-related parts of the framework presented in this thesis. The application uses the FRSH Core API, available by linking a library. When an application negotiates a new contract, the library performs admission-control via the Contract Broker *Common Object Request Broker Architecture* (CORBA) object (FCB Server), which in turn contacts the AQuoSA-specific Resource Manager. The latter performs the admission-test based on the currently admitted contracts, and the parameters provided by the application. This admission test may be potentially more complex then the simple utilization-one as currently implemented. Once the new contract has been admitted, the FRSH Core library performs the actual allocation via the FRSH Resource Allocator (FRA) library for the CPU, which has a proper plug-in for communicating with the AQuoSA scheduler via the AQuoSA user-space API.

When the FRA allocates resources corresponding to a contract within AQuoSA, it sets the budget to the values indicated in the FRSH contract, or to ones scaled-up by the spare-capacity capability of the contract broker.

It must be noted that (see Figure 4.1), in the FRSH/FORB over AQuoSA scheme, the CORBA interactions occur only for those actions that do not have strict real-time requirements, and not for monitoring actions typically required during a real-time task activation. For example, a new contract set-up involves the FCB, FRM and FRA components, whilst reading the current budget (e.g., as needed for implementing *anytime* computing algorithms) or the server deadline are actions managed quickly through a set of function calls to the FRA library. On a related note, if configured properly, the FCB and FRM CORBA objects may be given precise scheduling guarantees within the framework, in order to provide minimum guarantees on the contract set-up time, if needed.

Note that, in the just described architecture, Linux tasks that do not use resource reservations via the FRSH API are still managed by the default Linux scheduler.

4.2 Disk (BFQ scheduler)

To provide individual applications with timing guarantees on disk access we [Sojka et al., 2010] have to consider that requests response time is something highly variable and dependant on physical disk parameters. Moreover, many applications (e.g., video streaming) only issue *synchronous* requests to the disk, i.e., they send the request and then block waiting for it to complete. Therefore, work conserving approaches tend to introduce a lot of seeks, as they see only one request per application, and delaying the dispatch of a request (which is done by some classes of disk schedulers) is does not help either, since it actually prevents the application to issue its next ones.

The Budget Fair Queuing (BFQ [Valente and Checconi, 2010]) algorithm is a timestamp-based proportional-share disk scheduler designed to provide strong guarantees on disk bandwidth distribution even in presence of synchronous workloads. Bandwidth distribution guarantees can be turned into soft timeliness guarantees,

based on the mere knowledge of the aggregate throughput in the context of some workload scenario.

The algorithm maintains a per-application queue, and a B-WF^{2}Q+ (a slightly modified version of the Worst-case Fair Weighted Fair Queueing Plus algorithm) scheduler selects the queue to be dispatched to the disk device. Each application is also assigned a budget, representing the numbers of sectors to which it is entitled after being selected, and the scheduler involves some idling (usually referred to as *anticipation*) in case an application has no pending request but it still has some budget left.

The interested reader can find an overview of traditional elevator algorithms in [Silberschatz et al., 2008], while BFQ is detailed in [Valente and Checconi, 2010].

4.2.1 Integration of BFQ in FRSH/FORB

As any other resource, BFQ has been integrated within FRSH/FORB by implementing a BFQ Resource Manager and a BFQ Resource Allocator.

The BFQ Resource Manager (FRM) performs admission-test for disk contracts and lets a new one enter the system only if the service time over the period it is asking can be guaranteed. It is possible to ask for a *background contract*, which results in no service guarantees, i.e., the requests will be served when all the reserved applications are "idle". This can be used for the applications that do not need specific disk access guarantees, to avoid wasting reserved bandwidth for them.

The BFQ Resource Allocator (FRA) calculates the actual BFQ weight ϕ_i of a request associated to a contract by a bind operation based on budget and period contract attributes. The worst-case aggregate throughput figures of the disk device are required, both in the admission test (FRM) and allocation phases (FRA). For that reason, it can be either specified manually at FRM starting time (if known in advance), or it is automatically calculated by the FRM with a benchmarking procedure.

New contract negotiation and the first bind of an application to the VRES require CORBA interactions between the framework modules, and therefore should be done before starting the actual processing of the application. The runtime usage of the disk – i.e., issuing read and write requests – is not affected by the overhead of contacting different components neither locally nor on remote machines.

4.3 Wireless LAN

FRSH/FORB framework supports communication over Wi-Fi network (also called *Wireless LAN* (WLAN)). The part of the framework responsible for Wi-Fi resource is called *FRSH WLAN Protocol* (FWP). This protocol takes advantage of IEEE 802.11e standard [IEEE, 2005]. More specifically it uses medium access technique called *Enhanced Distributed Channel Access* (EDCA) which provides differentiated access to medium by means of four access categories called (in decreasing "priorities") *voice*, *video*, *best effort* and *background*. Within these categories, the classical *exponential back-off algorithm* is used to lower the probability of collision. Note that although EDCA improves communication capabilities for real-time applications,

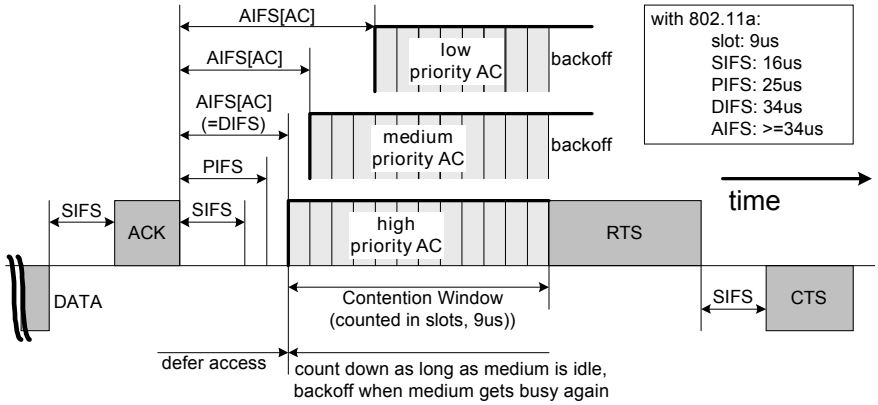

Figure 4.2: Principles of EDCA MAC algorithm (source: [Mangold et al., 2002]).

it still uses a probabilistic approach in the medium access algorithm and the guarantees provided by this algorithm are not "hard". FWP provides FRSH API for creating communication endpoints, binding them to virtual resources (VRES) and sending/receiving messages over them. Internally, FWP uses *User Datagram Protocol* (UDP) protocol for sending the messages.

4.3.1 Enhanced Distributed Channel Access (EDCA)

Wireless networks based on IEEE 802.11 [IEEE, 1999] WLANs became extremely popular over the last decade. Similarly as for wired network, it turned out early that QoS provided by these networks is not sufficient [Ni et al., 2004]. At that time there were several proposals for how to add QoS to WLANs. In 2005 some of these proposal were standardized in a new standard IEEE 802.11e [IEEE, 2005]. One of the standardized extensions was EDCA medium access method, which provides class-based differentiated QoS for IEEE 802.11 WLANs.

The EDCA is an extended version of legacy *Carrier Sense Multiple Access/Collision Avoidance* (CSMA/CA) and defines four *Access Categories* ACs named AC_VO (voice), AC_VI (video), AC_BE (best effort) and AC_BK (background). Each category provides different QoS and the difference is based on the values of parameters used by *Medium Access Control* (MAC) algorithm.

EDCA Medium Access Control Algorithm

The basis of EDCA algorithm can be described as follows (see also Figure 4.2): if a station has something to transmit, it has to wait for an *Arbitration Interframe Space* (AIFS) since the time it detects idle medium. After the end of AIFS starts an *Contention Window* (CW). The station is supposed to start transmitting after a contention window timer, which is used to count backoff slots, reaches zero value. The contention window timer is initialized to a uniformly distributed random value between zero and CW and is decreased on every backoff slot boundary when the

AC	CWmin	CWmax	AIFSN	TXOP limit
AC_VO	(aCWmin + 1/4) − 1	(aCWmin + 1)/2 − 1	2	1.504 ms
AC_VI	(aCWmin + 1/2) − 1	aCWmin	2	3.008 ms
AC_BE	aCWmin	aCWmax	3	0
AC_BK	aCWmin	aCWmax	7	0

Table 4.1: Default EDCA parameters for IEEE 802.11g PHY. The aCWmin value is defined as 31 for rates 1, 2, 5.5 and 11 Mbps and 15 for other rates offered by 802.11g. The value of aCWmax is 1023.

medium is idle. After a successful transmission the value CW is set to CW_{min}, unsuccessful transmission causes the CW value to double unless it reaches CW_{max}. Another principle used to lower the probability of collision and to circumvent the hidden node problem is RTS/CTS mechanism (see Figure 4.2). Instead of sending directly a long data packet a station first sends a short *Request to Send* (RTS) packet. Upon successful reception of RTS by *Access Point* (AP) it responds with *Clear to Send* (CTS) packet. The advantage here is that the probability of colliding short packets as CTS or RTS is lower than for long data packets. Also CTS packet is typically received by other stations even if they did not receive the corresponding RTS packets (the sender was hidden for them).

The access categories differs in the values of these AC specific parameters:

– ***Arbitration Interframe Space Number* (AIFSN)** determines the length of AIFS, i.e. the interval during that medium is idle before backoff algorithm is started. The length of AIFS is computed as follows:

$$\text{AIFS}[AC] = \text{AIFSN}[AC] * slot\_time + \text{SIFS}\,[\mu\text{s}] \tag{4.1}$$

Values of *slot_time* and *Short Inter-Frame Space* (SIFS) depend on physical layer. For IEEE 802.11a $slot_time = 9\,\mu$s and SIFS $= 16\,\mu$s, IEEE 802.11g has $slot_time = 20\,\mu$s and SIFS $= 10\,\mu$s.

The minimal value of AIFSN is 2 (in which case the length of AIFS equals to *Distributed (Coordination Function) Interframe Space* (DIFS) used in IEEE 802.11) for non-AP station and the maximal value is 15. For AP station the minimal value is 1.

– **Minimum and maximum values of contention window (CW)**: CW_{min} and CW_{max}. See above the brief description of these parameters. Both values are power of two.

– **TXOP limit** determines the duration of permission to transmit. During this interval, the station uses only SIFSs to delimit individual frames and hence the other stations has no chance to decrement their contention window timers and transmit another packets.

The table 4.1 shows default settings of AC parameters. A TXOP limit value of 0 indicates that only single data frame (in addition to RTS/CTS exchange) can be transmitted at any rate for each TXOP.

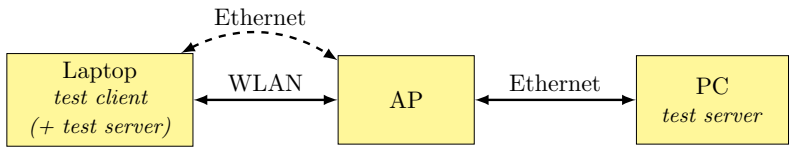

Figure 4.3: Our testbed setup

EDCA parameters are stored locally at the QSTA and can be dynamically updated by the QoS access point (QAP) that distributes them to STAs in the management frames (the beacon, and in probe and re-association response frames). This adjustment allows the stations in the network to adjust to changing conditions, and gives the QAP the ability to manage overall QoS performance.

Contention-based medium access is susceptible to severe performance degradation due to the overload and noise on medium especially. In overload conditions, the contention windows become large, and more and more time is spent in backoff delays rather than by sending data. Admission control is needed to regulate the amount of data contending for the medium.

4.3.2 Testbed setup

Before developing the admission control algorithm for *FRSH Resource Manager* (FRM) we wanted to evaluate the properties of real IEEE 802.11e hardware to check that its behavior corresponds to the theoretical results presented in many theoretical papers such as [Xiao, 2004, Vittorio and Lo Bello, 2007]. The experiments were conducted on a testbed (see Figure 4.3) where one station was a laptop running Linux 2.6.24 with Ovislink WMM-3000PCM Cardbus adapter containing Ralink RT2600 chip (*rt61pci* driver). The station was associated to an AP Linksys WRT54G ver. 7, which was connected by 100 Mbps Ethernet to a PC running Linux 2.6.22. This way we had two stations (Laptop and AP) competing for the wireless medium.

It is clear that with this setup the probability of collision is quite low and the measured results can be different if there are more stations.

There are two testing applications. The *test server* serves as a loopback i.e. it listens for incoming UDP packets and sends them back to whoever who sent them, using the same *Type of Service* (TOS) flags. The *test client* produces data streams of desired average bandwidth, packet size and access category, sends them to the *test server* and processes the responses. Both server and client add time-stamps to every packet payload so that the client can measure round-trip time and (if time in the server and client stations is synchronized) one-way delays. The scheduling policy of both *test client* and *test server* sending/receiving threads is set to *SCHED_FIFO* to minimize the influence of CPU scheduler to measured times. It was not necessary to use any real-time extensions to Linux kernel as we have used low baud rates and the latencies caused by the OS scheduler was several orders of magnitude below network latencies.

The parameters of EDCA queues were set to the default values defined in [IEEE, 2005] , which are mentioned in Table 4.2. Transmission bit-rate was fixed to 1 Mbit/s

AC	AIFSN	CWmin	CWmax	Burst
AC_VO	2	3	7	15
AC_VI	2	7	15	30
AC_BE	3	15	1023	0
AC_BK	7	15	1023	0

Table 4.2: EDCA parameters for experiments.

on both sides to eliminate automatic rate control algorithms.

Since it was not possible to synchronize the clocks of the two computers with precision of a few hundred microseconds using Network Time Protocol (NTP) and we didn't want to use a more complicated synchronization techniques [Guo, 2006],for some experiments the setup was different: Both *test server* and *test client* were running in the same machine but the communication between WLAN and Ethernet interfaces were handled externally thanks to the *send-to-self* patch[1] for Linux kernel (see dashed line in Figure 4.3).

4.3.3 Experiments

The results of our experiments are presented in the graphs below in the form of cumulative histogram of delay. The horizontal axis represents the measured time (divided by two in the case of round-trip time) and the vertical axis represents the percentage of received packets with delay less or equal to the value on the horizontal axis. The exact parameters of streams generated by *test client* application are shown above every plot. The packet size values don't include any headers (UDP, IP, MAC). As the measured time includes the transmission time of the packet, we keep the packet size of all streams the same to eliminate the influence of different transmission time. Also, to excite all possible behavior of the stochastic system represented by IEEE 802.11 MAC layer, the delay between send attempts is an evenly distributed random number with mean value of desired period and maximal deviation equal to 50% of period. All experiments ran for 60 seconds.

Basic Experiments

The experiment in Fig. 4.4 shows the delays of all access categories (voice, video, best-effort, background) under non-saturation condition. The worst-case delay of AC_VO was around 70 ms and the one of AC_BK was 240 ms.

In the second experiment (see Figure 4.5), we have slightly increased the bandwidth of all streams and AC_BK queue got into saturation. We can see that we have received only 10 packets per second instead of requested 17 and the worst-case delay increased to 5.4 seconds. As we show below, the major part of this delay is caused by waiting in transmission queues.

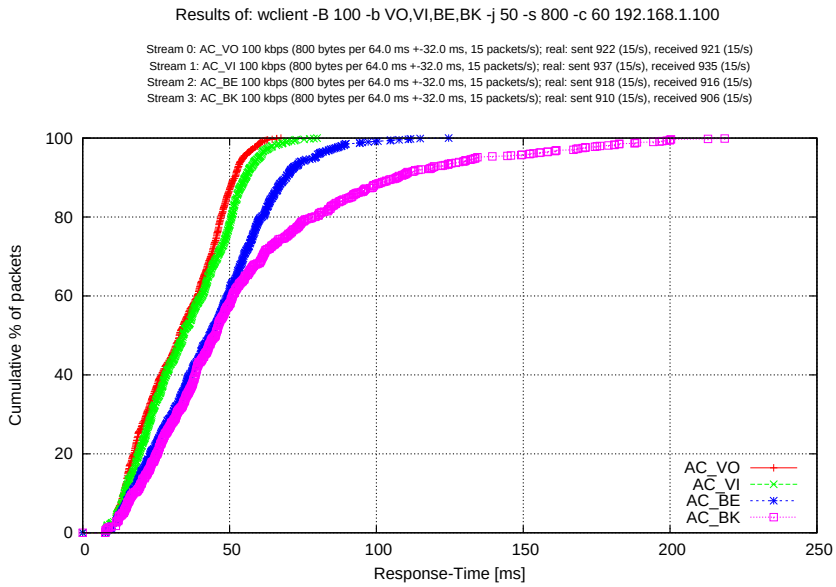

Figure 4.4: Delay of all access categories under non-saturation condition.

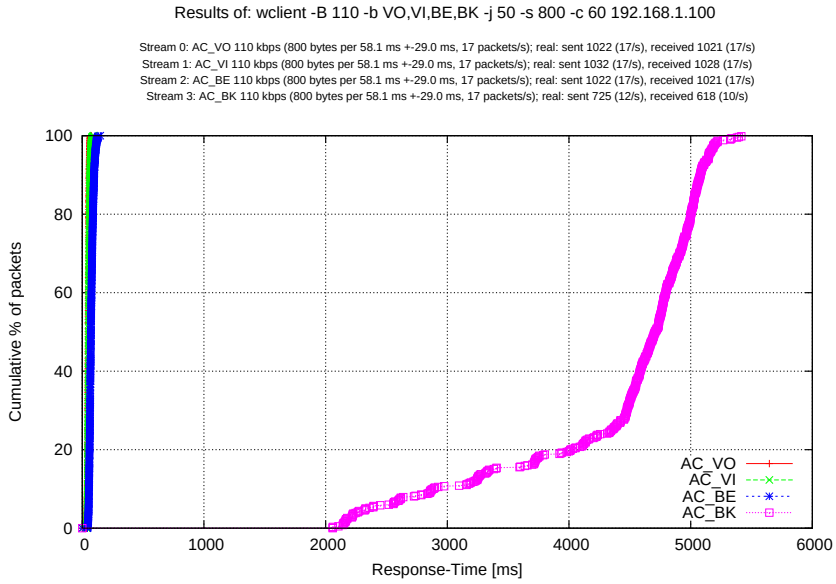

Figure 4.5: Delay of all access categories where AC_BK is under saturation.

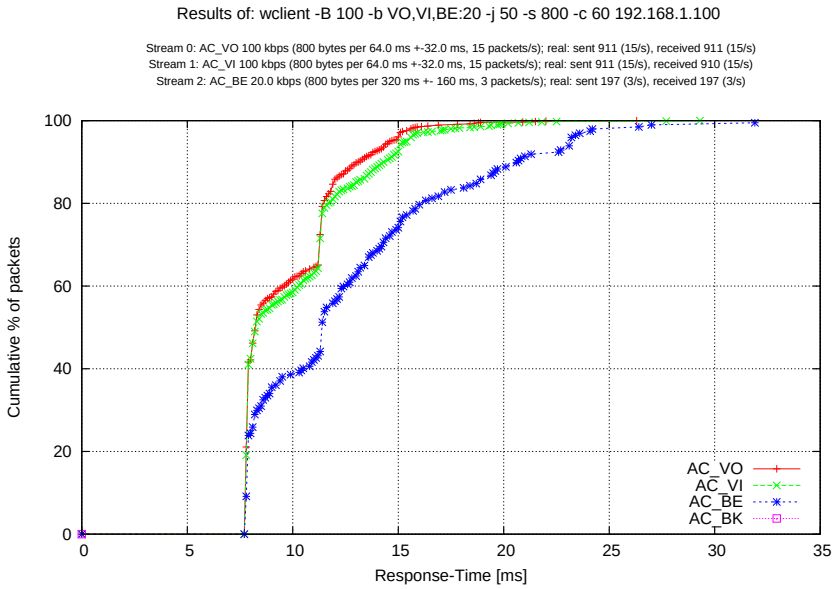

Figure 4.6: Influence of AC_BE at 20 kbps on AC_VO and AC_VI.

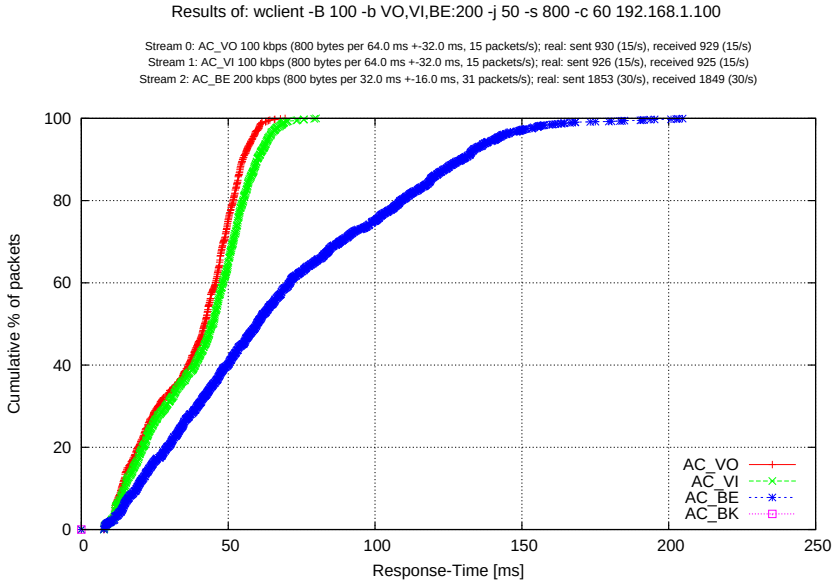

Figure 4.7: Influence of AC_BE at 200 kbps on AC_VO and AC_VI.

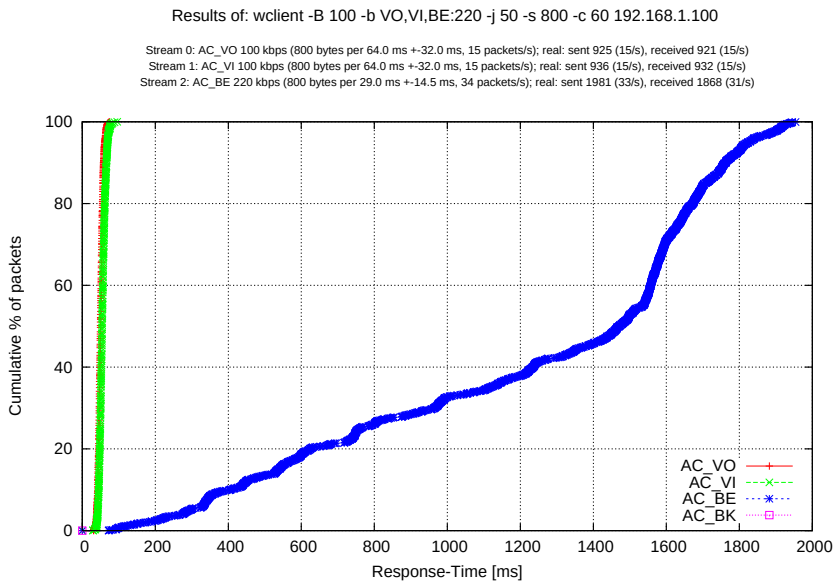

Figure 4.8: Influence of AC_BE at 220 kbps on AC_VO and AC_VI.

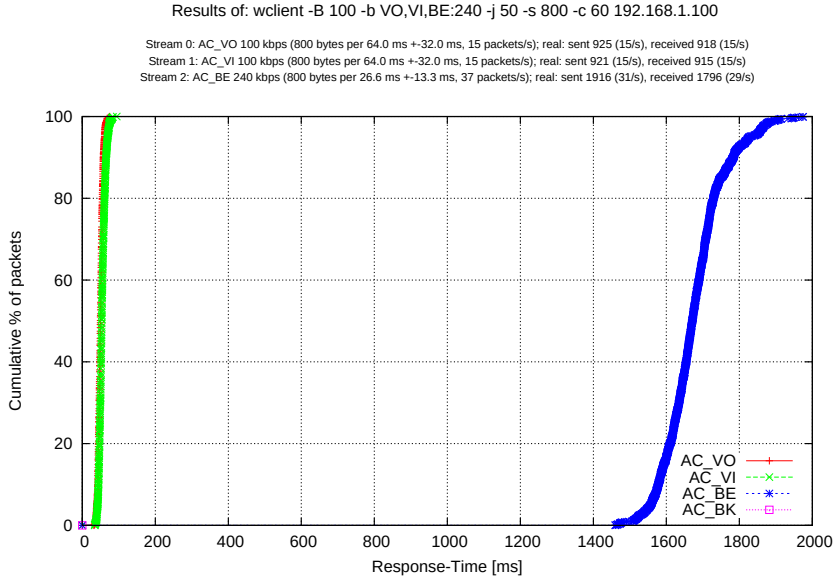

Figure 4.9: Influence of fully saturated AC_BE to AC_VO and AC_VI.

Access Category Interdependencies

In these experiments we have measured the influence of "low-priority" AC_BE stream of different bandwidths to the delays of AC_VO and AC_VI streams. The bandwidth of AC_VO and AC_VI was fixed to 100 kbit/s and the bandwidth of AC_BE was changed from zero to 340 kbit/s.

Figure 4.6 shows a non-saturated case with 20 kbps AC_BE stream. All subsequent AC_BE bandwidths produced similar results until 200 kbps (Figure 4.7), where AC_BE bandwidth reached the saturation boundary and the worst-case delay increased to 210 ms. If AC_BE bandwidth is slightly increased to 220 kbps (Figure 4.8), the worst-case delay increases rapidly to 2 seconds and remains that high even for higher requested bandwidths. In the saturated case, the real AC_BE bandwidth did not exceed 200 kbps as can be seen from the rates of received packets ($31\,\mathrm{pkt/s} = 31 \times 800 \times 8\,\mathrm{bps} \doteq 198\,\mathrm{kbps}$). The worst-case delay of AC_VO and AC_VI ranges from 30 ms in the non-saturated case, to 100 ms in all saturated cases. This means that the influence of a "low-priority" stream to the delay of "high-priority" streams is limited and therefore this communication is suitable for soft real-time applications.

The ramp between 50 and 1550 ms on the saturated (AC_BK) graph in Fig. 4.8 corresponds to the state where OS/HW queues are being filled. In the beginning, the queues are empty so that packets don't wait in queues and the delay is short. As the queues are filled more and more the time spent in queues becomes longer. Then there is notable turn at 1550 ms, which corresponds to the state where the queues are fully filled and packets starts to be dropped. The shape of the curve between 1450 and 2000 ms is similar to the one from Figure 4.9. It means, that when the test from Fig. 4.9 started, the queues were already filled from the previous experiment and the experienced delay is the longest possible one.

Influence of the Queue Size

When we decreased the maximum size of socket buffers (with `setsockopt` and `SO_SNDBUF`), we were able to decrease the maximum delay (see Figure 4.10). With zero byte buffers (the top graph), the lowest delay was achieved, but obviously when multiple threads use the same socket (in non-blocking mode) then the packet loss was higher (not depicted in Figure 4.10).

Differences Between Access Point and Station

Since the AIFSN, CWmin and CWmax parameters can have different values for AP and non-AP *Stations* (STAs) in the same network [IEEE, 2005], it was also evaluated how one-way communication delay depends on the direction (non-AP to AP and vice versa). This experiment used the testbed setup with both *test client* and *test server* in one station (dashed lines in Fig. 4.3), which enables precise measurement of one-way delays. The results can be seen in Figure 4.11. The top figure shows that the delays are shorter when the AP is the sender. This is obvious because the AP must have precedence over non-AP stations or otherwise it would be overloaded by

[1] `http://www.ssi.bg/~{}ja/#loop`

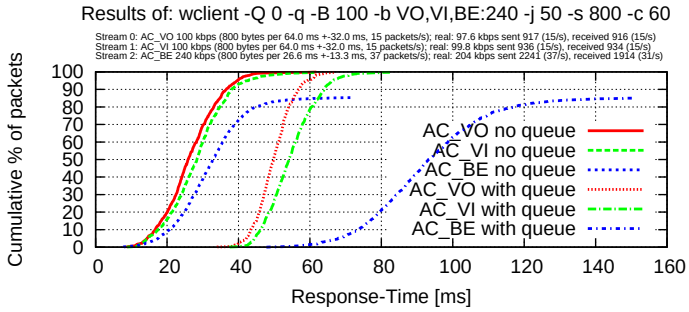

Figure 4.10: Influence of socket send queue size to delays. Two scenarios with SO_SNDBUF set to 0 and 3000.

packets coming from other stations. The bottom figure is provided for completeness and shows the data from the experiment in the same form as in the previous figures, i.e. the sum of two one-way delays divided by two.

The difference between AP and non-AP STA is problematic as we do not know the exact settings of AP's EDCA parameters and hence it is impossible to incorporate the AP behavior in an exact analysis.

Summary

From the results presented in this section we conclude that, in order to use IEEE 802.11e networks for real-time communication:

1. Saturation must be avoided for high priority ACs (VO, VI).
2. If saturation is not avoided for non-real-time (background) access categories, communication delay of real-time traffic increases (approximately by 100%).
3. Decreasing the number of socket buffers lowers delays but degrades performance and increases packet loss.

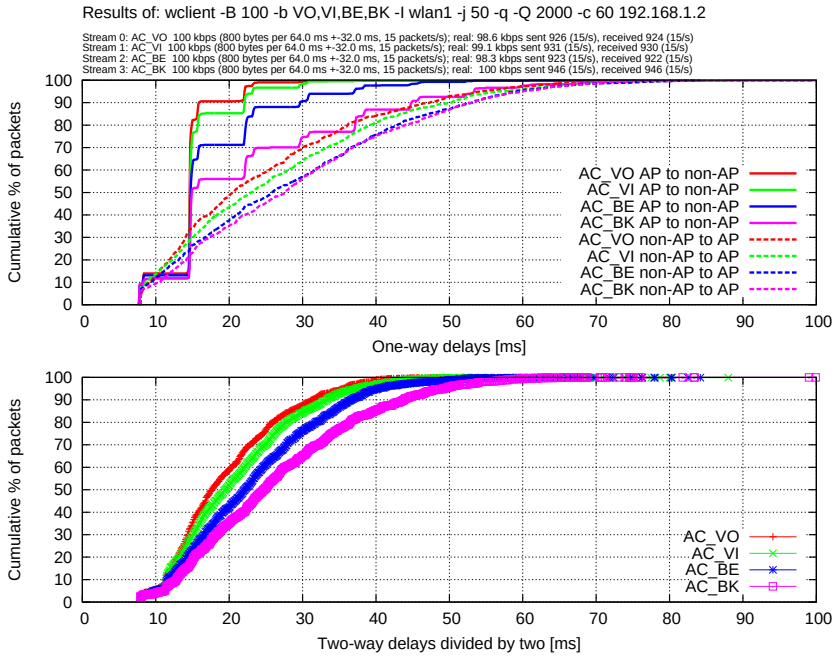

Figure 4.11: Difference in communication delays between AP and non-AP transmitters.

4.3.4 Simple Admission Test

With respect to the summary in the previous section and before implementing quite difficult and computation demanding admission test based on [Engelstad and Østerbø, 2006], we have decided to start with a very simple and not much universal *utilization based* admission test whose goal is only to avoid saturation. Later, it turned out that such a test is sufficient for the use in FRSH/FORB framework (see section 5.2).

For each contract (stream) we determine how many UDP packets may be sent according to the negotiated budget B (bytes per period) and *Maximum Transmission Unit* (MTU). For each packet we calculate its transmission time including all possible overheads:

- lower layer headers (UDP, *Internet Protocol* (IP), *Logical Link Control* (LLC), MAC),
- *Acknowledge* (ACK) packet and SIFS,
- *Physical Layer Convergence Protocol* (PLCP) preamble and PLCP header (and signal extension for *Extended Rate PHYs* (ERP) rates defined by IEEE 802.11g) for both data and ACK packets,
- AIFS and
- estimation of backoff time (see below).

AC	value
C_{VI}	6
C_{VO}	5
C_{BE}	2
C_{BK}	2

Table 4.3: Values of the constants used in the estimation of the backoff times.

The following equations provide the details of calculations in the admission test.

$$t_{\mathrm{frame}}(bytes) = t_{\mathrm{plcp}} + 8 \cdot bytes/bitrate \tag{4.2}$$

$$b(payload) = payload + l_{\mathrm{udp,ip,llc,mac,fcs}} \tag{4.3}$$

$$t_{\mathrm{tx}}(p) = t_{\mathrm{bkoff}} + t_{\mathrm{frame}}(b(p)) + t_{\mathrm{sifs}} + t_{\mathrm{frame}}(ack) \tag{4.4}$$

$$t_B = \lfloor B/\mathrm{MTU} \rfloor \, t_{\mathrm{tx}}(\mathrm{MTU}) + t_{\mathrm{tx}}(B \bmod \mathrm{MTU}) \tag{4.5}$$

These expressions are the same for all access categories, with the only exception which is the backoff time t_{bkoff}. Its calculation depend on AC as detailed in the next paragraph.

The estimation of the backoff time is based on AC parameters. First, average backoff time is calculated for free medium (i.e. when there is only one transmitting station) and the result is then multiplied by an empirical constant C_i, which is different for each AC. The intention of the multiplication is to roughly represent the influence of collisions. The backoff time calculated according to the is given by the following equation:

$$t_{\mathrm{bkoff},i} = C_i \cdot t_{\mathrm{slot}} \cdot (\mathrm{AIFSN}[i] + \mathrm{CW}_{\mathrm{min}}[i]/2). \tag{4.6}$$

The values of C_i are given in Table 4.3. The t_{slot} equals to 20 μs as defined in [IEEE, 1999].

Because of this simple estimation, this test is not very precise in the general case, because the relationships between the traffic and the number of collisions are more complex and retransmission time is not counted. On the other hand, the purpose of the admission test is to avoid saturation and therefore keep the number of collisions low. For that reason, we do not need complicated models to estimate EDCA back-off time and we can use constant values for this delay, one for each access category.

For each of the already accepted and new contracts, the admission test calculates the length of bus occupancy t_{B_k} and divides it by the contract period T_k to get so called *partial utilization value*. These numbers are then summed to form the *total utilization* U:

$$U = \sum_{\forall k} \frac{t_{B_k}}{T_k} \tag{4.7}$$

If the total utilization U is less than 96% (empirically determined value) the new contract is accepted, otherwise it is rejected.

To illustrate the properties of this test three experiments have been performed to measure the throughput of the WLAN channel. The results were compared with the

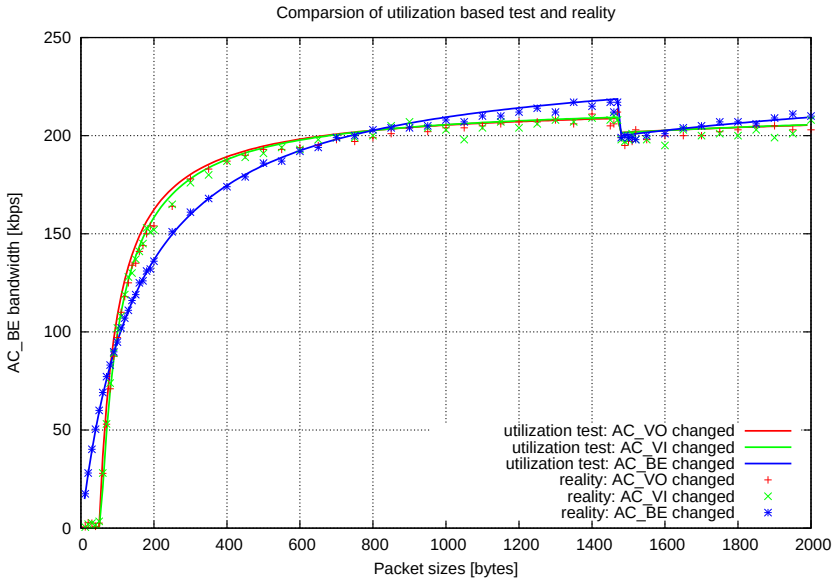

Figure 4.12: Comparison of the utilization based test with measured results for three different experiments.

results given by the utilization based test. After tuning the empirical constants, the results matched quite well. Of course, the situation might be different for another experiments but given the typical properties of the wireless networks (packet loss, sensitivity to external disturbances etc.) and the results from section 5.2, even such a simple test is sufficient for most real-world soft real-time applications.

In the experiments, we have measured the saturation bandwidth of AC_BE as a function of the size of packets in AC_VO, AC_VI and AC_BE respectively (see Fig. 4.12). In each of the three tests, the following streams were generated: AC_VO – 100 kbps, AC_VI – 100 kbps and AC_BE – 500 kbps. Since the Wi-Fi bandwidth was set to 1 Mbps and every packet is transmitted two times (once from source station to AP and once from AP to destination station), the network was fully saturated by these streams and the delays are similar as in Figure 4.9. The difference from the previous experiments is that the size of packets in one stream was being changed while the other two streams were formed by packets with 800 bytes of UDP data payload. By changing the size of the packets, we try to determine whether the overhead calculation used in the admission tests holds for various sizes of packets. Under these conditions the real (saturation) bandwidth of AC_BE was measured by observing the number of successfully received AC_BE packets during a time interval.

Figure 4.12 shows the measured saturation bandwidth together with the theoretical available bandwidth derived by the admission test algorithm. The theoretical bandwidth was achieved by binary chopping algorithm, which finds the maximum bandwidth of AC_BE stream which is admitted by the test. The solid lines correspond to the theoretical AC_BE bandwidth allowed by the admission test and

the points represent the measured saturation bandwidth for a particular experiment. The step at packet size of 1472 bytes is caused by IP protocol fragmentation (at MTU size) – at this point the UDP datagram was split into two packets and therefore the overhead doubles.

4.3.5 Integration of FWP in FRSH/FORB

As for any other resource, FWP implements resource manager and resource allocator components. FWP resource manager is responsible for two things:

1. It assigns the stream to the one of the four EDCA access categories according to the deadline specified by the application in the contract.

2. It checks that the overall bandwidth requested by all applications is lower than the bandwidth available. Currently, the available bandwidth is specified manually when the manager is started. The check is performed as described in the previous section.

   Currently, FWP works only when transmission bitrate is fixed. Since Wi-Fi network interface cards (NIC) normally change bitrate dynamically to cope with changing channel conditions, this constraint is quite limiting. In [Sojka et al., 2008], section 3.6.2, we describe, how FRSH/FRSH framework could support dynamically changing bitrate.

FWP resource allocator creates FWP virtual resources and configures their internally used sockets in such a way that the messages are sent through the EDCA access category specified by the manager. Every FWP VRES employs a traffic limiter to ensure that applications do not send more data within a period than they requested in the contract. If the application exhausts its budget, it is either blocked until the next replenishment time (in case of synchronous send) or the message is queued and sent by VRES at the next replenishment time (asynchronous send).

4.4 Wireless Sensor Networks

FRSH/FORB framework has also been integrated with *Wireless Sensor Networks* (WSNs). As opposed to distributed systems connected by a conventional network, WSNs are based on nodes with limited computational power. Their purpose is usually to collect some data measured in the nodes and to transport the data to a central point. The WSN nodes cannot run the full FRSH/FORB framework and hence the framework does not have control over the node's resources. Instead, the whole WSN is treated as a single resource which is used by FRSH/FORB applications as a data source.

There are two types of WSN integrated in FRSH/FORB. Both have different architecture and provide different guarantees. The following sections describe the integration of these two WSNs to FRSH/FORB. The complete description of the involved protocols and admission tests is provided in [Sojka et al., 2008].

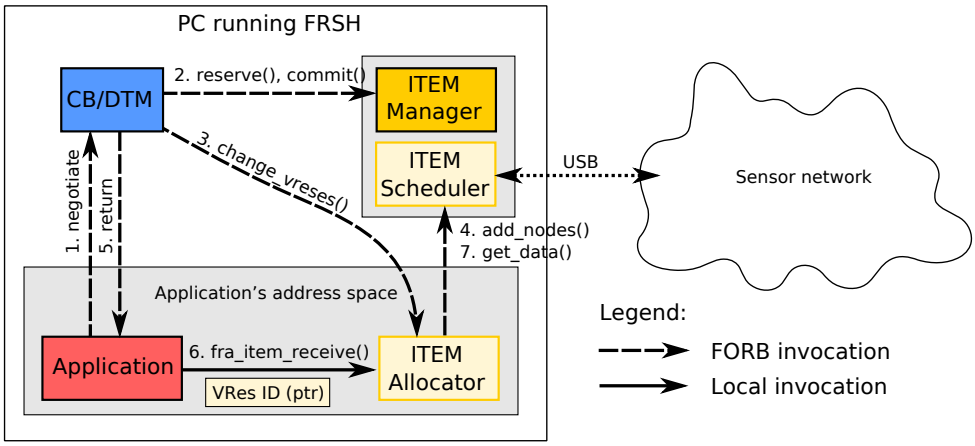

Figure 4.13: Integration of ITEM protocol with FRSH/FORB.

4.4.1 ITEM Network

Integrated TDMA and E-ASAP (ITEM) was developed by J. Trdlička and is described in-depth by [Sigh et al., 2008]. It is an implementation of adaptive *Time Division Multiple Access* (TDMA) protocol for use in WSN. The network operates in cycles and the length of the cycle depends on the number of nodes the data is gathered from. The more nodes, to longer cycle. The ITEM-based network allows the FRSH/FORB application to specify in the contract from which nodes it wants to receive data and what should be their maximal age (deadline). The contract is accepted if the network can fulfill the request and it is rejected when there are requests to gather data from too many nodes with respect to the lowest deadline specified in any contract. As an example consider a case where one application wants to receive data frequently from a single node. The network is therefore configured to short TDMA cycle to fulfill that request. Later, If another application requests data reception from, let say, 32 nodes with sufficiently long deadline, this request cannot be fulfilled with the short TDMA cycle, which is needed for the first application.

The modules of ITEM resource and their interaction during contract negotiation are shown in Figure 4.13, which is a simplified version of Figure 3.3 from page 30. The support of ITEM resource consists of the following modules:

ITEM resource manager provides the admission test to check whether the deadlines requested by applications can be met, given the number of nodes which are requested to send data.

ITEM scheduler is a module responsible for configuring the wireless sensor network according to the application requests (contracts) and for distribution the received data to applications. Technically, to simplify the framework setup this module is implemented in the same process as ITEM resource manager, but logically, these modules are completely independent.

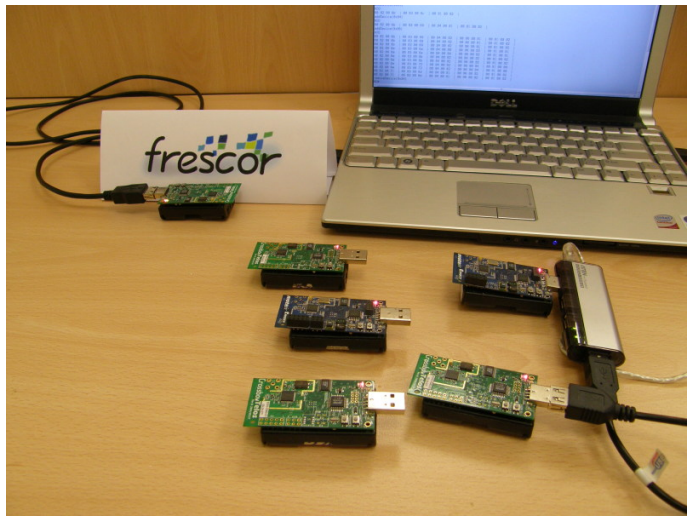

Figure 4.14: Demonstration of ITEM wireless sensor network with FRSH/FORB (FRESCOR project).

ITEM allocator is a lightweight module whose purpose is only to redirect VRES manipulation and data reception requests to the ITEM scheduler.

The contract negotiation process corresponds to the one described in Section 3.3.2. After a contract for the ITEM resource is negotiated, the application can use *fra_item_receive()* function to gather sensory data from the network. FORB is used to retrieve the data from the ITEM scheduler.

4.4.2 Cluster-Tree Network Supporting Variable Data Flows

Supporting time-sensitive WSN applications implies to predict and guarantee bounded end-to-end communication delays. Thus, the FRSH/FORB framework provides the worst-case analysis using the Network Calculus framework [Boudec and Thiran, 2004] to ensure that data traffic generated by accepted contracts does not exceed any user-defined deadlines. Similarly to ITEM resource, each FRSH/FORB contract specifies the sensory data flows generated by a given set of sensor nodes. The difference from ITEM is in the underlying network protocol (IEEE 802.15.4/ZigBee) and network topology (Cluster-Tree). Detailed description of the used analysis method can be found in [Jurčík et al., 2008], additional implementation details are provided in [Sojka et al., 2008].

In this kind of WSN, each data flow is defined by the following parameters and hence these parameters have to be specified in the contracts for this resource:

- *node_id* is the address of the data flow's source (16-bit address in case of IEEE 802.15.4/ZigBee protocols),
- *budget* is the size of the generated packet's payload [bits] and

– *period* is the time between two consecutive packets [sec].

FRSH Resource Manager (FRM) uses this information to calculate the burst size b and arrival rate r, which are the parameters needed by the (Network Calculus based) analytical framework [Jurčík et al., 2008].

4.5 FPGA

From the FRSH/FORB framework point of view, FPGA is an additional computing resource. The FPGA is able to constitute one or more FPGA cores. Each core can substitute a part of software for a particular task, lowering overall CPU load. Within the frame of resource reservation, the goal of the framework is to decide, whether to execute a task entirely in software, or whether to utilize the FPGA and use an FPGA core to accelerate the task.

This section describes the integration of FPGAs resource into the FRSH/FORB framework. It is based on [Peca et al., 2009] which also contains additional information and case studies.

4.5.1 FPGA reconfiguration capabilities

One or more FPGA cores can occupy the FPGA at once. In dynamic environments i.e. in application domain of FRSH/FORB framework, where application needs change over time, it may be desirable to reconfigure the FPGA during run-time, i.e. to interchange currently used FPGA core set by a different one. There are two possible reconfiguration paradigms: dynamic and static.

Dynamic reconfiguration

With dynamic (often called partial) reconfiguration, content of the FPGA is changed only partially during the reconfiguration. Individual FPGA cores can be loaded into free FPGA areas, preserving other cores, already present in the FPGA.

The main advantage of the dynamic reconfiguration is its flexibility in that FPGA cores can be loaded independently up to available capacity. The run-time reconfiguration can proceed during uninterrupted operation of running FPGA cores. There are several disadvantages of dynamic reconfiguration [Peca et al., 2009]. The main disadvantage is difficult design. Also, a resulting FPGA implementation of cores is slightly suboptimal due to design constraints, in comparison with the static reconfiguration.

The dynamic reconfiguration is a promising paradigm, however, it still is not a mature technology in industrial practice. Although FPGA manufacturers offer tools and application notes for implementation of dynamic reconfiguration [Xilinx, 2004], and a research has been done [Kohout, 2007, Donato et al., 2005], it is still a very difficult way. For the integration with FRSH/FORB framework, the dynamic reconfiguration was not employed.

Static reconfiguration

For every desirable set of FPGA cores, an FPGA bitstream[2] is created (compiled) offline. Then, it is possible to replace whole core set by another one.

There are no reconfiguration specific design constraints imposed. Every desirable core set is compiled as a whole, as one large hardware, containing all the selected FPGA cores. If there are N FPGA cores present in the system, there are up to 2^N core sets. However, substiantially smaller number has to be actually compiled. Some of combinations can be impractical or useless in an application. Also, there is no need to compile a set, if its superset already fits into the FPGA (unless we concern about power consumption). Again, if a set cannot fit into the FPGA as a whole, it will not be compiled, as well as all of its supersets. The limit case of this paradigm is that only each one of the FPGA cores itself is compiled, and only one of the cores at a time can be loaded into the FPGA.

The following difficulties are encountered even in simple case of static reconfiguration:

- **HW/HW state transition** In contrary to dynamic reconfiguration, whole content of the FPGA is replaced during each reonfiguration. Thus, if a task, running on an FPGA core, has to continue after the reconfiguration, it should be interrupted and its state must be transferred to new incarnation of the same FPGA core. However, the same core in a new bitstream may be placed in a different location, moreover, it may use slightly different logic building blocks (due to optimizations during compilation). Solution of such a general transition is a very difficult task. If the state transition is not implemented, the reconfiguration can proceed only when all FPGA cores are inactive.

- **HW/SW state transition** The issue of HW/SW transition is the very same as in case of the dynamic reconfiguration, see [Peca et al., 2009].

- **Real-time loading** The FPGA is loaded whole at once, without preservation of any running cores or content. Thus, it is possible to use common software tools. If a real-time operation is required, a loading time should be taken into account. Also, an interface which assure deterministic timing should be used on a CPU side. On the FPGA side, e.g. a *Joint Test Action Group* (JTAG) boundary-scan interface may be used.

The static reconfiguration is simple to use, a design is not specifically constrained. However, the HW/HW state transition is very difficult. The other big disadvantage is that every desirable combination of FPGA cores must be precompiled and stored somewhere in a memory. With growing number of the FPGA cores, there is a combinatoric explosion of possible core sets. Selecting only few of them results in a waste of possibly utilizable FPGA capacity. Static reconfiguration is considered for integration with FRSH/FORB framework.

[2]Bitstream is a serialized representation (block of data), desribing an FPGA content on the lowest possible level.

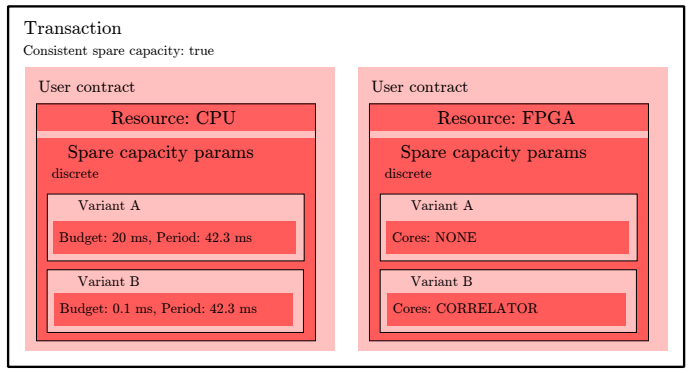

Figure 4.15: Example of data structures describing the transaction involving CPU and FPGA in the contract framework. There are two variants of possible task execution: A – software only and B – FPGA accelerated.

4.5.2 FRSH/FORB contracts for FPGA resources

To support FPGAs in the contract framework a way for applications to specify their requirements in contracts has to be defined. In the context of FRSH/FORB, FPGA cannot work as stand-alone computing entity; it is used as a coprocessor which means it is always accompanied by CPU. Therefore, the contract for FPGA resource has always to be accompanied by a CPU contract forming a transaction (see Section 3.3.4).

The responsibilities of the contract framework with respect to the FPGAs are the following:

1. Decide which cores should be loaded to the FPGA depending on application requirements.

2. For applications that can run their tasks either entirely in software or accelerated by an FPGA core, decide which application will run which variant.

For the framework to provide this functionality, applications must specify which FPGA cores they need as well as their CPU requirements for accelerated and software only (if available) variants.

As was mentioned above, for proper functionality, contracts for FPGA and for CPU have to be specified in a transaction. An example of such a transaction is depicted in Figure 4.15. It shows a real transaction used in the case study from [Peca et al., 2009]. The software only variant (denoted as A) needs to utilize the CPU for 20 ms every 42.3 ms, while the FPGA accelerated variant (B) needs only 0.1 ms on CPU and an FPGA core called *CORRELATOR*. The transaction requests consistent allocation of spare capacity, which means that the framework must allocate the same-named variants for both contracts, i.e. it has no sense to use simultaneously variant B for CPU and variant A for FPGA.

5 Framework Evaluation

In this section experimental results of the validation of the proposed framework are presented. The experimental validation aims to gather overhead figures for the contract negotiations in the framework (Section 5.1), and to highlight its capabilities in the provisioning of guarantees to individual applications. In Section 5.2, we show the capability of the Wi-Fi resource to temporally isolate applications from each other. Readers interested in temporal isolation capabilities of CPU and disk resource are kindly referred to [Sojka et al., 2010], where co-authors evaluated these resources. Finally, we present the experimental results gathered on the integrated case-study presented in Section 5.3, where contracts for the three types of resources (CPU, network and disk) are all used at the same time.

All experimental results have been gathered on a Pentium 4 at 2.4 GHz with 2 GB of RAM, running a Linux OS with a 2.6.29.1 kernel patched with BFQ and AQuoSA.

5.1 Negotiation Overhead

First, we measured the overhead of the negotiation procedure. To measure only the overhead of the framework and not the computation times of schedulability analysis and of VRES creation for a particular resource, we created a dummy resource, whose manager and allocator did nothing. In the experiment, we successively negotiated ten thousand contracts and measured the time of every single contract negotiation. The results are shown in Figure 5.1, with the lines labeled as "Negotiation". In case of local negotiation, both contract broker, resource manager and allocator were running on the same node. For remote negotiation, the manager was running on the second computer connected by a 100 Mbps Ethernet. The result is that the remote negotiation has a slightly higher overhead (as expected) and that in both cases the negotiation time is almost linearly dependent on the number of contracts in the system.

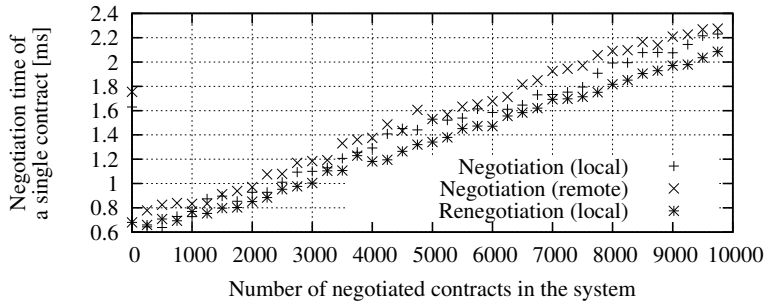

Figure 5.1: Contract negotiation time as a function of the number of negotiated contracts.

Then, we evaluated the overhead involved in renegotiation of existing contracts. This evaluation was done similarly to the previous experiment: we had several contracts in the system and we measured the time needed to renegotiate a single contract. The result is depicted again in Figure 5.1, with the line labeled as "Renegotiation". It can be seen that renegotiation takes, in average, slightly less time than the initial negotiation. The reason is that renegotiation involves less work to be done.

5.2 FRSH WLAN Protocol (FWP)

To evaluate the FWP protocol we mounted four Wi-Fi network interface cards (NICs) on our testbed PC, and an EDCA enabled Wi-Fi access point. The transmission bitrate was fixed to 12 Mbit/s. The Linux kernel was patched with *send-to-self* patch[1] which allows the messages addressed to the same computer to be sent over the external network. The messages were sent through one NIC and received through another NIC. Therefore, we did not need synchronized clocks on multiple computers to measure the communication delay.

Our testing application generated multiple data streams composed of messages with a 1024 bytes size, sent every 20 ms. The streams were received by the same application in different threads and the communication delays were measured. The messages of the i^{th} stream were sent from the ($i \mod 4$)-th NIC to the (($i + 1$) mod 4)-th NIC. Every test was run for 20 seconds so that every stream transmitted one thousand messages. We compared the results with FWP and without it.

The first experiment shows the consequence of limiting the total used bandwidth in the resource manager. The results can be seen in Figure 5.2. The horizontal axis shows the number of simultaneously generated streams and the vertical axis shows the maximal measured communication delay, its 95^{th} percentile and the packet loss. From the figure, it can be seen that the communication delay increases when the utilization grows. The highest bandwidth allowed by the FWP resource manager corresponds to eight streams. When the same experiment is repeated without FWP

[1] More information is available at `http://www.ssi.bg/~ja/#loop`.

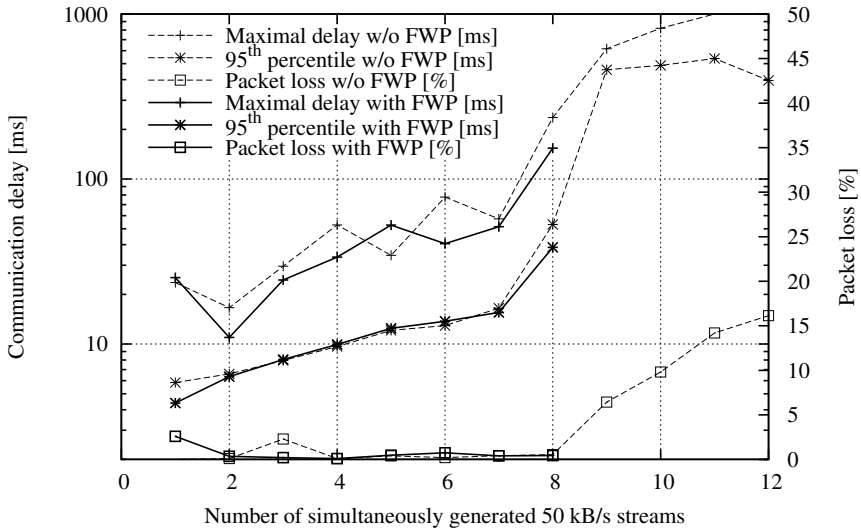

Figure 5.2: Illustration of how FWP resource manager maintains feasible bandwidth allocation.

(dashed lines), both communication delays and packet loss rise dramatically (note the logarithmic scale used for the delay axis) for nine simultaneous streams and beyond. By limiting the total bandwidth (here at eight streams), FWP is able to keep delays and packet loss low. Also note that the maximal delay is strongly influenced by the non-determinism of the EDCA medium access algorithm and by external disturbances. This explains why the maximal delay curve relative to FWP was occasionally higher than the one without it (for five streams).

In the second experiment we highlight the influence of the traffic limiter in FWP virtual resources (see the last paragraph of Section 4.3.5). The previous experiment was modified so that the delay between sending of messages in one stream was not fixed to 20 ms, but was a random variable uniformly distributed between 0 and 40 ms. The results can be seen in Figure 5.3. In order to see the difference, we had to bypass the FWP resource manager in all experiments, because the differences showed up only when the medium was saturated which is what the manager tries to prevent (see the limit of 8 streams in Figure 5.2). However, such situation may happen even when the manager is in use with disturbances which lower the link quality and decrease the available bandwidth. The results show that the maximum experienced delay (lines labeled as $+$) is approximately the same with and without the traffic limiter. The difference can be found in the 95^{th} percentile (lines labeled as $*$). For low utilization values, when the traffic limiter is active, the maximal delay is obviously close to the VRES period because some packets are delayed by the limiter. Without the limiter the delay is lower. However, the limiter helps when the medium is more saturated. For ten or more streams, the packet loss (lines labeled as $\square$) is lower with FWP than without it. Furthermore, for seven and more streams, the delay rises slower with the limiter than without it.

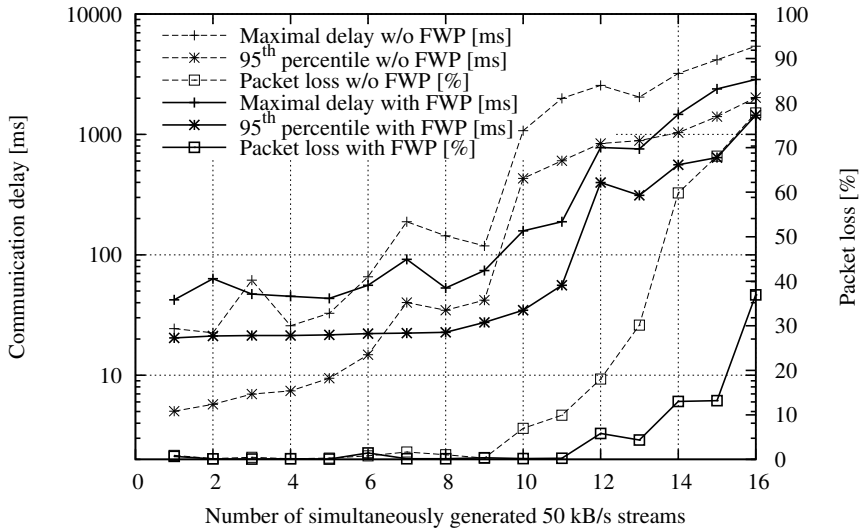

Figure 5.3: Demonstration of how traffic limiter in FWP VRES helps when Wi-Fi channel gets saturated.

A careful reader may wonder why there is "non-zero" packet loss for nine and more streams in Figure 5.2 and in Figure 5.3 only for twelve and more streams (the non-dashed line in the latter figure should roughly correspond to the dashed line in the former figure). The reason is the difference in channel conditions caused by external disturbances. When the experiment was run during working hours (the first one), other Wi-Fi networks on close channels disturbed us, while the second experiment was run in the evening when other wireless traffic was lower.

5.3 Integrated Case-Study

The proposed framework has been evaluated from the perspective of usability and achievable experimental results by realizing a concrete case-study application. It is constituted by a video-surveillance system with multiple cameras deployed in a building. Cameras are physically connected to the camera controller which communicates via Wi-Fi with the video server recording the video on a hard disk. The video is on-line and off-line surveyed by the operator, who dynamically decides upon the cameras to be recorded and the required quality of the video. Given the limited resources (CPU, WiFi and disk) the system presented in this paper allows the operator to dynamically (on-line) add/remove cameras and to change the video quality as long as the resource capacity is not exceeded (demonstrated in Figure 5.8).

The main components of the applications are the following (see Figure 5.4):

Camera Controller grabs videos from multiple connected video cameras, encodes them for transmission and sends them over the Wi-Fi network to the video

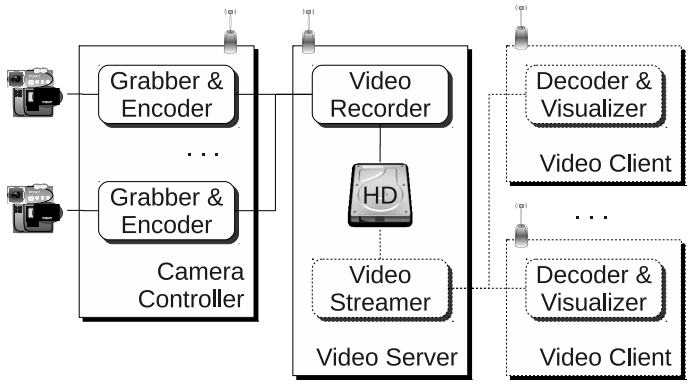

Figure 5.4: Case study block diagram.

server;

Video Server embeds two distinct components: the *Video Recorders* receive the video streams from the camera controller, re-encode them to an on-disk format and store them on a local hard drive; the *Video Streamer* reads back the stored videos and streams them over the network for being visualized by the video client(s);

Video Clients decode and visualize video streams, transmitted by the video streamer, on a local display.

In the following, we consider a concrete set-up of the general structure presented in Figure 5.4: one instance of the Camera Controller acquiring videos from up to three connected cameras and a single video client. This setup is depicted in Figure 5.5 together with resources involved in individual components.

The application has been realized by exploiting the open-source multimedia library FFMPEG[2], and the FRSH API described previously. The Video Client has been realized by using the VLC media player[3].

The video grabbing rate was selected to be 30 *frames per second* (fps) and the size of one frame was 320×240 pixels. The acquired video was encoded to an MPEG-4 stream with an `h263` codec and a bitrate of 1 Mbit/s. The stream was transmitted to the recording server using the *Real-Time Transport Protocol* (RTP)[4] which is based on the non-reliable UDP protocol. The recording server decoded each received stream, re-encoded it and stored it in MPEG-4 format onto the local disk. The video streamer is capable of streaming the recorded video either at full quality (same as used by the camera controller) or at lower quality 15 fps, 160×120, 100 kbit/s. Due to the environmental set-up and the distance between the Camera Controller and the Video Server, the wireless link between the Camera Controller and the Video Server was operating at a fixed bitrate of 12 Mbit/s.

[2]More information is available at: `http://ffmpeg.org/`.
[3]More information is available at: `http://www.videolan.org/vlc`.
[4]More information is available at: `ftp://ftp.isi.edu/in-notes/rfc3550.txt`.

Planned parameters	
Video rate	30 fps
Video resolution	320x240
Maximal video bandwidth	1 Mbit/s
Measured parameters	
Average frame size	3192 B
Avg video bandwidth	3192*30*8 = 751 kbit/s
I-frame every	12 frames = 0.4 s
Avg (max) I-frame size	8377 (8825)
Avg (max) P-frame size	2697 (5990)
CPU load of video encoding	15 %
CPU load of video recording	6 %

Table 5.1: Application parameters.

5.3.1 Parameter Tuning

The biggest difference between developing an application with and without the FRSH/FORB framework is that the developers need to provide contract parameters to the framework. It should be easy for strictly periodic applications with constant workload but it is more difficult for an application involving video compression where the workload differers every period (every processed video frame). This section summarizes our experience with determining proper contract parameters.

To properly setup contract parameters for a video processing application, some knowledge of video encoding and processing is required: The video stream is composed of different types of frames (I-frame, P-frame) and each type requires different CPU processing time, network and disk bandwidth. I-frames represent the full video frames while P-frames contain only differences from the previous frame(s). In our experiments, the size of encoded I-frames was, in average, three times bigger than the size of P-frames.

A correct set-up of the contract parameters is obviously determined by the application parameters. The parameters affecting resources requirements have been identified and measured. They are summarized in Table 5.1.

A correct set-up of the contract parameters has been fine-tuned based on a benchmarking phase. It was sufficient to benchmark the individual components separately because, as can be seen from the results in Section 5.3, the framework guarantees that after integration the negotiated parameters are reserved for the components in the same way as when the components were benchmarked in isolation.

Wi-Fi contract With the setting given in Table 5.1, the Wi-Fi network becomes the most limiting resource. It allows for transmission of approximately four streams, but due to a small "safety margin" the FWP manager admits only three streams. Although the maximal video bandwidth is 1 Mbit/s, the FWP manager needs to account for the real communication overhead (packet fragmentation, UDP and IP headers, MAC/LLC overhead – inter-frame spaces, contention window size etc.), which is in this case 47 %. Also note that every packet is transmitted two times

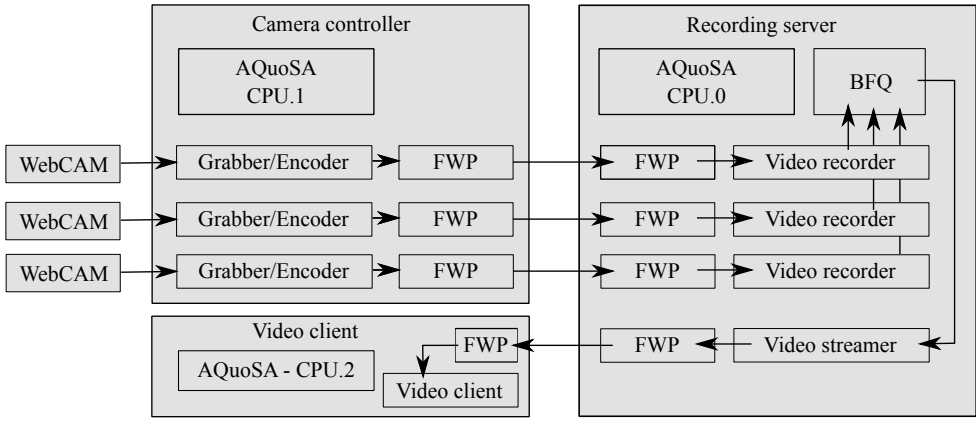

Figure 5.5: Detailed case study block diagram.

– once from the source station to the access point (AP) and once from the AP to the destination station. Therefore we get the total used Wi-Fi bandwidth as $3 \times 1\,\text{Mbit/s} \times 1.47 \times 2 = 8.82\,\text{Mbit/s}$.

As a consequence of different sizes of I-frames and P-frames, if the contract period is set to match the video frame rate, and the budget is set to be big enough for processing every I-frame, then approximately 64% ($1 - 3192/8825$) of the reserved bandwidth would be wasted due to the low resource utilization by P-frames. Since the Wi-Fi network is the bottleneck in our scenario, it was decided to set the period in the Wi-Fi contracts to 1 second and the budget to 125 KB, which corresponds to the maximum stream bandwidth. Deadline was set to 1/30 seconds so that the proper EDCA access category was used by FWP. The exact values of Wi-Fi contract attributes can be seen in the screen shot of a simple framework monitoring application in Figure 5.6. The list on the left side of the figure shows negotiated Wi-Fi contracts. For every video transmission there are two contracts: one for RTP protocol itself and one for accompanying RTCP protocol. The right side of the screen shot shows the attributes of the highlighted RTP contract.

CPU contract The CPU capacity on both the camera controller and the recording server was sufficient (one stream needs on average 15% of CPU on the camera controller and 6% on the recording server). Given the maximum of three streams, we can waste some CPU bandwidth by reserving more CPU than is actually needed. The period was set to match the frame rate and the budget was set to 25% of the period on the sender side, and to 10% of the period on the receiver side. It was experimentally checked that these values are sufficient even for processing the biggest I-frames.

Disk contract The disk throughput was measured to be 22 MB/s. Therefore, storing 125 KB/s video streams represented very low load for the disk. However, disk performance depends not only on bandwidth but also on seek patterns and

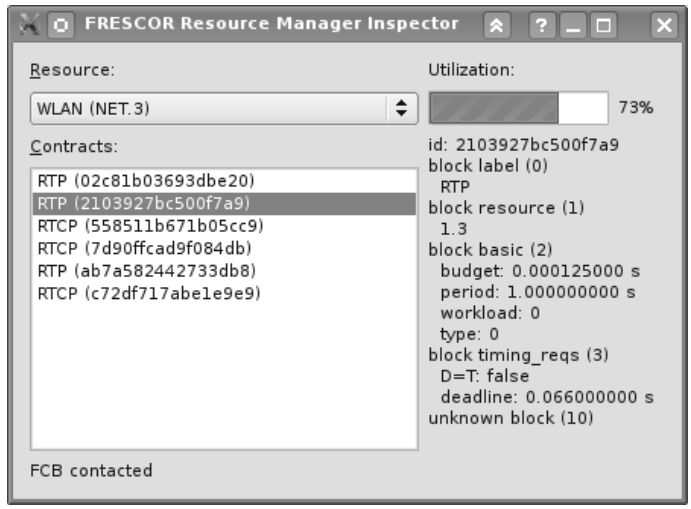

Figure 5.6: Screen shot of the graphical application for inspecting negotiated contracts in resource managers.

therefore it was very important to setup the contracts correctly. It can be seen in Figure 5.7 d) that the additional disk load has significant performance impact even on such low-bandwidth streams. It must be noted that in the current version of the framework, there is no special API for accessing the disk and in order to get the benefit from using disk reservations, applications must use "direct I/O" services instead of classical "buffered I/O" services when accessing the disk. In our case it was not straightforward to convert FFMPEG libraries to use direct I/O and it prolonged the case-study development time a lot. For future versions it would be beneficial if this limitation is removed.

The disk contract period was chosen to match the frame rate and the budget was set to 5 kB.

Summary Summarizing, the parameters for the various contracts in the FRSH API have been set-up as in Table 5.2. The results of experimental case study are presented in Section 5.3.

5.3.2 Experience Report

In this section we report on our experience with the framework which we gained during development of the case-study application.

– It was very helpful to have a *central view of the state of the framework*. We had a real-time monitoring application (see Figure 5.6) and the log of all framework operations (the excerpt is shown in Figure 5.8). It helped us to find quickly the reasons for reservation failures. We were able to generate the log because

Camera Controller	
Grabber/encoder budget	9 ms
Grabber/encoder period = deadline	1/30 s
FWP Budget	125 kB
FWP Period	1 s
FWP Deadline	1/30 s
Recording Server	
Writer CPU budget	5 ms
Writer CPU period = deadline	1/30 s
Writer Disk budget	5 kB
Writer Disk period	1/30 s
Streamer Disk budget	5 kB
Streamer Disk period	1/30 s
Streamer FWP Budget	12 (125) kB
Streamer FWP Period	1 s
Streamer FWP Deadline	1/15 (1/30) s
Video Client	
CPU budget	5 ms
CPU period = deadline	1/30 s

Table 5.2: Parameter values set in the FRSH contracts. The two values for Streamer correspond to the low and full video quality.

we setup the framework in a way that all contract negotiations went through the contract broker agent running in the recording server.

– Resource reservation helped us in *discovering certain errors* earlier than during integration phase. It happened when the actually used video stream bandwidth was higher (by mistake) than it was allowed by the negotiated network contract. This mistake was noticed due to jerky video on the video client. It would not be noticed without the framework because the available network bandwidth was sufficient for that single video stream.

– Determining the contract parameters often requires a benchmarking phase. In our case study, this benchmarking was done manually, which is a time consuming and error prone process. It would be much easier if the framework provided *resource usage statistics* such as the minimum/maximum/average consumed budget, deadline miss and budget overrun counts etc. Therefore, we plan to add such functionality to the framework in the future.

5.3.3 Experimental Results

In the case study, we ran the involved applications with and without the FRSH framework and under different loads. Every experiment lasted for 500 frames (cca 16 seconds). During those experiments several timing metrics were measured. The first metric was the average number of frames per second processed by the video recorder application. The second metric was the standard deviation of the time

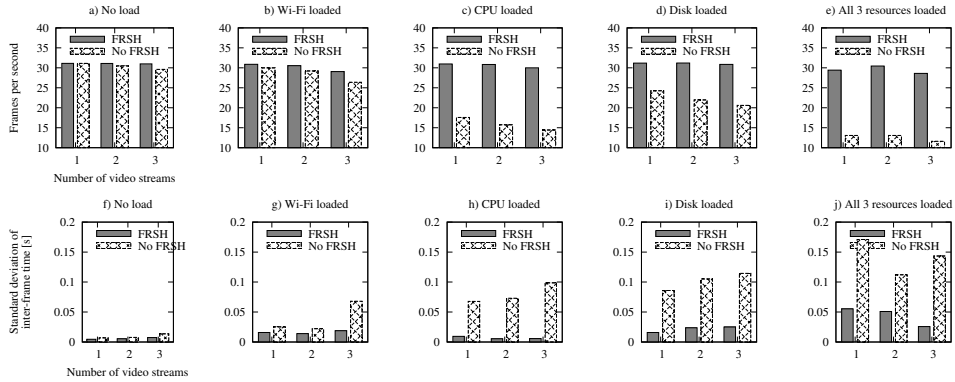

Figure 5.7: Results of the case study.

interval between the end of processing of two consecutive frames. The results can be seen on the graphs in Figure 5.7. Graphs a) and f) represent the case when all resources were loaded only by the applications of our case study. There are no significant differences in the measured frame rates, and the standard deviations show that the execution with FRSH is only slightly more regular than the one without FRSH. The reason why the measured frame rate is greater than 30 is that our cameras supplied approximately 31 frames per second even if we requested only 30 frames per second.

Graphs b) and g) show the metrics when the Wi-Fi network was loaded by a concurrently running communication. We connected two additional computers to the Wi-Fi network and let them interchange some data (all zeros) as fast as possible using the `netcat`[5] program. These communications were not under control of the FRSH framework (it can be considered and disturbances) and we setup two simultaneous streams running in opposite directions.

It can be seen that the load on the Wi-Fi channel influences the achieved frame rate. Clearly the impact increases with the number of transmitted streams but it is smaller when the FRSH framework is employed. The explanation of why the framework cannot guarantee a constant frame rate is that EDCA is not a deterministic medium access protocol and changing the EDCA access category can only increase the *probability* of faster medium access. On the other hand, one may wonder why the impact on the frame rate is not higher when running without FRSH. This can be explained by the netcat use of the *Transmission Control Protocol* (TCP) protocol, which automatically adapts its bandwidth according to the detected channel capacity. We tried to generate a more aggressive load (UDP floods) on the Wi-Fi link, but the camera controller started disconnecting from the network and the experiment could not be finished. We blame the used network adapter and/or its Linux driver for this problematic behavior.

Graphs c) and g) represent the case where the CPU on the video server was loaded by 20 additional CPU intensive non-FRSH applications. Here we can see

[5] `http://netcat.sourceforge.net/`

```
Time[s] Message
-------------------------------
 0.004: Waiting for requests
 0.111: Registering manager "AQuoSA" (0.0)
 0.115: Registering manager "AQuoSA" (0.1)
 0.121: Registering manager "AQuoSA" (0.2)
 0.125: Registering manager "WLAN" (1.3)
 5.219: Registering manager "Disk BFQ" (3.0)

 5.389: Negotiation request: NET.3 RTP
 5.391: Negotiation request: NET.3 RTCP
 5.396: Negotiation request: CPU.1 camera_ctrl
 5.402: Negotiation request: NET.3 RTP
 5.462: Negotiation request: NET.3 RTP
 5.463: Negotiation request: NET.3 RTCP
 5.465: Negotiation request: NET.3 RTCP
 5.468: Negotiation request: CPU.1 camera_ctrl
 5.469: Negotiation request: CPU.1 camera_ctrl

 9.259: Negotiation request: CPU.0 recorder
 9.261: Negotiation request: DISK.0 stream0.mp4
 9.565: Negotiation request: CPU.0 recorder
 9.606: Negotiation request: DISK.0 stream2.mp4
 9.622: Negotiation request: CPU.0 recorder
 9.663: Negotiation request: DISK.0 stream1.mp4

10.502: Negotiation request: CPU.2 client
10.519: Negotiation request: NET.3 RTP
10.521: Negotiation request: NET.3 RTCP
10.523: Negotiation request: CPU.0 client_streamer
10.559: Negotiation request: DISK.0 stream.mp4
13.931: Renegotiation request: CPU.0 client_streamer
13.933: Renegotiation request: NET.3 RTP
13.942: Contract(s) was/were rejected
17.235: Cancelation request: CPU.0 client_streamer
17.235: Cancelation request: DISK.0 stream.mp4
17.236: Cancelation request: NET.3 RTP
17.237: Cancelation request: NET.3 RTCP
17.240: Cancelation request: CPU.2 client

29.477: Cancelation request: CPU.0 recorder
29.477: Cancelation request: DISK.0 stream2.mp4
29.548: Cancelation request: CPU.0 recorder
29.548: Cancelation request: DISK.0 stream1.mp4
29.574: Cancelation request: CPU.0 recorder
29.575: Cancelation request: DISK.0 stream0.mp4
```

Figure 5.8: Log of the contract broker running in the video server.

that AQuoSA is highly successful in keeping the requested frame rate and regular execution (low variance of inter-frame times).

Similarly the disk scheduler (*Budget Fair Queuing* (BFQ)) achieves constant frame rate — see graphs d) and i)) — when the disk was loaded by two processes which read from two different places on the disk as fast as possible.

Finally, we ran all the three above mentioned loads simultaneously. The results are presented in graphs e) and j). The framework was able to keep the resources available for the applications in a way that no significant loss of quality was detected. The small decrease of quality can be attributed to the Wi-Fi network, which, in this case, constitutes the actual bottleneck. When the same experiment was run without the FRSH framework, the results are, as expected, very bad—only approximately 12 frames per seconds were successfully transported. Given the fact that in such a case it is very likely that the I-frames are lost, the recorded video is almost useless. With the FRSH framework, the recorded video is of good quality with only occasional small disturbances caused by dropped frames.

To highlight the dynamic nature of our framework, in Figure 5.8 we provide the timed log of important operations executed by the contract broker agent in the recording server, which has "connected" all resource managers needed for the case study. Shortly after the contract broker was started, five resource managers registered to it. According to Figure 5.4 there were three CPUs (CPU.0 – video server, CPU.1 – camera controller, and CPU.2 – video client), one disk and one Wi-Fi network. The disk resource manager probes for available disk throughput for five seconds after start and registers itself after the probe is finished. Then, at 5.38, three video steaming applications were started in the camera controller. Approximately four seconds later, three recording applications were started in the video server and they negotiated their CPU and disk contracts. A second later (10 seconds after start), the video client started on the 3rd computer to play back a formerly recorded stream. Initially, the stream was played back at low quality, but at time 13, the operator decided to increase the quality. The renegotiation happened while the old reservation was still in effect, so the video playback was not interrupted. Unfortunately, the Wi-Fi bandwidth was not available to satisfy that request so the quality remained the same until time 17 when the video client was terminated. Finally, approximately 25 seconds after the start, all the recorder applications were terminated and their reservations were canceled.

6

Integer Programming-Based Approach to Schedulability Analysis for Tasks with Offsets

Embedded systems are often characterized by the existence of various constraints which must be respected during the design of the system. For example their computation power is low, size of the memory is limited, systems are battery powered, etc. Significant number of embedded systems are also real-time systems, which means that their behavior is constrained in time. For checking whether the timing constraints are satisfied, there exist many schedulability tests, one of them being the rate-monotonic analysis presented in Section 2.1.2. The other constraints, such as memory requirements, can also be checked after the system is designed, but it is better to consider the constraints already during the design phase. Moreover, the designed system is often required to be optimal in some sense, e.g. we want to minimize its price or energy consumption. These requirements lead to the fact that various optimization techniques are used during the design phase of embedded real-time systems [Baruah and Fisher, 2005]. It is natural that we want the optimization technique to respect all constraints of the system – either time related or not.

The optimization techniques (also called *Mathematical Programming* (MP)) require the system to be represented with parameters, decision variables, and constraints over the parameters and decision variables. The goal of optimization is described with an objective function, which is defined over the same set of variables. Generic solvers can be utilized to nd the optimal solution [Davare et al., 2007].

One widely used optimization technique is *Linear Programming* (LP). This technique can be used when both constraints and objective function are linear expressions. Linear programming is a polynomial problem. Some problems, however, cannot be solved by linear programming even if their constraints and are linear expressions but some (or all) decision variables are restricted to have integer values.

Such problems are known as *Integer Linear Programming* (ILP) problems and are NP-hard.

This chapter describes an attempt to formulate the problem of response-time analysis for tasks with offsets as an ILP problem. The advantage of using MP for schedulability analysis is that the problem can by customized by system-specific issues by simply adding additional constraints [Davare et al., 2007]. The initial idea was to combine the design optimization process and schedulability analysis into one step, similarly as described e.g. in [Zheng et al., 2007].

In the context of contract-based resource reservation framework presented in Chapter 3, the goal is to optimize the distribution of spare capacity as described in Section 3.4. It would be nice to have a fast and efficient optimization technique, which can simultaneously take into an account the optimization goal and the schedulability of the system. The work in this chapter is the first step in this direction.

Schedulability analysis for task with offsets is a generic name for response-time analysis techniques, which take into account task offsets. The offsets brings additional information to the analysis process and allows it to give less pessimistic results. Moreover, these techniques are capable of analyzing tasks with self-suspensions and, to some extent, distributed systems. The concept of such an analysis was introduced in [Tindell and Clark, 1994] under the name "*Holistic schedulability analysis*". This technique was later generalized and formalized in [Palencia and González Harbour, 1998] and is commonly called *offset-based response-time analysis*. The authors derive an exact algorithm for solving the NP-hard [Ridouard et al., 2004] problem as well as polynomial-time algorithm for upper-bound approximate analysis. The approximate analysis was later improved in [Mäki-Turja and Nolin, 2008]. The exact algorithm has exponential complexity and as such it is not applicable to industrial-size problems.

The goal of this chapter was to formulate the schedulability analysis for tasks with offset as an ILP problem and compare the time needed to solve the problem by the solver with the time needed by the original exact algorithm. The expectation was the the branch-and-bound algorithm inside ILP solvers could solve the problem faster than the original algorithm, which performs exhaustive search. It turned out, that the size of our ILP formulation is generally the same as the number of steps in the original algorithm and therefore even generation of the ILP program has big complexity. Despite of that we present our results as it may serve as a basis for future research.

The outline of this chapter is as follows: For the sake of completeness sections 6.1 and 6.2 cite [Palencia and González Harbour, 1998] to give an overview of the original exact analysis algorithm and to introduce notation and expressions which are referred from the later sections. The cited text was extended with several figures with the aim of making the problem easier to understand. Section 6.2.1 summarizes the original exact algorithm and Section 6.2.2 shows how can be the original algorithm applied to the analysis of distributed and multi-processor systems. Then, in Section 6.2.3, we show how such algorithm could be applied to the resource reservation framework described earlier in this thesis. The ILP formulation is derived in Section 6.3, where we simplified the computational model to tasks with deadlines shorter than

periods. Finally we present experimental results in Section 6.4 and give conclusions in Section 6.5.

6.1 Computational Model

The real-time system considered for analysis is composed of tasks executing in the same processor (the extension for distributed systems is provided in Section 6.2.2), which are grouped to *transactions*. Each transaction Γ_i is activated by a periodic sequence of external events with period T_i and contains a set of m_i tasks. The relative phasing between the different external events is arbitrary. Each task is activated (released) when a relative time—called the *offset*—elapses after the arrival of the external event. We can assume this offset to be static, i.e., it does not change from one activation to the next. This restriction can be eliminated as is shown [Palencia and González Harbour, 1998]. Each activation of a task releases the execution of one instance of that task, which is called a *job*.

It is assumed that each task has its unique priority and that the task set is scheduled using a preemptive fixed priority scheduler. Notice that although offsets represent a kind of precedence constraints, in offset-based analysis tasks are activated at a time equal to the arrival of the external event plus the offset, and they execute at their assigned priority regardless of whether tasks of the same transaction and smaller offsets have finished or not.

Each task will be identified with two subscripts: the first one identifies the transaction to which it belongs and the second one the position that the task occupies within the tasks in its transactions, when they are ordered by increasing offsets. In this way, τ_{ij} will be the j-th task of transaction Γ_i. With an offset Φ_{ij} and worst-case execution time of C_{ij}. In addition, each task is allowed to have its activation time delayed by an arbitrary amount of time between 0 and the maximum jitter for that task which is called J_{ij}. This means that the activation time of task τ_{ij} may occur at any time between $t_0 + \Phi_{ij}$ and $t_0 + \Phi_{ij} + J_{ij}$, where t_0 is the instant at which the external event arrived.

Figure 6.1 shows an example of such system. The horizontal axis represents time. Down-pointing arrows represent periodic external events, gray boxes represent task execution. Up-pointing arrows represent task activation times and dashed lines under each transaction axis represent task jitter values.

As deadlines are allowed to be larger than one period, at each time there may be several activations of the same task pending. Both the offset Φ_{ij} and the jitter J_{ij} are allowed to be larger than the period of its transaction T_i. The response time of each task τ_{ij} is defined as the difference between its completion time and the instant at which the associated external event arrived. The worst-case response time will be called R_{ij}. Each task may have associated global deadline D_{ij}, which is also relative to the arrival of the external event.

It is assumed that if tasks synchronize for using shared resources in a mutually exclusive way they will be using a hard real-time synchronization protocol such as the priority ceiling protocol (see section 2.1.1). Under this assumption, the effects of lower priority tasks on a task under analysis τ_{ab} are bounded by an amount called

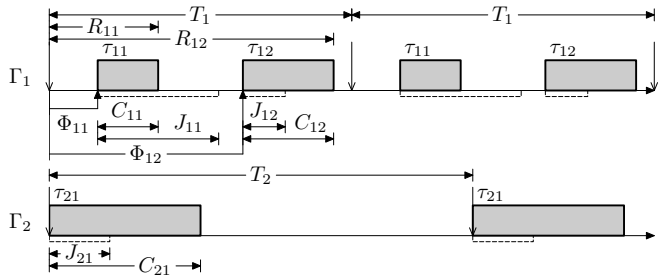

Figure 6.1: Computational model of a system composed of transactions with static offsets

the blocking term B_{ab} calculated as maximum of all the critical sections of lower priority tasks that have a priority ceiling higher than or equal to the priority of τ_{ab}.

6.2 Original Exact Response-Time Analysis

In this section, the algorithm which computes exact values of task's response times will be described. As this is NP-hard problem [Ridouard et al., 2004], the complexity of this algorithm is exponential with respect to the number of tasks in the system.

The rest of this section deals with the analysis of response time of one task τ_{ab}. To analyze the whole system, it is necessary to execute the described algorithm for each task in the system.

To find the worst-case response time of a task τ_{ab} under analysis, it is necessary to build the worst-case scenario for this task. Finding this scenario rests in finding such a combination of higher priority tasks having the highest contribution to τ_{ab} response time. The time when this combination occurs is called the critical instant. Recall that in the case where all tasks are independent and deadlines are less or equal to periods, it is the time when all the tasks with the higher priority are activated simultaneously with τ_{ab}. This no longer holds for tasks with offsets, as it might be impossible for some sets of task to be activated at the same time. The conditions under which task τ_{ij} has the worst-case contribution to the response time of the task under analysis, τ_{ab}, are formulated in two theorems later in this section.

When the response time of a particular task is analyzed, the offset of a higher priority task may be changed by adding or subtracting whole periods of that later task, without any effect on the response time of the lower priority task, since one instance of a task is indistinguishable from another instance. Therefore, in order to simplify the analysis, a *reduced task offset*, ϕ_{ij}, is considered and it's value is always within 0 and T_i.

$$\phi_{ij} = \Phi_{ij} \bmod T_i \tag{6.1}$$

In order to calculate the worst-case contribution of task τ_{ij} to the response time of lower priority tasks, each job (activation) of task τ_{ij} must be categorized into one of the following sets:

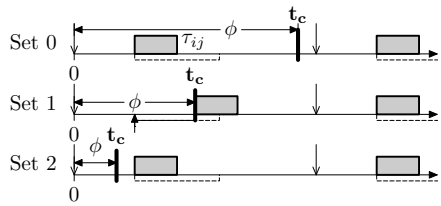

Figure 6.2: Contribution of a task τ_{ij} to the response time of lower priority task τ_{ab} (not depicted), whose critical instant occurs at t_c

Set 0: Activations that occur before the critical instant and that cannot occur inside the busy period even with the maximum jitter delay.

Set 1: Activations that occur before or at the critical instant and that can be delayed by an amount of jitter that causes them to coincide with the critical instant.

Set 2: Activations that occur after the critical instant.

This categorization can be accomplished only if the phase relation between the task activation pattern and the critical instant is known. As this phase relation is not known now, we will mark it as ϕ and later it will be shown how to compute it based on Theorem 3.

Phase relation between the transaction arrival and the critical instant, ϕ, is the time interval between activation of transaction Γ_i that occurred immediately before or at critical instant and that critical instant.

Notice that $0 \leq \phi < T_i$. Examples of tasks from each set as well as the values of ϕ are shown in Fig. 6.2. The task from Set 1 is depicted as being delayed to the critical instant t_c.

Theorem 2 (from [Palencia and González Harbour, 1998]) *Given a task τ_{ab} critical instant, t_c, and a phase relation ϕ between the arrival pattern of transaction Γ_i and the critical instant, the worst case contribution of task τ_{ij} to the response time of τ_{ab} occurs when the activations in Set 1 have an amount of jitter such that they all occur exactly at the critical instant and when the activations in Set 2 have amount of jitter equal to zero.* □

The original paper contains the full proof. Here, in the following two paragraphs, only a proof sketch is provided.

Figure 6.3 shows possible scenarios for calculating the contribution of task τ_{ij} to the response time of lower priority tasks. On the top level axis a) there is depicted a job of task τ_{ij}, its offset ϕ_{ij} and its jitter (dashed line). Axis b) shows the scenario where the critical instant occurs after where the third activation of the task would occur if it had no jitter. According to Theorem 2, the worst-case contribution of task τ_{ij} happens when all the tasks that could be activated before critical instant are delayed to the critical instant t_c if that is possible. On the axis b), jobs belonging to Set 1 are τ_{ij}^{-2} through τ_{ij}^{0} so they produce the wost-case contribution if their

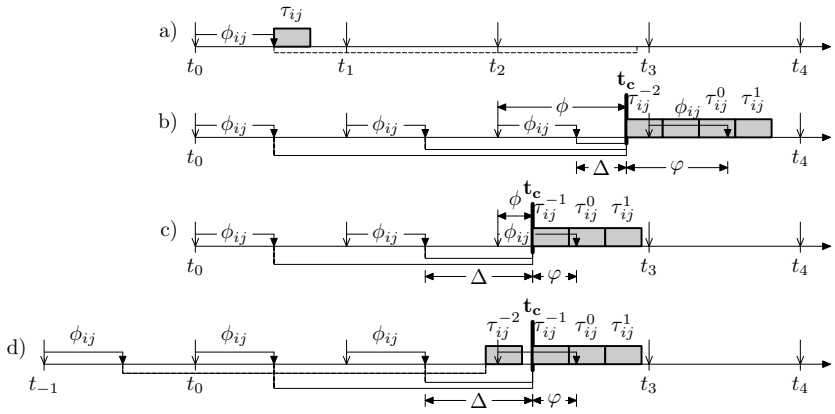

Figure 6.3: Scenarios for calculating the contribution of task τ_{ij} to the response time of lower priority tasks.

activations are delayed as it is shown in the figure by the solid lines under the axis. There is also job τ_{ij}^1, which is activated at $t_3 + \phi_{ij}$, which is after the critical instant. Therefore this job is categorized to Set 2 and according to theorem 2, this activation has to occur without any jitter. If it has jitter greater than zero the job execution might fall after the end of the busy period and the contribution of τ_{ij} would not be the worst.

Axis c) shows another scenario in which the critical instant occurs between the activation of the transaction at time t_2 and the activation of the job τ_{ij}^1. Here, there are two jobs τ_{ij}^{-1} and τ_{ij}^0 in Set 1 and job τ_{ij}^1 in Set 2. Axis d) shows the same scenario but adds job τ_{ij}^{-2}, which belongs to Set 0 because its execution cannot interfere with the examined busy period even when the task is released at the latest possible time (as shown in the Figure). Note that if the execution of τ_{ij}^{-2} could have interfered with the busy period, the critical instant would had occurred at the time of activation of this job.

Based on Theorem 2, the number of activations in each set can be calculated. The activations from Set 1 will accumulate at critical instant and the number of these activations will be called n_{ij}. Axis b) in Figure 6.3 has $n_{ij} = 3$ whereas axes c) and d) have $n_{ij} = 2$.

To calculate n_{ij}, auxiliary symbol Δ has to be defined as the difference in time between the time at which last activation in Set 1 would occur if it had no jitter delay, and the critical instant. In the example in Figure 6.3, $\Delta = t_c - t_2 + \phi_{ij}$ for axis b) and $\Delta = t_c - t_1 + \phi_{ij}$ for axis c).

It can be seen that:

$$\Delta(\phi) = \begin{cases} \phi - \phi_{ij} & \text{if } \phi \geq \phi_{ij} \\ T_i + \phi - \phi_{ij} & \text{if } \phi < \phi_{ij} \end{cases} \tag{6.2}$$

or equivalently:

$$\Delta(\phi) = (\phi - \phi_{ij}) \bmod T_i \tag{6.3}$$

Please note that in this and all the following equations the result of modulo operation is always greater than or equal to zero. Usually modulo operation is defined such that its result is negative if the first operand is negative.

The first activation of τ_{ij} in Set 1 corresponds to the event arriving at t_0, which is the first one whose activation may occur at or after the critical instant. Therefore, this is the first activation that simultaneously verifies:

$$t_0 + \phi_{ij} + J_{ij} \geq t_c \tag{6.4}$$

and:

$$t_0 - T_i + \phi_{ij} + J_{ij} < t_c \tag{6.5}$$

By looking at Figure 6.3 it can be seen that:

$$t_c = t_0 + (n_{ij} - 1)T_i + \phi_{ij} + \Delta(\phi) \tag{6.6}$$

and replacing it in the two previous equations gives:

$$t_0 + \phi_{ij} + J_{ij} \geq t_0 + (n_{ij} - 1)T_i + \phi_{ij} + \Delta(\phi) \tag{6.7}$$

$$t_0 - T_i + \phi_{ij} + J_{ij} < t_0 + (n_{ij} - 1)T_i + \phi_{ij} + \Delta(\phi) \tag{6.8}$$

from which is derived:

$$n_{ij} - 1 \leq \frac{J_{ij} - \Delta(\phi)}{T_i} \quad \text{and} \quad n_{ij} - 1 > \frac{J_{ij} - \Delta(\phi)}{T_i} - 1 \tag{6.9}$$

Given that n_{ij} is an integer number, the solution to the above expressions is:

$$n_{ij}(\phi) = \left\lfloor \frac{J_{ij} - \Delta(\phi)}{T_i} \right\rfloor + 1, \tag{6.10}$$

where half square brackets represent the floor operation.

In order to determine the effect of activations belonging to Set 2, the time at which the first of them occurs has to be known; the others will occur at periodic intervals after the initial one. Let's call the time difference between the critical instant and that first activation in Set 2 as φ. Given the definition of Δ we have:

$$\varphi(\phi) = T_i - \Delta(\phi) = T_i - \Big((\phi - \phi_{ij}) \bmod T_i\Big) \tag{6.11}$$

Substituting Δ with φ in equation (6.10) we get:

$$n_{ij}(\phi) = \left\lfloor \frac{J_{ij} + \varphi(\phi)}{T_i} \right\rfloor \tag{6.12}$$

According to Theorem 2, the worst-case contribution of τ_{ij} to the busy period of a lower priority task is equivalent to n_{ij} activation at the critical instant, plus a sequence of periodic activation starting at φ time units after the critical instant.

Without loss of generality, let's set the origin of time at the critical instant. Then, the number of activations in Set 2 until time t is

$$n_{ij}^{S2}(\phi, t) = \left\lceil \frac{t - \varphi(\phi)}{T_i} \right\rceil \tag{6.13}$$

and the worst-case contribution a task τ_{ij} to the response time of τ_{ab} at time t is determined by:

$$\begin{aligned} W(\tau_{ij}, \phi, t) &= n_{ij}(\phi) C_{ij} + n_{ij}^{S2}(\phi, t) C_{ij} = \\ &= \left(\left\lfloor \frac{J_{ij} + \varphi(\phi)}{T_i} \right\rfloor + \left\lceil \frac{t - \varphi(\phi)}{T_i} \right\rceil \right) C_{ij} \end{aligned} \tag{6.14}$$

The total interference of the tasks of transaction Γ_i on the execution of τ_{ab} is obtained by taking into account the contributions of all higher priority tasks:

$$W(\Gamma_i, \phi, t) = \sum_{\forall j \in \mathrm{hp}_i(\tau_{ab})} W(\tau_{ij}, \phi, t), \tag{6.15}$$

where hp_i is defined as a set of tasks belonging to transaction Γ_i with the priority greater to the priority of τ_{ab}.

Now, it must be determined how to calculate ϕ, the phase between the arrival pattern of Γ_i and the critical instant. The calculation is based on the following theorem:

Theorem 3 (from [Palencia and González Harbour, 1998]) *The worst-case contribution of transaction Γ_i to a task τ_{ab} critical instant is obtained when the first activation of some task τ_{ik} in $\mathrm{hp}_i(\tau_{ab})$ that occurs within the busy period coincides with the critical instant, after having experienced the maximum possible delay, i.e., the maximum jitter, J_{ij}.* □

Proof By definition of the busy period (Definition 1 at page 11), right before the critical instant there are no pending tasks of priority higher than the priority of τ_{ab}. Now suppose that we choose a critical instant that does not coincide with the activation of some task in $\mathrm{hp}_i(\tau_{ab})$ (see task τ_{ik} in Fig. 6.4 a). Let us focus on the first activation of a task belonging to $\mathrm{hp}_i(\tau_{ab})$ that occurs within busy period, τ_{ik}. If we cause the arrival of the events of Γ_i to occur earlier while keeping the same activation pattern for all its tasks, until task τ_{ik} coincides with the critical instant (Fig. 6.4 b) all the jobs of tasks belonging to $\mathrm{hp}_i(\tau_{ab})$ that were in the busy period continue to be in that same busy period, but we have brought more jobs of those tasks, and perhaps other additional tasks, closer to the busy period, thus increasing the chance of additional interference on task τ_{ab}. Thus by making the first job of τ_{ik} coincide with the critical instant we can only make the contribution worse.

Now it is necessary to check that the worst-case contribution of transaction Γ_i is obtained when a job of a task τ_{ik} that initiates the busy period has experienced the worst-case delay, equal to J_{ik}. Figure 6.5 contains a helpful example to the following explanation. There are depicted jobs of τ_{ik} and their actual jitter values j_{ik}. Upper indices of jobs and their associated parameters are numbered by the number of period they come from, i.e. τ_{ik}^0 corresponds to the period starting at t_0.

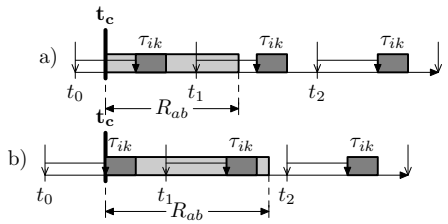

Figure 6.4: Calculation of critical instant phase – part 1. The lighter box represents a lower priority task τ_{ab} from another transaction than Γ_i.

Let us call I the set of such jobs of tasks belonging to $\mathrm{hp}_i(\tau_{ab})$ that initiate busy period, and let us suppose that each of these jobs has a jitter value j_{ik} less than the maximum for its associated task, J_{ik}. On axis A) $I = \{\tau_{ik}^0, \tau_{ik}^1\}$. Now let us move back (i.e., earlier in time) the event arrivals of transaction Γ_i, and simultaneously, increase the jitter delay of all the jobs in I by the same amount of time, so that all these jobs continue to be activated at the same time as before; jitter delays for all other jobs not being in I remain unchanged (and thus they are activated earlier). Under these conditions we will move back the event arrivals until we reach the point when either: a) one of the jobs in I reaches its maximum jitter; or b) when a job in the busy period that did not belong to I gets aligned with the critical instant (because it is activated earlier). In case b) (see axis B), we insert the new job (τ_{ik}^2) to set I and continue the process of moving back the event arrivals of Γ_i in an iterative manner, until we reach condition a), under which one or more of the activations that start the busy period have experienced their maximum jitter. This is what axis C) shows: the job τ_{ik}^0 reaches its maximum jitter.

Notice that during this process, none of the activations that belonged to the busy period has been moved to a point before critical instant, and thus all the jobs that belonged to the busy period remain in it. However, because the event arrivals of Γ_i occur earlier, it is possible that jobs which previously occurred after the end of busy period (e.g. τ_{ik}^3 on axis A) are now activated inside the busy period (axis B), thus making it longer and increasing the response times for the task under analysis, τ_{ab}. Therefore the theorem follows. ■

By applying Theorem 3, and supposing that we know that task τ_{ik} is one that originates the busy period, we can determine the phase between the event arrivals and the critical instant:

$$\phi = (\phi_{ik} + J_{ik}) \bmod T_i \tag{6.16}$$

Substituting this expression in equation (6.11) we obtain the phase φ_{ijk} between any task τ_{ij} and the critical instant created by τ_{ik}:

$$\begin{aligned} \varphi_{ijk} &= \varphi(\phi)|_{\phi=(\phi_{ik}+J_{ik}) \bmod T_i} = \\ &= T_i - \Big(\big((\phi_{ik} + J_{ik}) \bmod T_i - \phi_{ij} \big) \bmod T_i \Big) \end{aligned} \tag{6.17}$$

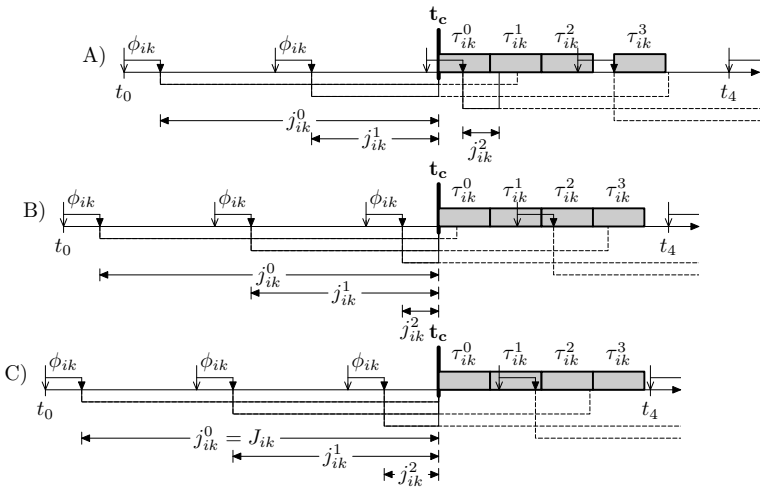

Figure 6.5: Calculation of critical instant phase – part 2

and applying the properties of the modulus function,

$$\varphi_{ijk} = T_i - \big((\phi_{ik} + J_{ik} - \phi_{ij}) \bmod T_i\big) \tag{6.18}$$

Using this value, we can now obtain the expression of the worst-case contribution of transaction Γ_i when the critical instant is created with τ_{ik}. This function will be called $W_{ik}(\tau_{ab}, t)$, and is obtained by replacing (6.18) in equations (6.14) and (6.15).

$$\begin{aligned} W_{ik}(\tau_{ab}, t) &= W(\Gamma_i, \phi, t)|_{\phi=(\phi_{ik}+J_{ik}) \bmod T_i} = \\ &= \sum_{\forall j \in \mathrm{hp}_i(\tau_{ab})} \Bigg(\underbrace{\left\lfloor \frac{J_{ij} + \varphi_{ijk}}{T_i} \right\rfloor}_{n_{ijk}} + \underbrace{\left\lceil \frac{t - \varphi_{ijk}}{T_i} \right\rceil}_{n_{ijk}^{S2}} \Bigg) C_{ij} \end{aligned} \tag{6.19}$$

In order to obtain the worst-case response time of task τ_{ab} the above equation needs to be applied for all transaction in the system. The main problem now is that for each transaction Γ_i we need to find the task τ_{ik} with which the critical instant will be created. In order to perform the exact analysis, it is necessary to check all possible variations of one task out of every transaction and choose the variation that leads to the worst case response time for the task under analysis.

The number of variations, and thus of different critical instant possibilities that need to be checked, is determined by the number of tasks of priority higher than that of the task under analysis that exist in each transaction in the system. We also have to take into account that the task under analysis itself may originate the critical instant for its transaction. Thus the total number of variations is:

$$\begin{aligned} N_v(\tau_{ab}) &= (N_a(\tau_{ab}) + 1) \cdot N_1(\tau_{ab}) \cdots = \\ &= (N_a(\tau_{ab}) + 1) \cdot \prod_{\forall i \neq a} N_i(\tau_{ab}) \end{aligned} \tag{6.20}$$

where $N_i(\tau_{ab})$ is the number of tasks belonging to $\mathrm{hp}_i(\tau_{ab})$. Each of $N_v(\tau_{ab})$ variations is characterized by a tuple v indexes, one for each transaction. Each index v_i identifies the task of transaction Γ_i that initiates the critical instant, i.e. $v \in \mathcal{V}$, $\mathcal{V} = \{(v_1, v_2, \ldots) : v_i \in \mathrm{he}_i(\tau_{ab})\}$, where $\mathrm{he}_i(\tau_{ab}) = \begin{cases} \mathrm{hp}_i(\tau_{ab}) \cup \{\tau_{ab}\} & \text{if } i = a \\ \mathrm{hp}_i(\tau_{ab}) & \text{otherwise.} \end{cases}$

For convenience, the jobs of the task under analysis will be numbered using letter p, with consecutive numbers ordered according to the activation time that they would have had if they had no jitter. In addition, the value of $p = 1$ will be assigned to the activation of τ_{ab} that occurs in the interval $(0, T_a]$. This means that the activation that occurred in $(T_a, 2T_a]$ gets the value $p = 2$, etc. Similarly, the activation that would have occurred in the interval $(-T_a, 0]$ but that was delayed to the critical instant corresponds to $p = 0$, the one in $(-2T_a, -T_a)$ to $p = -1$, etc. Notice that activations that occurred after the critical instant are numbered with positive numbers while previous activations have $p \leq 0$. The jobs in Figure 6.3 are numbered according to this numbering scheme whereas the jobs in Figure 6.5 are not.

For each variation v the completion time of each of the jobs of τ_{ab} in the busy period will be obtained. This time $w^v_{ab}(p)$ is obtained by considering the execution of τ_{ab} together with the interference from all other tasks in the system:

$$w^v_{ab}(p) = B_{ab} + (p - p^v_{0,ab} + 1)C_{ab} + \sum_{\forall i} W_{iv_i}\left(\tau_{ab}, w^v_{ab}(p)\right) \tag{6.21}$$

where $p^v_{0,ab}$ corresponds to the lowest-numbered job in the busy period, and is equal to:

$$p^v_{0,ab} = -\left\lfloor \frac{J_{ab} + \varphi_{abv_a}}{T_a} \right\rfloor + 1 \tag{6.22}$$

The solution to equation (6.21) is obtained as in the normal rate monotonic equation (2.8) by starting from a value of $w^v_{ab}(p) = 0$, and iterating until two consecutive iterations produce the same value. This analysis has to be repeated for all the jobs present in the busy period. The length of the busy period, which will be called L^v_{ab}, may be obtained with the following equation:

$$L^v_{ab} = B_{ab} + \underbrace{\left(\left\lceil \frac{L^v_{ab} - \varphi_{abv_a}}{T_a} \right\rceil - p^v_{0,ab} + 1\right)}_{n'_{ab}} C_{ab} + \underbrace{\sum_{\forall i} W_{iv_i}\left(\tau_{ab}, L^v_{ab}\right)}_{\text{interference}} \tag{6.23}$$

where the content of the first parentheses (n'_{ab}) represents the number of jobs of task τ_{ab} participating in the busy period and the sum represents interference from all higher priority tasks. L^v_{ab} represents the first instant after the critical instant at which all jobs of τ_{ab} and of all higher priority tasks have been completed.

With the length of busy period, the maximum value of p that needs to be checked can be calculated:

$$p^v_{L,ab} = \left\lceil \frac{L^v_{ab} - \varphi_{abv_a}}{T_a} \right\rceil \tag{6.24}$$

The global response time is obtained by subtracting from the obtained completion time the instant at which the external event that activated the transaction arrived. According to our numbering scheme, the first activation of τ_{ab} after the critical instant corresponds to the value $p = 1$ and, by definition it corresponds to instant φ_{abv_a}. Consequently the p-th activation occurs at $\varphi_{abv_a} + (p-1)T_a$. Since the task is activated Φ_{ab} time units after the event arrival, the event arrival for each job p occurs at time $t(p) = \varphi_{abv_a} + (p-1)T_a - \Phi_{ab}$. Therefore the global worst-case response time for job p is:

$$\begin{aligned} R^v_{ab}(p) &= w^v_{ab}(p) - t(p) = \\ &= w^v_{ab}(p) - \varphi_{abv_a} - (p-1)T_a + \Phi_{ab} \end{aligned} \tag{6.25}$$

Notice that in the above equation is used the real offset Φ_{ab} instead of the reduced offset ϕ_{ab}, which was used when calculating interference of higher priority tasks on task under analysis. To calculate the global worst-case response time for task τ_{ab} the maximum for among all potential critical instant must be determined:

$$R_{ab} = \max_{v \in \mathcal{V}} \left(\max_{p=p^v_{0,ab}}^{p^v_{L,ab}} \left(R^v_{ab}(p) \right) \right) \tag{6.26}$$

By applying the described analysis to each task in the system, the global worst-case response times can be obtained and, by comparing them with deadlines, it can be determined whether the system meets its timing requirements. However, although the analysis the analysis technique is exact, it represents an NP-hard algorithm in which the number of cases to check grows exponentially with the number of tasks.

6.2.1 Summary

As the above described algorithm is quite complex for the first-time reader this section provides a short summary of how to use this algorithm to calculate the worst-case response time of one particular task τ_{ab}. In order to compute response times of all tasks the algorithm must be repeated for all the tasks.

1. For each variation vector $v \in \mathcal{V}$, i.e. for all possible combinations of tasks from each transaction that initiate τ_{ab} busy period, do:
   (a) Calculate $p^v_{0,ab}$ according to (6.22).
   (b) Calculate L^v_{ab} by iteratively solving (6.23).
   (c) Calculate $p^v_{L,ab}$ according to (6.24).
   (d) For each p satisfying $p^v_{0,ab} \leq p \leq p^v_{L,ab}$ do:
      i. Solve equation (6.21) iteratively to obtain $w^v_{ab}(p)$. This computation will use (6.19) to compute the worst-case interference from transaction Γ_i.
      ii. Calculate $R^v_{ab}(p)$ according to (6.25)
   (e) Store the maximum value of $R^v_{ab}(p)$ to $R^v_{ab}*$
2. R_{ab}, the worst-case response time of task τ_{ab}, is equal to the maximum of all $R^v_{ab}*$.

6.2.2 Analysis of Multiprocessor and Distributed Systems

In multiprocessor or distributed systems it is usual that the system can be modelled with "transactions" composed of several tasks, like in the computational model described in Section 6.1. For example in the distributed system where a task on the first node produces some data and then sends them to the second node in which these data are processed can be modelled as transaction of three tasks. The first task represents the production of the data on the first node, the second task is the message transmitted on the bus and the last task is the data processing on the second node. If a real-time communication bus based on fixed priorities, such as CAN bus, is used it can be directly modelled as another processor, accounting the non-preemptability of message packets as additional blocking time.

When calculating the worst-case response time of a task on one processor, evidently it cannot be preempted by a task on another processor. Hence the definition of $\mathrm{hp}_i(\tau_{ab})$ must be refined to contain only tasks that belong to the same processor as τ_{ab}:

$$\begin{aligned}\mathrm{hp}_i(\tau_{ab}) \stackrel{\mathrm{def}}{=} \{j \in \Gamma_i : \mathrm{priority}(\tau_{ij}) > \mathrm{priority}(\tau_{ab}) \wedge \\ \mathrm{processor}(\tau_{ij}) = \mathrm{processor}(\tau_{ab})\}\end{aligned} \tag{6.27}$$

In multiprocessor and distributed systems, usually only the first task in the transaction has known offset and jitter. Offset is zero and jitter is the same as the jitter of the external event – often zero as in the case of hardware timer. Offsets and jitters of subsequent tasks in the transaction depend on response times of the preceding tasks. As an example, consider a CPU task that is activated by reception of a message from network. Its activation happens somewhere between the best-case and worst-case response time of the message. Since the worst-case response time of the preceding task in not known until the response-time analysis is computed, it is not possible to set the exact offset and jitter of the task. This problem can be solved by running the response-time analysis in iterations. For the first iteration, tasks offsets are set to the lower bound on the best-case response-times of preceding tasks and jitters are set to zero. Then, worst-case response times are calculated and jitters are increased according to the just computed response times. This is repeated until we get the same results in the two subsequent iterations.

By increasing the task jitter in the above iterative process, the effect of the task on the response-time of lower priority tasks worser and therefore the calculated response times cannot decrease from iteration to iteration.

For example, suppose that computation times C_{ij} are exact computation times, i.e. the execution of task τ_{ij} always takes C_{ij} units. Offset and jitters for the first iteration are set as

$$\begin{aligned}&\Phi_{i1} = 0, \\ &\Phi_{ij} = \Phi_{ij-1} + C_{ij-1}, \qquad j = 2, \ldots, m_i \\ &J_{ij} = 0, \qquad j = 1, \ldots, m_i\end{aligned} \tag{6.28}$$

After each iteration, jitters are updated according to:

$$J_{ij} = R_{ij-1} - \phi_{ij}, \qquad j = 2, \ldots, m_i. \tag{6.29}$$

Section 3.4	Section 6.1	Note
Γ_i^v	Γ_i	v corresponds to a transaction variant selected by the spare capacity distribution algorithm.
c_{ij}^v	τ_{ij}	Task τ_{ij} is executed by the VRES resulting from negotiation of c_{ij}^v.
$T(\Gamma_i^v)$	T_i	
$C(c_{ij}^v)$	C_{ij}	
$D(c_{ij}^v)$	D_{ij}	
$D(\Gamma_i)$	D_{im_i}	
	Φ_{ij}, J_{ij}	Calculated according to (6.28) and (6.29).

Table 6.1: Notation mapping

6.2.3 Applicability to the Resource Reservation Framework

This section shows, how could be the analysis for multiprocessor and distributed systems applied to the system represented by resource reservation framework from Chapter 3. This can only be done when all involved resources are scheduled by fixed-priority schedulers, which is for example when a distributed system uses *Controller Area Network* (CAN) [CiA, 2001] for communication between nodes. CAN bus uses a medium access protocol, which schedules messages strictly by their priority and non-preemptability of the message transmission can be modelled as additional blocking time.

Due to high complexity of the exact analysis, its use as on-line admission test is limited to small systems only. However, for certain class of systems [Traore et al., 2006] derive an exact analysis with pseudo-polynomial complexity. Additionally [Mäki-Turja and Nolin, 2008] present pseudo-polynomial approximate analysis of generic systems. Mapping of the resource reservation framework model to the model expected by these faster algorithms is analogous to what follows in this section.

Since the analysis involves all resources in the system, it should be implemented in the resource independent level i.e. in the contract broker. Table 6.1 shows the mapping between symbols used in this chapter and symbols introduced in Section 3.4 to describe the applications running under the FRSH/FORB framework. Basically, the information needed for the analysis is contained in the contract so with this mapping, the implementation of the analysis is straightforward.

6.3 ILP Formulation

As was shown in Section 6.2.2, response-time analysis of multiprocessor and distributed systems involves several iterations of the response-time calculation since the response times are dependent on task jitters and task jitters are dependent again on response times. In this section we formulate the schedulability analysis for tasks with offsets as an ILP problem, where both response-times and jitters are variables of a system of inequalities and are therefore calculated at once by an ILP solver.

In [Sojka, 2006] we tried to formulate the ILP problem directly as equations

from Section 6.2 with the aim of finding the variation vector v, which leads to the worst-case response-time, by the ILP solver, rather than iterating over all possible variants. That approach did not give correct results because we were searching for maximum response-time and equation (6.21) resp. (6.23) can have multiple solutions and only the smallest one represents the correct completion time resp. the length of the busy period. Traditionally, that equations are solved by fixed point iteration, where the iteration starts from zero and the value increases until a fixed point is found. Such fixed point solution is the smallest solution of the equation. The ILP solver, however, can find any solution, not only the smallest one.

More formally, we have been finding the maximal response time of task τ_{ab} as a maximum given by expression (6.26). The response time R_{ab} can be also written as a function of its parameters over which we perform the maximization:

$$\text{maximize } R_{ab}\left(v, p, L'_{ab}, w'_{ab}\right), \tag{6.30}$$

where $L'_{ab} = \min\left\{L_{ab}(v,p)\right\}$ and $w'_{ab} = \min\left\{w_{ab}(v,p)\right\}$.

The problem is that parameters L'_{ab} and w'_{ab} are not independent variables but instead they are another functions of variables v and p and involve non-linear minimum operator. For that reason (6.30) cannot be used as an objective in ILP formulation.

In the following, another formulation is derived, which does not suffer from the above mentioned problem. In Section 6.3.1 we recall common approaches to formulating response-time analysis for tasks without offsets as an ILP problem. In Section 6.3.2 we return back to tasks with offsets and restricted the computational model introduced above to deadlines to be less than or equal to transaction periods. Then, Section 6.3.3 presents schedulability conditions with integer variables for schedulability of a single task and finally, in Section 6.3.4, we derive conditions for schedulability of the whole system where jitters depends on response-times and vice versa.

6.3.1 ILP Approaches to Schedulability Analysis

This section recalls two basic approaches to formulation of schedulability analysis for fixed priority tasks as ILP problem. In this section, we consider a system with N independent tasks (without offsets) with deadlines less or equal to the task periods. Without the loss of generality we assume that task indices are ordered according to decreasing task priority, i.e. the set of indices of tasks having higher priority than i-th task is $\mathrm{hp}(i) = \{1, \ldots, i-1\}$.

Response Time-Based Formulations

The response time R_i of i-th task can be calculated as a solution to equation (2.7), which can be then written as

$$R_i = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i}{T_j} \right\rceil C_j. \tag{6.31}$$

This equation can be formulated as an ILP problem and solved by an ILP solver:

$$\text{minimize} \quad R_i \tag{6.32}$$

$$\text{subject to} \quad R_i = C_i + \sum_{j=1}^{i-1} n_j C_j, \quad n_j \in \{0, 1, 2, \ldots\} \tag{6.33}$$

$$\frac{R_i}{T_k} \leq n_k < \frac{R_i}{T_k} + 1, \quad k = 1, \ldots, i-1 \tag{6.34}$$

Equation (6.31) can have several solutions and only the smallest one represents the actual response time. Therefore, we must search for the minimum response time in objective function (6.32).

Often, we are not interested in exact response times but only whether the given real-time system is schedulable or not. A task is schedulable whenever its response-time is less than or equal to its deadline and the system is schedulable whenever all tasks are schedulable. This is expressed by the following conditions:

$$R_i \leq D_i, \quad i = 1, \ldots, N \tag{6.35}$$

In [Seto et al., 1998], it has been shown that a sufficient and necessary schedulability condition for i-th task can be expressed by the following conditions on a set of integers $[n_1, \ldots, n_i]$ with $n_i = 1$:

$$\sum_{j=1}^{i} n_j C_j \leq D_i, \tag{6.36}$$

$$\sum_{j=1}^{i} n_j C_j \leq n_k T_k, \quad k = 1, \ldots i-1 \tag{6.37}$$

where $n_j \in \mathbb{Z}$. These conditions can be also derived by substituting (6.33) in (6.34). Since these conditions are proven to be sufficient and necessary, the sharp inequality in (6.34) is redundant as also follows from objective function (6.32).

For the same reason that (6.31) can have more than one solution, there might be more than one set of integers satisfying the above inequalities. It means that the real response time might be less than the value of the left-hand side of (6.36), i.e.

$$R_i \leq \sum_{j=1}^{i} n_j C_j \leq D_i, \tag{6.38}$$

but the schedulability is still guaranteed even without specifying an objective function.

This formulation is the basis for the schedulability analysis of tasks with offsets presented in Section 6.3.3.

Request Bound Function-Based Formulation

An alternative formulation of schedulability condition was found by [Lehoczky et al., 1989] and it can be expressed for i-th task as follows:

$$\bigvee_{t \in \mathcal{S}_i} \sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil C_j \leq t, \text{ where} \tag{6.39}$$

$$\mathcal{S}_i = \{kT_j : j = 1, \ldots, i-1,\ k = 1, \ldots \lfloor D_i/T_j \rfloor\} \cup \{D_i\}. \tag{6.40}$$

Here $\vee$ stands for logical OR operation and $\cup$ represents the union of sets. Such formulation can be used in different scenarios, e.g. when computation times are not known and are represented as variables in the ILP program. This is not possible in (6.36) since multiplication of two variables (n and C) is not a linear expression. This formulation was used e.g. by [Zeng and Natale, 2010].

6.3.2 Restricted Computational Model

To derive the ILP formulation of schedulability analysis for tasks with offsets, we restrict the computational model presented in Section 6.1 so that task deadlines must be less then or equal to the respective transaction periods, i.e. $D_{ij} \leq T_i$. This simplifies some expressions and makes the ILP formulation easier to derive. We will attempt to remove this restriction in our future work.

Now, we will show how this restriction changes the individual expressions derived in Section 6.2. Expression (6.1) can be simplified to $\phi_{ij} = \Phi_{ij}$. Further, every schedulable task τ_{ij} must satisfy $\phi_{ij} + J_{ij} + C_{ij} \leq D_{ij}$. Therefore, assuming that tasks have non-zero execution time, we get:

$$\phi_{ij} + J_{ij} < T_i. \tag{6.41}$$

From this condition, it can be seen that the number of activations of task τ_{ij} from Set 1, n_{ijk}, introduced in expression (6.12) can get only values zero or one, but the expression itself remains unchanged. Also all expressions (6.13) through (6.20) remain unchanged.

Now, we update the expression for completion time of task τ_{ab}. The completion time (6.21) depends on the number of the job, p. In our restricted model we do not have to investigate all values of $p = p^v_{0,ab}, \ldots, p^v_{L,ab}$ because either $p = p^v_{0,ab}$ or $p^v_{0,ab} > p^v_{L,ab}$. In the first case (6.21) can be rewritten as

$$\begin{aligned} w^v_{ab} &= B_{ab} + C_{ab} + \sum_{\forall i \in \mathrm{HP}(\tau_{ab})} W_{iv_i}(\tau_{ab}, w^v_{ab}) = \\ &= B_{ab} + C_{ab} + \sum_{\substack{\forall i \in \mathrm{HP}(\tau_{ab}) \\ \forall j \in \mathrm{hp}_i(\tau_{ab})}} \Bigg(\underbrace{\left\lfloor \frac{J_{ij} + \varphi_{ijv_i}}{T_i} \right\rfloor}_{n_{ijv_i}} + \underbrace{\left\lceil \frac{w^v_{ab} - \varphi_{ijv_i}}{T_i} \right\rceil}_{n^{S2}_{ijv_i}} \Bigg) C_{ij}, \end{aligned} \tag{6.42}$$

where $\mathrm{HP}(\tau_{ab})$ is a set of indices of all transactions containing at least one task with priority higher than the priority of τ_{ab}.

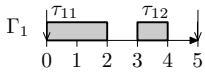

Figure 6.6: Example system of non-interfering tasks.

In the second case ($p^v_{0,ab} > p^v_{L,ab}$) this expression does not give correct result. This happens when there are tasks in a transaction which cannot interfere due to their offsets. As an example, consider the system from Figure 6.6 where the tasks have offsets $\phi_{11} = 0$ and $\phi_{12} = 3$, and zero jitters. When (6.42) is used to calculate the completion time of τ_{12} in the case of the critical instant created with τ_{11}, we get $w^{(1)}_{12} = C_{12} + 1 \cdot C_{11} = 3$ which is not true. In this case τ_{12} does not complete within the busy period created with τ_{11} and therefore $w^{(1)}_{12}$ does not exist.

The response time calculated according to (6.25) is derived from the completion time $w^v_{ab}(p)$ by subtracting from it the activation time $t(p)$ of p-th job. Using an equivalent expression with w^v_{ab} in our ILP formulation is problematic, because the ILP problem would have no solution. For this reason, we look for another expression to calculate the response time from. Such an expression is the length of busy period (6.23). For our restricted model, this expression can be rewritten as

$$L^v_{ab} = \sum_{\substack{\forall i \in \mathrm{HE}(\tau_{ab}) \\ \forall j \in \mathrm{he}_i(\tau_{ab})}} \Bigg(\underbrace{\left\lfloor \frac{J_{ij} + \varphi_{ijv_i}}{T_i} \right\rfloor}_{n_{ijv_i}} + \underbrace{\left\lceil \frac{L^v_{ab} - \varphi_{ijv_i}}{T_i} \right\rceil}_{n^{S2}_{ijv_i}} \Bigg) C^{ab}_{ij}, \tag{6.43}$$

where $C^{ab}_{ij} = \begin{cases} C_{ij} & \text{if } ij \neq ab \\ B_{ab} + C_{ab} & \text{if } ij = ab \end{cases}$,
$\mathrm{he}_i(\tau_{ab})$ is a set of tasks from i-th transaction with priority higher than or equal to τ_{ab} and $\mathrm{HE}(\tau_{ab}) = \{i : \mathrm{he}_i(\tau_{ab}) \neq \emptyset\}$.

Note that if w^v_{ab} exists, $L^v_{ab} = w^v_{ab}$. If we now use L^v_{ab} in (6.25) instead of w^v_{ab}, we get:

$$R^v_{ab} = L^v_{ab} - t = L^v_{ab} - \varphi_{abv_a} + \underbrace{\left\lfloor \frac{J_{ab} + \varphi_{abv_a}}{T_a} \right\rfloor}_{n_{abv_a}} T_a + \Phi_{ab}, \tag{6.44}$$

where we substituted the value p for $p^v_{0,ab}$. The activation time t depends on whether task τ_{ab} is activated after the critical instant (in Set 2), in which case $n_{abv_a} = 0$ or at the critical instant (in Set 1) with $n_{abv_a} = 1$. If w^v_{ab} does not exist, $R^v_{ab} \leq 0$.

Since w^v_{ab} must exist for at least one v, the worst-case response time is, similarly as in (6.26), given by

$$R_{ab} = \max_{v \in \mathcal{V}} R^v_{ab}. \tag{6.45}$$

6.3.3 Linear Schedulability Conditions

In the following we derive conditions that are equivalent to schedulability condition $R_{ab} \leq D_{ab}$. We can rewrite this condition as

$$\max_{v \in \mathcal{V}} R_{ab}^{v} \leq D_{ab} \quad \Rightarrow \quad \bigwedge_{v \in \mathcal{V}} R_{ab}^{v} \leq D_{ab}, \tag{6.46}$$

where $\wedge$ represents logical AND.

Lemma 1 *The value of φ_{ijk} from (6.43) can be expressed as $\varphi_{ijk} = n'_{ijk}T_i - (\phi_{ik} + J_{ik} - \phi_{ij})$, where n'_{ijk} is an integer variable satisfying:*

$$0 \leq \phi_{ik} + J_{ik} - \phi_{ij} - (n'_{ijk} - 1)T_i < T_i. \qquad \square$$

Proof The lemma follows from (6.18) and from properties of modulo operator, i.e. $a \bmod b = a - nb|_{0 \leq a-nb<b,\ n \in \mathbb{Z}}$. ■

Lemma 2 *The number of activations, n_{ijk}, of task τ_{ij} that can be delayed to the critical instant created with τ_{ik}, satisfies:*

$$0 < (n_{ijk} + 1)T_i - J_{ij} - \varphi_{ijk} \leq T_i. \qquad \square$$

Proof This lemma follows from (6.43) and from a property of the floor function: $\lfloor x \rfloor = n|_{x-1<n\leq x,\ n \in \mathbb{Z}}$. ■

Theorem 4 *In the system of tasks with offsets, where task deadlines are less then or equal to transaction periods, task τ_{ab} is schedulable if and only if there exist integers $n_{ijk}, n'_{ijk}, n^{S2}_{ijk}$, $i \in \mathrm{HE}(\tau_{ab})$, $j,k \in \mathrm{he}_i(\tau_{ab})$ satisfying:*

$$\sum_{\substack{i \in \mathrm{HE}(\tau_{ab}) \\ j \in \mathrm{he}_i(\tau_{ab})}} (n_{ijv_i} + n^{S2}_{ijv_i})C^{ab}_{ij} - \varphi_{abv_a} + n_{abv_a}T_a + \Phi_{ab} \leq D_{ab}, \qquad \forall v \in \mathcal{V} \tag{6.47}$$

$$\sum_{\substack{i \in \mathrm{HE}(\tau_{ab}) \\ j \in \mathrm{he}_i(\tau_{ab})}} (n_{ijv_i} + n^{S2}_{ijv_i})C^{ab}_{ij} \leq n^{S2}_{lmv_l}T_l + \varphi_{lmv_l}, \qquad \forall v \in \mathcal{V} \forall l \in \mathrm{HE}(\tau_{ab}) \forall m \in \mathrm{he}_l(\tau_{ab}) \tag{6.48}$$

$$\varphi_{ijk} = \phi_{ij} - \phi_{ik} - J_{ik} + n'_{ijk}T_i, \qquad \forall i \in \mathrm{HE}(\tau_{ab}) \forall j,k \in \mathrm{he}_i(\tau_{ab}) \tag{6.49}$$

$$0 \leq \phi_{ik} + J_{ik} - \phi_{ij} - (n'_{ijk} - 1)T_i < T_i, \quad \forall i \in \mathrm{HE}(\tau_{ab}) \forall j,k \in \mathrm{he}_i(\tau_{ab}) \tag{6.50}$$

$$0 < (n_{ijk} + 1)T_i - J_{ij} - \varphi_{ijk} \leq T_i, \qquad \forall i \in \mathrm{HE}(\tau_{ab}) \forall j,k \in \mathrm{he}_i(\tau_{ab}). \tag{6.51}$$

$\square$

Proof First we show that if task τ_{ab} is schedulable, the above conditions hold. Since task τ_{ab} is schedulable the condition (6.46) holds. By substituting this condition from (6.44) and (6.43) we get (6.47), where n_{ijv_i} and n_{abv_a} are expressed by (6.51) according to Lemma 2 and φ_{ijv_i} and φ_{abv_a} are expressed by (6.49) and (6.50) according to Lemma 1. Now, it remains to show that $n^{S2}_{ijv_i}$ (the number of activations

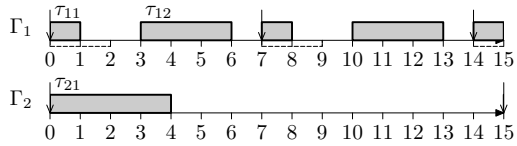

Figure 6.7: Example system of tasks with offsets.

of task τ_{ij} from Set 2) satisfies (6.48). The length of τ_{ab} busy period initiated with tasks τ_{iv_i} is according to (6.43) $L^v_{ab} = \sum_{i,j}(n_{ijv_i} + n^{S2}_{ijv_i})C^{ab}_{ij}$. During that time task τ_{ij} executed n_{ijv_i} times due to activations from Set 1 and $n^{S2}_{ijv_i}$ time due to activations from Set 2. Therefore the end of the busy period must be less or equal to the $(n^{S2}_{ijv_i} + 1)$-th activation of τ_{ij} in Set 2, which happens at $n_{ijv_i}T_i + \varphi_{ijv_i}$, i.e. $L^v_{ab} \leq n_{ijv_i}T_i + \varphi_{ijv_i}$, which is exactly (6.48).

Now, we show the second implication, i.e. if the above conditions hold, the task τ_{ab} is schedulable. The first condition (6.47) says that response time of task τ_{ab} is less or equal to its deadline so it follows that τ_{ab} is schedulable if the numbers n_{ijk} and n^{S2}_{ijk} represent the correct number of executions of task τ_{ij} during τ_{ab} busy period. From Lemma 2 we see that n_{ijk} is correct. The second condition (6.48) ensures that n^{S2}_{ijk} is so high that $(n^{S2}_{ijk} + 1)$-th activation that occurs at or after the end of τ_{ab} busy period. Therefore n^{S2}_{ijk} represents the correct number of executions of τ_{ij} in (6.47) and the task τ_{ab} is schedulable. ■

Example 2 We demonstrate how to apply Theorem 4 on an example from Figure 6.7. There are two transactions with periods $T_1 = 7$ and $T_2 = 15$ and three independent tasks i.e. $B_{ab} = 0$. Task parameters are: $C_{11} = 1$, $\phi_{11} = 0$, $J_{11} = 2$ $C_{12} = 3$, $\phi_{12} = 3$, $J_{12} = 0$, $C_{21} = 5$, $\phi_{21} = 0$, $J_{21} = 0$ and $D_{21} = 15$. Task τ_{21} has the lowest priority and the conditions for its schedulability are shown below.

$$n_{111} + n^{S2}_{111} + 3n_{121} + 3n^{S2}_{121} + 4n_{211} + 4n^{S2}_{211} + \varphi_{211} - 15n_{211} \leq 15 \tag{6.52}$$
$$n_{112} + n^{S2}_{112} + 3n_{122} + 3n^{S2}_{122} + 4n_{211} + 4n^{S2}_{211} + \varphi_{211} - 15n_{211} \leq 15 \tag{6.53}$$

$$\begin{aligned}
n_{111} + n^{S2}_{111} + 3n_{121} + 3n^{S2}_{121} + 4n_{211} + 4n^{S2}_{211} &\leq \varphi_{111} + 7n^{S2}_{111}\\
n_{111} + n^{S2}_{111} + 3n_{121} + 3n^{S2}_{121} + 4n_{211} + 4n^{S2}_{211} &\leq \varphi_{121} + 7n^{S2}_{121}\\
n_{111} + n^{S2}_{111} + 3n_{121} + 3n^{S2}_{121} + 4n_{211} + 4n^{S2}_{211} &\leq \varphi_{211} + 15n^{S2}_{211}\\
n_{112} + n^{S2}_{112} + 3n_{122} + 3n^{S2}_{122} + 4n_{211} + 4n^{S2}_{211} &\leq \varphi_{112} + 7n^{S2}_{112}\\
n_{112} + n^{S2}_{112} + 3n_{122} + 3n^{S2}_{122} + 4n_{211} + 4n^{S2}_{211} &\leq \varphi_{122} + 7n^{S2}_{122}\\
n_{112} + n^{S2}_{112} + 3n_{122} + 3n^{S2}_{122} + 4n_{211} + 4n^{S2}_{211} &\leq \varphi_{211} + 15n^{S2}_{211}
\end{aligned}$$

$$
\begin{array}{lll}
\varphi_{111} = -2 + 7n'_{111} & 0 \leq 9 - 7n'_{111} < 7 & 0 < 5 - \varphi_{111} + 7n_{111} \leq 7 \\
\varphi_{112} = -3 + 7n'_{112} & 0 \leq 10 - 7n'_{112} < 7 & 0 < 5 - \varphi_{112} + 7n_{112} \leq 7 \\
\varphi_{121} = 1 + 7n'_{121} & 0 \leq 6 - 7n'_{121} < 7 & 0 < 7 - \varphi_{121} + 7n_{121} \leq 7 \\
\varphi_{122} = 7n'_{122} & 0 \leq 7 - 7n'_{122} < 7 & 0 < 7 - \varphi_{122} + 7n_{122} \leq 7 \\
\varphi_{211} = 15n'_{211} & 0 \leq 15 - 15n'_{211} < 15 & 0 < 15 - \varphi_{211} + 15n_{211} \leq 15
\end{array}
$$

The above conditions are satisfied by the following values of integer variables:

$$
\begin{array}{lll}
n_{111} = 1 & n^{S2}_{111} \in \{1, 2\} & n'_{111} = 1 \\
n_{112} = 0 & n^{S2}_{112} \in \{1, 2\} & n'_{112} = 1 \\
n_{121} = 0 & n^{S2}_{121} = 2 & n'_{121} = 0 \\
n_{122} = 1 & n^{S2}_{122} = 1 & n'_{122} = 1 \\
n_{211} = 1 & n^{S2}_{211} = 0 & n'_{211} = 1
\end{array}
$$

Note that n^{S2}_{111} and n^{S2}_{111} can take two different values. Similarly to what was explained around (6.38) for tasks without offsets, here the response time R_{21} is equal to the bigger value of left-hand sides of (6.52) and (6.53) when all n^{S2} have the lowest possible value. If their values are higher, the left-hand sides of (6.52) and (6.53) are greater than the real response time. □

6.3.4 Schedulability of Multiprocessor and Distributed Systems

Now, with Theorem 4, it is easy to formulate the schedulability conditions for multiprocessor and distributed systems where jitters depends on response times and response times depend on jitters. Response time R_{ab} of the task under analysis must satisfy:

$$
R_{ab} \geq \sum_{\substack{i \in \mathrm{HE}(\tau_{ab}) \\ j \in \mathrm{he}_i(\tau_{ab})}} (n_{ijv_i} + n^{S2}_{ijv_i}) C^{ab}_{ij} - \varphi_{abv_a} + n_{abv_a} T_a + \Phi_{ab}, \quad \forall v \in \mathcal{V} \tag{6.54}
$$

and the jitter of the subsequent task in the transaction, τ_{ab+1}, can be then calculated according to (6.29) as

$$
J_{ab+1} = R_{ab} - \Phi_{ab+1}. \tag{6.55}
$$

By putting together conditions (6.47) – (6.51) for each task in the system with conditions (6.54) and (6.55) for all but the last task in every transaction and setting $J_{i0} = 0$, and tasks offsets according to (6.28) we obtain an ILP program for Schedulability Analysis of Tasks with Offsets in Multiprocessor or distributed Systems, which we call SATOMS in the following. Besides integer variables from Theorem 4, this program contains additional variables $J_{ij} \in \mathbb{R}$, $j > 1$ which represent task jitters.

Complexity of the ILP Program

The time needed by ILP solvers to solve the problem depends mostly on the number of integer variables. The number of integer variables used in SATOMS formulation can be calculated as follows.

We start with determining the number of variables needed for deciding the schedulability of one task according to Theorem 4. We denote the number of tasks in the i-th transaction with priority higher than or equal to the priority of τ_{ab} as $\bar{N}_i(\tau_{ab})$ and $\sum_i \bar{N}_i$ as $\bar{N}$. For every transaction in the system, we need $3\bar{N}_i^2(\tau_{ab})$ variables and for the whole system we need $3\sum_i \bar{N}_i^2(\tau_{ab})$ variables. In order to relate this number to the total number of involved tasks $\bar{N}(\tau_{ab})$, we can sum inequality $\bar{N}_i(\tau_{ab}) \leq \bar{N}_i^2(\tau_{ab})$ over all transactions and we get

$$\bar{N}(\tau_{ab}) = \sum_i \bar{N}_i(\tau_{ab}) \leq \sum_i \bar{N}_i^2(\tau_{ab}) \leq \left(\sum_i \bar{N}_i(\tau_{ab}) \right)^2 = \bar{N}^2(\tau_{ab}), \tag{6.56}$$

which means that the number of integer variables $V(\tau_{ab}) = 3\sum_i \bar{N}_i^2(\tau_{ab})$ lays between three times the number of involved tasks and three times the square of the number of involved tasks, i.e.

$$3\bar{N}(\tau_{ab}) \leq V(\tau_{ab}) \leq 3\bar{N}^2(\tau_{ab}). \tag{6.57}$$

Now, the SATOMS formulation includes schedulability conditions for each of M tasks in the system, so the total number of variables V is

$$V = 3 \sum_{m=1}^{M} \sum_i \bar{N}_i^2(\tau_m) \tag{6.58}$$

In fact, this number can be made a little bit lower because variables n and n' depend only on task parameters and can be shared between schedulability conditions for each task. On the other hand, values of n^{S2} depend on the task being analyzed and cannot be shared.

From (6.58) and (6.57) we get the following bounds for the number of integer variables V:

$$1.5M^2 \leq V \leq 3M^3. \tag{6.59}$$

The upper bound represents the case where there is only one transaction in the system whereas the lower bound is the case when there is only one task in every transaction, which is equivalent to tasks without offsets.

6.4 Experimental Results

We implemented SATOMS algorithm described in the previous section and the iterative algorithm described in Section 6.2.2. Implementation of SATOMS

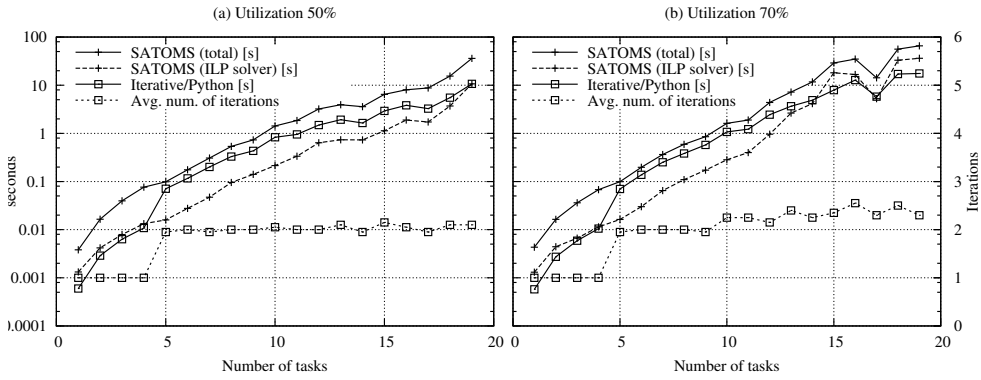

Figure 6.8: Comparison of average computation times of different implementations. System utilization is 50% in (a) and 70% in (b).

algorithm uses Python (PuLP[1] library) to generate the ILP program which is then solved by CPLEX solver. The iterative algorithm was implemented solely in Python.

We compared time complexity of the two algorithms on randomly generated task sets. Each task set comprised $m = 1, \ldots, 19$ tasks grouped into $1 + \lfloor m/5 \rfloor$ transactions and the utilization of the whole system was 0.5 and 0.7 respectively. This total utilization was randomly (uniformly) distributed among transactions and the partial utilization assigned to each transaction was also randomly distributed among tasks in the transaction. Transaction periods were randomly chosen between 20 and 1000.

For each number of tasks we generated 20 different systems, analyzed them and plot the average of these 20 runs in Figure 6.8. The graphs show the total time of SATOMS algorithm, which includes preparation of the ILP formulation and solving the ILP problem. Since in most cases the preparation took longer time than actual solving, we plot the time needed by the solver separately (dashed line). For the iterative algorithm, we plot the total time and the number of iterations (dotted line) needed to find the result.

From the graphs can be seen that SATOMS algorithm is approximately 8 times slower than the iterative algorithm but the time needed by the ILP solver is in most cases smaller than the time of the iterative algorithm.

It must be noted that our earlier experiments show that our prototype implementation of the original exact response-time analysis in Python is approximately 10 to 50 times slower than the implementation in MAST [MAST, 2010] tool, which is written in compiled language Ada. We also did not measure the overhead of PuLP library which calls the solver so it might be that actual time needed by the solver is even less than presented in the graphs.

[1] `http://code.google.com/p/pulp-or/`

6.5 Conclusion

We derived a formulation of schedulability analysis for tasks with offsets in the form of linear inequalities with integer variables suitable for solving by ILP solvers. We used this formulation as the basis for schedulability analysis of multiprocessor and distributed systems where tasks jitters depend on response times and vice versa (SATOMS). Experimental results show that such a formulation is not suitable for solving of industrial-size problems. The contribution of this method is that it may serve as a basis of a special-purpose branch-and-bound algorithm which will use heuristics that can speedup the computation.

7 Conclusions

7.1 Summary

This thesis presented FRSH/FORB framework for the management of multiple heterogeneous resources shared across a set of distributed real-time applications. The framework exposes to the application developers the FRSH API, which has been designed to allow access to real-time scheduling services resources, such as CPU, disk and network, in a way that is as uniform as possible. This way, users do not need to deal with different APIs for reserving resources on the underlying OS, but they can declare the application requirements using natural attributes such as deadlines or throughput figures, instead of priorities. The framework uses the application provided information to effectively schedule the workload. This allows for an easier deployment of real-time applications over a distributed system, especially in those cases in which the system is open and dynamic. One of the main strengths of FRSH/FORB framework is its modularity with respect to support of additional resources, which was shown by integration of six different resources into the framework.

The evaluated resources provide a great level of temporal isolation for distributed soft real-time applications. This was shown by the presented experimental results, gathered on a real implementation of the framework on the Linux OS. Specifically we evaluated and stress tested resource reservation technique for wireless LAN, which was developed in this thesis. More importantly, we reported results from a real case-study application, developed around the theme of video recording, showing the main benefits of adopting the framework. Also, we reported about our experience in how the proposed framework was used, and specifically how the resource requirements of the case-study application were determined. This constitutes a valuable experience that can be leveraged by future researchers/developers who may want to make use of it. We also identified areas where the framework could be improved to bring a greater value for its users.

Finally, the last chapter presented a novel approach to schedulability analysis for tasks with offsets where the analysis was formulated as a set of linear conditions with integer variables. It was shown how this formulation can be used to analyze multiprocessor and distributed systems and we compared the performance of our technique with the previously known iterative technique. Our method is not directly usable for bigger systems but we believe that it can serve as a basis for future, more efficient, techniques.

7.2 Goals

The goals set at the beginning of this thesis were successfully completed as detailed below.

1. The resource reservation framework was designed and implemented as described in Chapter 3. Support for new resources can be easily added as we demonstrate in Chapter 4 where six different resources were integrated into the framework. The integration of FPGA resource in Section 4.5 demonstrates the support for task migration between resources.

2. The framework was extensively evaluated and the results are provided in Chapter 5.

3. An admission test for Wi-Fi networks was developed and it is described in Section 4.3.4. Wi-Fi networks were integrated into the framework and the admission test was evaluated in Section 5.2.

4. Schedulability analysis for tasks with offsets was formulated as an integer linear programming problem in Section 6.3 and the performance of this approach was evaluated in Section 6.4.

A

FRSH API Change Proposal

A.1 Introduction

In this document we are trying to summarize our experience with FRSH API defined in [Gonzáles Harbour and Tellería de Esteban, 2008]. Our experience is based on development of an alternative FRSH implementation, where we aimed at better abstraction of the API from the underlying implementation. With such an experience we propose some changes to the API.

It should be noted that this version of the document represents my personal view of the situation. But, on the other hand, many of the presented opinions were influenced by discussions in our group here at CTU.

The problems of the current API [Gonzáles Harbour and Tellería de Esteban, 2008] can be classified into several groups:

A **Negotiation time vs. run-time services.** We think, that it is important to clearly say whether an service is meant to be used during negotiation time or during run-time. There are a few cases in the current API, where these use-cases are intermixed. Negotiation services should only be used to specify information needed for admission test (i.e. application requirements). Once the negotiation was successful, run-time services (bind, get_remaining_budget, ...) are used by applications. This division is essential to support applications with admission test computed off-line, so that the developers know which functions may be used in the "off-line" mode. This division should make clear which services (the negotiation-time ones) can take long time to complete due to possible (remote communication, redistribution of spare capacity etc.) and which should be fast (run-time services).

B **Resource dependent services.** Since FRSH framework aims at being multi-resource framework, it must be clear which functions apply to which resource. The current API seems to be heavily influenced by CPU-centric thinking.

Every resource type must provide at least different "bind" function. For CPU we bind threads to VREServes, for networks communication endpoints, for disk probably file descriptors, etc.

C **Fundamental, internal and helper functions.** There is no clear boundary between helper and internal functions and the functions which are really needed by applications. This makes the API very huge and hard to learn.

It might be useful to define minimal FRSH API and to introduce a helper library which will provide additional services built on top of the minimal API. This will make porting FRSH framework to other platforms easier.

D **Problems in dynamic/open environments running multiple independent applications.** Some parts of the API suppose that the developer has full control of all applications running in the system. This was probably caused by the fact that the first implementations were done on simple real-time executive (MarteOS) and not on systems with full memory protection like Linux.

E **Attempt to define OS compatibility layer.** Replacement of native OS services with FRSH services is a questionable thing. On one side it makes the FRSH applications portable between different platforms and FRSH implementations, on the other side it probably limits the services provided by the native platform and makes porting legacy application more difficult. While limiting the functionality of the underlying platform is in line with the goal of achieving higher predictability, increasing complexity of porting existing applications might be a problem.

A possible solution might be a dual approach, where a simple compatibility layer is provided by FRSH together with an implementation specific interface which allows to manage native OS entities by FRSH.

A.2 Specific problems in the current API

A.2.1 frsh_contract_*_resource_and_label()

I do not see any reason why to put manipulation of resource and label into a single function. Labels are mainly used for debugging and distributed negotiations, whereas resource must be set in all cases in the contract. So I'd introduce two separate functions for this.

A.2.2 frsh_contract_set_timing_reqs() A B

This function contains parameters which determine signals used by the framework to notify the application about events like deadline-miss or budget overrun. In our opinion this should be better specified as parameter to `frsh_thread_bind()` as this information is not needed by the admission test and in case of remote negotiation the application preparing the contract might not have enough knowledge of the remote application to specify the signals correctly.

Moreover an implementation of the framework might use different means of communications to inform the application about the occurred events. This is also the case with non-CPU contracts, e.g. for networks, budget overrun can be usually determined at queueing/sending time.

A.2.3 Group contract negotiation C

Group contracts represents a strange concept. On one side it is only an optimization to save several runs of the admission test, on the other side it has many in common with (distributed) transactions. The only difference from transactions is that a transaction can comprise of contracts for different resources which are "balanced" so that they have the same period.

We propose the group contract negotiation to be the basic service in FRSH API and the other functions like `frsh_contract_negotiate()`, `frsh_contract_cancel()` etc. will be provided in the helper library by building on top of group contract negotiation.

A.2.4 frsh_vres_get_*() B

Some of these functions are only useful for CPU. For example `frsh_vres_get_job_usage()` has little sense for networks, where the application probably knows how many bytes it has already sent.

A.2.5 frsh_service_thread_* C D

B

These functions should not be part of the public API since they are too implementation specific. The budget and period would be better specified only as some configuration parameters. Moreover it has no sense if multiple applications try to change these values simultaneously. The service thread only deals with CPU contracts. There is no similar function for other resources.

A.2.6 frsh_resource_get_vres_from_label() B D

This function was probably meant only for CPU contracts, it is not clearly defined for networks, because it is not clear what it means "negotiated in the same processing node". Also its applicability is questionable in systems with memory protection, where it is not desirable for one application to access resources of another application.

A.2.7 Shared objects

I didn't follow this area much, but I'm embarrassed about this whole thing. It's clear that we need to manage critical sections in order to calculate blocking time, but if I am new to FRSH, I'd not use this API because of its complexity and overhead. The other problems I can see is the lack of support for nested critical sections and inability to specify different WCETs for different spare capacity allocations.

So, I do not have here a specific proposal for change, but only a notion that there must be a better abstraction for shared objects.

Also, if we keep the API, it should be split to internal and external part as some functions (`frsh_csect_get_blocking_time()`) are only needed by admission test code. C.

A.2.8 Spare capacity

The distribution of spare capacity is supposed to be based on *importance* and *weight* parameters and it is only considered within boundary of a single resource. If multiple contracts are bound by a transaction, it is often necessary to allocate the spare capacity to the transaction as a whole and not to the separate contracts independently.

`frsh_resource_get_capacity()` should probably be an internal function C not exposed to applications. I cannot see any reason why it is useful for application to know this information since the available capacity returned by this function may already be outdated when the application processes it.

`frsh_vres_set_stability_time()` suffers from B. It has probably only sense for "local" resources such as CPU and disk. For networks it might involve remote communication, which is probably not acceptable. Therefore, this function also falls to category A.

See also transaction support in A.2.13.

A.2.9 Networking API E

FRSH framework uses FNA API [Vila Carbó et al., 2007]. Using a special purpose API for network communication makes porting legacy applications to FRSH hard. It would be useful if FRSH can provide BSD Sockets API, which is the most used networking API. There are two possible approaches how to provide this API:

A. Provide BSD sockets layer on top of FNA API.

B. BSD sockets API will be provided natively by FRSH implementation.

These two possibilities are discussed in the following sections.

BSD sockets on top of FNA API

This is possible without changing current FNA API, but it also has limitations in that there can be a conflict between FRSH sockets layer and native BSD sockets API of the OS. It will be probably possible to solve this conflict by some C preprocessor macro tricks.

Native BSD sockets

For operating systems without this BSD sockets API, it is just a matter of renaming FNA functions and their parameters.

```
frsh_contract_negotiate(contract, &vres);
s = frsh_get_vres_socket(vres)
/* use the socket as the application wants */
```

Figure A.1: Usage of native BSD sockets in FRSH applications

For systems with BSD sockets API (e.g. Linux, RTEMS) FNA functionality must be inserted in BSD sockets API implementation. In case of Linux it means, that virtual resources must be implemented in the kernel. We think this is the right approach since it allows to run non-FRSH applications and the kernel assures, that these applications will not use the network bandwidth reserved by FRSH applications.

The application will use the functions like `send()` and `recv()` to exchange the messages and FRSH/FNA functions (e.g. `frsh_vres_get_remaining_budget()`) to operate on VRES. Under Linux, these FRSH/FNA functions will be implemented using IOCTL mechanism on sockets to communicate with VRES implementation in kernel.

Therefore, this implementation would not provide functions like `frsh_send_endpoint_create()` and all the information like destination and stream_id will be provided in contract. It has another advantage, because for some network (e.g. WiFi), this information has influence on bandwidth usage and is therefore necessary for admission test.

If it is not desirable, to specify all socket parameters in contracts, other possibility is to have API where the sockets are created in a normal way and then binded to VRES similarly as in the CPU case (see Fig. A.2).

```
frsh_contract_negotiate(contract, &vres);
s = socket(AF_INET, SOCK_DGRAM, 0);
frsh_bind_socket(vres, s);
/* use the socket as the application wants */
```

Figure A.2: Alternative possibility of using native BSD sockets in FRSH applications

A.2.10 Two-step negotiation C

Two-step negotiation should be an internal API. For application developers it only adds complexity.

A.2.11 frsh_network_get_* E

This kind of information should be provided by a protocol specific mean. It is not possible to write application independently on used network protocol (e.g. transfer video over CAN-bus).

A.2.12 FRESCOR Network Adaptation (FNA)

The main problems of FNA are:

- There are services, that apply to all resource and not only to networks (e.g. `fna_contract_negotiate()`, ...) B. There should be a generic interface for implementations to implement these services. Such "virtualization" (in the *Object Oriented Programming* (OOP)-sense) would be beneficial for all resource types.
- Missing bind callback called when `frsh_send_endpoint_bind()` is called by an application.

A.2.13 Distributed (multi-resource) transactions

In FRSH, the way how distributed transactions are implemented builds on the fact, that there exist support for reserving individual resources and transactions are implemented as higher level layer on top of this "resource reservation" layer. This is IMHO correct approach, but I think that its implementation in FRSH has some problems because the reservation layer is not as simple as it should be. I believe that the approach taken in FRSH/FORB implementation is more useful. The fundamental difference is that in FRSH/FORB spare capacity is not redistributed in reservation layer (as in the case of FRSH) but in the higher layer – the same one which handles transactions.

I can also see that the problem of distributed transactions is very difficult but everybody want a solution for it. Therefore I believe that this "higher layer" should be an integral part of the framework as it could represent big added value of the framework. This is how FRSH/FORB was designed.

A.3 Conclusion

From the above text, one can conclude that the whole FRSH API is wrong and it is not useful in its current form. That's not true – the API is already useful for the prototyping work and it can serve well to many existing applications. This document only tries to find places where the API could be improved in order to be accepted by a more wide development community.

Bibliography

[Abeni and Buttazzo, 1998] Abeni, L. and Buttazzo, G. (1998). Integrating multimedia applications in hard real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, Madrid, Spain.

[Alur and Dill, 1994] Alur, R. and Dill, D. L. (1994). A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235.

[Baruah and Fisher, 2005] Baruah, S. K. and Fisher, N. (2005). Code-size minimization in multiprocessor real-time systems. In *IPDPS*. IEEE Computer Society.

[Bini, 2009] Bini, E. (2009). Modeling preemptive EDF and FP by integer variables. In *4th Multidisciplinary International Scheduling Conference, proceedings*.

[Bini et al., 2009] Bini, E., Nguyen, T. H. C., Richard, P., and Baruah, S. K. (2009). A Response-Time bound in Fixed-Priority scheduling with arbitrary deadlines. *IEEE Trans. Comput.*, 58(2):279–286.

[Bordin et al., 2008] Bordin, M., Panunzio, M., and Vardanega, T. (2008). Fitting schedulability analysis theory into model-driven engineering. In *20th Euromicro Conference on Real-Time Systems*, pages 135–144. IEEE Computer Society.

[Boudec and Thiran, 2004] Boudec, J. L. and Thiran, P. (2004). Network calculus: A theory of deterministic queuing systems for the internet. In *Lecture Notes in Computer Science (LNCS)*. Springer-Verlag.

[Burns et al., 2003] Burns, A., Dobbing, B., and Vardanega, T. (2003). Guide for the use of the Ada Ravenscar Profile in high integrity systems. Technical Report YCS-2003-348, University of York.

[Burns and Wellings, 2001] Burns, A. and Wellings, A. (2001). *Real Time Systems and Programming Languages: Ada 95, Real-Time Java and Real-Time C/POSIX*. Addison Wesley, 3 edition.

[Buttazzo, 2005] Buttazzo, G. C. (2005). Rate monotonic vs. EDF: judgment day. *Real-Time Syst.*, 29(1):5–26.

[Cancila et al., 2010] Cancila, D., Passerone, R., Vardanega, T., and Panunzio, M. (2010). Toward correctness in the specification and handling of non-functional attributes of high-integrity real-time embedded systems. *Industrial Informatics, IEEE Transactions on*, 6(2):181 – 194.

[CiA, 2001] CiA (2001). CAN history. `http://www.can-cia.org/index.php?id=161`.

[Davare et al., 2007] Davare, A., Zhu, Q., Natale, M. D., Pinello, C., Kanajan, S., and Sangiovanni-Vincentelli, A. (2007). Period optimization for hard real-time distributed automotive systems. In *Proceedings of the 44th annual Design Automation Conference*, pages 278–283, San Diego, California. ACM.

[Deng et al., 2008] Deng, G., Schmidt, D. C., Gill, C. D., and Wang, N. (2008). QoS-enabled component middleware for distributed real-time and embedded systems. In Lee, I., Leung, J. Y.-T., and Son, S. H., editors, *Handbook of Real-Time and Embedded Systems*. Chapman & Hall/CRC.

[Döhmen et al., 2008] Döhmen, G., Enzmann, M., Andersson, H., and Hördt, C. (2008). SPEEDS methodology – a white paper. [online] http://www.speeds.eu.com/downloads/SPEEDS_WhitePaper.pdf, last visited 3/2009.

[Donato et al., 2005] Donato, A., Ferrandi, F., Santambrogio, M. D., and Sciuto, D. (2005). Operating system support for dynamically reconfigurable soc architectures. In *IEEE International SOC Conference, 2005. Proceedings.*

[Engelstad and Østerbø, 2006] Engelstad, P. and Østerbø, O. (2006). The delay distribution of IEEE 802.11e EDCA and 802.11 DCF. In *Performance, Computing, and Communications Conference, 2006. IPCCC 2006. 25th IEEE International.*

[FRESCOR, 2009] FRESCOR (2009). FRESCOR project. Online, `http://frescor.org/`.

[FRSH/FORB, 2010] FRSH/FORB (2010). FRSH/FORB framework. Online, `http://frsh-forb.sourceforge.net/`.

[Gonzáles Harbour and Tellería de Esteban, 2006] Gonzáles Harbour, M. and Tellería de Esteban, M. (2006). Architecture and contract model for integrated resources. Deliverable of the FRESCOR project (D-AC2v1), `http://www.frescor.org/index.php?page=publications`.

[Gonzáles Harbour and Tellería de Esteban, 2008] Gonzáles Harbour, M. and Tellería de Esteban, M. (2008). Architecture and contract model for integrated resources II. Deliverable of the FRESCOR project (D-AC2v2), `http://www.frescor.org/index.php?page=publications`.

[Guo, 2006] Guo, F. (2006). *Implementation Techniques for Scalable, Secure and QoS-Guaranteed Enterprise-Grade Wireless LANs.* PhD thesis, Stony Brook University.

[Huber, 2008] Huber, B. (2008). *Resource Management in an Integrated Time-Triggered Architecture.* Doctoral thesis, Technische Universitt Wien.

[IEEE, 1999] IEEE (1999). Wireless LAN medium access control (MAC) and physical layer (PHY) specification.

[IEEE, 2005] IEEE (2005). Medium access control (MAC) quality of service enhancements.

[Jurčík et al., 2008] Jurčík, P., Severino, R., Koubaa, A., Alves, M., and Tovar, E. (2008). Real-time communications over cluster-tree sensor networks with mobile sink behaviour. In *14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), Proceedings*, pages 401–412.

[Kohout, 2007] Kohout, L. (2007). Partial dynamic reconfiguration in Xilinx FPGA circuits. In *CAK Embedded Systems Colloquium*. `http://rtime.felk.cvut.cz/kolokvium/2007/presentations/kohout_lukas.pdf` (last visited: Nov 16 2008).

[Lehoczky et al., 1989] Lehoczky, J., Sha, L., and Ding, Y. (1989). The rate monotonic scheduling algorithm: exact characterization and average case behavior. In *Real Time Systems Symposium, 1989., Proceedings.*, pages 166–171.

[Lehoczky et al., 1987] Lehoczky, J., Sha, L., and Strosnider, J. (1987). Enhanced aperiodic responsiveness in hard real-time environments. In *Proceedings of IEEE Real-Time System Symposium*, pages 261–270.

[Liu, 2000] Liu, J. W. S. (2000). *Real-Time Systems*. Prentice Hall, Upper Saddle River, NJ, USA.

[Mäki-Turja and Nolin, 2008] Mäki-Turja, J. and Nolin, M. (2008). Efficient implementation of tight response-times for tasks with offsets. *Real-Time Systems*, 40(1):77–116.

[Mangold et al., 2002] Mangold, S., Choi, S., May, P., Klein, O., Hiertz, G., and Stibor, L. (2002). IEEE 802.11e wireless LAN for quality of service. In *European Wireless, Proc.*

[MAST, 2010] MAST (2010). MAST (Modeling and Analysis Suite for Real-Time Applications).
`http://mast.unican.es/`.

[McGuire et al., 2009] McGuire, N., Okech, P. O., and Zhou, Q. (2009). Analysis of inherent randomness of the Linux kernel. In *Eleventh Real-Time Linux Workshop*.

[Ni et al., 2004] Ni, Q., Romdhani, L., and Turletti, T. (2004). A survey of QoS enhancements for IEEE 802.11 wireless LAN. *Wireless Communications and Mobile Computing, Journal*, 4(5):577 – 566.

[Object Management Group, 2008] Object Management Group (2008). *Common Object Request Broker Architecture (CORBA) Specification, Version 3.1, Part 1: CORBA Interfaces*. Number formal/2008-01-04. [Online].

[Palencia and González Harbour, 1998] Palencia, J. C. and González Harbour, M. (1998). Schedulability analysis for tasks with static and dynamic offsets. In *Proceedings of the 19th Real-Time Systems Symposium*, pages 26–37. IEEE Computer Society Press.

[Palopoli et al., 2009] Palopoli, L., Cucinotta, T., Marzario, L., and Lipari, G. (2009). AQuoSA — adaptive quality of service architecture. *Software – Practice and Experience*, 39(1):1–31.

[Peca et al., 2009] Peca, M., Píša, P., Sojka, M., Krákora, J., and Hanzálek, Z. (2009). Reconfigurable FPGAs. FRESCOR deliverable D-ND4, Czech Technical University in Prague.

[Peiro et al., 2007] Peiro, S., Masmano, M., Ripoll, I., , and Crespo, A. (2007). PaRTiKle OS, a replacement for the core of RTLinux-GPL. In *9th Real-Time Linux Workshop*, page 6, Linz, Austria. Real-Time Systems Group, Polytechnic University of Valencia.

[Pinto et al., 2006] Pinto, A., Bonivento, A., Sangiovanni-Vincentelli, A. L., Passerone, R., and Sgroi, M. (2006). System level design paradigms: Platform-Based design and communication synthesis. *ACM Transactions on Design Automation of Electronic Systems*.

[Real and Crespo, 2004] Real, J. and Crespo, A. (2004). Mode change protocols for Real-Time systems: A survey and a new proposal. *Real-Time Syst.*, 26(2):161–197.

[Ridouard et al., 2004] Ridouard, F., Richard, P., and Cottet, F. (2004). Negative results for scheduling independent hard real-time tasks with self-suspensions. In *The 25th IEEE International Real-Time Systems Symposium*.

[Rivas and Harbour, 2000] Rivas, M. A. and Harbour, M. G. (2000). Early experience with an implementation of the POSIX.13 minimal real-time operating system for embedded applications. In *25th IFAC Workshop on Real-Time Programming*.

[Rivas and Harbour, 2001] Rivas, M. A. and Harbour, M. G. (2001). Marte os: An ada kernel for real-time embedded applications. In *Ada-Europe*, Leuven, Belgium.

[Schmidt, 2006] Schmidt, D. C. (2006). Guest editor's introduction: Model-Driven engineering. *Computer*, 39(2):25–31.

[Seto et al., 1998] Seto, D., Lehoczky, J. P., and Sha, L. (1998). Task period selection and schedulability in Real-Time systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, page 188. IEEE Computer Society.

[Sigh et al., 2008] Sigh, I., Trdlička, J., and Hanzálek, Z. (2008). ITEM – implementation of integrated TDMA and E-ASAP module. In *Work-In-Progress Session of 20th Euromicro Conference on Real-Time Systems, Proceedings*, pages 64–67.

[Silberschatz et al., 2008] Silberschatz, A., Galvin, P. B., and Gagne, G. (2008). *Operating System Concepts*. Wiley Publishing.

[Sojka, 2006] Sojka, M. (2006). Optimization-based approach to response-time analysis for tasks with offsets. In *International Student Conference on Electrical Engineering (POSTER)*, Prague, Czech Republic. Czech Technical University in Prague.

[Sojka et al., 2008] Sojka, M., Molnár, M., Trdlička, J., Jurčík, P., Smolík, P., and Hanzálek, Z. (2008). Wireless networks – documented protocols, demonstration. FRESCOR Deliverable D-ND3v2, Czech Technical University in Prague.

[Sojka et al., 2010] Sojka, M., Píša, P., Faggioli, D., Cucinotta, T., Checconi, F., Hanzálek, Z., and Lipari, G. (2010). Modular software architecture for flexible reservation mechanisms on heterogeneous resources. *Journal of Systems Architecture*. Under review.

[Sprunt et al., 1989] Sprunt, B., Sha, L., and Lehoczky, J. (1989). Aperiodic task scheduling for hard-real-time systems. *Real-Time Systems*, 1(1):27–60.

[Stankovic and Ramamritham, 1989] Stankovic, J. A. and Ramamritham, K., editors (1989). *Tutorial: hard real-time systems*. IEEE Computer Society Press, Los Alamitos, CA, USA.

[Stanovich et al., 2010] Stanovich, M., Baker, T. P., Wang, A., and Harbour, M. G. (2010). Defects of the POSIX sporadic server and how to correct them. In *Real-Time and Embedded Technology and Applications Symposium, IEEE*, volume 0, pages 35–45, Los Alamitos, CA, USA. IEEE Computer Society.

[Telleria de Esteban, 2008] Telleria de Esteban, M. (2008). Implementation of a flexible real-time scheduling middleware based on contracts. Master's thesis, Universidad de Cantabria. http://www.mtelleria.com/tesis_master/tesis_mte.pdf.

[Tindell and Clark, 1994] Tindell, K. and Clark, J. (1994). Holistic schedulability analysis for distributed hard real-time systems. *Microprocess. Microprogram.*, 40(2-3):117–134.

[Traore et al., 2006] Traore, K., Grolleau, E., Rahni, A., and Richard, M. (2006). Response-Time analysis of tasks with offsets. In *Emerging Technologies and Factory Automation, 2006. ETFA '06. IEEE Conference on*, pages 1–8.

[Valente and Checconi, 2010] Valente, P. and Checconi, F. (2010). High throughput disk scheduling with fair bandwidth distribution. *IEEE Transactions on Computers*, 99(PrePrints).

[Vila Carbó et al., 2007] Vila Carbó, J., Sangorrín López, D., Hernández Orallo, E., and Smolík, P. (2007). General purpose networks. FRESCOR Deliverable D-ND2v1, Universidad Politcnica de Valencia.

[Vittorio and Lo Bello, 2007] Vittorio, S. and Lo Bello, L. (2007). An approach to enhance the QoS support to real-time traffic on IEEE 802.11e networks. In *6th Intl Workshop On Real Time Networks*.

[Waszinowski and Hanzálek, 2003] Waszinowski, L. and Hanzálek, Z. (2003). Analysis of Real Time Operating System Based Applications. In *Proceedings of the 1st International Workshop on Formal Modeling and Analysis of Timed Systems*. Springer-Verlag.

[Xiao, 2004] Xiao, Y. (2004). Performance analysis of IEEE 802.11e EDCF under saturation condition. In *Communications, 2004 IEEE International Conference on*, volume 1, pages 170–174.

[Xilinx, 2004] Xilinx (2004). Two flows for partial reconfiguration: Module based or difference based. Application note, Xilinx Inc.

[Zeng and Natale, 2010] Zeng, H. and Natale, M. D. (2010). Improving Real-Time feasibility analysis for use in linear optimization methods. In *22nd Euromicro Conference on Real-Time Systems*.

[Zheng et al., 2007] Zheng, W., Zhu, Q., Natale, M. D., and Vincentelli, A. S. (2007). Definition of task allocation and priority assignment in hard Real-Time distributed systems. In *Real-Time Systems Symposium, IEEE International*, volume 0, pages 161–170, Los Alamitos, CA, USA. IEEE Computer Society.

Curriculum vitae

Michal Sojka was born in Prague, Czech Republic, in 1978. He received his master degree in cybernetics and control engineering from Faculty of Electrical Engineering at Czech Technical University (CTU FEE) in Prague, Czech Republic in 2003.

Since 2003 he has been working towards the doctor of philosophy degree (Ph.D.) in electrical engineering and information technology at the Department of Control Engineering, CTU FEE in Prague. From 2003 to 2004, he participated in European project OCERA (Open Components for Embedded Real-time Applications), and from 2006 to 2009 in EU project FRESCOR (Framework for Real-time Embedded Systems based on COntRacts). He also participated in the organization of the international summer schools "ARTIST2 2006" and "Real-Time Linux Intro 2007" and he was a member of organizing committee of the international conference ECRTS'08. His research interests include scheduling and analysis of real-time systems, software development methodologies and robotics.

His teaching activities at CTU FEE involve courses on Real-Time Programming, Open-Source Development, Computers for Control and Distributed Control Systems. He maintains several software packages used in many projects ranging from CTU diploma theses to projects of external companies. Around 2008 he cooperated with AŽD Prague on development of industrial communication stack for European Train Control System and in 2009 he did consulting of VxWorks operating system for Rockwell Automation.

Author's Publications

Journal Papers

1. M. Sojka, P. Píša, D. Faggioli, T. Cucinotta, F. Checconi, Z. Hanzálek, and G. Lipari, "Modular software architecture for flexible reservation mechanisms on heterogeneous resources," *Journal of Systems Architecture*, 2010, under review, authorship 35%.

Conference Papers

2. O. Špinka, J. Krákora, M. Sojka, and Z. Hanzálek, "Low-cost avionics system for ultra-light aircraft," in *11th IEEE International Conference on Emerging Technologies and Factory Automation*. IEEE, 2006, pp. 102–109, authorship 20%.

3. M. Sojka, "Optimization-based approach to response-time analysis for tasks with offsets," in *International Student Conference on Electrical Engineering (POSTER)*. Prague, Czech Republic: Czech Technical University in Prague, 2006.

4. P. Šůcha, M. Kutil, M. Sojka, and Z. Hanzálek, "TORSCHE scheuling toolbox for Matlab," in *IEEE Symposium on Computer-Aided Control System Design*, 2006, pp. 50 – 52, authorship 20%.

5. M. Peca, M. Sojka, and Z. Hanzálek, "SPEJBL – The biped walking robot," in *Preprints 7th IFAC International Conference on Fieldbuses and nETworks in industrial and embedded systems (FET)*. Toulouse: Universite Toulouse, 2007, pp. 63–70, authorship 5%.

6. M. Sojka, M. Molnár, and Z. Hanzálek, "Experiments for real-time communication contracts in IEEE 802.11e EDCA networks," in *Factory Communication Systems. IEEE International Workshop on*, 2008, pp. 89 – 92, authorship 80%.

7. M. Sojka and Z. Hanzálek, "Modular architecture for real-time contract-based framework," in *4th IEEE International Symposium on Industrial Embedded Systems*, 2009, pp. 66 – 69, authorship 90%.

8. M. Sojka, P. Píša, M. Petera, O. Špinka, and Z. Hanzálek, "A comparison of Linux CAN drivers and their applications," in *5th IEEE International Symposium on Industrial Embedded Systems (SIES)*, 2010, authorship 35%.