Abstract  We investigate the mathematical properties of event bound functions as they are used in the worst-case response time analysis and utilization tests. We figure out the differences and similarities between the two approaches. Based on this analysis, we derive a more general form do describe events and event bounds. This new unified approach gives clear new insights in the investigation of real-time systems, simplifies the models and will support algebraic proofs in future work. In the end, we present a unified analysis which allows the algebraic definition of any scheduler. Introducing such functions to the real-time scheduling theory will lead two a more systematic way to integrate new concepts and applications to the theory. Last but not least, we show how the response time analysis in dynamic scheduling can be improved.

Keywords  Scheduling Theory; Feasibility Test; Response Time Analysis; Static Scheduling; Dynamic Scheduling; Unification of Scheduling Theory; Dirac Delta; Heaviside Function

Acknowledgements  This work was supported by the Deutsche Forschungsgemeinschaft (DFG) under grand SL 47/17-1. Credits to Iwan Feras Fattohi for his critical comments and Kilian Kempf for proofreading. Finally, the paper is dedicated to Ulrich Slomka and Ulrich Herzog.

1 Introduction

If we have a careful review of existing work in real-time scheduling theory, we found two different approaches to satisfy the real-time capability of an embedded system: the feasibility test or in more general the utilization based approach and
the response time analysis. The feasibility tests compute the utilization of a
hardware resource as the response time analysis focus on the behaviour of tasks.
In system analysis, both approaches are useful. However, looking to related work,
the two approaches are different in one small detail: while the utilization based
tests computations are built on the floor operator, the response time analysis
uses the ceiling operator. Nevertheless, if we look closer to previous work, this
leads to problems in formulating a utilization based test for static scheduling and
a response time analysis for dynamic scheduling. However, in the practical use of
the scheduling theory, the analysis of static scheduling prefers the response time
analysis while the analysis in dynamic scheduling prefers the utilization based test.
The reason for this observation is that the mathematical expressiveness of both
functions is limited: the floor and the ceiling operator does not support algebraic
properties such as distributivity and commutativity. Besides, these operators are
not analytical in the sense that calculus is not well supported. The work of
[Biondi et al., 2014] and [Stigge and Yi, 2013] shows the limitations on floor
and ceil operators in the context of the real-time scheduling theory. Both papers
postulate new analysis techniques if an event count with more mathematical
expressiveness is known. From a practical perspective, real-time analysis work
always covers one concrete problem, and the algorithms published are solving
just this particular problem. Combining different ideas is difficult because the
task models often change. Sometimes different algorithms are used to address
different problems in the application of the theory. [Anssi et al., 2013] discusses
different real-time analysis methods to compute task response times to cover
multiple issues in the automotive industry and find that different approaches are
necessary to cover all aspects needed.

This paper presents an approach to address both problems directly. If we look
at another domain in science and engineering, the problem of discrete and con-
tinuous behaviour was already addressed. In digital signal processing and digital
control theory both worlds, the discrete and the continuous nature of systems
are combined. The idea of this paper is to adapt mathematical models used in
physics, signal- and control theory to the problem of real-time scheduling analy-
sis. As a result, we present

– a new universal mathematical framework, which allows replacing geometric
proofs given by diagrams and known from previous work with new algebraic
and analytical methods,
– a new generic approach to formulate interfering tasks in different scheduling
policies
– and therefore a unified formulation of the feasibility- and the response time
analysis in static and dynamic scheduling based on just one equation.
– Additionally we adopt assumptions of the analysis of arbitrary deadlines to
the analysis of response times in dynamic scheduled systems and found a
deterministic and tighter analysis as in previous work.
2 Related work

Real-time systems are computer systems whose software must complete calculations within fixed deadlines. For this purpose, individual tasks of the program are divided into individual and independently executable tasks. To ensure a response of tasks to given deadlines, a real-time system requires an operating system that generates predictable execution sequences. If an operating system delivers predictable schedules a mathematical model can be derived and deadline compliance can be calculated. During the Apollo missions to the moon, the first today-like real-time computer was used for guidance and navigation (Apollo Guidance Computer, AGC, [Mindell, 2008], p.221 ff). During this time software engineers expect a task utilization of 80% will guarantee a correct real-time behavior of the AGC. However, on the 20th of July 1969 during the first manned landing, the computer of the lunar module Eagle gave a program alarm at decent to the surface of the moon. During the whole landing the computer had to be reseted three times and the mission was short before abort. A later analysis at NASA figured out, that a wrong real-time behavior and the missing of the deadline of a flight critical task led to the problem and the 1202 program alarm of the AGC. Later on, a mathematical analysis of Liu and Layland, 1973 showed that the assumption a utilization of 80% on static real-time scheduling (rate monotonic scheduling, RMS) resulted in missing deadlines. Liu and Layland, 1973 showed that the utilization limit of a static real-time task set is dependent on the number of tasks, and in the limit on a large number of tasks is only 69%. However, while this limit is only necessary and not sufficient, it was necessary to develop further real-time tests. While Liu and Layland, 1973 considered the utilization of a task set in static and dynamic scheduling, other researchers followed an different approach, computing the response times of all tasks of a task set, as given by Joseph and Pandya, 1986. Since both, Liu and Layland, 1973 as well as Joseph and Pandya, 1986 assumed implied deadlines defined by the period of events, Leung and Whitehead, 1982 showed that deadline monotonic scheduling (DMS) was the optimal priority assignment when the deadline is smaller than the period and Lehoczky, 1990 introduced a schedulability test on given checkpoints to DMS.

This first work in real-time scheduling theory were limited to uni-processor systems. An extention to distributed systems gives Tindell and Clark, 1994 by introducing a jitter based periodic event model. Later on, the response time analysis was generalized by Richter, 2005 to integrate more complex event models. The response time analysis as given by Lehoczky, 1990 is limited to systems with static priorities. The extension for dynamically scheduled (earliest deadline first, EDF) real-time systems Palencia and Harbour, 2003 and Palencia and Harbour, 2005 needs to distinguish between different dynamic cases during analysis. This makes the approach complex. The real-time analysis distinguishes between load analysis (processor load) Liu and Layland, 1973 and response time analysis Lehoczky, 1990. Therefore, both directions are discussed.
independently in literature. The utilization based approach was extended and improved by [Baruah et al., 1993], who introduced shorter deadlines to the analysis to dynamic scheduling. However, this work supports only the periodic event model. A more general approach to model different and complex worst-case event patterns was first introduced by Gresser, 1993. This event stream model could be very easily combined with Baruah’s approach [Albers and Slomka, 2004]. Because the analysis algorithm has a bad run-time complexity some approximations are introduced by [Albers and Slomka, 2004] and [Albers and Slomka, 2005] for dynamic scheduling and by [Fisher and Baruah, 2005] for static scheduling. While the work of Gresser, 1993 does not model event bursts in an appropriate way, [Albers et al., 2006] introduce hierarchical event streams. Additionally, [Guan and Yi, 2014a] use Baruah’s utilization based scheduling test to design a novel response time analysis for dynamic scheduling. Other extension are the multiframe- [Mok and Chen, 1997], the generalized multiframe [Baruah et al., 1999] and the recurring real-time task model [Baruah, 2003]. These techniques allow the modeling of periodic task sequences with jobs with different execution times and extend real-time scheduling theory to the domain of stream processing systems [Baruah, 2010], [Moyo et al., 2010] and with the most powerful model of [Stigge et al., 2011].

In addition to these works, which can be assigned to the classical theory of real-time systems (scheduling theory), the real-time behavior of task systems can also be verified with the real-time calculus (RTC). The real-time calculus is based on the network calculus [Cruz, 1991a], [Cruz, 1991b], [Boudec, 1998] which describes a mathematical framework for analyzing the flow of data in networks. [Naedele et al., 1998] and [Thiele et al., 2000] introducing the real-time calculus and apply their work [Thiele et al., 2001], [Thiele et al., 2002] and [Chakraborty et al., 2003] to the analysis of network processors. It was shown that the classical methods can be replaced by the real time calculus. In contrast to prior work, the real-time calculus allows the calculation of systems with many different scheduling strategies as static- (DMS) and dynamic scheduling (EDF), time-division multiplex access (TDMA) and others. While the approach is modular it also allows hierarchical scheduling. Finally, by [Künzli et al., 2007] and [Richter, 2005] response time analysis as given by the classical theory were combined with real-time calculus to build an analysis that highlights the strengths of each technique. The disadvantage of this work is that the modelling is not generic and must be redefined for each system to be modelled.

However, the existing work is split in utilization based techniques, response time analysis and the real-time calculus. Each approach has its advantages and disadvantages. Sometimes authors like to combine the different work but often they are missing event bound functions with different properties as given by the established theory. The need for new approaches is given in [Stigge and Yi, 2013], [Biondi et al., 2014] and [Guan and Yi, 2014a]. Other authors prefer an analysis technique independent from the application structure [Künzli et al., 2007] and [Richter, 2005].
The goal of the presented work is to combine all different techniques in one single framework. Because the approach uses advanced techniques given by theoretical physics and signal theory it is more compact and expressive than previous work in the real-time domain. Because of its expressiveness it allows the formulation of a closed algebraic method which is open to different problems in real-time analysis. This leads to an easy formulation of utilization based and response-time based analysis in static as well as in dynamic scheduling. The approach allows an easy combination of both scheduling techniques without the overhead to formulate different equations and algorithms. It combines different event models and gives a new approach to the response time analysis of dynamic task systems. For the first time in literature we present an approach which allows the formulation of an explicit function to describe different schedulers.

3 Model of computation

Different computational models to analyze real-time systems exist. In this work, we consider the bounded execution time model. We are assuming that the execution flow in real-time systems separates into different tasks. A task is a kind of programming function assigned to an external or internal interrupt - an event - of the system. The tasks are periodically time- or event-controlled. Each event requests a task, and the concrete instance which occurs is called a job. Each job must be executed in a limited time interval: the deadline. In the bounded execution time model, tasks are preempted by higher priority tasks. The priority of the execution of a job can be assigned statically or dynamically. Bounding jobs of a task to a deadline allows any scheduling permutation without any sophisticated scheduling algorithm. In static scheduling, like rate monotonic/deadline monotonic (RMS/DMS) scheduling, the priorities are assigned statically to each task depending on the request rate of the triggering events. In dynamic scheduling, like the earliest deadline first (EDF) policy, the priority of each job depends on the next approaching deadline. Therefore, the scheduling priority is not strictly assigned to tasks. In the classical scheduling theory by Liu and Layland, 1973, the feasibility of a task set in the sense if deadlines met, is proved by computing the utilization of resources like processors or the maximal response time of any worst-case job.

3.1 Events

A timing relationship between events is needed to compute a task set’s utilization or the response time of the worst-case job or all other jobs as well. The established model defines a sequence of periodic events and the distance in time between events is denoted by a single value: the period \( p \in \mathbb{R}_0^+ \). Because each task has different periods, a function \( p(\tau) := p(\tau) \) may always return the period of the considered task. This event model has been extended to the sporadic event
model where the period interprets as minimal inter-task arrival time. The periodic event model with jitter allows considering distributed systems in holistic real-time analysis [Tindell and Clark, 1994] and [Tindell et al., 1994a]. This model was extended to include task offsets [W. Tindell, 1994] and arbitrary deadlines to the response time analysis [Tindell et al., 1994b]. However, a more general model on events was first introduced by [Gresser, 1993], has limitations to express bursty event patterns. Hierarchical event streams give a shorthand formulation to solve this problem. In this paper, we consider the periodic or sporadic event model and the event stream model in parallel. The periodic model in this work is used to give the reader a simple link to previous work, while the event stream model is more general and includes all derivatives like the sporadic, the bursty, or the periodic model with jitter.

**Definition 1 (Event stream)** An event stream is an array of event tuples or an event list:

\[ E = \left\{ (\rho_\epsilon, \phi_\epsilon) \right\} \]  

(1)

The event stream must be valid, which means the order of the time intervals \( \phi_\epsilon \) must be subadditive or superadditive. If the event list does not fulfil the requirement of subadditivity, we call it an event sequence.

An event tuple consists of the period \( \rho \) of an event and a minimal distance \( \phi \) to another event. The position of the event tuple in the stream array has a meaning: The first tuple initializes the stream. It always has \( \phi = 0 \). The second tuple describes the minimal distance between two events, the third between three events and so on. Therefore each tuple represents the minimal distance of the related number of events and its periodical repetition. In this work, each event tuple is indexed by \( \epsilon \). Therefore, \( \rho_\epsilon \) denotes the period of event \( \epsilon \) and \( \phi_\epsilon \) the minimal distance \( \phi \) between \( \epsilon \) events. Note, that in this model sporadic events can be described easily: an event which occurs only once has an infinite period.

**Example 1 (Event model: periodic)** Assume an event which occurs periodically every \( \rho \) time:

\[ E_{\text{periodic}} = \left\{ (\rho, 0) \right\} \]  

(2)

The minimal distance of the first initial event is \( \phi = 0 \). The event recurs with the period \( \rho \).

**Example 2 (Event model: periodic with jitter)** Assume a sequence of events which are not exactly periodic. If each event jitters around a given period, the worst-case behaviour is given by the following event pattern:

\[ E_{\text{jitter}} = \left\{ (\infty, 0), (\rho, \rho - 2j) \right\} \]  

(3)
The first tuple describes just the occurrence of one event. Therefore the minimal distance is set to 0. Note that this tuple is only needed to mark position one in the event stream. In the worst-case, an event of an event sequence has a maximal positive jitter (occurs at $j^+$ after 0) and the next following event has a maximal negative jitter - occurs at $j^-$ before $\rho$. Then the minimal distance between two events is $\phi = \rho - j^- - j^+ = \rho - 2j$, if $j^- = j^+$. From now, in the worst-case, each following event can only occur with a maximal negative jitter. Any other behaviour leads to a relaxed event sequence, and therefore, the considered case gives the densest occurrence of events. In the worst-case, the release of all other events is bounded by $\rho$.

However, if events occur bursty, the event stream model becomes complex. The reason is that each event in a burst has to be described explicitly. [Albers et al., 2006] give a more compact model: the hierarchical event stream:

**Definition 2 (Hierarchical event stream)** A hierarchical event stream is an array of event quadruple:

$$\mathcal{E} = \left\{ \left( p, n, \phi, \mathcal{E}_c \right) \right\}$$  \hspace{1cm} (4)

The first tuple in the quadruple is the same as defined originally for event streams. The second tuple additionally defines the hierarchical embedded event stream with a bound $n$. The bound $n$ defines how many events of a second event stream count from the embedded event stream. In this notation, a burst is described by embedding an event stream with a short period inside an event stream with a longer period. If the long or outer period is greater than the shorter or inner period multiplied by the bound, a non-overlapping burst occurs. In [Albers et al., 2006] and [Albers et al., 2008], different conditions and normalizations on hierarchical event streams are discussed. This result expresses the bursts described in [Tindell et al., 1994b] very compact. However, the formulation of a request bound function for bursty event streams is complex in both approaches.

3.2 Tasks

The inter-arrival pattern of events only describes the occurrence of events. At each event, an independent part of a program is executed by the operating system. Such an execution unit is called a task $\tau$. A real-time application separates into several tasks. Therefore each task is an element of a task set: $\Gamma := \{ \tau_1, \tau_2, \ldots, \tau_n \}$. All tasks must schedule on the given processor in a way that all deadlines met. A scheduler is optimal if no algorithm exists, which produces a better valid schedule. In [Liu and Layland, 1973] was proven that RMS is optimal for static, and EDF is optimal for dynamic scheduling. Therefore an execution time must be added to the model. Because the execution of a task's job varies and we are only interested in worst-case bounds [Liu and Layland, 1973]. In the real-time analysis, a task is defined by an inter-arrival pattern of events
and the two execution times. In the bounded execution model, the relative deadline specifies the time a task has to finish after being requested. If all tasks are independent, it is not necessary to consider the best case execution time. This parameter is only needed if tasks with data dependencies are running on different processors [Graham, 1976].

**Definition 3 (Execution time)** The execution time of a task is the time the execution of the task needs if a processor exclusively executes the task with no interruption by other tasks. The execution time may depend on data attributes given to the task. Therefore we distinguish between the worst-case or maximal ($c^+$, WCET) and best-case or minimal execution time ($c^-$, BCET).

As we consider the bounded execution model, a deadline must be assigned to each task. The deadline is a time interval in which the execution of a task must finish. It is distinguished between a relative deadline ($d$) and an absolute deadline ($D$).

**Definition 4 (Relative Deadline)** The relative deadline $d_\tau$ of a task bounds the execution of any job related to the request time $t_\tau$ of this job.

**Definition 5 (Absolute Deadline)** The absolute deadline $D_{\tau,e}^n$ of the $n$'th job is related to $t = 0$. Therefore the $n$'th absolute deadline of the job is

$$D_{\tau,e}^n = \phi_{\tau,e} + np_{\tau,e} + d_{\tau,e}$$  \hspace{1cm} (5)

During the execution of the task set, the operating system has to schedule jobs of the task set. The operating system determines the execution order of the jobs based on the relative or absolute deadline assigned to each job. In some cases, fixed priority numbers given by the programmer replacing deadline-based scheduling.

**Definition 6 (Static priority)** Let $\pi \in \mathbb{N}$ and assume two independent tasks $\tau$ and $\tau'$, a task $\tau'$ has a higher assigned priority than task $\tau$, if $\pi_{\tau'} > \pi_\tau$ and assume a task with higher priority preempts tasks with lower priority. The set of all higher priority tasks of task $\tau$ is

$$\tau^>_\tau := \{ \tau' \in \Gamma \mid \pi_{\tau'} > \pi_\tau \}$$  \hspace{1cm} (6)

Therefore, a task can be specified formally:

**Definition 7 (Task)** A task $\tau \in \Gamma$ is a quadruple including the inter-arrival pattern of events $\mathcal{E}$, the worst-case and best-case execution time of a task and a relative deadline $d$ by which the task execution bounds:

$$\tau := \{ \mathcal{E}, c^+, c^-, d, \pi \}$$  \hspace{1cm} (7)
Note, that the relative deadline can be replaced or amended by a static priority $\pi$. Access to the data structure of a task can be granted by task dependent functions: $p_\tau = p(\tau), c^+_\tau = c^+(\tau)$ etc.

In some work, to each job of a task different execution times assigned. In such a case, the execution times of a task specified by a vector. Job-related execution times are introduced by the multi-frame task model [Mok and Chen, 1997]. If job-related deadlines added, this is called the generalized multi-frame model [Baruah et al., 1999]. Therefore jobs must introduced in the task model:

Definition 8 (Job) A job is the instance of a task $\tau \in \tau$ triggered by any event of the event stream related to a task.

3.2.1 Problem formulation

Why the mathematical formulation of event requests is different in the demand bound test and the response-time analysis as shown in figure 1. The demand bound test has to check left points of the demand bound while the busy window approach looks for an intersection on the right side of the request bound. The initial value of the response time analysis gives the worst-case execution time of the considered task. In contrast to the demand bound test, the time point $t = 0$ does not matter because the minimal response time is always equal to the best- or worst-case execution time. As this time interval is the starting point of the fixed-point iteration, 0 never occur in the equation. However, to find the intersection with the resource function, the bound must be left-continuous. Therefore, at the end of the busy interval, an event should not count. If a task finishes its execution and at the same moment a new task requests, this request is superfluous.

1 More general: $f_{k,l} = f(k,l)$
It is obvious that the event bound $E_γ(t)$ is not equivalent to $E^*_γ(t)$, while $\left[ \frac{\alpha}{n} + 1 \right] = 1 \neq \left[ \frac{\alpha}{n} \right]$. The right-continuous event bound and the left-continuous event bound differ in all-time points $t_n = np_r$.

Is it necessary to use two different functions? The goal of this work is to find a function which is right-continuous in all $t_n = np_r$ except the last one which should be left-continuous. Therefore, the utilization test and the response time analysis should use the same function except at the end of the considered timing interval. Remember, the demand bound test evaluates all event requests until the hyper-period, and the response-time analysis counts all events until the result of the last iteration. If we extend the event bound function $E_γ : \mathbb{R}^2 \to \mathbb{R}$ we can specify a bound time interval $\Delta_{a,b} := [t_a, t_b) = [a, b)$ which restricts the time in which the event bound counts. If we integrate the hyper-period as bounding restriction to the demand bound function, it is possible to formulate a general unified event bound. Let us discuss this idea in more general:

**Problem 1 (Unified event bound function or unified event bound, ueb)** Investigate if a function which counts all events in the time interval $\Delta_{a,b} = [t_a, t_b)$ at each time point $\forall n \in \mathbb{N}_0 : t_n := np_r + t'$. Such a function is called the unified event bound.

Such a function is equivalent to the right-continuous event bound except at $t = t_b$. Note that for the response time analysis only this point in time is relevant and it is not necessary that all other points of the function are left-continuous. Therefore this function can be used for feasibility tests as well as for response time analysis. A function with these properties are postulated in [Stigge and Yi, 2013] and [Biondi et al., 2014]. Because in any related work no uniform bound is given, both papers accepted an over-approximation by using the right-continuous request bound.

**Problem 2 (Postulated demand bound test)** Assume the existence of a unified event bound function and a hyper-period $P = \text{lcm}_{\epsilon \in \mathcal{E}} p_{\epsilon}$ of the tasks periods. Then the demand bound test in the periodic event model can be written as:

\[
D_Γ(t, P) \leq t \quad (8)
\]

\[
\sum_{\tau \in \mathcal{E}} R_\tau(t - d_\tau, P) \leq t \quad (9)
\]

\[
\sum_{\tau \in \mathcal{E}} E_\tau(t - d_\tau, P) c_\tau^+ \leq t \quad (10)
\]

**Problem 3 (Postulated Response Time Analysis)** If a unified event bound function exists, the response time in the periodic event model can be written as:

\[
r_{\tau,n}^+ := c_\tau^+ + \sum_{\tau' \in \Gamma_\tau} E_\tau(r_{\tau,n-1}^+, r_{\tau,n-1}^+) c_{\tau'}^+ \quad (11)
\]

\[
c_\tau^+ + \sum_{\tau' \in \Gamma_\tau} E_\tau(t, t) c_{\tau'}^+ - t = 0 \quad (12)
\]
Assume the following definition for interfering tasks: $\tau' \in \tau \cup \Gamma_{\tau}$. Then the response time analysis can be reformulated as the well known fixed-point iteration. In other words, the response time analysis will become a root-finding problem:

$$\sum_{\tau' \in \tau \cup \Gamma_{\tau}} E_{\tau}(t, t) c_{\tau'}^+ - t = 0 \quad (13)$$

The paper is organized as follows: First, we derive a unified event bound function using methods from calculus and distribution theory. Second, we will show how hierarchical event streams can be easily described and computed by using the Dirac delta function. Based on this idea, we develop a unified real-time scheduling theory considering static and dynamic priorities in one holistic approach for feasibility and response time analysis as well. For the first time, we derive both analysis techniques from only one axiom, the average load of a processor. As a special treat, we can develop a tighter response time analysis as given in related work for dynamic scheduling at the end by just adding the same assumption to dynamic scheduling as already done to model task with arbitrary deadlines already done in static scheduling. A running example of a full utilized task set illustrates each idea if necessary. In the end, we will compute the response times of some interesting tasks set in static, dynamic and hierarchical scheduling. The paper ends with an appendix concluding the used mathematical symbols and explaining special notations from theoretical physics.

3.3 The unified event bound function

During the next section, we develop a strict formal view to events as known in signal theory. The idea is to express all needed mathematical properties in the model implicitly without any informal or hidden assumptions. First, we introduce events, and then we show how they can be count in an alternative way compared to the floor and ceil operation. We discuss the mathematical properties and will show how the new method is related to previous work.

3.4 A mathematical view on events and tasks

In real-time systems analysis or scheduling theory, events and jobs introduced semi-formal. Tasks or better jobs were often given as geometrical objects such as rectangles in Gantt charts. Then the length of the rectangle models execution demand of the job and the place of the rectangle determines by its position in time. The height of the rectangles does not matter and is most often given to 1 as seen in figure 2a. The goal of the following section is to formalize release times and time durations appreciatively. The goal is to transform informal geometric proofs to analytical descriptions which are computed algebraically.

\footnote{In this work hierarchical scheduling means a mixed scheduling policy where any dynamic or static scheduler may embed a scheduler of any lower hierarchy. Therefore we follow the definition of \cite{Wandeler2006} or \cite{Lipari2005}}
3.4.1 Modeling jobs

In each computer system, a computational activity has a duration or in other words, an execution time. The time between the release of a job and its non-preempted execution end starts at a defined point in time $t_a$ and ends later at a second point in time $t_b$. If the job is not interrupted by any other activity this time is called the worst-case execution time $c^+$. However, if we assume independent tasks on a unique processor, we can concentrate on $c^+$. Calling $t_a$ the request time, each job of a task ends after $c^+$ if no other job interrupts the execution. Therefore the job finishes at $t_b = t_a + c^+$. Figure 2a. shows such a simple behaviour as it is described in most of the previous work by a Gantt-Chart. Therefore, during the execution of a job, the processor is busy and has a utilization of one. In contrast to related work, we first look for an algebraic formulation of this behaviour. Formally the geometric Gantt-Chart description of a job can be replaced by a composition of Heaviside functions.

**Definition 9 (Heaviside function)** Assume $x \in [0, 1]$. The Heaviside or step function $H: \mathbb{R} \to \{0, 1\}$ is defined as

$$H(t) = \begin{cases} 0 & t < 0 \\ x & t = 0 \\ 1 & t > 0 \end{cases} \quad (14)$$

Note that different definitions of the Heaviside function exist. The above definition supports the requirements needed in this work best. In some cases $x \in \emptyset$. However, this is not important in this paper.
Based on this definition, it is easy to introduce the concept of the Dirac delta function or shortly the delta function, which becomes our base to define events formally:

**Definition 10 (Dirac delta)** The Dirac delta function is given by:

\[
H(t - s) = \int_{-\infty}^{t} \delta(t' - s) \, dt'
\]  

(15)

This equation does not define a function in a traditional, well-known way. Therefore it is also called a distribution. It was first introduced by Paul Dirac in the early 1930s and is a well-established mathematical tool in theoretical physics and signal theory [Bracewell, 2000]. As we will see later, the idea of Paul Dirac can be applied to find and define the unified event bound. It is very important to have in mind that \(\delta(t) = 0\) for all \(t \neq 0\) which directly follows from the definition.

Let us next consider how any job of task \(\tau\) with execution time \(c + \tau\) requested at time \(t_r\) can be modeled. Let us first assume that all jobs have the same execution demand. Therefore we call the task homogenous.

**Lemma 1 (Dirac job)** A job requested at time \(t_r\) is described by

\[
c^+_{\tau}(t, t_a) = \int_{-\infty}^{t} \delta(t' - t_r) \cdot c^+_{\tau} \, dt'
\]  

(16)

**Proof** A non-preemptive real-time task instance or job needs two Heaviside functions for its algebraic description: one to represent the request \(H(t - t_a)\) and one to model the completion of the task \(H(t_b - t)\). Consider figure 2b. Multiplying both functions builds a rectangle of height one defined by the execution function \(c : \mathbb{R} \rightarrow \{0, 1\}\):

\[
c_{\tau}(t, t_a) = H(t - t_a) \cdot H(t_b - t) = H(t - t_a) \cdot H(t_a + c + \tau - t)
\]  

(17)

Alternatively it is possible to use

\[
c_{\tau}(t, t_a) = H(t - t_a) - H(t - t_b) = H(t - t_a) - H(t - [t_a + c])
\]  

(18)

It is important to note that such a description does not consider preemption, and therefore, it does not support the bounded execution model completely. As a consequence, it is necessary to model the behaviour of interfering computational loads such as interrupts and higher priority jobs explicitly. The idea of the following is to describe the occurrence frequency of jobs and their requested load concerning the available computation time in a given time interval \([t_a, t]\). Let us first rewrite the equation for the computational load without changing anything\(^4\):

\[
c_{\tau}(t, t_a) = \int_{-\infty}^{t} H(t' - t_a) \cdot H(t_a + c^+_{\tau} - t') \, dt'
\]  

(19)

\(^4\) The integral looks a little bit oversized because both Heaviside functions return only 1. However, introducing the integral is crucial as we will see later.
The complete non-preempted execution starting at \( t_a \) is given by
\[
c^+_a(t_a) = \lim_{t \to \infty} \int_{-\infty}^{t} H(t' - t_a) \cdot H(t_a + c^+_a - t') \, dt' = \int_{-\infty}^{\infty} H(t' - t_a) \cdot H(t_a + c^+_a - t') \, dt'
\] (20)

The release of each job needs a context switch at the beginning and end of execution. This context switch can be modelled by some time, \( \varepsilon \). If we assume that this time should not be added to the execution of the job as assumed in real-time scheduling theory, we can write
\[
c^+_a(t_a) = \int_{-\infty}^{\infty} \frac{1}{2\varepsilon} H(t' - (t_a + \varepsilon)) \cdot H((t_a + \varepsilon) + c^+_a - t') \, dt'
\] (21)

For our first event we are not interested when it starts so let us move it to the origin \( t_a = 0 \):
\[
c^+_a(0) = \int_{-\infty}^{\infty} \frac{1}{2\varepsilon} H(t' - \varepsilon) \cdot H(\varepsilon + c^+_a - t') \, dt'
\] (22)

Let us now modify equation (22) by describing the computational load not by its horizontal time:
\[
c^+_a(0) = \int_{-\infty}^{\infty} \frac{c^+_a}{2\varepsilon} \cdot H(t' - \varepsilon) \cdot H(\varepsilon - t') \, dt'
\] (23)

In real-time analysis, we are interested only to the load given by the real-time tasks itself. Such an assumption is permissible because the execution time of a job is much longer than the interruption time by the operating system, and then it should be ignored. Therefore we look what happens if the operating systems overhead approaches to 0. Mathematically the request span can be eliminated under the assumption that
\[
\int_{-\infty}^{\infty} \frac{1}{2\varepsilon} H(t - \varepsilon) \cdot H(\varepsilon - t) \, dt = 1
\]
The obvious solution eliminates the time \( 2\varepsilon \) by setting \( \varepsilon = 0 \) does not work in general. If we now assume a Heaviside function with \( x = 0 \) then \( H(t - \varepsilon)H(\varepsilon - t) = 0 \) and not 1. Therefore, we set \( \varepsilon \) in a way, that all possible Heaviside functions \( x \in [0, 1] \) will be supported as well. Mathematically we apply limit value analysis to the problem: If the term addressed by the integral is divided by \( 2\varepsilon \) and \( \varepsilon \to 0 \) we get
\[
\lim_{\varepsilon \to 0} \int_{-\infty}^{\infty} \frac{1}{2\varepsilon} \cdot H(t - \varepsilon) \cdot H(\varepsilon - t) \, dt = \int_{-\infty}^{\infty} \delta(0) \, dt' = 1
\] (24)

\* If scheduling overhead should be modelled assume a separate task describing the overhead of the operating system.
Assume the substitution \( \delta(0) = \lim_{\varepsilon \to 0} \frac{1}{2\varepsilon} \cdot \mathbb{H}(t - \varepsilon) \cdot \mathbb{H}(\varepsilon - t) \). Consider figure 2c. for illustration. Therefore,

\[
c^+_c(0) = \int_{-\infty}^{\infty} \delta(0) \cdot c^+_c \, dt'
\]

And if we like to consider a job requested at time \( t_r \):

\[
c^+_c(t_r) = \int_{-\infty}^{\infty} \delta(t' - t_r) \cdot c^+_c \, dt'
\]

3.4.2 Events as Dirac delta

Let us now apply the delta function by defining events as needed in real-time systems analysis in a strictly formal way:

**Definition 11 (Event)** An event \( \epsilon : \mathbb{R} \to [0, 1] \) is a request at a point in time \( t_\epsilon \in \mathbb{R} \) with infinitely thin time span:

\[
\epsilon(t_\epsilon) = \begin{cases} 
\int_{-\infty}^{t} \delta(t' - t_\epsilon) \, dt' = 1 & t = t_\epsilon \\
\delta(t - t_\epsilon) = 0 & t \neq t_\epsilon 
\end{cases}
\]

The time point \( t_\epsilon \) calls the request time of the event.

In other words, an event is a timeless state change in any system.

Computing only the area bounded by a given Heaviside function does not allow to consider preemption as needed in the bounded execution model. Multiplying a Dirac delta with any given WCET results in a peak with the amplitude of the execution time at the request time of the event, as shown in figure 3a. Running overtime \( t \) the value of these peaks is reduced exactly by \( t \) in the interval \( t \). Because the model considers the release time of events, we can add a peak of execution time at any time an interfering job of higher priority interrupts the execution of the considered job. Consider figure 3b which illustrates the idea. At time \( t = 0 \) task \( \tau_1 \) and \( \tau_2 \) are requested. After the specified period \( p_1 \) task \( \tau_1 \) is requested again. Figure 3b. shows the behaviour of the resulting function. Note, that such a saw-function is equal to the well-known request bound function of these two tasks subtracting \( t \). Changing the point of view transforms the established fixed-point iteration of the busy window approach to find the roots of the equivalent sawtooth-wave.
3.4.3 Event models

The definition of only one event does not support the modelling of tasks as a sequence of jobs. Therefore a formal description of a series or sequence of events is required to model sequential jobs. Mathematically this is expressed by a series of Dirac deltas called a Dirac comb in the case all events are strictly periodic. However, the general way to describe any sequence of requesting events is to describe event streams. An event stream can be described by a Dirac comb as well:

**Definition 12 (Event density)** A sequence of \( k \) events is given by:

\[
\mathbf{\lambda}_\tau(k, t) = \sum_{\epsilon \in \mathcal{E}} \delta(t - \phi_\epsilon - np_\epsilon)\]

(28)

calling \( \mathbf{\lambda}_\tau(k, t) \) an event stream or event density\(^6\) Therefore an event sequence\(^7\) specifying \( k \) events can be written with the event tuple:

\[
\epsilon = \langle \rho, \phi \rangle_k \quad (29)
\]

Moreover, as a short form notation a set of corresponding event tuples defines the event density formally:

\[
\mathcal{E} = \{ \langle \rho, \phi \rangle_k \} \quad (30)
\]

We choose the notation \( \langle a, b \rangle \) to distinguish the new approach clearly from the event stream notation.

This definition introduces a new perspective and insight into event streams. A mathematical equation now describes an event stream with precisely defined mathematical properties instead of only writing a weak set of tuples. To describe

\(^6\) This work introduces the term 'event density'. As we will see later, this is a more intuitive term than the name event stream as used in previous work.

\(^7\) We distinguish between the event density as a sum of Dirac deltas and the event tuple describing the parameters of the event density.
event densities which model valid event streams, we assume a maximal event density \( \Xi^+ \) and a minimal event density \( \Xi^- \). Both are event densities which have the mathematical property of sub- or super-additivity. Additionally, it is very easy to bound the number of events. Instead of previous models, the term event density allows specifying a fixed number of events as a sporadic or bursty event stream.

**Example 3 (Periodic event model)** The periodic event model describes an infinite number of periodic events. Assume \( \phi_e = 0 \), therefore \( k = \infty \) and the sequence of events is given by

\[
\Xi_r(\infty, t) = \sum_{n=0}^{\infty} \delta(t - np_r) = \sum_{n\in\mathbb{N}_0} \delta(t - np_r)
\]  

(31)

Assume a sporadic event which occurs only once. In the event stream model, the definition of the event bound requires to set the period of the given event tuple of the sporadic event to \( \infty \). Now the period is zero if an event occurs only once and the sum limits the occurrence:

**Example 4 (Sporadic event)** A event which is sporadic and which occurs only at \( t_e = 10 \text{ ms} \) is described by

\[
\mathcal{E} = \{ <0, 10 > \text{ ms} \} = \{ <0, 10 > \text{ ms} \} \text{ in contrast to } \mathcal{E} = \{ \infty \text{ ms} \}
\]  

(32)

as originally defined by [Gresser, 1993].

**Example 5 (Periodic event model with jitter)** First assume the established periodic event model with jitter:

\[
\mathcal{E}_{jitter} = \{ <0, 0 >, <p, p - 2j > \infty \} = \{ <0, 0 >, <p, p - 2j > \}
\]  

(33)

Now consider we only like to describe four events in this model:

\[
\mathcal{E}_{jitter} = \{ <0, 0 >, <p, p - 2j >_3 \}
\]  

(34)

Such a description is natural, easier to understand, and more potent than the original form.

We use a computer algebra system (CAS) [Cohen, 2003] to validate the approach. The CAS allows us to verify the algebraic structure of the work. Additionally, it is possible to consider numeric examples as well. Therefore a sample numeric task set is defined:

**Example 6 (Example task set)** Table 1 gives a task set used as a running example in the rest of the paper. Just for simplification, we only consider periodic tasks. Therefore we specify three tasks by their period, their worst-case execution time and the relative deadline which is given by the deadline as well. Additionally, the last column of the table states the input for the computer algebra system, as mentioned...
Table 1 Example task set

| Task | Period $p$ [t.u.] | Wcet $c$ [t.u.] | Relative deadline $d$ [t.u.] | CAS input |
|------|------------------|----------------|-------------------------|-----------|
| $\tau_1$ | 8                | 2              | 8                       | $\{\{\{0,8\},\infty\}\},2\}$ |
| $\tau_2$ | 16               | 4              | 16                      | $\{\{0,16\},\infty\},4\}$ |
| $\tau_3$ | 24               | 12             | 24                      | $\{\{0,24\},\infty\},12\}$ |

earlier. Note that this task set has a utilization equal to one. Therefore it is schedu-
able by dynamic scheduling and not by static scheduling as shown later in figure 12. Considering a utilization of one is essential to investigate the differences in static and dynamic scheduling and the tightness of a response-time analysis as seen later.

First we defined a function $EventDensity$ following definition 12 to build the algebraic equation from a given nested list as task description:

$$
\sum_{\text{IoE}=0}^{\infty} \text{DiracDelta}\left[-8 \text{IoE} + \text{td}\right] +
\sum_{\text{IoE}=0}^{\infty} \text{DiracDelta}\left[-16 \text{IoE} + \text{td}\right] +
\sum_{\text{IoE}=0}^{\infty} \text{DiracDelta}\left[-24 \text{IoE} + \text{td}\right]
$$

The variable $\text{IoE}$ (Instance of Event) denotes the number of the considered job and the function $EventDensity$ computes the Dirac comp as discussed earlier. Replacing it by $n$ or any other counting variable leads to the formal notation given earlier. Note that the sequence of variables given in the CAS output follows the rules defined in computer algebra. Therefore we do not change outputs of the CAS to be compatible with the equations defined.

3.4.4 To count or not to count

After defining the event density, we have to consider how to count the events. As we have seen in lemma 1, the execution time of a job is computed by integrating a couple of Dirac deltas. Therefore, we will find the number of events by integrating over a series of Dirac deltas or events which we called the event density. However, the integral gives us the freedom two mask given time intervals from event densities. As we will see, this is a significant advantage compared to the counting of events in related work.

3.4.5 Counting by integrating dirac deltas

To compute the execution demand of a processor, events and a series of events must be counted during a given time. This number of events then is multiplied
by the specified execution time of the task. By changing the event model to Dirac deltas, we have to count the number of deltas in a given timing interval. Integrating the series of Dirac deltas results in the number of events given in the time bounded by the limits of the integral:

**Lemma 2 (Finite event bound)** Assume any time interval \( \Delta_{a,b} := [t_a, t_b] \in \mathbb{R} \). The number of events then can be counted by a function \( E : \Gamma \times \mathbb{R} \times \mathbb{N} \rightarrow \mathbb{N} \)

\[
E_{r}(\Delta_{a,b})_k = \sum_{\epsilon \in E} \sum_{n=0}^{k-1} \delta(t' - \phi_\epsilon - np_\epsilon) dt' = \int_{t_a}^{t_b} \text{III}_r(k, t') dt'
\]

(35)

Also, in the particular case of the periodic event model with countless events, this simplifies to:

\[
E_{r}(\Delta_{a,b}) = \sum_{t \in \mathbb{N}} \delta(t' - np_r) dt'
\]

(36)

**Proof** We have to sum all events of an event density:

\[
\sum_{\epsilon \in E} \epsilon(t_s)
\]

(37)

According to the definition of an event \( \epsilon(t_s) = \int_{-\infty}^{\infty} \delta(t' - t_s) dt' \), we get for any event density in \( \Delta_{a,b} \)

\[
E_{r}(\Delta_{a,b})_k = \sum_{\epsilon \in E} \sum_{n=0}^{k-1} \delta(t' - \phi_\epsilon - np_\epsilon) dt'
\]

(38)

By definition the number of event tuples is limited and the series given by equation 38 converges absolute for \( k \in [0, \infty) \), because of the bound \( \Delta_{a,b} \) and \( n \in \mathbb{N} \). In the case of a finite \( k \) convergence of the sum is trivial. Therefore, the integral and the sum can be switched:

\[
E_{r}(\Delta_{a,b})_k = \int_{t_a}^{t_b} \sum_{\epsilon \in E} \sum_{n=0}^{k-1} \delta(t' - \phi_\epsilon - np_\epsilon) dt'
\]

(39)

Note, that the first assumption does not hold if \( \Delta_{a,b} \in (-\infty, \infty) \). However, as we will see later, the sum or event density is always limited.

By definition of the unified event bound, we do not want to count events at \( t_b \). However, by definition in calculus, the Riemann integral is bounded by the interval \([t_a, t_b]\) and this is not the postulated interval \( \Delta_{a,b} \in [t_a, t_b] \). Transforming the limits of the integral in a right-open interval can be done by masking the desired interval with the help of Heaviside functions. In this case, two variants of the infinite set of Heaviside functions as defined in definition 9 are needed:
Definition 13 (Upper Heaviside function) Assume the general Heaviside function with \( x \in [0, 1] \) and let \( x = 1 \). Then the upper Heaviside function \( \mathbb{H} : \mathbb{R} \to \{0, 1\} \) is

\[
\mathbb{H}(t) = \begin{cases} 
0 & t < 0 \\
1 & t \geq 0 
\end{cases}
\]  
(40)

Definition 14 (Lower Heaviside function) Assume the general Heaviside function with \( x \in [0, 1] \) and let \( x = 0 \). Then the lower Heaviside function \( \mathbb{H} : \mathbb{R} \to \{0, 1\} \) is

\[
\mathbb{H}(t) = \begin{cases} 
0 & t \leq 0 \\
1 & t > 0 
\end{cases}
\]  
(41)

By applying definition \(13\) and definition \(14\) to lemma \(2\) we can formulate the unified event bound as illustrated in figure \(4\):

**Theorem 1 (Unified event bound function, \(\text{ueb}\))** Assume \( E : \Gamma \times \mathbb{R}^2 \times \mathbb{N} \to \mathbb{N} \), then the number of events in any bounded interval \( \Delta_{a,b} = [t_a, t_b) \) can be counted by

\[
E_\tau(t, \Delta_{a,b})_k = \int_{t_a}^{t_b} \sum_{\epsilon \in E_\tau} \delta(t' - \phi_\epsilon - np_\epsilon) \cdot \mathbb{H}(t' - t_a) \cdot \mathbb{H}(t_b - t') \, dt'
\]  
(42)

in the special case of a periodic event model this becomes

\[
E_\tau(t, \Delta_{a,b}) = \int_{-\infty}^{t} \sum_{n \in \mathbb{N}_0} \delta(t' - np_\tau) \cdot \mathbb{H}(t' - t_a) \cdot \mathbb{H}(t_b - t') \, dt'
\]  
(44)

**Proof** Assume we count the events in a bounded interval \([t_a, t_b]\). Instead of the infinite interval given in equation \(43\) we bound the integral by its limits:

\[
E_\tau([t_a, t_b])_k = \int_{t_a}^{t_b} \sum_{\epsilon \in E_\tau, n=0}^{k-1} \delta(t' - \phi_\epsilon - np_\epsilon) \, dt'
\]  
(46)

The limits of the integration include \(t_a\) and \(t_b\) by definition. While \( \mathbb{H}(t' - t_a) \cdot \mathbb{H}(t_b - t') = 1 \) only if \( t_a \leq t \leq t_b \) and 0 in all other cases, equation \(46\) can be rewritten
as:

\[
E_\tau([t_a, t_b])_k = \int_{t_a}^{t_b} \sum_{\epsilon \in E} \sum_{n=0}^{k-1} \delta(t' - \phi_\epsilon - np_\epsilon) \, dt'
\]

(47)

\[
= \int_{-\infty}^{\infty} \sum_{\epsilon \in E} \sum_{n=0}^{k-1} \delta(t' - \phi_\epsilon - np_\epsilon) \cdot H(t' - t_a) \cdot H(t_b - t') \, dt'
\]

(48)

Changing the term \( H(t_b - t') \) to \( H(t_b - t) \) excludes \( t_b \) from the bound. Therefore, the integration over \( \Delta_{a,b} \in [t_a, t_b] \) can be formulated as

\[
E_\tau([t_a, t_b])_k = \int_{-\infty}^{\infty} \sum_{\epsilon \in E} \sum_{n=0}^{k-1} \delta(t' - \phi_\epsilon - np_\epsilon) \cdot H(t' - t_a) \cdot H(t_b - t') \, dt'
\]

(49)

We observe that this integral is not only bounded by \( \Delta_{a,b} \). Assume \( t_a \leq t < t_b \), then this function is also bounded by \( t \), and therefore we can write

\[
E_\tau([t_a, t_b])_k = \int_{t_a}^{t} \sum_{\epsilon \in E} \sum_{n=0}^{k-1} \delta(t' - \phi_\epsilon - np_\epsilon) \cdot H(t' - t_a) \cdot H(t_b - t') \, dt'
\]

(50)

This equation shows one of the most critical limitations of the established scheduling theory: In contrast to previous work, the unified event bound allows computing the number of events in any time interval. The computation of the bound can be moved to any time point \( t \in \mathbb{R} \). Therefore, the unified event bound is invariant in time. However, Theorem \( \ref{thm:heaviside-bound} \) has additional properties useful in real-time scheduling analysis: Defining the event bound by integrating over Dirac pulses and limiting this integral by two different Heaviside functions, the upper and lower Heaviside function, we find several and different bounds if we combine different descriptions for integral limits. Therefore, the number of events bounds by four different cases:

\[\text{Case } 1: \int_{a}^{t} \text{ and } \int_{t}^{b}\]

\[\text{Case } 2: \int_{a}^{t} \text{ and } \int_{t}^{b}\]

\[\text{Case } 3: \int_{a}^{t} \text{ and } \int_{a}^{b}\]

\[\text{Case } 4: \int_{a}^{t} \text{ and } \int_{a}^{b}\]

The proof for the periodic or sporadic model is obvious.

\[\square\]

In real-time scheduling theory, the starting time of the analysis interval is implicitly set to \( t_a = 0 \) by definition. Therefore, \( \Delta_{0,b} = [0, t_b] \). Then the unified event bound function can be written as

\[
E_\tau(t, t_b)_k = \int_{0}^{t} \sum_{\epsilon \in E} \sum_{n=0}^{k-1} \delta(t' - \phi_\epsilon - np_\epsilon) \cdot H(t' - t_a) \cdot H(t_b - t') \, dt'
\]

(52)

This equation shows one of the most critical limitations of the established scheduling theory: In contrast to previous work, the unified event bound allows computing the number of events in any time interval. The computation of the bound can be moved to any time point \( t \in \mathbb{R} \). Therefore, the unified event bound is invariant in time. However, Theorem \( \ref{thm:heaviside-bound} \) has additional properties useful in real-time scheduling analysis: Defining the event bound by integrating over Dirac pulses and limiting this integral by two different Heaviside functions, the upper and lower Heaviside function, we find several and different bounds if we combine different descriptions for integral limits. Therefore, the number of events bounds by four different cases:

The integral \( \int_{a}^{t} dt' \) is defined on the interval \( [a, t] \). Therefore the above simplification holds.
Fig. 4  To count or not to count

i  The bound of the number of events $k$.
ii  The lower timing bound $t_a$ defined by the lower Heaviside mask or, the lower limit of the integral.
iii  The timing bound $t$ given by the limitation of the Dirac comb or the upper limit of the integral.
iv  The above timing bound $t_b$ as defined by the upper Heaviside mask.

This first result shows that previous work defined different event bound functions because not considering the limits of time intervals like in calculus. We have proven that a unified function has to consider the limits of a well-defined integration problem as we can see in figure 4. Additionally, we know from distribution theory, that the Dirac delta is the derive of the Heaviside function [Bracewell, 2000]. As a consequence, it is obvious to call an event stream an event density: The event count in the real-time analysis is an integral over a dense series of Dirac deltas.

**Definition 15 (Heaviside mask)** The pair of Heaviside functions limits the integration interval by masking bounds:

$$M(t, \Delta_{a,b}) := H(t - t_a) \cdot H(t_b - t)$$  (53)

With $H(t - t_a)$ the left or early mask and $H(t_b - t)$ the right or late mask. Note, that both Heaviside functions can be upper or lower Heaviside functions. Therefore four different masks exist: $M(t, \Delta^E_{a,b})$, $M(t, \Delta^L_{a,b})$, $M(t, \Delta^E_{2})$ and $M(t, \Delta^L_{2})$.

**Example 7 (Example task set)** Let us consider the task set given in example 6. Assume we defined a function `DiracCount` to count events. Then the CAS gives the following output if we like to count the events in the interval $[0, T)$:
4 Unified analysis of real-time systems

In this section, we consider how the unified event bound can be used to solve real-time analysis problems. Furthermore, it is possible to model additional conditions on task scheduling without modification of the structure of our analysis equation. Therefore it is easy to derive variants to model bursty or hierarchical event patterns or hierarchical schedulers.

4.1 A general event model: The event spectrum

Simple event models become complex in bursty events. Different solutions address this problem [Tindell et al., 1994b] and [Albers et al., 2008]. However, both

\[ \int_{-\infty}^{\infty} \text{HeavisideDown}[T - td] \times \text{HeavisideUp}[td] \sum_{i=1}^{n} \text{DiracDelta}[-24 \text{IoE} + td] \, dt + \]

\[ \int_{-\infty}^{\infty} \text{HeavisideDown}[T - td] \times \text{HeavisideUp}[td] \sum_{i=1}^{n} \text{DiracDelta}[-16 \text{IoE} + td] \, dt + \]

\[ \int_{-\infty}^{\infty} \text{HeavisideDown}[T - td] \times \text{HeavisideUp}[td] \sum_{i=1}^{n} \text{DiracDelta}[-8 \text{IoE} + td] \, dt \]

Such schedulers are called hierarchical in real-time calculus. However, [Liu and Layland, 1973] call it mixed schedulers. Therefore term also differs from [Baruah et al., 2011], [Zhu et al., 2011] and [Ittershagen et al., 2013]. In this paper, we mention a scheduler which schedules all jobs with the same priority according to their dynamic deadlines.

---

Table 2 Relations between traditional [Liu and Layland, 1973], [Baruah et al., 1991] and unified theory in the interval \([0, t]\)
approaches are not intuitive and require different models to describe the synchronization of events. The two papers solve the problem in different ways but lack to give mathematical or formal approaches to their solutions. Applying now the mathematical toolset developed in section 3.3, a hierarchical event stream as described by [Albers et al., 2008] can be derived mathematically. Assume two independent event densities: One event density with a small period and a second one with a much larger one. Both densities together form a new bursty event stream if they are synchronized. The convolution of Dirac combs computes the composition of two event densities. Therefore, synchronization in real-time scheduling can be modelled by

**Theorem 2 (Hierarchical event density composition)** Any hierarchical event stream is a composition of two flat event densities and can be computed by the convolution of the two event densities:

\[
\Pi^{k,l}_{\xi_1,\xi_2} = \Pi^{k}_{\xi_1} \ast \Pi^{l}_{\xi_2}
\]

Proof To make the proof easy to follow, we assume \( \delta(\tau - t) = \sum_{\epsilon \in \mathcal{E}} \sum_{n=0}^{k-1} \delta(t - \phi_\epsilon - np_\epsilon) \) and \( t_n = \phi_{\epsilon_1} + np_{\epsilon_1} \) and \( t_m = \phi_{\epsilon_2} + mp_{\epsilon_2} \):

\[
\Pi^{k}_{\xi_1} \ast \Pi^{l}_{\xi_2} = \int_{-\infty}^{\infty} \Pi^{k}_{\xi_1} \cdot \Pi^{l}_{\xi_2} \, d\tau
\]

\[
= \int_{-\infty}^{\infty} \delta(\tau - t_n)_{\epsilon_1} \cdot \delta(t - [\tau + t_m])_{\epsilon_2} \, d\tau
\]

Substitute \( \xi = \tau - t_n \) and \( d\tau = d\xi \):

\[
\Pi^{k}_{\xi_1} \ast \Pi^{l}_{\xi_2} = \int_{-\infty}^{\infty} \delta(\xi)_{\epsilon_1} \cdot \delta(t - [\xi + t_n + t_m])_{\epsilon_2} \, d\xi
\]

\[
= \int_{-\infty}^{\infty} \delta(0)_{\epsilon_1} \, d\xi = 1
\]

For \( \xi \neq 0 \), the trivial solution is \( \Pi^{k}_{\xi_1} \ast \Pi^{l}_{\xi_2} = 0 \). The only nontrivial solution of the last equation is for \( \xi = 0 \). We get \( \int_{-\infty}^{\infty} \delta(0)_{\epsilon_1} \, d\xi = 1 \) and therefore

\[
\int_{-\infty}^{\infty} \delta(0)_{\epsilon_2} \, d\xi \cdot \delta(t - [t_n + t_m])_{\epsilon_2} = \delta(t - [t_n + t_m])_{\xi}
\]
Now let us resubstitute $t_n$ and $t_m$:

$$\begin{align*}
\mathcal{III}^n_{t,k} \ast \mathcal{III}^m_{t,l} &= \sum_{n=0}^{k-1} \sum_{m=0}^{l-1} \delta(t - [\phi_{\epsilon_1} + n\phi_{\epsilon_2} + \phi_{\epsilon_2}]) \\
&= \sum_{n=1}^{k-1} \sum_{m=1}^{l-1} \delta(t - [\phi_{\epsilon_1} + \phi_{\epsilon_2} + n\phi_{\epsilon_1}])
\end{align*}$$

(61)

(62)

In other words, the product of the Dirac delta becomes zero, exactly if $\tau - t_n = 0$ and if $\tau + t_m = t$. Therefore the theorem holds.

$\square$

**Definition 16 (Event spectrum)** As a result from the previous theorem the following 3-tuple describes hierarchical and synchronized event densities:

$$\epsilon = \langle \phi_{\epsilon_0} + \phi_{\epsilon_1}, \rho_{\epsilon_0}, \rho_{\epsilon_1} \rangle_{k,l}$$

(63)

with the hierarchical event density or event spectrum

$$\mathcal{III}^{k,l}_{\epsilon_0,\epsilon_1} = \sum_{n=1}^{k-1} \sum_{m=1}^{l-1} \delta(t - [\phi_{\epsilon_0} + \phi_{\epsilon_1} + n\phi_{\epsilon_0}])$$

(64)

The event spectrum is the most general form of an event model. An event spectrum can express all other known event models. According to lemma[1] the event bound is calculated only by integrating the event spectrum density. Additionally, theorem[2] gives us the possibility to compute composite event models during analysis. To best of our knowledge, no previous work in any known real-time analysis technique covers this aspect.

4.2 Task model: the request bound

The previous presented mathematical framework allows the formulation of advanced analysis techniques. Next, we discuss how to integrate the generalized multi-frame model and how easily interfering request bounds can be constructed to describe different scheduling policies.

4.2.1 Generalized request bound

The new approach to describe events with Dirac deltas is compelling: The advantage compared to established techniques is that the Dirac comb of definition[12] addresses each event separately, and therefore, each event may have different properties. As a result, the model allows assigning different execution times to different events without any additional effort. In the established analysis, the request bound is given by a multiplication of the event bound and the worst-case execution time of the task. However, because it is easy to address each event separately by the unified event bound, the multiframe-[Mok and Chen, 1997] and
the generalized multiframe model [Baruah et al., 1999] integrates easily into the new approach. Formulating the event- and the request bound unified allows addressing each job with separate execution time. Therefore, it is possible to model task sets with complex execution time behaviour. However, often it is not necessary to assign an own execution time to each job. In this case, the execution time vector contains fewer elements as events occur by a task. Then the execution time can be addressed by restricted access to the given vector: The length of the vector then bounds the access as it could be described by \( n \mod |C_{\tau,\epsilon}| \) as also given in the multiframe model:

**Definition 17 (Execution time vector)** The execution time vector introduced by [Mok and Chen, 1997] of \( k \) different execution times of a task is given by

\[
C_{\tau,\epsilon}[n] = C^n_{\tau,\epsilon} = [c^n_1, \ldots, c^n_k]
\]  

10 A detailed description gives the appendix A.

Note the style of the notation: The idea is to address each component of the vector by \( n \). If we like to address each event separately, it is not possible anymore to use the notation given in related work by defining request and demand bound functions. Addressing different events and jobs in one equation require to write an integral and two sum symbols every time. Therefore, it is necessary to introduce a short-form notation to simplify the writing and reading of event- and request bounds. Based on Einstein’s well-known shorthand notation [Einstein, 1997]

**Definition 18 (Short form notation for request bounds)** Assuming \( \delta(t' - \phi - n_{\tau,\epsilon})^{n_{\tau,\epsilon}} \cdot C_{\tau,\epsilon}[n] \) is a short-hand notation for \( \sum_{e \in E, n=0}^{k-1} \delta(t - \phi - n_{\tau,\epsilon})C_{\tau,\epsilon}[n] \) and \( C \) is a vector that contains different execution times for different jobs, it is possible to write

\[
\int_0^t \sum_{e \in E, n=0}^{k-1} \delta(t' - \phi - n_{\tau,\epsilon}) \cdot C_{\tau,\epsilon}[n] \cdot \mathbb{H}(t' - t_0) \cdot \mathbb{H}(t_b - t') \ dt'
\]

\[
= \int_{[t,t_0]} \delta(t' - \phi - n_{\tau,\epsilon})^{n_{\tau,\epsilon}} \cdot C_{\tau,\epsilon}[n] \cdot \mathbb{H}(t' - t_0) \cdot \mathbb{H}(t_b - t') \ dt'
\]

\[
= \int_{[t,t_0]} \delta(t' - \phi - n_{\tau,\epsilon})^{n_{\tau,\epsilon}} \cdot C_{\tau,\epsilon}[n] \ dt'
\]

\[
= \mathbb{E}_{\tau,\epsilon}^{n_{\tau,\epsilon}}(t, \Delta_{lb}^b) \cdot C_{\tau,\epsilon}[n] = \mathbb{R}_{\tau,\epsilon}^{n_{\tau,\epsilon}}(t, \Delta_{lb}^b)
\]

To complete the integration of the multiframe model first introduced by [Baruah et al., 1999] in this work, we need to redefine the concept of deadlines:

**Definition 19 (Deadline Vector)** The deadline vector is given by

\[
D_{\tau,\epsilon}[n] = D^n_{\tau,\epsilon} = [d_1, \ldots, d_k]
\]
4.2.2 The request bound of interfering jobs

The request bound function, as defined in general, does not distinguish between task priorities. Therefore it sums the requested execution times of all tasks. It is necessary to compute the interference of jobs to differentiate between the request of higher prior jobs that interrupt and interfere with a given job and other jobs that will have no impact on the final response. The following section will consider static as dynamic priorities as well. We look at how the same approach can solve both problems. Additionally, we find a unified solution of hierarchical scheduling of both algorithms which can be used in general to describe one of the two algorithms as well as a combination of them. First, we formulate an abstract interfering request bound which can easily be adapted to different scheduling criteria:

Theorem 3 (Interference request bound) Assume any criteria \( □ \) and \( ■ \geq □ \) has a higher or equal priority and any job of \( τ'_r \) interfere with \( τ_0 \). The interfering jobs execution time is selected by masking the request bound:

\[
R_{τ'_r}(t, Δ^h_0) = E_{τ'_r,τ_0}^{k}(t, Δ^h_0) \cdot C^{τ'_r}_τ[τ] \cdot H(τ'_r - τ_0) \tag{67}
\]

Proof The Heaviside function as given by definition [13] returns 1 if \( ■ \geq □ \), and the request of the higher priority task is added to the request bound. If the criterium of task \( τ'_r \) is smaller than the one of task \( τ \) the Heaviside function is equal to 0 modelling no interrupt.

The idea to describe interference of jobs is generalized to a bunch of different relations \( H : \mathbb{R} \to \{0, 1\} \) mapping any difference of real- or integer numbers to boolean values:

Definition 20 (Heaviside relation) Again, assume any criteria \( □ \), the main relations can be computed by the following heaviside functions:

\[
\begin{align*}
\&\& = \&\& \&\& := H(■ - □) \cdot H(□ - ■) = \delta_{□,■} \\
\&\& \leq □ := H(□ - ■) \\
\&\& \geq □ := H(■ - □) \\
\&\& < □ := H(□ - ■) \\
\&\& > □ := H(■ - □)
\end{align*}
\]

The Kronecker delta \( \delta_{□,■} \) is a well known short-form writing if criteria are equal.
Definition 21 (Taskscheduler) Any boolean equation of Heaviside relations models a scheduler in real-time analysis because it defines whether two tasks interfere or not. Assume any Heaviside relation $\xi \in \{H_{=}, H_{\leq}, H_{\geq}, H_{<}, H_{>}\}$ the function $S : f^2 \rightarrow \{0, 1\}$ represents a task scheduler which describes the interference of two tasks:

$$S_{\tau, \tau'} := \max\{\min \{\xi\}\}$$ (69)

Note, the operation $\max$ and $\min$ represents $or$ and $and$ on integers.

Example 8 (Static task scheduler) Assume static priorities as given in definition 6. Two jobs interfere if

$$S_{\pi, \pi'} := \max\{H_{\pi \tau} < \pi \tau' \} \min\{H_{\pi \tau} = \pi \tau', H_{\tau r} < t_{\tau'}\}$$ (70)

In deadline monotonic scheduling the priority is not needed, it is possible to write directly

$$S_{d, d'} := \max\{H_{d \tau} < d \tau' \} \min\{H_{d \tau} = d \tau', H_{\tau r} < t_{\tau'}\}$$ (71)

Example 9 (Dynamic task scheduler) In dynamic scheduling the job with the earliest absolute deadline is scheduled. Therefore we have only to change the relative deadline to the absolute deadline in definition 8. In this case, the consideration of the request time is mandatory because system designer and programmers can not guarantee different absolute deadlines if the specified relative deadlines are different.

$$S_{D, D'} := \max\{H_{D \tau} < D \tau' \} \min\{H_{D \tau} = D \tau', H_{\tau r} < t_{\tau'}\}$$ (74)

Note, that indifference to the static task scheduler the absolute deadline of each job must considered $\text{11}$ $H_{D \tau} < D \tau'$ and $H_{D \tau} = D \tau'$ are disjunct, therefore

$$S_{D, D'} := H_{D \tau} < D \tau' + H_{D \tau} = D \tau' \cdot H_{\tau r} < t_{\tau'}$$ (75)

The first step in the discussion is the formulation of the interfering request bound for static schedulers and task priorities specified by fixed numbers:

$\text{11}$ In scheduling theory, we assume that any job could be executed if absolute deadlines are equal. It can be described again by the upper Heaviside function without any assumption about request times. Again, arbitrary deadlines could be modeled easily, considering request times.
Corollary 1 (Interference request bound in static scheduling) Assume any static scheduler with a priority \( \pi \) assigned to each task. If task \( \tau' \) has a higher priority than task \( \tau \) and a higher number of \( \pi_{\tau'} > \pi_{\tau} \) specifies this behaviour, then the interference request bound \( R_{\tau,\tau'}^{\tau' > \tau} : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R} \) is given by:

\[
R_{\tau,\tau'}^{\tau' > \tau}(t, \Delta_b^\tau) = E_{\tau,\tau'}^{\tau' > \tau}(t, \Delta_b^\tau) \cdot C_{\tau',\tau}^{\tau' > \tau} \cdot \mathbb{H}(\pi_{\tau'} - \pi_{\tau}) + \delta_{\tau,\tau'} \cdot \mathbb{H}(t'_\tau - t'_{\tau'})
\]  

Contrarily, if task \( \tau' \) has a higher priority than task \( \tau \) and a lower number of \( \pi_{\tau'} < \pi_{\tau} \) specifies this behaviour, then the priority difference in the equation changes. If we assume deadline monotone scheduling the interfering request bound can express this directly:

\[
R_{\tau,\tau'}^{\tau' < \tau}(t, \Delta_b^\tau) = E_{\tau,\tau'}^{\tau' < \tau}(t, \Delta_b^\tau) \cdot C_{\tau',\tau}^{\tau' < \tau} \cdot \mathbb{H}(\pi_{\tau'} - \pi_{\tau}) + \delta_{\tau,\tau'} \cdot \mathbb{H}(t'_\tau - t'_{\tau'})
\]

Proof Consider theorem 3. For \( \pi_{\tau'} > \pi_{\tau} \) the Heaviside function \( \mathbb{H}(\pi_{\tau'} - \pi_{\tau}) = 1 \), and the execution request of task \( \tau' \) is added to the interference task set of \( \tau \). If two priorities are equal the job with the earliest request is scheduled. The interference mask become one if \( t'_\tau - t'_{\tau'} \geq 0 \). The proof of the other DMS equation is obvious.

According to this well-known definition of absolute deadlines, the interfering request bound in dynamic scheduling can be formulated by:

Corollary 2 (Interference request bound in dynamic scheduling) The interfering request bound \( R_{\tau,\tau'}^{D \tau' \geq D \tau} : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R} \) of higher priority tasks in dynamic scheduling is:

\[
R_{\tau,\tau'}^{D \tau' \geq D \tau}(t, \Delta_b^\tau) = E_{\tau,\tau'}^{D \tau' \geq D \tau}(t, \Delta_b^\tau) \cdot C_{\tau',\tau}^{D \tau' \geq D \tau} \cdot \mathbb{H}(D_{\tau'} - D_{\tau}) + \delta_{D \tau',D \tau} \cdot \mathbb{H}(t'_\tau - t'_{\tau'})
\]

Proof Assume dynamic scheduling and a given job \( \tau_{\epsilon,n} \). The request bound of this job is the sum of all execution times of job’s \( t'_{\epsilon,n} \) with an absolute deadline shorter than the job’s \( t_{\epsilon,n} \) deadline. According to theorem 3 the subtraction \( D_{\tau} - D_{\tau'} \) is positive if \( D_{\tau} < D_{\tau'} \), therefore in this case \( \mathbb{H}(D_{\tau} - D_{\tau'}) = 1 \). If \( D_{\tau} > D_{\tau'} \) the inequality \( \mathbb{H}(D_{\tau} - D_{\tau'}) = 0 \). This means \( \mathbb{H}(D_{\tau} - D_{\tau'}) \) selects the higher priority jobs in dynamic scheduling. This approach models scheduling were any task instance with an absolute deadline smaller than the absolute deadline of the considered task instance interfere in the considered task. However, what happens if two instances have the same absolute deadline? In this case, the instance with the smaller request is scheduled to avoid scheduling overhead. This behaviour is modelled by \( \delta_{D \tau',D \tau} \cdot \mathbb{H}(t'_\tau - t'_{\tau'}) \).

Example 10 (Interfering request bound) Again consider the task set given in example 6. The interfering request bounds in static and dynamic scheduling are given in

\footnote{This assumption does not hold in general. However, if \( \epsilon \) job is scheduled if deadlines are equal only the Heaviside mask should be modified to model such a scheduling behavior. But this leads to an analysis over approximation.}
In real-time scheduling theory, the structure of equations changes on any new problem. As we demonstrated the unified theory, the request bound of interfering jobs can be expressed by just one single equation choosing the correct parameters of the Heaviside mask.

**Theorem 4 (Interference request bound in hierarchical scheduling)** Assume a task set and assign a static priority to each task. Then tasks with different priorities schedule by static scheduling and tasks with equal priority schedule according to their deadlines dynamically. In the case of such hierarchical scheduling, the interfering request bound \( R_{t,t}' \) is

\[
R_{t,t}'(t, \Delta_a) = \sum_{a' \leq C_{a,t}} (t, \Delta_a) \cdot C_{a,t},
\]

\[
\max ( \tilde{H}(\pi_{t'} - \pi_I), \delta_{t,t'} \cdot [ \tilde{H}(D^t_a - D^t_{a'}) + \delta_{D^t_a,D^t_{a'}} \cdot \tilde{H}(t_{a'} - t_I) ] ) \quad (79)
\]

**Proof** If a task has a higher priority than the considered task, the function \( \tilde{H}(\pi_{t'} - \pi_I) = 1 \) else it is 0. If the priority of the tasks is equal and the absolute deadline
of the interfering task is smaller than the absolute deadline of the considered task the scheduling is described by \( \delta(r, \tau') \cdot [ \mathbb{H}(D^r_\tau - D^r_\tau) + \delta D_{\tau'} \cdot \mathbb{H}(t'_\tau - t'_\tau) ] \), as we already know from corollary 2. This function is only 1 if the absolute deadline of a potential interfering task \( \tau' \) is shorter than the deadline of the considered task \( \tau \) and the request time of the interfering task \( \tau' \) is earlier than the request time of the considered task \( \tau \). Therefore, the selecting criteria to identify an interfering task leads to 0 or 1 dependently on the tasks priority or absolute deadline. The function \( \max(\mathbb{H}(\pi_\tau - \pi_{\tau'}), \mathbb{H}(D^r_\tau - D^r_{\tau'}) = 1 \) if \( \max(0, 1) \), \( \max(0, 1) \) or \( \max(1, 1) \). This implements an or-operation between static and dynamic scheduling.

\( \square \)

4.3 Analysis preliminaries

We need some additional assumptions to derive feasibility tests or a response time analysis based on the interfering request bound. In this section, we will introduce the concept of the remaining load to compute the backlog, which is not proceeded by a processor during a given time interval. Besides, we will give some useful definitions related to a generalized analysis framework.

**Theorem 5 (Remaining load)** The remaining load of a job interfered with other jobs is the computational demand of a given time interval \([0, t]\) which cannot be computed by the processor during this time interval. Assume that the timing interval \( \Delta^r_\tau = t \), then the remaining load \( L^{\tau \geq \tau} : \mathbb{R}^2 \rightarrow \mathbb{R} \) is:

\[
L^{\tau \geq \tau}_{r, \tau'}(t) = \max_{0 \leq s \leq t} \{ R^{\tau \geq \tau}_{r, \tau'}(t, t) - R^{\tau \geq \tau}_{r, \tau'}(s, t) - t \} \tag{80}
\]

**Proof** If \( \forall t \in \mathbb{R} : L^{\tau \geq \tau}_{r, \tau'}(t) \geq 0 \) and \( L^{\tau \geq \tau}_{r, \tau'}(t) = R^{\tau \geq \tau}_{r, \tau'}(t, t) - t \), then

\[
R^{\tau \geq \tau}_{r, \tau'}(t, t) - L^{\tau \geq \tau}_{r, \tau'}(t) - t \leq 0 \tag{81}
\]

\[
R^{\tau \geq \tau}_{r, \tau'}(t - s, t) - L^{\tau \geq \tau}_{r, \tau'}(t - s) - t \leq 0 \tag{82}
\]

\[
R^{\tau \geq \tau}_{r, \tau'}(t, t) - R^{\tau \geq \tau}_{r, \tau'}(s, t) - L^{\tau \geq \tau}_{r, \tau'}(t) - L^{\tau \geq \tau}_{r, \tau'}(s) - t \leq 0 \tag{83}
\]

Because \( L^{\tau \geq \tau}_{r, \tau'}(t) - L^{\tau \geq \tau}_{r, \tau'}(s) \leq L^{\tau \geq \tau}_{r, \tau'}(t - s) \) (Boudec, 1998, p. 7) the following in equation holds:

\[
L^{\tau \geq \tau}_{r, \tau'}(t) \geq R^{\tau \geq \tau}_{r, \tau'}(t, t) - R^{\tau \geq \tau}_{r, \tau'}(s, t) - t \tag{84}
\]

\[
L^{\tau \geq \tau}_{r, \tau'}(t) = \sup_{0 \leq s \leq t} \{ R^{\tau \geq \tau}_{r, \tau'}(t, t) - R^{\tau \geq \tau}_{r, \tau'}(s, t) - t \} \tag{85}
\]

At this point, we can carefully review the properties of the unified event bound as given by theorem 1. Note that the domain of this function is a compact space: Because we defined the limits of the integral as an open interval, the domain \( t \) is compact. If we define any analysis in a bounded domain, then the unified event bound and therefore, the request and demand bound are compact. Bounding periodic task sets
to their hyper-period bounds the timing interval as well. Therefore, the supremum of the function is equal to its maximum: $\sup = \max$. Then

$$L^{\tau\geq t}_{\tau,\tau}(t) = \max \left\{ R^{\tau\geq t}_{\tau,\tau}(t, t) - R^{\tau\geq t}_{\tau,\tau}(s, t) - t \right\} \quad (86)$$

The proof builds on the leaky bucket algorithm. In the case no load is requested to a processor, the remaining load is equal to 0. Only if service is requested, the processor executes it with the rate of $t$. The above proof is directly adapted from the network calculus as given by [Boudec, 1998](p.10, f.). Because we defined a compact unified event-, request- and demand bound by using an integral and we model its limits by a Heaviside function, we can now combine the result of the network respective the real-time calculus with the work done in established scheduling theory. If supremum and infimum become maximum and minimum in all cases, we can further use effective maximization and minimization techniques supported by numerical mathematics and therefore it is easy to apply this theory to computer algebra systems or numerical math tools.

**Example 11 (Remaining load on the example task set)** Consider again example 6. Let us compute the remaining load of job $\tau_3$, job 1. Figure 6 shows the result for static and dynamic scheduling.

The resulting plot is shown on the left hand in figure 6. Additionally, we consider as an example, the remaining load of task $\tau_3$, job 1. Figure 6 shows the result for static and dynamic scheduling.
Definition 22 (Average load) Given any time interval $[a, b]$ the average load $\Delta^b_a = t$, the remaining load $U : \Gamma \times \mathbb{R} \to \mathbb{R}$ is the mean value of the requested load related to the interval. The average load of a task set is the sum of the average loads of each task.

$$U_R(\Delta^b_a) = \sum_{\tau \in \Gamma} U_{\tau}(\Delta^b_a) = \frac{1}{b-a} \left( L_R(a) + \int_a^b \sum_{\tau \in \Gamma} (\Pi^k_{\tau} \cdot c^+_\tau) \, dt \right)$$  \hspace{1cm} (87)

Lemma 3 (Average load by the unified request bound) The average load in any time interval $[a, b)$ of a task set on a processor is given by

$$U_R(\Delta^b_a) = \frac{L_R(t) + R_R(\Delta^b_a, \Delta^b_a)}{b-a}$$  \hspace{1cm} (88)

and in the special case in the interval $[0, t)$

$$U_R(\Delta^b_a) = U_R(t) = \frac{R_R(t, t)}{t}$$  \hspace{1cm} (89)

Now only tasks with the same or a higher priority should be considered, we use

$$U_{\tau, \tau'}^\geq(\Delta^b_a) = \frac{L_{\tau, \tau'}^\geq(t) + R_{\tau, \tau'}^\geq(\Delta^b_a, \Delta^b_a)}{b-a}$$  \hspace{1cm} (90)

and

$$U_{\tau, \tau'}^\geq(t) = \frac{R_{\tau, \tau'}^\geq(t, t)}{t}$$  \hspace{1cm} (91)

for the interval $[0, t)$.

Proof The average load of the requested jobs in any interval related to the duration of this interval. In some cases, there is some load left from previous intervals. That remains in an additional load:

$$U_R(\Delta^b_a) = \frac{1}{\Delta^b_a} \left( L_R(a) + \int_a^b \sum_{\tau \in \Gamma} (\Pi^k_{\tau} \cdot c^+_\tau) \, dt \right)$$

$$= \frac{1}{b-a} \left( L_R(a) + \int_a^b \sum_{\tau \in \Gamma} (\Pi^k_{\tau} \cdot c^+_\tau) \, dt \right)$$

$$= \frac{L_R(a) + R_R(\Delta^b_a, \Delta^b_a)}{b-a}$$

13 We assume only to count events in $[0, t)$ as given by our unified request bound. However, to define a utilization interval a closed interval is needed.
Note that the computation of the utilization requires a summation during \([a, b)\). Therefore, the request bound is given by \(R_R(\Delta_R^a, \Delta_R^b)\). Let us now consider the utilization in the interval \([0, t)\):

\[
U_R(t) = \sum_{\tau \in \Gamma} U_\tau(t)
\]

\[
= \sum_{\tau \in \Gamma} \left( \frac{1}{t} \cdot \int_0^t \left( \sum_{\tau \in \Gamma} \left( X_{\tau} \cdot c_\tau^+ \right) \right) dt' \right)
\]

\[
= \frac{1}{t} \cdot \int_0^t \sum_{\tau \in \Gamma} \left( X_{\tau} \cdot c_\tau^+ \right) dt'
\]

\[
= \frac{R_R(t, t)}{t}
\]

\(\square\)

**Example 12 (Average load of the example task set by different scheduling strategies)** The average load of job 1 and job 2 of task \(\tau_3\) are given in figure 5. Note that this diagram clearly shows the busy window of both jobs. The following CAS input produces the plots:

```cas
Plot[ Utilization[Task3[0]], (TaskSet, edf), t], Utilization[Task3[0]], (TaskSet, dms), t], 1, {t, 0, Hyperperiod[TaskSet]}, AxesOrigin -> {0, 0}, PlotLegends -> Placed["Expressions", Below], AxesLabel -> {t, U}]
```

4.4 Unified feasibility analysis

Feasibility tests build on utilization bounds. Therefore we have to check whether the utilization of a task set is always smaller than 1 or 100%. A utilization bound
given for any time interval based on the interference request bound, and the average load allows feasibility tests for static, dynamic, and hierarchical scheduling.

**Theorem 6 (Feasibility analysis for static and dynamic scheduling)** Assume a task set and a hierarchical scheduler. If a task has a higher priority than another task, it executes first, and if two tasks have the same priority, they are scheduling under earliest deadline first. The feasibility of a given independent task set executed by one computing resource with a hierarchical static, and a dynamic scheduler can then be guaranteed, if and only if

\[ \forall t \in P \Gamma : R_{\pi,\tau}^{\pi,\tau}(t, P) + R_{\tau,\tau'}^{\tau,\tau}(t, P) \leq t \]  

(92)

**Proof** The feasibility test can be derived from the utilization bound

\[ U_{\Gamma}(\Delta^b) = \frac{L_{\Gamma}(t) + R_{\Gamma}(\Delta^b, \Delta^b)}{t_b - t_a} \]  

(93)

and in the special case in the interval \([0, t)\)

\[ U_{\Gamma}(\Delta^b) = U_{\Gamma}(t) = \frac{R_{\Gamma}(t, t)}{t} \]  

(94)

A task set is feasible if \( \forall t \in [0, P_{\Gamma}] \) for any task the utilization \( u_{\Gamma}(t) \leq 1 \). If \( t_a = t_b = 0 \) then \( L(0) = 0 \) and therefore

\[ \forall t \in P_{\Gamma} : \frac{R_{\pi,\tau}^{\pi,\tau}(t, P) + R_{\tau,\tau'}^{\pi,\pi}(t, t)}{t} \leq 1 \]  

(95)

Now we separate all higher priority tasks from tasks with the same priority and multiply by \( t \):

\[ \forall t \in P_{\Gamma} : R_{\pi,\tau}^{\tau,\tau}(t, P) + R_{\tau,\tau'}^{\pi,\pi}(t, t) \leq t \]  

(96)

or

\[ \forall t \in P_{\Gamma}, \forall \tau \in \Gamma : \sum_{\tau' \in \Gamma} R_{\tau'}(t - d_{\tau'}, P_{\Gamma}) \cdot \delta_{\pi,\tau} \]  

(97)

\[ + \sum_{\tau' \in \Gamma} R_{\tau'}(t, t) \cdot H(\pi_{\tau'} - \pi_{\tau}) \leq t \]  

(98)

Note that we consider tasks with the same priority and tasks with higher priorities in independent terms because we want to derive a feasibility test for static and dynamic scheduling as well. Therefore we have to consider two cases:

A If \( \pi_{\tau'} = \pi_{\tau} \) then \( \delta_{\pi,\tau} = 1 \) and \( H(\pi_{\tau'} - \pi_{\tau}) = 0 \). The feasibility test for dynamic scheduling is then given by

\[ \forall t \in P_{\Gamma}, \forall \tau \in \Gamma : \sum_{\tau' \in \Gamma} E_{\tau'}(t - d_{\tau'}, P_{\Gamma}) \cdot c_{\tau'}^{b} = \sum_{\tau' \in \Gamma} R_{\tau'}(t - d_{\tau'}, P_{\Gamma}) \leq t \]  

(99)

---

14 To keep the proof simple we do not consider the request times.
In this case $\tau' = \tau$ and therefore
$$\forall t \in \mathcal{P}_\tau : \sum_{\tau \in \Gamma} E_{\tau}(t - d_\tau, \mathcal{P}_\tau) \cdot c_\tau^+ = \sum_{\tau \in \Gamma} R_{\tau}(t - d_\tau, \mathcal{P}_\tau) \leq t \quad (100)$$
which is equal to the processor demand test or problem [2].

B Consider $\pi_{\tau'}$ not $\pi_\tau$: Now $\delta_{\pi_{\tau'}, \pi_\tau} = 0$ only for the considered task $\tau$ and in all other cases $\delta_{\pi_{\tau'}, \pi_\tau} = 0$ and $\mathbb{E}(\pi_{\tau'} - \pi_\tau) = 1$ if a task $\tau'$ has a higher priority than the considered task $\tau$. Then we get
$$\forall t \in \mathcal{P}_\tau, \forall \tau \in \Gamma : \quad E_{\tau}(t - d_\tau, \mathcal{P}_\tau) \cdot c_\tau^+ + \sum_{\tau' \in \mathcal{P}_\tau} E_{\tau'}(t, t) \cdot c_{\tau'}^+ \leq t \quad (101)$$
or
$$\forall t \in \mathcal{P}_\tau, \forall \tau \in \Gamma : \quad R_{\tau}(t - d_\tau, \mathcal{P}_\tau) + \sum_{\tau' \in \mathcal{P}_\tau} R_{\tau'}(t, t) \leq t \quad (102)$$
where $\tau$ is the considered task and $\tau'$ are all higher priority tasks. This result is equivalent to
$$\forall t \in \mathcal{P}_\tau, \forall \tau \in \Gamma : \quad R_{\tau}(t - d_\tau, \mathcal{P}_\tau) \leq t - \sum_{\tau' \in \mathcal{P}_\tau} R_{\tau'}(t, t) \quad (103)$$
which is equal to the processor demand test for static scheduling originally given by [Baruah, 2003].

4.5 Unified response time analysis

In this section, we will derive a response time analysis based on the average load and the unified event bound to simplify the mathematical framework as given by related work. As a result of the section, we will see that the unified event bound solves problem [3].

**Theorem 7 (Unified response time analysis)** If a job is scheduled by any scheduling algorithm assuming priorities given by any relation between two different variables $\blacksquare \geq \Box$, the request time of the job is $t_{r, \epsilon}$. Let the response time $r_{r, \epsilon} = t_{f, \epsilon} - t_{r, \epsilon}$ be the difference of the finishing time $t_{f, \epsilon}$ and the request time. The response time $r_{r, \epsilon}$ is bounded by
$$\forall \tau_\epsilon \in \Gamma : \quad L_{\tau, \epsilon}(t_{r, \epsilon}) + R_{\tau, \epsilon}(r_{r, \epsilon}, t_{r, \epsilon}) - r_{r, \epsilon} = 0 \quad (104)$$

**Proof** Again, we start with the average load. The average load of any given time interval $[t_2, t_3]$ is given by:
$$U_{\tau}(\Delta_\epsilon^b) = \frac{L_{\tau}(a) + R_{\tau}(\Delta_\epsilon^b, \Delta_\epsilon^b)}{\Delta_\epsilon^b} \leq 1 \quad (105)$$
Now, we consider the interval $\Delta_{a}^{b} = t_{r,\epsilon}^{f} - t_{r,\epsilon}^{\text{f}} = r_{r,\epsilon}$ for each job of all tasks, therefore

$$\forall \tau \in \Gamma : \frac{L_{r}(t_{r,\epsilon}^{f}) + R_{r}(r_{r,\epsilon}, r_{r,\epsilon})}{r_{r,\epsilon}} \leq 1$$

We only have to consider all tasks of the same or a higher priority than the considered task’s priority:

$$\forall \tau \in \Gamma : \frac{L_{r,\tau'}(t_{r,\epsilon}^{f}) + R_{r,\tau'}(r_{r,\epsilon}, r_{r,\epsilon})}{r_{r,\epsilon}} \leq 1$$

During a busy period, the average load is positive and more significant than 100% because the requested execution demand is higher than the elapsed processor time. The average load will be smaller than 100% if the requested demand in a time interval is smaller than the processing time interval. In this case, the processor is idle. Therefore, the end of the busy period is exact if the average load is equal to 100%:

$$\forall \tau \in \Gamma : \frac{L_{r,\tau'}(t_{r,\epsilon}^{f}) + R_{r,\tau'}(r_{r,\epsilon}, r_{r,\epsilon}) - r_{r,\epsilon}}{r_{r,\epsilon}} = 0$$

As we want to compute the response time $r_{r,\epsilon}$, we also except task sets with an average load equal to 100%, which means after a job has finished the next higher priority job starts immediately, and the processor does not idle. As a result, we have to end the summation of task requests exactly at $r_{r,\epsilon}$ and we get

$$\forall \tau \in \Gamma : \frac{L_{r,\tau'}(t_{r,\epsilon}^{f}) + R_{r,\tau'}(r_{r,\epsilon}, r_{r,\epsilon}) - t_{r,\epsilon}^{f}}{r_{r,\epsilon}} = 0$$

Note that in the worst-case in static scheduling we only have to consider the first job of each task. The remaining load then is $L_{r,\tau'}(0) = 0$ and the worst-case response time becomes

$$\forall \tau \in \Gamma : R_{r,\tau'}(r_{r,\epsilon}^{+}, r_{r,\epsilon}^{+}) - r_{r,\epsilon}^{+} = 0$$

Theorem 7 describes an unified abstract form of the busy window approach. In real-time scheduling theory, static and dynamic scheduling are major scheduling algorithms. Therefore, the unified approach has to be adapted to static as well as to dynamic scheduling.

**Corollary 3 (Static response time analysis)** Assume a given task set with static priorities. The abstract given relation $\geq \Diamond$ is then replaced by $\geq \Pi_{r}$ formulating the static priority scheme. The response time then becomes

$$\forall \tau \in \Gamma : \frac{L_{r,\tau'}(t_{r,\epsilon}) + R_{r,\tau'}(r_{r,\epsilon}, r_{r,\epsilon}) - r_{r,\epsilon}}{r_{r,\epsilon}} = 0$$

**Proof** Replacing $\geq \Diamond$ with $\geq \Pi_{r}$, the proof follows directly from theorem 1 and theorem 7.

15 Including other scheduling schemes should be future work.
Corollary 4 (Dynamic response time analysis) Assume a given task set with dynamic priorities. The abstract given relation $\geq$ is then replaced by $D^n_\tau \leq D^n_\tau'$, formulating the dynamic priority scheme corresponding to EDF scheduling. The response time then becomes

$$\forall \tau \in \Gamma : \sum_{\tau' \leq \tau'} D^n_{\tau'} \leq D^n_{\tau'} \leq (r_{\tau,\epsilon}, r_{\tau,\epsilon}) - r_{\tau,\epsilon} = 0$$

(111)

Proof Replacing $\geq$ with $D^n_\tau \leq D^n_\tau'$, the proof follows directly from theorem 2 and theorem 7.

Example 13 (Response time analysis) Consider again the example task set given in table 1. The response time analysis implemented in the CAS supports static and dynamic scheduling. The CAS gives the following output for the example task set.

The output is printed as a list with the following format: \{ $t_{\tau',\epsilon}, L_{\tau'} \geq \tau (t_{\tau,\epsilon}), r_{\tau,\epsilon}, d_{\tau}$ \}. In the following each of this lists represent a task and the analysis provides the response times of each job.

A logical combination of the two selecting Heaviside functions describes a hierarchical scheduler, as shown in theorem 4.

Corollary 5 (Hierarchical response time analysis) Assume a scheduler which scheduled tasks by their given priorities and all tasks with the same priority by their deadline. A hierarchical busy window response time analysis is given by

$$\forall \tau \in \Gamma : \sum_{\tau' \leq \tau'} L_{\tau',\epsilon} (t_{\tau,\epsilon}) + R_{\tau',\epsilon} (r_{\tau,\epsilon}, r_{\tau,\epsilon}) \leq 0$$

(112)

Proof The analysis directly follows from theorem 4 and theorem 7.

The plots shown are originally given from the CAS.
Example 14 (Response time analysis with hierarchical scheduler) To consider hierarchical scheduling, the task set given in Table 1 is modified. To highlight the effect of hierarchical scheduling, we add a few tasks and to decrease the utilization of the following task set:

- Task1 := \{\{(0, 6), \infty\}, 1\}
- Task2 := \{\{(0, 9), \infty\}, 1, 8\}
- Task3 := \{\{(0, 12), \infty\}, 2\}
- Task4 := \{\{(0, 18), \infty\}, 3, 15\}
- Task5 := \{\{(0, 36), \infty\}, 7, 17\}
- Task6 := \{\{(0, 72), \infty\}, 9, 35\}

TaskSet := \{Task1, Task2, Task3, Task4, Task5, Task6\}

In this task set, the last number denotes the priority level, ignored under dynamic scheduling. In static scheduling, a task with the lowest number has the highest priority. The CAS computes the following output, plotted in Figure 11 and Figure 10:

```
ResponseTimeAnalysis[{TaskSet, dms}]
```

```
\{\{0, 0, 52, 35\}\}
```

HeRTA: Heaviside Real-Time Analysis

39
As a result, the schedule of the second task set is feasible if all tasks scheduled dynamically. If a hierarchical scheduler is used then the worst-case response time of all jobs of task $\tau_1$ is decreased because of its high priority, while task $\tau_6$ does not hold its deadline anymore, because of its low priority. All other tasks except task $\tau_5$ have the same priority and, therefore, are scheduled dynamically. However, task $\tau_2$ and task $\tau_3$ will miss their deadlines because of the high priority of task $\tau_1$.

4.5.1 A tighter response time analysis in dynamic scheduling

The sporadic and arbitrary model developed by [Tindell et al., 1994b] was applied to dynamic scheduling by [Spuri, 1996]. The work of [Palencia and Harbour, 1998] generalizes [Spuri, 1996] approach. However, if we analyze a fully utilized task set a general problem with the previous work arises.

**Example 15 (Spreading worst case in dynamic scheduling)** Let us now consider the schedules of the example task set as given in figure 6 as given in detail: Figure 12 gives the static schedule and does not hold the deadline of task $\tau_3$ because the task sets utilization is exact $U_T = 1$. Therefore it exists only a dynamic schedule. Related work [Spuri, 1996] and [Guan and Yi, 2014b] assumes that, if the absolute deadlines...
Fig. 11 Response Time Analysis plot of the second task set scheduled only by EDF

Fig. 12 Schedules of the example tasks set: in static (a.) and dynamic scheduling (b.-d.)

of jobs are equal, any of these jobs are scheduled. This assumption leads to different schedules as shown in figure [12], [12], and [12]. The worst-case in this scenario is that the worst-case response time of each task is equal to its relative deadline because of the chosen utilization. This leads to the worst case response times $r_1 = 8$, $r_2 = 16$ and $r_3 = 24$. However, as seen in figure [12], the worst case of different tasks occur in different schedules.

This example shows that EDF is underspecified, which leads to non-deterministic behaviour. Maybe this should be the reason that EDF is not entirely accepted in industry applications.
The behaviour of a dynamic scheduler like EDF is non-deterministic. If no additional criterion is given, a dynamic scheduler may dispatch any of the tasks if deadlines are equal. However, this is true for static scheduling as well, and this case is prevented by giving different priorities to tasks. In the model of arbitrary deadlines, the original concept given by [Liu and Layland, 1973] has been expanded by [Tindell et al., 1994b]. In this model, any job of a task with the same priority is dispatched if its request time is shorter than any request time of other jobs with the same priority. If we add this simple criterion to dynamic scheduling, the schedule of all jobs becomes deterministic. Because the previous work only addresses busy windows, it is not possible to formulate additional scheduling criteria. However, this can be done with the new approach presented in this paper.

Theorem 8 (Spreading worst case in dynamic scheduling) On the assumption that a dynamic scheduler is free to decide which job is scheduled if the absolute deadlines are equal the worst-case response time of different tasks occurs in different schedules and therefore the worst-case response time is over estimated:

\[ E^{n \leq k}(t, \Delta^0) C^n \cdot \mathbb{E}(D_n^0 - D_n^\tau) + \delta_{D^\tau, D^\tau'} \cdot \mathbb{H}(t^\tau - t^\tau') \leq E^{n \leq k}(t, \Delta^0) C^n \cdot \mathbb{H}(D_n^0 - D_n^\tau) \]

\[ (113) \]

Proof: If we consider the interfering request bound the proof follows directly from corollary 2:

\[ E^{n \leq k}(t, \Delta^0) C^n \cdot \mathbb{H}(D_n^0 - D_n^\tau) = E^{n \leq k}(t, \Delta^0) C^n \cdot \mathbb{H}(D_n^0 - D_n^\tau) + \delta_{D^\tau, D^\tau'} \]

Therefore, if we add any criteria in the case that the absolute deadlines are equal, e.g. the request time of a job like \( \delta_{D_n^\tau, D_n^\tau'} \cdot \mathbb{H}(t^\tau - t^\tau') \), then in some cases no interference will occur. If an interference will not occur in some cases the resulting request is lower:

\[ E^{n \leq k}(t, \Delta^0) C^n \cdot \mathbb{H}(D_n^0 - D_n^\tau) + \delta_{D^\tau, D^\tau'} \leq E^{n \leq k}(t, \Delta^0) C^n \cdot \mathbb{H}(D_n^0 - D_n^\tau) \]

\[ (115) \]

Because in periodically scheduling the critical jobs of \( n - 1 \) task will have a request time shorter than the request time of one task, the worst-case response time of all these jobs, except one, will be short compared to the situation the job can be free choose by the scheduler.

As we have seen in the schedules of example 3 given in figure 12b., 12c., and 12d., the worst-case behaviour of dynamically scheduled tasks spreads over different schedules. Theorem 8 proofs that a task set of \( n \) tasks will have shorter worst-case response bounds for \( n - 1 \) tasks and one worst-case time equal to related work. Therefore, to the best of our knowledge, we found a closer worst-case response time estimation bound than any related work. If we only assume a dynamic scheduler schedules the task with the lowest request time first, if
Hierarchical Scheduling

Average Load

Utilization Analysis

Response Time Analysis

the absolute deadlines are equal, then the worst case response time is tighter.$^{18}$
Consider example $^{15}$ again: If we add this additional criteria to the scheduler, the maximal response times become $r_1^* = 8$, $r_2^* = 10$ and $r_3^* = 20$. However, because of the presented assumptions and theorems, we can always be sure to be equal or better than previous work.$^{19}$

5 Conclusion

This paper was motivated by the question whether it exists one unified event- or request bound for all kind of analysis purposes in real-time scheduling theory. Such a function was discovered by applying mathematical techniques from theoretical physics and digital signal processing to the real-time analysis problem. It could be shown that such a unified request bound, and the definition of an average load in real-time systems allows to derive most of the established real-time analysis algorithms from only these two assumptions. This results in a utilization based analysis and task response time analysis by just one unified event bound. Additionally, static and dynamic scheduling is considered as well in just one equa-

\begin{table}
\centering
\begin{tabular}{|c|c|}
\hline
| Request Bound for | Static Scheduling | Dynamic Scheduling |
\hline
$R_{r,k}^{D_r} \leq (t, A_r^*) = \frac{E_{r,k}^{D_r} (t, A_r^*) \cdot C_{r,k}^* \cdot H(d_r - d_r') + \delta d_r, d_r' \cdot H(t_r' - t_r')}{U_r (\Delta_r^*)}$ & $R_{r,k}^{D_r} \leq (t, A_r^*) = \frac{E_{r,k}^{D_r} (t, A_r^*) \cdot C_{r,k}^* \cdot H(D_r^0 - D_r') + \delta D_r^0, D_r' \cdot H(t_r' - t_r')}{U_r (\Delta_r^*)}$ \\
\hline
\end{tabular}
\caption{Unified real time scheduling analysis}
\end{table}
tion. The new equation system also covers the analysis of bursty event sequences by introducing hierarchical event streams or event densities as a computation of the convolution of two independent event densities. As a beautiful result, the work allows easily defining hierarchical schedulers. Table 3 gives an overview of the concluding results of this work. We conclude that an interfering request bound for static and dynamic scheduling for the first time in real-time scheduling theory is described by using the same equation structure. Both aspects are covered if we use, for static scheduling, the relative deadlines, and for dynamic scheduling, the absolute deadlines in the equation of interference. It could easily be seen that a few equations with a general mathematical structure will cover the main aspects in preemptive static and dynamic scheduling in the bounded execution time programming of real-time systems. In addition to these results, we also noted that the well-known response time analysis in dynamic scheduling overestimates. In the context of our new mathematical model, we found a better limit for the response time in dynamic scheduling as given in related work.

In future work, the new model is extended to the adaptive rate model. Because of the rich mathematical models are given in calculus it should be interesting to investigate the impact of the work to the real-time calculus to extend modular models as well as to develop new models for modern fieldbus devices. As the general approach of interfering request bounds built on an abstract criterion, it should be easy to extend the work to multcriticallity systems as well as to other widely implemented scheduling algorithms such as time division multiplex (TDMA).

In this paper, we have not discussed the computational complexity of the problem. The first goal was to develop a new toolbox for real-time scheduling analysis. However, the complexity of the problem is exponential. Therefore approximation techniques already discussed has to be integrated into future work.

A Mathematical framework

The main focus of the paper is the adaption of the mathematics used in theoretical physics and digital signal theory. In this appendix, we explain a few notations which are typically not well-known or widely used in the real-time systems community. Additionally, a list of symbols clarifies the notation. One of the goals of this paper is to formulate an easy to use mathematical theory of real-time systems with a clear focus on intuitively simple equations. Therefore a table which lists all symbols is given at the end of the appendix.

For intuitive reading we write \( p_\tau \). This means a function which gives the period of the specified task; The idea is a short notation for \( p_\tau = p(\tau) \). The next two definitions are from theoretical physics. In the real-time analysis, we often write summations, and in this paper, we get integrals over two summations. However, writing this in each equation brings a lot of overhead and redundant information. Therefore, we adopt an index based writing notation to the problem:

**Definition 23** (Einstein’s notation)

\[
i \in \{1, \ldots, n\} : \quad c_1 x_1 + \ldots + c_3 x_3 = \sum_{i=1}^{n} c_i x_i
\]  

(116)
The notation was introduced by [Einstein, 1997] to simplify multidimensional equations in gravity. However, it can be used to simplify the notations in real-time analysis as well. In this work we use two modified forms to reduce the complexity of equations:

**Definition 24 (Modified Einstein’s Notation)**

\[
c_k x_i = c_1 x_1 + \cdots + c_k x_k = \sum_{i=1}^{k} c_i x_i
\]

\[
c_{i,j} x_{i,j} = \sum_{i=1}^{k} \sum_{j \in J} c_{i,j} x_{i,j}
\]

This idea can be adapted to the request bound:

\[
\int_{-\infty}^{\infty} \delta(t' - \phi - n\epsilon - n\epsilon \cdot \tau) dt' = \int_{[a,b)} \delta(t' - \phi - n\epsilon - n\epsilon \cdot \tau) \cdot \tau_{\tau',\phi} \cdot \tau_{\tau',\phi} \cdot \tau_{\tau',\phi} \cdot \tau_{\tau',\phi} dt'
\]

An other useful symbol is the Kronecker delta. This function only returns 1 if both arguments are equal. In all other cases it returns 0. It is used in physics as a short hand notation for matrices. In our case it is used to collect tasks of the same priority. It also can be used to collect jobs of the same tasks.

**Definition 25 (Kronecker Delta)**

The Kronecker delta is used to write matrices in a compact form. The function returns a 1 if the two elements given to the function are equal. In all other cases it returns 0. In real-time analysis this property can be used to identify tasks with the same priority.

\[
\delta_{i,j} = \begin{cases} 
1 & i = j \\
0 & i \neq j
\end{cases}
\]
The following table concludes all symbols used in the paper. The first part of the table gives the functions applied from theoretical physics. The second part list all time-related symbols, while the third part introduces event-related elements. The fourth part presents all parameters related to tasks, and the last part lists the symbols used in real-time analysis.

| Symbol | Meaning |
|--------|---------|
| $\delta(t)$ | Dirac delta |
| $\delta_{i,j}$ | Kronecker delta |
| $H(t)$ | Heaviside function |
| $\overline{H}(t)$ | upper Heaviside function |
| $\underline{H}(t)$ | lower Heaviside function |
| $t_a$ | point in time |
| $\Delta$ | time interval |
| $\Delta^+$ | maximal time interval |
| $\Delta^-$ | minimal time interval |
| $\Delta_0$ | interval related to time 0, equivalent to $t_0$ |
| $\Delta_0^+$ | interval between $t_a$ and $t_0$ |
| $p$ | period |
| $j$ | jitter |
| $\phi$ | minimal distance, phase or offset |
| $\mathcal{P}$ | hyper period, the least common multiplier of a set of periods |
| $\epsilon$ | event |
| $\mathcal{E}$ | event list |
| $\mathcal{X}$ | event sequence |
| $\mathcal{X}_\Delta$ | event stream or event density |
| $\mathcal{X}_\Delta^+$ | maximal event density |
| $\mathcal{X}_\Delta^-$ | minimal event density |
| Symbol | Description |
|--------|-------------|
| \( \tau, \tau_n \) | task |
| \( \tau_{n,k} \) | job |
| \( \tau' \) | interfering task |
| \( \tau'_\interfering \) | interfering job |
| \( \Gamma \) | task set |
| \( \pi \) | priority |
| \( \Gamma_{\tau} \) | higher priority task set of task \( \tau \) |
| \( d \) | relative deadline |
| \( D \) | absolute deadline \( D = p + d \) |
| \( c \) | computational load, execution time |
| \( c^- \) | best case execution time |
| \( c^+ \) | worst case execution time |
| \( C_{\tau,k}^+ \) | a vector of different worst case execution times |
| \( C_{\tau,k}^- \) | a vector of different best case execution times |
| \( \mathcal{D} \) | a vector of different relative deadlines |
| \( S\mathcal{D}_{\tau,\tau'} \) | scheduler, schedules jobs of tasks \( \tau, \tau' \) according to criterion \( \Box \) |

| Symbol | Description |
|--------|-------------|
| E | event bound function |
| R | request bound function |
| L | demand bound function |
| r_\tau | response time of a task |
| r_\tau^+ | maximal response time of a task |
| r_\tau^- | minimal response time of a task |
| U_\tau | utilization of a task |
| U_\Gamma | utilization of a task set |

**References**

Albers et al., 2006. Albers, K., Bodmann, F., and Slomka, F. (2006). Hierarchical event streams and event dependency graphs: A new computational model for embedded real-time systems. In *18th Euromicro Conference on Real-Time Systems, ECRTS’06, 5-7 July 2006, Dresden, Germany, Proceedings*, pages 97–106.

Albers et al., 2008. Albers, K., Bodmann, F., and Slomka, F. (2008). Advanced hierarchical event-stream model. In *Real-Time Systems, 2008. ECRTS’08. Euromicro Conference on*, pages 211–220. IEEE.

Albers and Slomka, 2004. Albers, K. and Slomka, F. (2004). An event stream driven approximation for the analysis of real-time systems. In *16th Euromicro Conference on Real-Time Systems (ECRTS 2004), 30 June – 2 July 2004, Catania, Italy, Proceedings*, pages 187–195.

Albers and Slomka, 2005. Albers, K. and Slomka, F. (2005). Efficient feasibility analysis for real-time systems with EDF scheduling. In 2005 Design, Automation and Test in Europe
Anssi et al., 2013. Anssi, S., Kuntz, S., Gérard, S., and Terrier, F. (2013). On the gap between schedulability tests and an automotive task model. *Journal of Systems Architecture*, 59(6):341–350.

Baruah, 2003. Baruah, S. (2003). Dynamic- and static-priority scheduling of recurring real-time tasks. *Real-Time Systems, 24*:93–128. 10.1023/A:1021711220939.

Baruah, 2010. Baruah, S. (2010). The non-cyclic recurring real-time task model. In *Real-Time Systems Symposium (RTSS), 2010 IEEE 31st*, pages 173 –182.

Baruah et al., 1999. Baruah, S., Chen, D., Gorinsky, S., and Mok, A. (1999). Generalized multiframe tasks. *Real-Time Systems, 17*:5–22. 10.1023/A:1008030427220.

Baruah et al., 1993. Baruah, S., Howell, R., and Rosier, L. (1993). Feasibility problems for recurring tasks on one processor. *Theor. Comput. Sci.*, 118(1):3–20.

Baruah et al., 1991. Baruah, S., Mok, A., and Rosier, L. (1991). Preemptively scheduling hard-real-time sporadic tasks on one processor. In *90 Real-Time Syst. Symp.*, pages 182–190. Publ by IEEE.

Baruah et al., 2011. Baruah, S. K., Bonifaci, V., d’Angelo, G., Marchetti-Spaccamela, A., Van Der Ster, S., and Stougie, L. (2011). Mixed-criticality scheduling of sporadic task systems. In *European Symposium on Algorithms*, pages 555–566. Springer.

Biondi et al., 2014. Biondi, A., Melani, A., Marconi, M., Natale, M. D., and Buttazzo, G. (2014). Exact interference of adaptive variable-rate tasks under fixed-priority scheduling. In *Proceedings of the 26th Euromicro Conference on Real-Time Systems (ECRTS)*.

Boudec, 1998. Boudec, J.-Y. L. (1998). Application of network calculus to guaranteed service networks. *Information Theory, IEEE Transactions on, 44*(3):1087 –1096.

Bracewell, 2000. Bracewell, R. (2000). *The Fourier Transform and Its Applications*. Electrical engineering series. McGraw Hill.

Cruz, 1991a. Cruz, R. L. (1991a). A calculus for network delay. i. network elements in isolation. *Information Theory, IEEE Transactions on*, 37(1):114 –131.

Cruz, 1991b. Cruz, R. L. (1991b). A calculus for network delay. ii. network analysis. *Information Theory, IEEE Transactions on*, 37(1):132 –141.

Einstein, 1997. Einstein, A. (1916,1997). *The Foundation of the General Theory of Relativity*. Princeton University Press.

Fisher and Baruah, 2005. Fisher, N. and Baruah, S. (2005). A polynomial-time approximation scheme for feasibility analysis in static-priority systems with bounded relative deadlines. In *Proceedings of the 13th International Conference on Real-Time Systems*.

Graham, 1976. Graham, R. (1976). Bounds on the performance of scheduling algorithms. *Computer and job scheduling theory*, pages 165–227.

Gresser, 1993. Gresser, K. (1993). An event model for deadline verification of hard real-time systems. In *Real-Time Systems. 1993. Proceedings. Fifth Euromicro Workshop on*, pages 118–123. IEEE.

Guan and Yi, 2014a. Guan, N. and Yi, W. (2014a). General and efficient response time analysis for edf scheduling. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*.

Guan and Yi, 2014b. Guan, N. and Yi, W. (2014b). General and efficient response time analysis for edf scheduling. In *Proceedings of the Conference on Design, Automation & Test in Europe, DATE ’14*, pages 255:1–255:6. 3001 Leuven, Belgium, Belgium. European Design and Automation Association.

Ittershagen et al., 2013. Ittershagen, P., Hartmann, P. A., Grußtner, K., and Rettberg, A. (2013). Hierarchical real-time scheduling in the multi-core era–an overview. In *Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2013 IEEE 16th International Symposium on*, pages 1–10. IEEE.
Joseph and Pandya, 1986. Joseph, M. and Pandya, P. (1986). Finding response times in a real-time system. The Computer Journal, 29(5):390–395.

Künzli et al., 2007. Künzli, S., Hamann, A., Ernst, R., and Thiele, L. (2007). Combined approach to system level performance analysis of embedded systems. In Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis, CODES+ISSS ’07, pages 63–68, New York, NY, USA. ACM.

Lehoczky, 1990. Lehoczky, J. P. (1990). Fixed priority scheduling of periodic task sets with arbitrary deadlines. In Real-Time Systems Symposium, 1990. Proceedings., 11th, pages 201–209.

Leung and Whitehead, 1982. Leung, J. Y.-T. and Whitehead, J. (1982). On the complexity of fixed-priority scheduling of periodic, real-time tasks. Performance evaluation, 2(4):237–250.

Lipari and Bini, 2005. Lipari, G. and Bini, E. (2005). A methodology for designing hierarchical scheduling systems. Journal of Embedded Computing, 1(2):257–269.

Liu and Layland, 1973. Liu, C. L. and Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. J. ACM, 20:46–61.

Mindell, 2008. Mindell, D. (2008). Digital Apollo: Human and Machine in Spaceflight. Inside Technology Series. MIT Press.

Mok and Chen, 1997. Mok, A. K. and Chen, D. (1997). A multiframe model for real-time tasks. Software Engineering, IEEE Transactions on, 23(10):635–645.

Moyo et al., 2010. Moyo, N. T., Nicollet, E., Lafaye, F., and Moy, C. (2010). On schedulability analysis of non-cyclic generalized multiframe tasks. In Real-Time Systems (ECRTS), 2010 22nd Euromicro Conference on, pages 271–278.

Naedele et al., 1998. Naedele, M., Thiele, L., and Eisenring, M. (1998). Characterising variable task releases and processor capacities. Technical Report 45, Computer Engineering and Networks Laboratory, ETH Zurich.

Palencia and Harbour, 1998. Palencia, J. C. and Harbour, M. G. (1998). Schedulability analysis of static tasks with static and dynamic offsets. In Real-Time Systems Symposium, 1998. Proceedings.. 1998 IEEE, pages 26–37.

Palencia and Harbour, 2003. Palencia, J. C. and Harbour, M. G. (2003). Offset-based response time analysis of distributed systems scheduled under edf. In Proceedings of the Proceedings. 15th Euromicro Conference on Real-Time Systems.

Palencia and Harbour, 2005. Palencia, J. C. and Harbour, M. G. (2005). Response time analysis of edf distributed real-time systems. In Journal of Embedded Computing, April, Nr. 2.

Richter, 2005. Richter, K. (2005). Compositional Scheduling Analysis Using Standars Event Models. PhD thesis, TU Braunschweig.

Spuri, 1996. Spuri, M. (1996). Analysis of deadline scheduled real-time systems. PhD thesis, Inria.

Stigge et al., 2011. Stigge, M., Ekberg, P., Guan, N., and Yi, W. (2011). The digraph real-time task model. In Real-Time and Embedded Technology and Applications Symposium (RTAS), 2011 17th IEEE, pages 71–80.

Stigge and Yi, 2013. Stigge, M. and Yi, W. (2013). Combinatorial abstraction refinement for feasibility analysis. In Proceedings of the 34th IEEE Real-Time Systems Symposium (RTSS).

Thiele et al., 2002. Thiele, L., Chakraborty, S., Gries, M., and Künzli, S. (2002). A framework for evaluating design tradeoffs in packet processing architectures. In Design Automation Conference, 2002. Proceedings. 39th, pages 880 – 885.

Thiele et al., 2001. Thiele, L., Chakraborty, S., Gries, M., Maxiaguine, A., and Greuter, J. (2001). Embedded software in network processors - models and algorithms. In Henzinger, T. and Kirsch, C., editors, Embedded Software, volume 2211 of Lecture Notes in Computer Science, pages 416–434. Springer Berlin / Heidelberg.

Thiele et al., 2000. Thiele, L., Chakraborty, S., and Naedele, M. (2000). Real-time calculus for scheduling hard real-time systems. In International Symposium on Circuits and SystemsISCAS 2000, volume 4, pages 101–104, Geneva, Switzerland.

Tindell et al., 1994a. Tindell, K. W., Burns, A., and Wellings, A. J. (1994a). An extendible approach for analyzing fixed priority hard real-time tasks. Real-Time Systems, 6:133–151. 10.1007/BF01088593.
Tindell et al., 1994b. Tindell, K. W., Burns, A., and Wellings, A. J. (1994b). An extendible approach for analyzing fixed priority hard real-time tasks. *Real-Time Systems*, 6(2):133–151.

Tindell and Clark, 1994. Tindell, K. W. and Clark, J. (1994). Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and Microprogramming*, 40(2-3):117 – 134. <ce:title>Parallel Processing in Embedded Real-time Systems</ce:title>.

Wandeler and Thiele, 2006. Wandeler, E. and Thiele, L. (2006). Interface-based design of real-time systems with hierarchical scheduling. In *12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS’06)*, pages 243–252. IEEE.

W.Tindell, 1994. W.Tindell, K. (1994). Adding time-offsets to schedulability analysis. Technical report, Department of Computer Science, University of York.

Zhu et al., 2011. Zhu, H., Goddard, S., and Dwyer, M. B. (2011). Response time analysis of hierarchical scheduling: The synchronized deferrable servers approach. In *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*, pages 239–248. IEEE.