Modular Performance Analysis of Cyclic Dataflow Graphs

Lothar Thiele, Nikolay Stoimenov

TIK-Report No. 306

Computer Engineering and Networks Laboratory,
Swiss Federal Institute of Technology ETH Zurich, 8092 Zurich, Switzerland,
thiele@tik.ee.ethz.ch, stoimenov@tik.ee.ethz.ch

June 2009

Abstract

Applications for parallel and distributed embedded systems are often specified as dataflow graphs with dependency cycles. Examples of corresponding models of computation are marked graphs or synchronous data flow graphs. Performance analysis is often used in the exploration of different implementation alternatives or in order to provide guarantees on the timing behavior. This paper describes a new approach to the modular performance analysis of cyclic dataflow graphs such as SDF graphs as existing component-based analysis methods are not able to faithfully deal with cycles in the event flow. The new method results in tight bounds on essential quantities like buffer sizes, end-to-end delays and throughput. Because of the generality of the approach, one can analyze not only systems that can be modeled as marked graphs but also implementations that contain buffers with finite sizes, that produce system-wide back-pressure caused by blocking write semantics. The embedding of the novel approach into a modular performance analysis method allows the analysis of distributed implementations that use resource sharing mechanisms such as fixed-priority scheduling and TDMA. The paper presents the new models and methods as well as experimental results.

1 Introduction

Applications that are implemented on distributed embedded systems can often be specified using dataflow graphs where nodes correspond to processes, and edges correspond to communication channels with a first in first out (FIFO) buffer semantics. In particular, this observation holds if the underlying algorithms perform computations on streaming data which is common for control-, media-, signal-, image- and transceiver-applications. This model of computation has received a lot of interest in the past as it naturally fits to distributed implementations, for example heterogeneous multiprocessors, MPSoC (multiprocessors on a chip) and large scale distributed systems in automotive and avionics. There are several subclasses of dataflow models such as Kahn Process Networks, Marked Graphs [20], and Synchronous Dataflow Graphs [16], for an overview see [17]. Many results are available concerning their deadlock behavior, schedulability, and mapping onto multiprocessor systems [2, 25].

The performance analysis of applications that have been mapped onto distributed or parallel computation and communication platforms has received much attention recently, see e.g. [5, 12, 21, 27]. It enables the analysis of essential system characteristics such as end-to-end delays, upper bounds on buffer spaces, and throughput. It

is based on information about the worst-case execution times, communication times, and the resource sharing strategies. The formal analysis can be used for design space exploration, e.g. binding of processes to computing resources, mapping of channels to communication paths, and selecting scheduling strategies, or for final verification of system properties after the design step.

In many of the above mentioned application domains we are faced with applications that contain cyclic dependence behavior where the result of a certain process output may depend on previous outputs of the same process, possibly transformed via a set of intermediate processes. Such applications exhibit iterative behavior that is combined with loop carried dependencies. Another prominent example is related to the use of finite buffers in the implementation of a given application which is usually modeled as a one which has infinite buffers but contains additional cyclic dependencies.

However, the analysis of cycles in the dataflow of applications poses tremendous difficulties for performance analysis, in particular for any modular and component-based approach. The cycles in the information flow between the individual processes of the application lead to global, system-wide state dependencies. As a result, the timing behavior of a process (and as a result its use of the available resources) not only depends on predecessor processes that provide the data streams that are to be processed, but also on successor processes and the process itself. A typical special case is the use of finite buffers with blocking write semantics: If they are full, they put back-pressure to preceding process executions and may cause a system-wide slow-down or even blocking. Ignoring dependency cycles, for example by just cutting them or by replacing finite buffers by infinite ones, leads to unsafe performance analysis results.

Following the above discussion, there is a need for extending the model of computation that can be handled efficiently by modular component-based performance analysis methods towards cyclic behavior in the event flow.

1.1 Related Work

The present paper specifically deals with a subclass of dataflow graphs called marked graphs, see e.g. [20, 23]. They are characterized by the fact that each process can fire if there is at least one token in each input queue and the firing adds one output token to each output queue. For graphs where each process has a fixed delay (execution time), there are many results in the literature that characterize the timing behavior. They all suppose that there is a fixed deterministic processing time for tokens in each node. Early results in [23] have been generalized and connected to eigenvalue problems in max/plus algebra, see [1, 7] and more recently [10]. The results are not directly applicable to more complex interaction with the resources as envisioned in this paper: non-deterministic delays, various resource sharing mechanisms such as TDMA and fixed priority, non-deterministic timing behavior of input streams.

The class of synchronous dataflow graphs (SDF) has been introduced in [16] as an untimed model. Unlike marked graphs, they are characterized by fixed token consumption and production rates other than 1. Because of the practical importance of this model of computation, many results are available that describe properties of an implementation on single or multiple processors. The processes in an SDF graph, also called actors, are annotated with execution times for analysis [25]. The above mentioned restrictions on the scope of the performance analysis for marked graphs also hold here.

Very often, SDF graphs are converted to equivalent marked graphs, also denoted homogeneous SDF graphs (HSDF) [25], for the purpose of performance analysis. The same method can be used by the analysis framework described in this paper. Therefore, the new results can be generalized to the class of SDF graphs as well.

Acyclic dataflow graphs with fixed token consumption and production rates of processes as well as finite buffer capacities can be modeled as SDF graphs by adding to each edge with finite buffer an edge in the opposite direction which represents the available capacity. Based on this concept, there have been several results based on the classical delay models, e.g. computing buffer sizes under throughput constraints, see [29], and computing throughput while respecting sequence constraints by additional edges, see [22]. In all of these cases, resources are not

explicitly modeled and therefore: (a) only limited resource sharing methods can be analyzed, and (b) modularity and composability is limited.

More complex interactions between resources and process executions can be faithfully modeled using the closely related concepts of dioids [1] and network calculus [6, 8, 15]. The concepts of arrival and service curves allow a much more general modeling of the system environment and has been applied to model communication networks. In [3], results from [6] have been applied to chains of processes with finite buffer sizes. Recently, very similar results have been described in [4]. These results are restricted only to systems with finite buffer sizes, use course-grained approximations of resources as the maximal processing or communication capability is set to infinity, and they are not compositional in terms of resources, i.e. they are pessimistic for systems with resource sharing.

The SymTA/S approach [12] has been extended towards cyclic data dependencies in [13]. The analysis is based on classical real-time analysis, i.e. worst-case response times. By iterating relations for individual processes, the overall system behavior is obtained, see also [27]. The approach is limited in terms of traffic models, i.e. periodic with jitter and bursts, as well as in terms of resource models. Recently, the SymTA/S approach has been extended towards the class of SDF graphs, see [24]. Here, an SDF application is encapsulated by providing input and output interfaces to it. The analysis of the dataflow graph itself is based on the classical results described above as well as simulation. Therefore, they are restricted to simple delay-based resource interactions with upper and lower bounds on the execution times.

Recently, the above models and methods have been generalized to distributed real-time systems, denoted as real-time calculus (RTC) and modular performance analysis (MPA) [5, 28]. The method allows to consider complex communication and computation resource models, scheduling policies such as fixed priority, EDF, TDMA and hierarchical using servers. On the other hand, complex state-dependent behavior such as cyclic data-dependencies as in marked graphs can not be modeled as well as implementations with finite buffer sizes. Recently, there have been extensions towards cyclic *resource* dependencies [14], which do not extend directly to cyclic dataflow dependencies as described in this paper. In addition, the MPA framework has been extended towards AND/OR activations as described in [11] which are essential components in dataflow graph representations however, cycles in the dataflow have not been tackled.

1.2 Contributions

In summary, the problem statement can be formulated as follows: Given a set of marked graphs as depicted in Figure 1 that share a set of computation and communication devices by means of e.g. fixed priority scheduling or TDMA (time division multiple access) and show a complex interaction with the environment. MG1 (marked graph 1) has bounded buffers between processes $v_1 \rightarrow v_2$ and $v_2 \rightarrow v_3$ limited to a maximal size of 1 and 2, respectively. Determine essential system characteristics such as end-to-end delays, buffer sizes and throughput of the whole system.

The paper presents the following new results:

- Modular performance analysis methods are extended to the class of marked graphs. Unlike other known methods, the approach takes into account a general model for resource interaction based on the concept of service curves that covers 'periodic' and 'bounded delay' resource models as special cases, and a general stream model that covers 'periodic', 'periodic with jitter/bursts' and 'sporadic' as special cases.
- The analysis covers systems with cyclic data dependencies, finite buffer sizes, non-deterministic resource behavior, TDMA and fixed priority scheduling policies. It can also be embedded into compositional frameworks such as SymTA/S or MPA.
- Performance bounds are obtained by using upper and lower bound representations which yield higher accuracy than known methods.

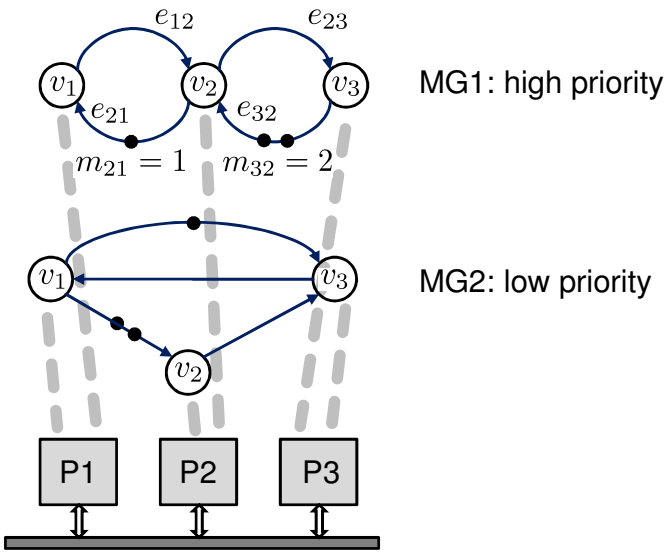

Figure 1: Visualization of problem statement where two marked graphs are mapped to a distributed platform and the individual processes share the available resources using some scheduling scheme.

- Experimental results are provided that show the applicability of the new method to selected case studies and the advantages with respect to known approaches in terms of accuracy of the results.

The paper describes a stepwise abstraction that leads from a characterization of a marked graph in time domain to an abstract representation in *time interval* domain which is then used to (a) determine essential performance indicators and to (b) embed the analysis into a compositional framework. Section 2 contains the time domain characterization and introduces the essential notation of a service function to describe resources. Section 3 introduces an abstraction of the service function, i.e. it represents resource capabilities in the time interval domain and analyzes marked graphs under this abstraction. Section 4 introduces the final abstraction, namely the representation of data streams in the time interval domain which is the main prerequisite for modular performance analysis. Section 5 contains the experimental results that show the applicability and tightness of the analysis.

2 Model Definition

In this section, we will define the basic elements of the analysis framework. The analyzed system will be modeled as a marked graph, i.e. as a set of processes that (a) communicate via FIFO buffers with unlimited capacity and (b) at the time of firing, consume and produce one token at any input and output, respectively. Finite size buffers will be modeled using cycles in the dataflow graph.

2.1 Dataflow Graph

Let us first define a generic dataflow graph, i.e. the basic underlying model of the forthcoming analysis, see also Figure 1.

Definition 2.1. *A dataflow graph (V, E, M) is defined as a set of processes $v \in V$ and a set of channels $e \in E$ where $E \subseteq V \times V$. To each channel there is associated a number of initial token $M : E \to \mathbb{R}^{\geq 0}$, i.e. $m_{ij} \in \mathbb{R}^{\geq 0}$ denotes the number of token associated to channel $e_{ij} = (v_i, v_j)$ connecting process $v_i \in V$ with $v_j \in V$.*

The term 'token' is used in a very general sense. It should be interpreted as any amount of data, not necessarily integer. This way, we will be able to model systems in a flow-based as well as in a discrete-event setting.

It will be useful to assign input and output ports to each process $v_i \in V$. We denote the input port of v_i associated to channel $e_{ji} = (v_j, v_i)$ as (j, i) and the output port associated to $e_{ik} = (v_i, v_k)$ as (i, k).

2.2 Arrival Functions

The timing properties of an event stream can be described using the concept of an arrival function R: $R(t) \in \mathbb{R}^{\geq 0}$ which denotes the number of token that arrived in the time interval $[0, t)$, $t > 0$, and $R(0)$ denotes the initial number of token in the stream.

It will be useful for the analysis if we partially order the set of all arrival functions. In particular, we say that $R \geq R'$ if and only if $R(t) \geq R'(t)$ for all $t \geq 0$. If we are dealing with n-dimensional vectors of arrival functions $R = (R_i : i = 1, ..., n)$, then we say that $R \geq R'$ if and only if $R_i(t) \geq R'_i(t)$ for all $t \geq 0$, $i = 1, ..., n$.

It is known from lattice theory, see e.g. [9], page 63, that the set of arrival functions ordered by $\geq$ as defined above forms a complete lattice. The bottom $\perp$ and top $\top$ element of the set are defined as 0 and ∞ for all $t \geq 0$, respectively, where $\top \geq R \geq \perp$ for all arrival functions R. The 'complete' means that there exists a least upper bound and a greatest lower bound for each finite or infinite subset of arrival functions.

Example 2.2. *Figure 2 shows two examples of arrival functions. R_1 represents a periodic arrival pattern of discrete tokens with period p and R_2 represents a continuous flow with rate ρ/σ. In both cases, the streams of token start at time τ.*

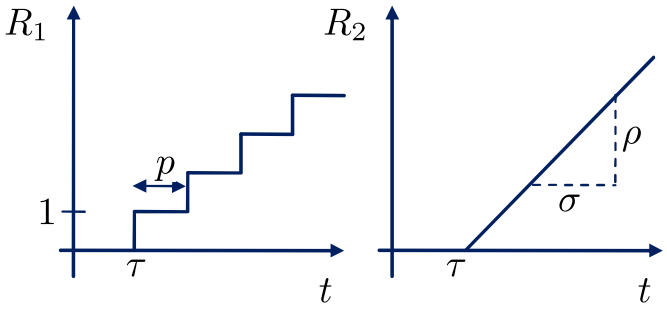

Figure 2: Two simple arrival functions.

2.3 Processes and Mappings

The operation of a single process v_i can be described as the mapping from a vector of input arrival functions to a vector of output arrival functions. The input arrival function R^{in}_{ji} is associated to an input port (j, i) of v_i and the output arrival function R^{out}_{ik} is associated to an output port (i, k).

Definition 2.3. *A process $v_i \in V$ with n input ports and m output ports maps an n-dimensional vector of input arrival functions R^{in} to an m-dimensional vector of output arrival functions R^{out} by means of a deterministic mapping Π_i, i.e. $R^{out} = \Pi_i \circ R^{in}$ where $R^{in} = (R^{in}_{ji} : e_{ji} \in E)$ and $R^{out} = (R^{out}_{ik} : e_{ik} \in E)$. We will also call the mapping Π_i, the transfer function of process v_i.*

In the following, we will restrict ourselves to the class of monotone processes. Loosely speaking, if we consider two distinct traces and we feed more token to a process in one of them ($\overline{R^{in}} \geq R^{in}$), than the process does produce at least as many output token as for the other one ($\overline{R^{out}} \geq R^{out}$).

Definition 2.4. *A monotone process Π satisfies: $\overline{R} \geq R \Rightarrow \Pi \circ \overline{R} \geq \Pi \circ R$.*

Note that not all possible processes satisfy this condition. Nevertheless, a large class of interesting processes are monotone, e.g. the considered class of marked graphs.

Example 2.5. *A simple process type is denoted as AND. It fires immediately when there are token available at all of its input ports. If we restrict it to two inputs R_1^{in} and R_2^{in}, we find its transfer function as*

$$\textit{AND:} \quad R^{out}(t) = \min\{R_1^{in}(t), R_2^{in}(t)\}. \tag{1}$$

2.4 Service Function and Greedy Processing Components (GPC)

The elementary process described in the above example does not interact with available resources at all. On the other hand, it would be highly desirable to express the fact, that a process may need resources in order to operate on available input token. The concept of a service function C allows us to describe the availability of a resource (such as a processor or a communication device). $C(t) \in \mathbb{R}^{\geq 0}$ denotes the number of token that can be processed in the time interval $[0, t)$, $t > 0$ where $C(0) = 0$. In this paper, the unit of the service function is the same as the one of the arrival function, more general concepts for characterization of these units are described in [18].

Example 2.6. *Note that the concept of service functions allows us to model any complex resource behavior, i.e. the resource may be available with a resource rate of 1 token unit in $[0, t_1)$ and not available in $[t_1, t_2)$ which is the case when another task is running on the resource or other data are communicated, or the time slot allocated to the process has finished. This is expressed with $C(t) = 1$, $0 \leq t < t_1$ and $C(t) = 0$, $t_1 \leq t < t_2$.*

Now, let us consider a component with a single input which uses a resource. It takes an input arrival function $R_{in}(t)$ and produces an output arrival function $R^{out}(t)$ by means of a service function $C(t)$. Input token are processed, always when there are resources available. Therefore, we call the corresponding process a greedy processing component (GPC).

Definition 2.7. *A greedy processing component (GPC) with service function C is defined by the transfer function*

$$\textit{GPC:} \quad R^{out}(t) = \inf_{0 \leq \lambda \leq t} \{R^{in}(\lambda) + C(t) - C(\lambda)\}. \tag{2}$$

The remaining unused service from such a component is given by

$$C'(t) = C(t) - R^{in}(t). \tag{3}$$

The above definition can be related to the intuitive notion of a greedy process as follows: The output between some time λ and t can not be larger than the available service: $C(t) - C(\lambda)$, and therefore, $R^{out}(t) \leq R^{out}(\lambda) + C(t) - C(\lambda)$. As the component can not output more than what was available at the input, we have $R^{out}(\lambda) \leq R^{in}(\lambda)$ and therefore, $R^{out}(t) \leq R^{in}(\lambda) + C(t) - C(\lambda)$. There is some last time λ^* before t when the buffer was empty. At λ^*, we clearly have $R^{out}(\lambda^*) = R^{in}(\lambda^*)$. In the interval from λ^* to t, the buffer is never empty and all available resources are used to produce output token: $R^{out}(t) = R^{out}(\lambda^*) + C(t) - C(\lambda^*) = R^{in}(\lambda^*) + C(t) - C(\lambda^*)$. As a result, we obtain (2).

Note that the above resource and timing semantics model almost all practically relevant processing and communication components, e.g. processors that operate on tasks and use queues to keep ready tasks, communication networks and buses, etc. As a result, we are not restricted to modeling processing time with a fixed delay. The service function can be chosen to represent a resource that is available only in certain time intervals (e.g. TDMA scheduling) or which is the remaining service after a resource has been used for other higher priority tasks (e.g. fixed priority scheduling).

Following the above results, we can define the notion of a *Greedy Marked Graph Process*, and say that an activated Greedy Marked Graph Process simultaneously removes token from each input channel and adds token

to each output channel with a rate that is determined by the available service. A Greedy Marked Graph Process is activated if there is a positive number of token in each input channel.

Using the above characterization and (1), (2), it follows that a Greedy Marked Graph Process can be modeled as a concatenation of an AND and a GPC component as shown in Figure 3.

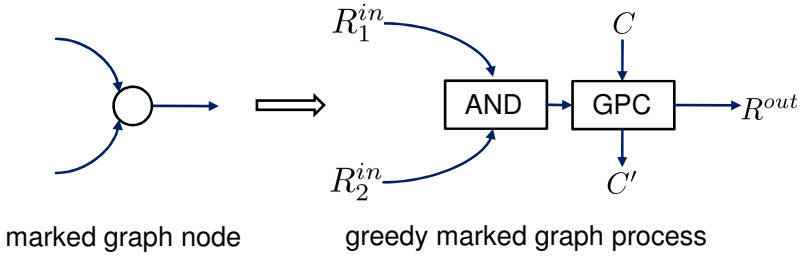

Figure 3: Marked graph node and its representation as a greedy marked graph process.

2.5 Execution Semantics of Marked Graphs with Greedy Processes

In this section, we move one step further towards the performance analysis of marked graphs with cyclic dependencies. To this end, we first define the operation of a network of greedy process nodes using fixed points of a system equation. Note, that we are still describing the operation of the marked graph in time domain, i.e. without any sort of abstraction.

In order to determine the semantics of a marked graph, we will derive a set of system equations. To this end, let us first define a step function with height s:

$$I^s(t) = \begin{cases} 0 & \text{if } t = 0 \\ s & \text{if } t > 0 \end{cases}$$

If we now look at the semantics of a channel containing s initial tokens, it provides at its output as many tokens as have been submitted to its input plus the number of initial tokens s.

Now, we can set up a set of equations that describe the semantics of a whole marked graph (V, E, M)

$$(R_{ik}^{out} : e_{ik} \in E) = \Pi_i \circ (R_{ji}^{in} : e_{ji} \in E) \quad \forall v_i \in V, \tag{4}$$

$$R_{ij}^{in} = R_{ij}^{out} + I^{m_{ij}} \quad \forall e_{ij} \in E, \tag{5}$$

where Π_i denotes the input-output transfer function of a single greedy marked graph process v_i. If we combine (4) and (5), we get a single equation of the form

$$R = \Pi \circ R, \tag{6}$$

where $R = (R_{ij}^{out} : e_{ij} \in E)$ is a vector of arrival functions that contains as elements all output arrival functions of processes and Π is the combined mapping of the whole dataflow graph. Note that, the combined mapping Π is monotone if all process mappings Π_i, $v_i \in V$ are monotone.

In order to solve (6), we can use results from lattice theory, see [9], page 187. It follows that if the mapping Π is monotone, then the fixed-point equation (6) has a least and a greatest fixed-point, R^l and R^u, respectively.

We can strengthen this result by assuming δ-causality for all processes of a dataflow graph, i.e. changes at the input of a process are not visible before a (small) time lag $\delta > 0$: if $R(s) = R'(s)$ for all $s \leq t - \delta$ then we have $(\Pi \circ R')(t) = (\Pi \circ R)(t)$. Then we can determine all solutions of (6) inductively, starting from the initial conditions at $t = 0$. As the mappings of the processes are deterministic, the solution to (6) is unique.

Example 2.8. *Let us look at the simple dataflow graph MG1 shown in Figure 1 and determine the corresponding mapping $R = \Pi \circ R$ by concatenating (1) and (2):*

$$R^{out}_{1,2}(t) = \inf_{0\le\lambda\le t}\{R^{out}_{2,1}(\lambda) + I^1(\lambda) + C_1(t) - C_1(\lambda)\},$$
$$R^{out}_{2,3}(t) = \inf_{0\le\lambda\le t}\{\min\{R^{out}_{3,2}(\lambda) + I^2(\lambda), R^{out}_{1,2}(\lambda)\} + C_2(t) - C_2(\lambda)\},$$
$$R^{out}_{3,2}(t) = \inf_{0\le\lambda\le t}\{R^{out}_{2,3}(\lambda) + C_3(t) - C_3(\lambda)\},$$
$$R^{out}_{2,1}(t) = \inf_{0\le\lambda\le t}\{\min\{R^{out}_{3,2}(\lambda) + I^2(\lambda), R^{out}_{1,2}(\lambda)\} + C_2(t) - C_2(\lambda)\},$$

where the resources available to v_1, v_2 and v_3 are described by the service functions $C_1(t)$, $C_2(t)$ and $C_3(t)$, respectively. The functionality corresponds to a simple processing chain with finite buffer sizes of 1 and 2, respectively.

3 Resource Abstraction and System Equations

In order to arrive at efficient methods for the modular performance analysis of marked graphs, we will need to introduce several abstractions. Instead of calculating the resulting arrival functions for a single service function $C_i(t)$ in time domain, we will use upper and lower bounds on $C_i(t)$. This will enable us to consider a wide class of processes and process characteristics as well as to derive computationally feasible analysis methods that provide statements about the behavior of a system under a whole set of resource behaviors. This first step introduces non-determinism as the service function is not provided explicitly anymore, but only bounds on it.

3.1 Service Curves and GPC Abstractions

Following the ideas of network calculus [8, 15], we define upper and lower bounds on service functions, denoted as *service curves*. This way, we abstract from the concrete time domain and operate in the *time interval domain*.

Definition 3.1. *Upper and lower service curves, β^u and β^l, map positive time intervals $\Delta \in \mathbb{R}^{\ge 0}$ to the maximal and minimal amount of available resources in any time interval of length Δ. They satisfy $\beta^u(0) = \beta^l(0) = 0$ and*

$$\beta^l(\Delta) \le C(t+\Delta) - C(t) \le \beta^u(\Delta) \quad \forall t \ge 0, \Delta > 0.$$

Example 3.2. *Figure 4 shows three examples of service curves that model (a) a fully available resource that leads to a delay of τ for each unit of input token, (b) a TDMA resource that is available only in periodically repeating time slots and (c) a service curve that models a locally synchronous behavior with cycle time τ, i.e. every τ an input token unit can be processed.*

Now, we can upper and lower bound the mapping of a GPC as given in (2) by

$$R^{out}(t) \le \inf_{0\le\lambda\le t}\{R^{in}(\lambda) + \beta^u(t-\lambda)\},$$
$$R^{out}(t) \ge \inf_{0\le\lambda\le t}\{R^{in}(\lambda) + \beta^l(t-\lambda)\}.$$

Using the Min-plus algebra convolution operator

$$(a \otimes b)(\Delta) = \inf_{0\le\lambda\le\Delta}\{a(\lambda) + b(\Delta-\lambda)\}, \tag{7}$$

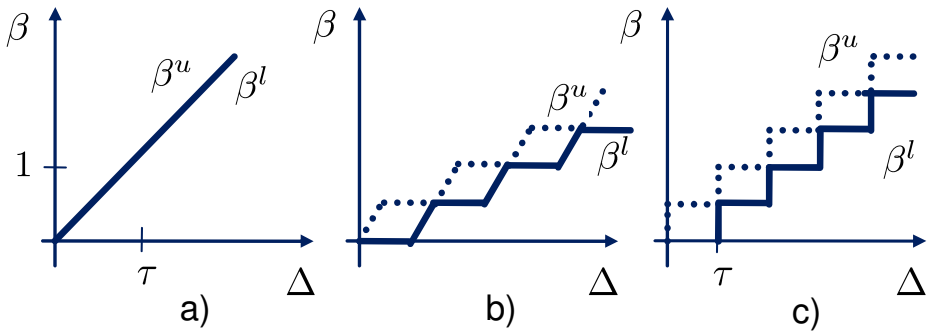

Figure 4: Three examples of service curves.

we obtain a more concise notation, see also [6, 8, 15]:

$$R^{in} \otimes \beta^l \leq R^{out} \leq R^{in} \otimes \beta^u. \tag{8}$$

In other words, for a single GPC component we can bound the number of token that arrive in $[0, t)$ by abstracting the available service using β^u and β^l. The next step is to apply this abstraction to the whole marked graph. As we will see, we then get upper and lower bounds on the number of token that arrive in any channel in the graph.

3.2 Bounds for the Marked Graph and System Equations

So far, the (concrete) execution semantics of a marked graph has been described by the single equation (6). Now, we will investigate the influence of the resource abstraction introduced in (8).

The approach is based on replacing the mapping Π of the whole marked graph by 'larger' and 'smaller' mappings Π. Then the resulting arrival functions R provide upper and lower bounds on the system behavior, respectively. We say that a mapping Π^u is larger or equal than a mapping Π if the relation holds pointwise or more generally:

$$\begin{aligned} \Pi^u \geq \Pi \quad &\Leftrightarrow \quad \Pi^u \circ R \geq \Pi \circ R \quad \forall R, \\ \Pi^l \leq \Pi \quad &\Leftrightarrow \quad \Pi^l \circ R \leq \Pi \circ R \quad \forall R. \end{aligned}$$

This leads us to the following result:

Theorem 3.3. *Let $\mathcal{R}$ be a complete lattice. Let be given a monotone mapping Π with a unique fixed-point $R \in \mathcal{R}$. Define R^l and R^u to be the greatest and least fixed-point of $R^l = \Pi^l \circ R^l$ and $R^u = \Pi^u \circ R^u$, respectively. Then*

$$R^l \leq R \leq R^u.$$

Proof. Under the assumptions of the theorem, we find that the smallest fixed-points of $\underline{R} = \Pi \circ \underline{R}$ and $R^u = \Pi^u \circ R^u$ satisfy $\underline{R} \leq R^u$, see [9], page 199. As we have $R \geq \underline{R}$ for all R that satisfy $R = \Pi \circ R$, we find $R \geq \underline{R} \leq R^u$. As the fixed-point of Π is unique, we finally get $R = \underline{R} \leq R^u$. The proof for R^l is similar. □

As a result of the theorem one can directly show that $R^l \leq R \leq R^u$ holds for any fixed-point of $R^l = \Pi^l \circ R^l$ and $R^u = \Pi^u \circ R^u$.

In other words, if we replace the mapping of the dataflow graph by one that is either not smaller or not larger, then we get upper or lower bounds on the arrival functions, i.e. on the number of token that passed through the

processing elements at each moment in time. This result will now be used in order to replace the service functions $C(t)$ by their abstractions, the service curves $\beta^u(\Delta)$ and $\beta^l(\Delta)$.

To this end, we will determine the above mappings Π^u and Π^l explicitly from the given marked graph structure. As a result, we obtain abstract system equations whose solutions yield upper and lower bounds on the behavior of any marked graph.

Starting point is again the modeling of each process of a marked graph by a Greedy Marked Graph Process, see also (1), (2), (8) and Figure 3.

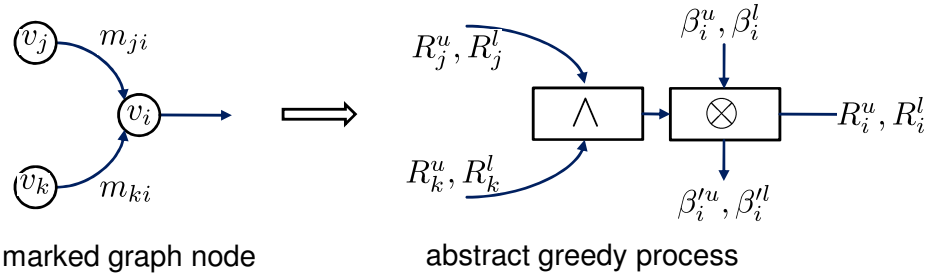

Figure 5: Marked graph node with m_{ji}, m_{ki} initial tokens on edges e_{ji}, e_{ki} and its abstract representation.

Using the notation introduced so far and the minimum-operator $\wedge$ with $a \wedge b = \min\{a, b\}$, we obtain for the simple two input-case as depicted in Figure 5

$$R_i = [(R_j + I^{m_{ji}}) \wedge (R_k + I^{m_{ki}})] \otimes \beta_i, \tag{9}$$

where β_i can be replaced by β_i^u and β_i^l in order to obtain the equations for the upper and lower bounds R_i^u and R_i^l, respectively, where $R_i^u \geq R_i \geq R_i^l$. Using elementary calculus, we can reformulate the above equation to

$$R_i = [(R_j + I^{m_{ji}}) \otimes \beta_i] \wedge [(R_k + I^{m_{ki}}) \otimes \beta_i],$$

and finally,

$$R_i = \beta_i \wedge [R_j \otimes (\beta_i + m_{ji})] \wedge [R_k \otimes (\beta_i + m_{ki})].$$

For the first main result of this paper, we will make use of the matrix notation $S = (S_{ij})$ (S contains elements S_{ij}), the vector notations $R = (R_i)$ and $\beta = (\beta_i)$ as well as the the matrix product $C = A \otimes B$ with $c_{ij} = \bigwedge_{(k)}(a_{ik} \otimes b_{kj})$. Note again the definition of $\otimes$ in (7) and $a \wedge b = \min\{a, b\}$.

Theorem 3.4. *Given a marked graph (V, E, M) and service curves β^u, β^l associated to its nodes that describe bounds on the corresponding available resources, see Definition 3.1. Define the upper and lower system matrices of the graph as $S^{u,l} = (S_{ij}^{u,l})$ with*

$$S_{ij}^{u,l} = \begin{cases} \beta_i^{u,l} + m_{ji} & e_{ji} \in E \\ \infty & e_{ji} \notin E \end{cases} \tag{10}$$

Then we can write the system equations for the marked graph as

$$R^u = \beta^u \wedge S^u \otimes R^u, \tag{11}$$

$$R^l = \beta^l \wedge S^l \otimes R^l, \tag{12}$$

where R^u and R^l denote upper and lower bounds on any vector of execution traces of the marked graph with

$$R^u \geq R \geq R^l. \tag{13}$$

3.3 Solving the System Equation

Finally, we need to determine solutions to (11) and (12) in order to determine bounds on the event sequences between the processes, i.e. tight bounds on the vector of arrival functions R in (13).

To this end, we make use of the corresponding results for distributive dioids as described in [1], page 193. All solutions to (11, 12) can be determined as

$$R = y \wedge S^* \otimes \beta \qquad \forall y \; : \; y = S \otimes y, \tag{14}$$

where for simplicity we omit the superscripts u or l that relate to (11) or (12), respectively. The matrix S^* denotes the min-closure of S which is defined as

$$S^* = \bigwedge_{k=0}^{\infty} S^{(k)}, \tag{15}$$

where $S^{(k)} = S \otimes S^{(k-1)}$ for $k \geq 1$ and

$$S^{(0)} = \begin{pmatrix} I^\infty & \infty & \cdots \\ \infty & I^\infty & \cdots \\ \cdots & \cdots & \ddots \end{pmatrix}$$

Investigating the structure of the S^* more closely yields the following interpretation: *An element S^*_{ji} of S^* is the minimal 'path length' of all (including cyclic) paths from node i to node j in the marked graph. The 'path length' is defined as the sum of all tokens along the path plus the convolution of all service curves on the path, except that of node i. If $i = j$, then the value of $S^*_{ii}(0)$ is set to 0.* We will come back to the structure of S^* in more detail in Section 3.4.

In order to determine as tight bounds as possible, we should find now the *least fixed-point* of $R^u = \beta^u \wedge S^u \otimes R^u$ and the *greatest fixed-point* of $R^l = \beta^l \wedge S^l \otimes R^l$.

The greatest solution to $R^l = \beta^l \wedge S^l \otimes R^l$ is simply obtained as $R^l = (S^l)^* \otimes \beta^l$, see [1], page 192. In order to determine the least solution to $R^u = \beta^u \wedge S^u \otimes R^u$, we need to determine the least y with $y = S^u \otimes y$, i.e. $\underline{y} = \inf\{y \; : \; y = S^u \otimes y\}$.

The least fixed-point of $y = S^u \otimes y$ with $\underline{y} = \inf\{y \; : \; y = S^u \otimes y\}$ is given by $\underline{y} = \lim_{k\to\infty}((S^u)^{(k)} \otimes \bot)$ where $\bot(t) = 0$ for all $t \geq 0$. This result can easily be shown using elementary techniques from lattice theory, see [9], and by noting that S^u is monotone. This is the result of the following theorem.

Theorem 3.5. *The least fixed-point of $y = S \otimes y$ with $\underline{y} = \inf\{y \; : \; y = S \otimes y\}$ is given by*

$$\underline{y} = \lim_{k\to\infty} (S^{(k)} \otimes \bot).$$

Proof. Let us consider the sequence $y_0 = \bot$, $y_1 = S \otimes \bot$, $y_2 = S^{(2)} \otimes \bot$, ... Then we can easily see that it is increasing as $y_1 = S \otimes \bot \geq \bot = y_0$ and $y_k = S \otimes y_{k-1} \geq S \otimes y_{k-2} = y_{k-1}$ if $y_{k-1} \geq y_{k-2}$. Here we use the fact that S is monotone. Therefore, we have $y_0 \leq y_1 \leq y_2 \leq ....$ As a result, the limit $y' = \sup_{k\to\infty}(S^{(k)} \otimes \bot)$ exists.

Now, we will show that $\underline{y} \geq y_k$ for all $k \geq 0$, i.e. the least fixed-point is lower bounded by y_k. Of course, we have $\underline{y} \geq y_0 = \bot$. Therefore, we also find $\underline{y} = S \otimes \underline{y} \geq S \otimes y_0 = y_1$ and in general $\underline{y} = S^{(k)} \otimes \underline{y} \geq S^{(k)} \otimes y_0 = y_k$. Again, we make use of the monotonicity of S. As a result we have $\underline{y} \geq y'$.

Finally, we note that $y' = \lim_{k\to\infty}(S^{(k)} \otimes \bot)$ actually is a fixed-point, i.e. $y' = S \otimes y'$ and therefore, $\underline{y} = y'$. $\square$

We can now show that for all meaningful marked graphs, we can simplify the calculation of the least fixed point and therefore, R^u. The proof uses some interpretation of $(S^u)^{(k)}$ which will be given in the next section.

Given a marked graph where the sum of initial token in each directed cycle of the network is strictly larger than 0. Then $\underline{y} = \lim_{k\to\infty}((S^u)^{(k)} \otimes \bot) = \top$, where we have $\bot(t) = 0$ and $\top(t) = \infty$ for all $t \geq 0$. More specifically, we have the following theorem.

Theorem 3.6. *Given a dataflow graph which models a marked graph. Suppose that the sum of initial token in each directed cycle of the network is strictly larger than 0. Then* $\underline{y} = \lim_{k\to\infty}((S^u)^{(k)} \otimes \bot) = \top$ *and therefore,*

$$R^u = (S^u)^* \otimes \beta^u. \tag{16}$$

Proof. If we show that $\underline{y} = \lim_{k\to\infty}((S^u)^{(k)} \otimes \bot) = \top$, then the theorem follows from (14). To this end, we proof that all elements $(S^u)^{(k)}_{ji}$ of $(S^u)^{(k)}$ will approach ∞ for $k \to \infty$. Let us distinguish between two situations. At first, there is no path containing a directed cycle in the dataflow graph between a pair of nodes i and j. Based on the result (19), we find $(S^u)^{(k)}_{ji} = \infty$ for all $k > |V|$ where $|V|$ denotes the number of nodes of the dataflow graph. Now, let us suppose that there is a path from i to j that contains a cycle. If $k > |V| \cdot K$, then any path of length k contains at least K cycles. Moreover, let δ denote the minimal sum of initial token in any cycle of the dataflow graph. Then we find using (19) that $(S^u)^{(k)}_{ji} \geq K\delta$ for $k > |V| \cdot K$. As $\delta > 0$, we find $\lim_{k\to\infty}(S^u)^{(k)}_{ji} = \infty$. □

Note that the result also shows that the fixed points of the system equations (11) and (12) are unique for both cases, $S = S^u$ and $S = S^l$, respectively, see also [6].

The main results of this section can be summarized in the following theorem.

Theorem 3.7. *Given a marked graph* (V, E, M) *and service curves* β^u, β^l *associated to its nodes. Suppose that the sum of initial token in each directed cycle of the network is strictly larger than 0. Then we can determine tight upper and lower bounds on any vector of execution traces of the marked graph* $R^u \geq R \geq R^l$ *with the arrival functions*

$$R^l = (S^l)^* \otimes \beta^l, \tag{17}$$
$$R^u = (S^u)^* \otimes \beta^u, \tag{18}$$

where we use S^u *and* S^l *from Theorem 3.4, and the corresponding closures* $(S^u)^*$ *and* $(S^l)^*$ *from (15).*

3.4 Interpretations

In this section, we will interpret the above equations (17) and (18) in terms of properties of the underlying marked graph, e.g. paths, initial tokens and service curves associated to the elementary processes.

Obviously, the matrix $S^{(n)} = \bigotimes_{0\leq k\leq n} S$ plays a central role in the solution to the system equation for a marked graph. As will be shown, there is a close relation to results in max-plus algebra, see [1], page 110.

As has been defined already in (10), the elements of the system matrix S are given as $S_{ij} = m_{ji} + \beta_i$ if edge (j, i) exists in the marked graph. Using this definition, one can now determine the elements of $S^{(n)}$ which are denoted as $S^{(n)}_{ij}$. Here, we use the following notation:

- A path p in the marked graph is a set of connected edges, i.e. $p = \{(i_0, i_1), (i_1, i_2), ..., (i_{n-1}, i_n)\}$.
- The length n of a path is defined as $n = |p|$.
- The set of nodes of a path is defined as $V(p) = \{i_0, ..., i_n\}$.

- The set of all paths of length n from node i to node j is denoted as $P^n(i,j) = \{p \ : \ (|p| = n) \wedge (i = i_0) \wedge (j = i_n)\}$.
- The set of all paths from i to j is denoted as $P(i,j)$.

Based on the definition of S and $S^{(n)} = \bigotimes_{1 \leq k \leq n} S$ we find

$$S_{ij}^{(n)} = \bigwedge_{p \in P^n(j,i)} \left[\sum_{(r,s) \in p} m_{rs} + \bigotimes_{(r,s) \in p} \beta_s \right], \tag{19}$$

for $n > 0$. If there exists no path of length n from j to i, then $S_{ij}^{(n)} = \infty$. The matrix S^+ is defined as

$$S^+ = \bigwedge_{k=1}^{\infty} S^{(k)}.$$

Using the definition of S, we find

$$S_{ij}^{+} = \bigwedge_{p \in P(j,i)} \left[\sum_{(r,s) \in p} m_{rs} + \bigotimes_{(r,s) \in p} \beta_s \right], \tag{20}$$

if there exists a path from j to i, and $S_{ij}^{+} = \infty$ otherwise. Note that the min-closure S^* can simply be determined as $S^* = S^{(0)} \otimes S^+$ (or $S_{ij}^* = S_{ij}^+$ if $i \neq j$ and $S_{ii}^* = \min\{I^\infty, S_{ii}^+\}$).

Now, we can more explicitly determine the resulting upper and lower arrival functions in the dataflow graph as stated in (17) and (18). Using elementary arithmetic, we obtain

$$R_i = \beta_i \wedge \bigwedge_{p \in P(j,i)} \left(\sum_{(r,s) \in p} m_{rs} + \bigotimes_{k \in V(p)} \beta_k \right), \tag{21}$$

where the equation holds for the upper and for the lower bounds, i.e. by adding the superscript u or l to R and β. As an interpretation one can say that R_i is the minimal function that is larger than β_i and larger than the 'path length' of any path in the marked graph that ends at node i. Here, the 'path length' is determined as the sum of all initial tokens on the path plus the convolutions of all service functions on the path, including that of the path origin.

Example 3.8. *It is useful to derive explicit formulas for special forms of service curves. The linear approximations used here are common in the analysis of networked systems, see e.g. [15]. Therefore, let us consider the special case of β_i as depicted in Figure 6, where β^l is a rate-latency function defined as*

$$\beta^l(\Delta) = \max\{0, \sigma^l \cdot (\Delta - \tau)\},$$

and β^u is a peak rate function defined as

$$\beta^u(\Delta) = \sigma^u \cdot \Delta.$$

As a shorthand notation for the above, we can also write $\beta^l = (\tau, \sigma^l)$ and $\beta^u = (\sigma^u)$.
From the definition of the convolution operator we can conclude that

$$\bigotimes_{i \in I} \beta_i^l = (\sum_{i \in I} \tau_i, \bigwedge_{i \in I} \sigma_i^l),$$

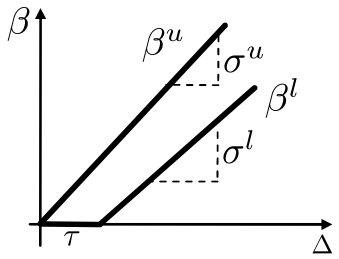

Figure 6: A simple set of service curves.

where I is a multiset of node indices $i \in V$. One should note that the indices i are not necessarily disjoint, i.e. the result changes if a service curve appears several times in the convolution.

For the upper service curve, the situation is even simpler. Here, we obtain

$$\bigotimes_{i \in I} \beta_i^u = (\bigwedge_{i \in I} \sigma_i^u).$$

Now, we can make the explicit formula (21) more concrete by just replacing the convolution of the service curves by the above expressions

$$R_i^l = (\tau_i, \sigma_i^l) \wedge \bigwedge_{p \in P(j,i)} \left(\sum_{(r,s) \in p} m_{rs} + (\sum_{k \in V(p)} \tau_k, \bigwedge_{k \in V(p)} \sigma_k^l) \right),$$

$$R_i^u = (\sigma_i^u) \wedge \bigwedge_{p \in P(j,i)} \left(\sum_{(r,s) \in p} m_{rs} + (\bigwedge_{k \in V(p)} \sigma_k^u) \right).$$

3.5 Marked Graphs with Inputs

So far, we have been dealing with marked graphs that are autonomous, i.e. they do not have any stream of input token from the environment. Token sources can enter the AND elements like any other channel, see Figure 5. Therefore, we can start from the elementary system equation (9) and integrate system inputs. Let us define the system input matrix G with elements (G_{ij}), where $G_{ii} = G_i(t)$ if there is input at node v_i with arrival function $G_i(t)$, otherwise $G_{ii} = G_i(t) = \infty$ for all t if there is no input at node v_i, and all other non-diagonal matrix elements are set to 0, e.g. see Figure 7. Then we obtain

$$R_i = [(R_j + I^{m_{ji}}) \wedge (R_k + I^{m_{ki}}) \wedge G_i] \otimes \beta_i.$$

Using this information in the fixed-point equation discussed in the previous section, we replace (17, 18) in Theorem 3.7 with

$$R^l = (S^l)^* \otimes G \otimes \beta^l, \tag{22}$$

$$R^u = (S^u)^* \otimes G \otimes \beta^u. \tag{23}$$

3.6 Transfer Functions of Marked Graphs

As a last preparatory step for embedding marked graphs into any compositional performance analysis framework, we need to determine the transfer functions of a marked graph: How does a single input stream G_s at source node v_s influence an internal stream R_d at some destination node v_d?

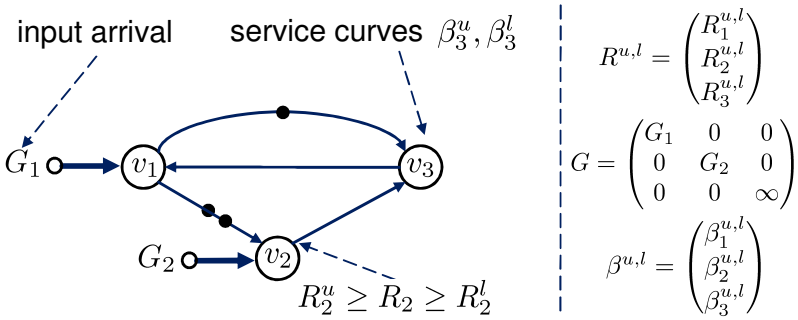

Figure 7: Marked graph with input arrival functions G, service curves β^u, β^l associated to its nodes and the resulting traces characterized by arrival functions R and their bounds R^u, R^l.

Since this information is already contained in (22) and (23), we only have to rewrite the equations such that we (a) only evaluate the destination arrival function $R_d^{u,l}$ and (b) have only one explicit input, namely G_s at the source node v_s. In order to simplify the notation, we suppose that the graph has only a single input at node v_s, an extension to the general case is straightforward. As a result of the whole exercise we find

$$R_d^l = (\beta_{sd}^l \otimes G_s) \wedge h_{sd}^l, \tag{24}$$

where $\beta_{sd}^l = (S^l)_{ds}^* \otimes \beta_s^l$ and $h_{sd}^l = \bigwedge_{j \neq s}((S^l)_{dj}^* \otimes \beta_j^l)$, and similarly

$$R_d^u = (\beta_{sd}^u \otimes G_s) \wedge h_{sd}^u, \tag{25}$$

where $\beta_{sd}^u = (S^u)_{ds}^* \otimes \beta_s^u$ and $h_{sd}^u = \bigwedge_{j \neq s}((S^u)_{dj}^* \otimes \beta_j^u)$.

Here, we note that $\beta_{sd}^{u,l}$ denote the cumulative service curves for the path from source node v_s to destination v_d and $h_{sd}^{u,l}$ denotes an 'offset' term. It is a function that is independent from the input arrival, i.e. it represents the constant part in the transfer function which is the response of the marked graph if the input stream does not contain any token. Note also, that (24) and (25) are scalar functions and not matrices or vectors anymore. Figure 8 visualizes the concept of a transfer function for a path in a marked graph.

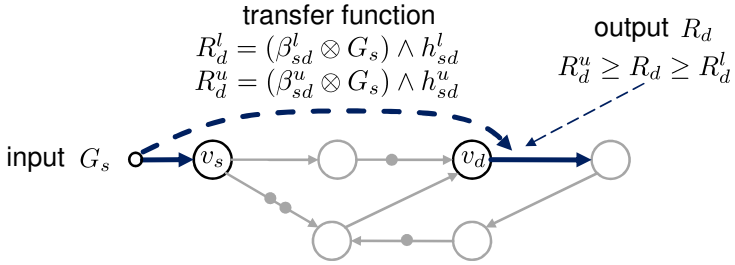

Figure 8: Visualization of transfer functions.

4 Performance Analysis

In this section, we will do the last abstraction on marked graphs. After replacing the service functions $C_i(t)$ that represent the availability of a resource for executing node v_i in the time domain by their corresponding service

curves $\beta_i^u(\Delta)$ and $\beta_i^l(\Delta)$ in the time interval domain, we now do the same for the arrival functions $R_i(t)$ and $G_i(t)$. This last abstraction is necessary for several reasons.

The whole analysis will be done now in the time interval domain. This way, we can embed the analysis not only in the Modular Performance Analysis framework, see for example [28], but also relate it to classical real-time analysis that expects stream characterizations like periodicity, jitter and burst size.

We will be able to determine performance bounds on end-to-end delays, necessary buffer spaces and the remaining service, i.e. after a given resource has been used for executing a certain marked graph node. The last one enables composability in terms of resources which makes possible the analysis of various resource sharing strategies such as fixed priority and TDMA.

4.1 Arrival Curves

In contrast to arrival functions $R(t)$ that count the number of token that occurred in $[0, t)$, arrival curves determine upper and lower bounds on the number of tokens in any *time interval* Δ, for examples see Figure 9.

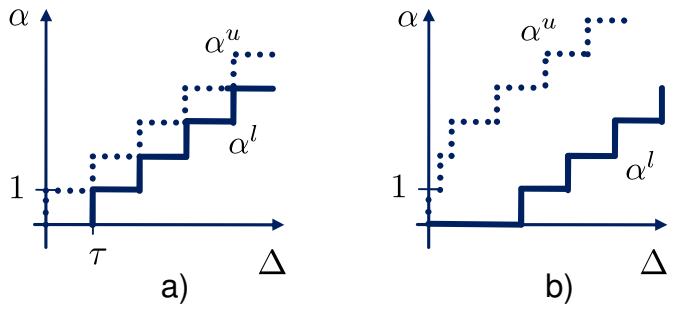

Figure 9: Arrival curves of periodic stream (a) and periodic stream with jitter and limited burst (b).

Definition 4.1. *Upper and lower arrival curves α^u, α^l map positive time intervals $\Delta \in \mathbb{R}^{\geq 0}$ to the maximal and minimal amount of token in any time interval of length Δ. They satisfy $\alpha^u(0) = \alpha^l(0) = 0$ and*

$$\alpha^l(\Delta) \leq R(t + \Delta) - R(t) \leq \alpha^u(\Delta) \quad \forall t \geq 0, \Delta > 0.$$

In order to simplify the following discussions, we will use the notation of the Min-plus deconvolution operator defined as

$$(a \oslash b)(\Delta) = \sup_{\lambda \geq 0} \{a(\Delta + \lambda) - b(\lambda)\}, \tag{26}$$

and the Max-plus deconvolution operator defined as

$$(a \overline{\oslash} b)(\Delta) = \inf_{\lambda \geq 0} \{a(\Delta + \lambda) - b(\lambda)\}.$$

Now we can make use of Definition 4.1 and obtain the tightest arrival curves (i.e. the least upper and greatest lower curves) of some internal stream $R(t)$ and some input stream $G(t)$. Here, we use α and γ to denote arrival curves related to internal streams (arrival functions R) and external input streams (arrival functions G):

$$\alpha^u = R \oslash R, \quad \alpha^l = R \overline{\oslash} R, \tag{27}$$

$$\gamma^u = G \oslash G, \quad \gamma^l = G \overline{\oslash} G. \tag{28}$$

4.2 Bounds on Buffer Size and End-to-End Delay

Let us now determine an upper bound B_{sd} on the difference between the number of token that arrived at the input to some source node v_s and left at some destination node v_d at any time. For example, if we set $v_s = v_d$, then B_{ss} is an upper bound on the number of token that are stored in front of the marked graph, i.e. at its input queue. In other words, one can then guarantee that an input queue of size B_{ss} would be sufficient to store all necessary token.

The above definition of B_{sd} directly yields

$$G_s(t) - R_d(t) \leq G_s(t) - ((\beta^l_{sd} \otimes G_s) \wedge h^l_{sd})(t) \leq B_{sd}.$$

Using the transfer functions from (24) and (25), the arrival curve corresponding to the input in (28) and the definition of the convolution operator in (7), we obtain

$$\begin{aligned} &G_s(t) - ((\beta^l_{sd} \otimes G_s) \wedge h^l_{sd})(t) = \\ =&G_s(t) + \max\{-h^l_{sd}(t), \sup_{0\leq\lambda\leq t}(-G_s(t-\lambda) - \beta^l_{sd}(\lambda))\} \\ \leq&\max\{\gamma^u_s(t) - h^l_{sd}(t), \sup_{0\leq\lambda\leq t}(\gamma^u(\lambda) - \beta^l_{sd}(\lambda))\}. \end{aligned}$$

As a result, we find

$$B_{sd} \leq \max\left\{\sup_{\lambda\geq 0}\{\gamma^u_s(\lambda) - h^l_{sd}(\lambda)\}, \sup_{\lambda\geq 0}\{\gamma^u_s(\lambda) - \beta^l_{sd}(\lambda)\}\right\}. \tag{29}$$

In other words, the maximal backlog as defined above can be determined as the maximum of the maximal vertical distances between the functions γ^u_s (the upper arrival curve corresponding to the input stream) and h^l_{sd} as well as between γ^u_s and β^l_{sd}.

A more detailed interpretation of B yields the following statement: Let us suppose that some path from s to d has an initial sum of tokens M_{sd}, then on this path, there will never be more than $M_{sd} + B$ token. This statement is based on the fact that a firing of a node other than d does not change the number of token on any path from the input of s (i.e. including the input buffer of s) to d . Therefore, the only possibility to change the number of token on a path (including the input buffer) is to change $G(s)$ or R_d, and their difference is bounded by B.

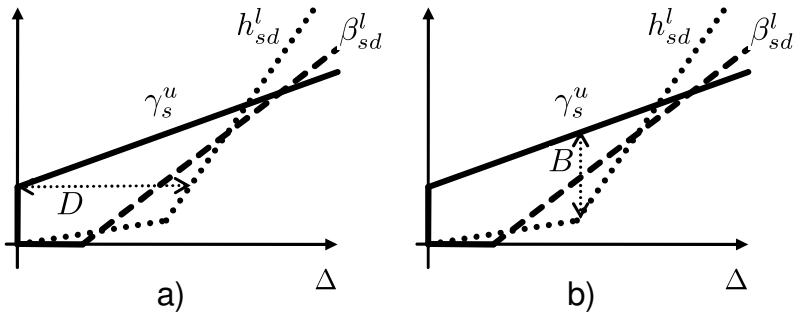

Figure 10: Visualization of delay (a) and backlog (b) bounds in a marked graph.

Let us now determine a bound D_{sd} on the end-to-end delay of token, i.e. the maximal time a token needs from a system input at the source node v_s to the output of a destination node v_d. In order to faithfully determine such a bound on the end-to-end delay we suppose that the system does not produce an output if the input stream is empty, i.e. $G_s(t) = 0$. Therefore, we request that $\beta^l_{sd}(0) = 0$.

In order to simplify the notation let us first define the maximal horizontal distance of two functions A and B as

$$h(A,B) = \inf\{\tau \geq 0 \,:\, B(t+\tau) \geq A(t)\ \forall t \geq 0\}.$$

Then an upper bound on the end-to-end delay of any token between an input G_s at source node v_s to a destination node v_d is given by

$$D_{sd} = h(G_s, R_d) = \inf\{\tau \geq 0 \,:\, R_d(t+\tau) \geq G_s(t)\ \forall t \geq 0\}.$$

Using similar arguments as in the case of the necessary buffer space, we obtain that

$$\begin{aligned}
&G_s(t) - R_d(t+\tau) = \\
&= \max\{G_s(t) - h^l_{sd}(t+\tau), \\
&\qquad \sup_{0\leq\lambda\leq t+\tau} (G_s(t) - G_s(\lambda) - \beta^l_{sd}(t+\tau-\lambda))\} \\
&\leq \max\{0, \gamma^u_s(t) - h^l_{sd}(t+\tau), \\
&\qquad \sup_{0\leq\lambda\leq t} (\gamma^u_s(t-\lambda) - \beta^l_{sd}(t-\lambda+\tau))\} \\
&\leq \max\{0, \sup_{\lambda\geq 0}\{\gamma^u_s(\lambda) - h^l_{sd}(\lambda+\tau)\}, \\
&\qquad \sup_{\lambda\geq 0}(\gamma^u_s(\lambda) - \beta^l_{sd}(\lambda+\tau))\}.
\end{aligned}$$

As a result we get

$$D_{sd} \leq \max\{h(\gamma^u_s, \beta^l_{sd}), h(\gamma^u_s, h^l_{sd})\}. \tag{30}$$

In other words, the maximal delay as defined above can be determined as the maximum of the maximal horizontal distances between the functions γ^u_s and h^l_{sd} as well as between γ^u_s and β^l_{sd}. The interpretation of the delay and buffer bounds in marked graphs is visualized in Figure 10.

4.3 Output Arrival Curves

In this section, we will describe a method that allows to compute bounds on the token streams at any node in a given marked graph. In comparison to the results in Theorem 3.7, these bounds are now given in terms of arrival curves, i.e. not in the time domain but in the time interval domain as can be seen when comparing Figure 8 and Figure 11. Again, this is necessary to abstract from the concrete time domain and leads to composability in terms of resources and event streams.

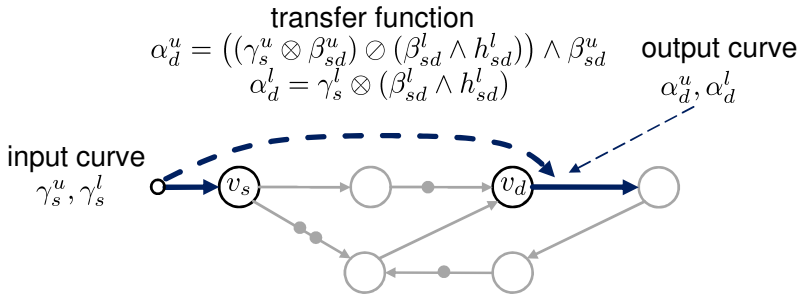

Figure 11: Transfer functions in time interval domain.

The derivation starts from the transfer functions developed in (24) and (25). In order to apply the relations known from real-time calculus, see [26], we first approximate the transfer functions from input v_s to node v_d as

follows:

$$R_d^l = (\beta_{sd}^l \otimes G_s) \wedge h_{sd}^l \geq (\beta_{sd}^l \otimes G_s) \wedge (h_{sd}^l \otimes G_s) \geq (\beta_{sd}^l \wedge h_{sd}^l) \otimes G_s,$$
$$R_d^u = (\beta_{sd}^u \otimes G_s) \wedge h_{sd}^u \leq \beta_{sd}^u \otimes G_s.$$

As a result, we find that upper and lower bounds on the output stream at node v_d in the time domain $R_d^{u,l}$ can be determined by convolving the input stream function G_s with a certain service curve. This fact enables to directly use the results shown in [26] to compute the corresponding output arrival curves. Therefore, we obtain the following theorem:

Theorem 4.2. *Given a marked graph (V, E, M) and service curves β^u, β^l associated to its nodes according to Theorem 3.7. Suppose that the network has a single input stream at node v_s with arrival curves γ_s^u, γ_s^l. Then we can determine upper and lower arrival curves α_s^u, α_s^l associated to any node v_d with*

$$\alpha_d^u = \Big((\gamma_s^u \otimes \beta_{sd}^u) \oslash (\beta_{sd}^l \wedge h_{sd}^l)\Big) \wedge \beta_{sd}^u, \tag{31}$$
$$\alpha_d^l = \gamma_s^l \otimes (\beta_{sd}^l \wedge h_{sd}^l). \tag{32}$$

where we use β_{sd}^l and β_{sd}^u as defined for (24) and (25), respectively.

4.4 Remaining Service

In order to enable compositionality, we need to determine the remaining service curve $\beta_d'^l, \beta_d'^u$ at any node v_d, see also Figure 5. This enables us to use the remaining service as an input to some other process and thereby, represent fixed-priority scheduling where the first process, i.e. the one that gets the initial service β_d^l, β_d^u from the provided resource, has a higher priority in comparison to the process that just gets the remaining service $\beta_d'^l, \beta_d'^u$.

To this end, we start from the balancing equation of a greedy process component according to Definition 2.7, i.e. the remaining service equals the available service reduced by the produced output, $C_d'(t) = C_d(t) - R_d'(t)$. Therefore, we obtain

$$C_d'(t+\Delta) - C_d'(t) = [C_d(t+\Delta) - C_d(t)] - [R_d'(t+\Delta) - R_d'(t)],$$

which is bounded by

$$\beta_d'^l(\Delta) \geq \beta_d^l(\Delta) - \alpha_d^u(\Delta), \qquad \beta_d'^u(\Delta) \leq \beta_d^u(\Delta) - \alpha_d^l(\Delta).$$

Using the fact that the remaining service curves are monotone functions, we can tighten the bounds as follows:

$$\beta_d'^l(\Delta) = \sup_{0 \leq \lambda \leq \Delta} \{\beta_d^l(\lambda) - \alpha_d^u(\lambda)\}, \tag{33}$$
$$\beta_d'^u(\Delta) = (\inf_{\Delta \leq \lambda} \{\beta_d^u(\lambda) - \alpha_d^l(\lambda)\})^{\geq 0}. \tag{34}$$

In the last two subsections, we have determined the output arrival curves at any process and the corresponding remaining services curves of any process in a marked graph. These representations can then be used in order to compose the marked graph model with other parts of the application. For example, the output of a marked graph characterized by its arrival curve can be linked to the input of some other application. One can also link the remaining service to another performance model and analyze a fixed priority setting this way, see also [5, 28].

5 Experimental Results

5.1 Comparison

In this section we compare the performance analysis results computed with the method proposed in this paper with results computed with the methods proposed in [3] and [4]. These two methods are very similar in their approach and therefore we have implemented only the more recent one described in [4]. For simplicity in this section, we will refer to our method as MG (for marked graph) and the method proposed in [4] as FB (for finite buffer).

System: The system used for evaluation is shown in Figure 12a. It is a simple chain of three tasks $T1, T2, T3$ which processes a bursty input event stream characterized with period 4 ms, jitter 20 ms, and minimum interarrival distance between two events 1 ms. All tasks have constant execution time of 1 ms. They are mapped on an MpSoC with three processing elements $PE1, PE2, PE3$. $PE1$ exhibits complex behavior due to being shared with other tasks, it may not be available to task $T1$ for 2 ms, then it may provide service of maximum 20 events/ms which eventually slows down to a long term rate of 1 event/ms. Similarly, $PE2$ may not be available for 2 ms, has a maximum speed of 2 events/ms, and a long term rate of 1 event/ms. $PE3$ may not be available for 1 ms, and has a constant rate of 0.5 events/ms. For simplicity, the communication hardware is not shown here and it is not modeled.

Each task is activated by events that arrive in a FIFO buffer mapped to the same processing element as the task. Tasks $T1$ and $T2$ have buffers with unlimited capacity. Task $T3$ has a buffer with a finite size $B = 1$. The semantics of the buffer are blocking-write which means that task $T2$ needs to block if the buffer at $T3$ is full. When $T2$ blocks, the service provided by $PE2$ is available to be used by other lower priority tasks mapped on $PE2$.

Model: The system is modeled with a simple marked graph that has a single cycle with one initial token, see Figure 12b. The abstract model of greedy marked graph processes that is used for performance evaluation of the system with method MG is shown in Figure 12c.

Scenario: We compare methods MG and FB in terms of tightness of the computed performance metrics for the system in Figure 12a. More specifically, we compare the bounds on the output of task $T2$: α_2^u, α_2^l computed with the two methods, and the bounds on the remaining service of task $T2$: $\beta_2'^u, \beta_2'^l$ again for both methods. The parameters of interest are indicated with a question mark '?' in Figure 12c. We have chosen these parameters because they are essential for computing bounds on other performance metrics such as end-to-end delays and buffer sizes. Any inaccuracy in computing the chosen parameters will have an influence on all other computed metrics for the system.

Results: Bounds for the output event stream of $T2$ computed with methods FB and MG are shown in Figure 13a. Note, that method FB does not compute the lower bound α_2^l. For the upper bound α_2^u, method MG is more tight and it accurately shows the fact that there cannot be a burst of events coming out of task $T2$ since the buffer of $T3$ is of finite size. Even for the long term rate, method FB shows some error.

Bounds for the remaining service of $T2$ are shown in Figure 13b. Note, that method FB does not compute an upper bound on the service $\beta_2'^u$. For the lower bound $\beta_2'^l$, method MG is again tighter. This is due to the fact that method FB computes a pessimistic bound on the event output of the task which is then used for computation of the remaining service.

The tightness of the results computed with method MG can be observed even for simple systems such as the one used here. We expect that the difference in results will be more visible for more complex systems. MG is a more general method than method FB since it can analyze not only systems with finite buffers but any system that can be modeled with a marked graph. And more importantly, this gain in generality does not lead to pessimism in the computed results.

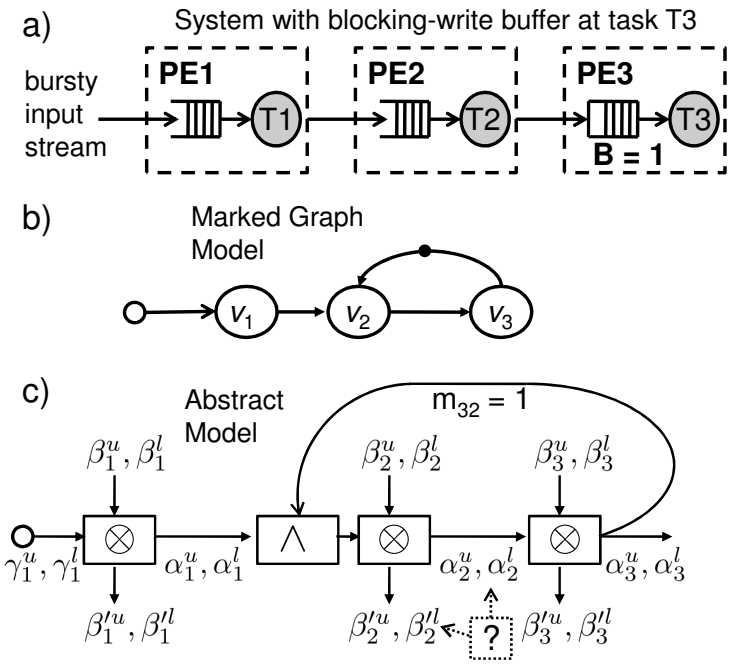

Figure 12: A system with a single finite buffer (a), with its marked graph model (b), and the abstract greedy marked graph processes model (c).

5.2 Validation

In this section, we validate our approach with a more complex scenario and compare the analysis results to simulation measurements.

System and models: We use an application from the area of software defined radio. It is adapted from [19]. The Wireless LAN (WLAN) and TD-SCDMA applications run simultaneously. Both of them are modeled as marked graphs as depicted in Figure 14. In contrast to [19], we will use an idealized scenario. The underlying multiprocessor architecture consists of 5 independent cores where communication time is supposed to be negligible.

It is assumed, that processor 5 provides a TDMA schedule which partitions the period into two equal time slices, named 5.1 and 5.2. Processors 1-4 have speeds of 100M cycles/sec, processor 5 provides 200M cycles/sec. The following Table 1 lists the mapping of the nodes to the processors and the number of cycles each of the nodes in [19] needs on the respective processor. The TDMA-scheduler in processor 5 is assumed to have a period of 0.2 ms and the slot lengths for 5.1 and 5.2 are equal, i.e. 0.1 ms. We further assume, that the inputs to the two applications are periodic with periods equal to 0.2 ms and 0.7 ms, for the WLAN and TD-SDMA applications, respectively.

Let us suppose that we use fixed priority scheduling where all nodes of the WLAN application have higher priority than those of the TD-SCDMA marked graph.

Experimental setup: Simulation models of the two applications have been implemented in the Real-Time Simulation (RTS) Toolbox (www.mpa.ethz.ch). It is a framework for discrete-event simulation which uses the component structure of MPA [5, 28] however, instead of using abstracted event and resource models, it uses traces which are produced randomly following the specifications of the processing cores and the input streams. The processes are simulated assuming their worst-case execution demands. The analysis computations have been performed with the RTC (Real-Time Calculus) Toolbox (www.mpa.ethz.ch).

Scenario: We compare results from analysis computed with inequality (30) and simulation for the maximum token delays from the input node to the outputs of all nodes in the graph for both applications. The simulation

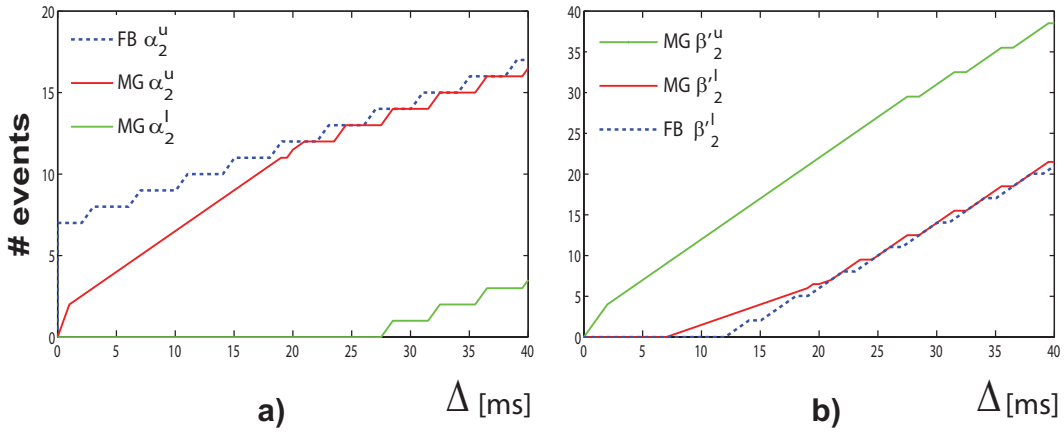

Figure 13: Comparison of methods FB and MG for the output event stream (a), and the remaining service (b). Note, that method FB does not compute lower bounds on the output event streams, and upper bounds on the remaining services.

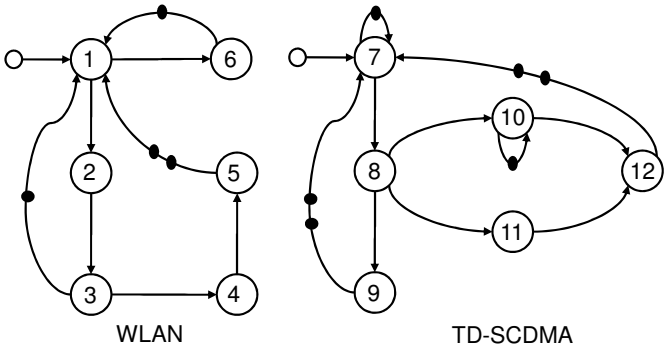

Figure 14: Marked graphs that model WLAN and TD-SCDMA applications.

has been performed with several traces and from all of them the maximum observed end-to-end delays have been selected.

Results: The results are summarized in Table 2. They show the tightness of the analysis and the feasibility of the method for the performance analysis of cyclic data flow graphs.

6 Concluding Remarks

The paper presents a new modular performance analysis framework for distributed implementations of cyclic dataflow graphs, in particular marked graphs. It substantially generalizes previous analysis approaches in that general non-deterministic resource interactions can be modeled by means of service curves. This way, it is possible to model implementations with finite buffer sizes, model dynamic scheduling where the processes of different marked graphs are scheduled according to a fixed priority scheme and take into account other scheduling disciplines like TDMA. We plan to extend the approach towards more general dataflow models such as conflict-free Petri Nets.

Table 1: Mapping and cycles for each of the processes in Figure 14.

node	1	2	3	4	5	6
cycles	2k	0.31k	0.33k	0.42k	4k	2k
core	1	2	4	3	5.1	5.2
node	7	8	9	10	11	12
cycles	50k	12.5k	20k	3.3k	0.25k	50k
core	1	2	5.1	3	4	5.2

Table 2: Maximum end-to-end delays observed in simulation compared to analytical results.

output	1	2	3	4	5	6
simulation [ms]	0.02	0.023	0.026	0.031	0.12	0.11
analysis [ms]	0.02	0.023	0.027	0.031	0.151	0.13
output	7	8	9	10	11	12
simulation [ms]	0.56	0.688	1.025	0.721	0.691	1.28
analysis [ms]	0.56	0.688	1.048	0.726	0.692	1.316

References

[1] F.L. Baccelli, G.J. Olsder, J.P. Quadrat, and G. Cohen. *Synchronization and Linearity: An Algebra for Discrete Event Systems*. Wiley Series on Probability and Mathematical Statistics: Probability and Mathematical Statistics, 1992.

[2] Shuvra S. Bhattacharyya, Praveen K. Murthy, and Edward A. Lee. Synthesis of embedded software from synchronous dataflow specifications. *J. VLSI Signal Process. Syst.*, 21(2):151–166, 1999.

[3] Amit Bose, Xiaoyue Jiang, Bin Liu, and Gang Li. Analysis of manufacturing blocking systems with network calculus. *Perform. Eval.*, 63(12):1216–1234, 2006.

[4] A. Bouillard, L.T.X. Phan, and S. Chakraborty. Lightweight modeling of complex state dependencies in stream processing systems. In *Real-Time and Embedded Technology and Applications Symposium, 2009. RTAS 2009. 15th IEEE*, pages 195–204, April 2009.

[5] Samarjit Chakraborty, Simon Künzli, and Lothar Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *DATE '03: Proceedings of the conference on Design, Automation and Test in Europe*, page 10190, Washington, DC, USA, 2003. IEEE Computer Society.

[6] C.S. Chang. *Performance Guarantees in Communication Networks*. Springer-Verlag, London, UK, 2000.

[7] G. Cohen, D. Dubois, J. Quadrat, and M. Viot. A linear-system-theoretic view of discrete-event processes and its use for performance evaluation in manufacturing. *Automatic Control, IEEE Transactions on*, 30(3):210–220, Mar 1985.

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

[9] B.A. Davey and H.A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 2nd edition, 2002.

[10] A. H. Ghamarian, M. C. W. Geilen, S. Stuijk, T. Basten, B. D. Theelen, M. R. Mousavi, A. J. M. Moonen, and M. J. G. Bekooij. Throughput analysis of synchronous data flow graphs. In *ACSD '06: Proceedings of the Sixth International Conference on Application of Concurrency to System Design*, pages 25–36, Washington, DC, USA, 2006. IEEE Computer Society.

[11] Wolfgang Haid and Lothar Thiele. Complex task activation schemes in system level performance analysis. In *CODES+ISSS '07: Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, pages 173–178, New York, NY, USA, 2007. ACM.

[12] Marek Jersak and Rolf Ernst. Enabling scheduling analysis of heterogeneous systems with multi-rate data dependencies and rate intervals. In *DAC '03: Proceedings of the 40th conference on Design automation*, pages 454–459, New York, NY, USA, 2003. ACM.

[13] Marek Jersak, Kai Richter, and Rolf Ernst. Performance analysis for complex embedded applications. *International Journal of Embedded Systems*, 1(1/2):33–49, 2005.

[14] Bengt Jonsson, Simon Perathoner, Lothar Thiele, and Wang Yi. Cyclic dependencies in modular performance analysis. In *EMSOFT '08: Proceedings of the 8th ACM international conference on Embedded software*, pages 179–188, New York, NY, USA, 2008. ACM.

[15] Jean-Yves Le Boudec and Patrick Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*, volume 2050 of *Lecture Notes in Computer Science*. Springer, 2001.

[16] E.A. Lee and D.G. Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235–1245, Sept. 1987.

[17] E.A. Lee and T.M. Parks. Dataflow process networks. *Proceedings of the IEEE*, 83(5):773–801, May 1995.

[18] Alexander Maxiaguine, Simon Künzli, and Lothar Thiele. Workload characterization model for tasks with variable execution demand. In *DATE '04: Proceedings of the conference on Design, automation and test in Europe*, page 21040, Washington, DC, USA, 2004. IEEE Computer Society.

[19] Orlando Moreira, Frederico Valente, and Marco Bekooij. Scheduling multiple independent hard-real-time jobs on a heterogeneous multiprocessor. In *EMSOFT '07: Proceedings of the 7th ACM & IEEE international conference on Embedded software*, pages 57–66, New York, NY, USA, 2007. ACM.

[20] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, Apr 1989.

[21] Traian Pop, Petru Eles, and Zebo Peng. Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems. In *CODES '02: Proceedings of the tenth international symposium on Hardware/software codesign*, pages 187–192, New York, NY, USA, 2002. ACM.

[22] P. Poplavko, T. Basten, M. Bekooij, J. van Meerbergen, and B. Mesman. Task-level timing models for guaranteed performance in multiprocessor networks-on-chip. In *CASES '03: Proceedings of the 2003 international conference on Compilers, architecture and synthesis for embedded systems*, pages 63–72, New York, NY, USA, 2003. ACM.

[23] Raymond Reiter. Scheduling parallel computations. *J. ACM*, 15(4):590–599, 1968.

[24] Simon Schliecker, Steffen Stein, and Rolf Ernst. Performance analysis of complex systems by integration of dataflow graphs and compositional performance analysis. In *DATE '07: Proceedings of the conference on Design, automation and test in Europe*, pages 273–278, San Jose, CA, USA, 2007. EDA Consortium.

[25] Sundararajan Sriram and Shuvra S. Bhattacharyya. *Embedded multiprocessors: Scheduling and synchronization*. CRC Press, 2000.

[26] Lothar Thiele, Samarjit Chakraborty, Matthias Gries, and Simon Künzli. A framework for evaluating design tradeoffs in packet processing architectures. In *DAC '02: Proceedings of the 39th conference on Design automation*, pages 880–885, New York, NY, USA, 2002. ACM.

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

[28] Ernesto Wandeler, Lothar Thiele, Marcel Verhoef, and Paul Lieverse. System architecture evaluation using modular performance analysis: a case study. *Int. J. Softw. Tools Technol. Transf.*, 8(6):649–667, 2006.

[29] Maarten Wiggers, Marco Bekooij, Pierre Jansen, and Gerard Smit. Efficient computation of buffer capacities for multi-rate real-time systems with back-pressure. In *CODES+ISSS '06: Proceedings of the 4th international conference on Hardware/software codesign and system synthesis*, pages 10–15, New York, NY, USA, 2006. ACM.