Modular Performance Analysis of Cyclic Dataflow Graphs

Lothar Thiele Nikolay Stoimenov
Computer Engineering and Networks Laboratory, ETH Zurich
8092 Zurich, Switzerland
thiele@tik.ee.ethz.ch stoimenov@tik.ee.ethz.ch

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 dataflow (SDF) 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 time division multiple access (TDMA). The paper presents the new models and methods as well as experimental results.

Categories and Subject Descriptors

C.4 [**Computer Systems Organization**]: Performance Of Systems—*modeling techniques*

General Terms

Design, Performance, Theory, Verification

Keywords

Dioid algebras, marked graphs, real-time calculus, real-time systems, synchronous dataflow

1. INTRODUCTION

Applications that are implemented on distributed embedded systems can often be specified using dataflow graphs. 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 [22], 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, 28].

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. [6, 12, 24, 30]. 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 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. 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 extend-

EMSOFT'09, October 12–16, 2009, Grenoble, France.

ing 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. [22, 26]. 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. Many results are available that characterize their timing behavior but they all suppose that there is a fixed deterministic processing time for tokens in each node. Early results in [26] have been generalized and connected to eigenvalue problems in max/plus algebra, see [1, 8]. The results are not directly applicable to more complex interactions 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. 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 [28]. 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) [28], 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 [32], and computing throughput while respecting sequence constraints by additional edges, see [25]. 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.

Other methods compute timing separations between events in various models considering either fixed delays, see [5, 11, 23], or probabilistic delay distributions, see [20, 33]. The method in [18] computes buffer sizes and output traces of nodes in latency-insensitive systems using max/plus algebra. All of these methods do not model explicitly the available resources and the resource interactions.

More complex interactions between resources and process executions can be faithfully modeled using the closely related concepts of dioids [1] and network calculus [7, 9, 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, 4], results from [7] have been applied to chains of processes with finite buffer sizes. These results are restricted only to systems with finite buffer sizes, use coarse-grained approximations of resources as the maximal processing or communication capability is set to infinity, and 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. The overall system behavior is obtained by iterating relations for individual processes, see also [30]. The approach

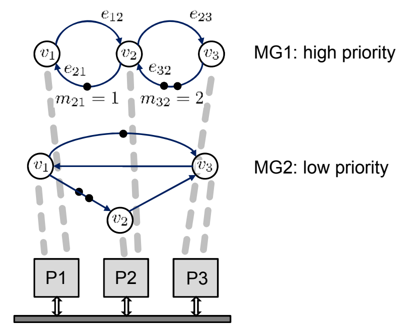

Figure 1: Visualization of problem statement where two marked graphs are mapped to a distributed platform and the individual processes share the available resources using some scheduling scheme.

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 [27]. The analysis is based on the classical results described above as well as simulation, and is 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) [6, 31]. 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.

1.2 Contributions

In summary, the problem statement can be formulated as follows: Given a set of marked graphs as depicted in Fig. 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.

- 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 Fig. 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 tokens $M : E \to \mathbb{R}^{\geq 0}$, i.e., $m_{ij} \in \mathbb{R}^{\geq 0}$ denotes the number of tokens 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 tokens that arrived in the time interval $[0, t)$, $t > 0$, and $R(0)$ denotes the initial number of tokens 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 $t \geq 0$, $i = 1, ..., n$.

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 tokens start at time τ.*

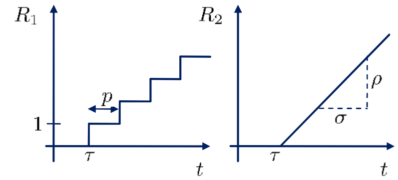

Figure 2: Two simple arrival functions.

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_{ji}^{in} is associated to an input port (j, i) of v_i and the output arrival function R_{ik}^{out} 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 $R^{in} = (R_{ji}^{in} : e_{ji} \in E)$ and $R^{out} = (R_{ik}^{out} : e_{ik} \in E)$. 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 tokens to a process in one of them ($\overline{R^{in}} \geq R^{in}$), then the process does produce at least as many output tokens as for the other one ($\overline{R^{out}} \geq R^{out}$).

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

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. If we restrict it to two inputs R_1^{in} and R_2^{in}, we find its transfer function as*

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

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 tokens. 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 tokens 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 [19].

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 per time 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 $C(t) = t$, $0 \leq t < t_1$, and $C(t) = t_1$, $t_1 \leq t < t_2$.*

Now, let us consider a component with a single input which uses a resource. It takes an input arrival function

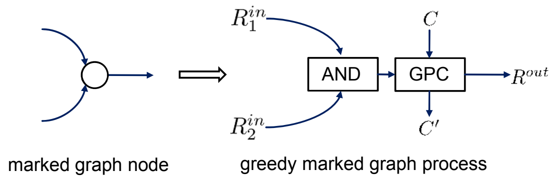

Figure 3: Marked graph node and its representation as a greedy marked graph process.

$R_{in}(t)$ and produces an output arrival function $R^{out}(t)$ by means of a service function $C(t)$. Input tokens 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*

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

The remaining unused service from such a component is given by

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

The above definition of a greedy process can be derived as follows: The output between some time λ and t cannot be larger than the available service: $C(t) - C(\lambda)$, and therefore, $R^{out}(t) \le R^{out}(\lambda) + C(t) - C(\lambda)$. As the component can not output more than what was available at the input, we have $R^{out}(\lambda) \le R^{in}(\lambda)$ and therefore, $R^{out}(t) \le R^{in}(\lambda) + C(t) - C(\lambda)$. 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 tokens: $R^{out}(t) = R^{out}(\lambda^*) + C(t) - C(\lambda^*) = R^{in}(\lambda^*) + C(t) - C(\lambda^*)$. 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 tokens from each input channel and adds tokens 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 tokens in each input channel. It also follows that multiple output channels of a single process have equal traces.

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

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, 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 [10], page 187. It follows that if the mapping Π is monotone, then the fixed-point equation (6) has a least and a greatest fixed-points, 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 $s \le t - \delta$ 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 Fig. 1 and determine the corresponding mapping $R = \Pi \circ R$ by concatenating (1) and (2):*

$$\begin{aligned}
R_{1,2}^{out}(t) &= \inf_{0 \le \lambda \le t} \{R_{2,1}^{out}(\lambda) + I^1(\lambda) + C_1(t) - C_1(\lambda)\} \\
R_{2,3}^{out}(t) &= \inf_{0 \le \lambda \le t} \{\min\{R_{3,2}^{out}(\lambda) + I^2(\lambda), R_{1,2}^{out}(\lambda)\} \\
&\qquad + C_2(t) - C_2(\lambda)\} \\
R_{3,2}^{out}(t) &= \inf_{0 \le \lambda \le t} \{R_{2,3}^{out}(\lambda) + C_3(t) - C_3(\lambda)\} \\
R_{2,1}^{out}(t) &= \inf_{0 \le \lambda \le t} \{\min\{R_{3,2}^{out}(\lambda) + I^2(\lambda), R_{1,2}^{out}(\lambda)\} \\
&\qquad + C_2(t) - C_2(\lambda)\}
\end{aligned}$$

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

To develop efficient methods for the modular performance analysis of marked graphs, we will need to introduce several abstractions. For example, 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

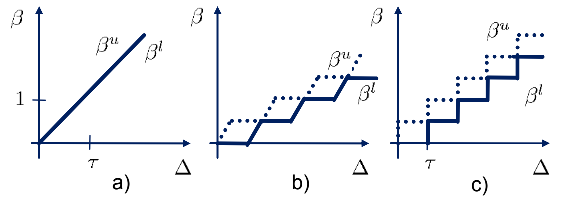

Figure 4: Three examples of service curves.

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 [9, 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 $\Delta \in \mathbb{R}^{\geq 0}$ 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*

$$\beta^l(\Delta) \leq C(t+\Delta) - C(t) \leq \beta^u(\Delta) \quad \forall t \geq 0, \Delta > 0$$

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

$$R^{out}(t) \leq \inf_{0 \leq \lambda \leq t}\{R^{in}(\lambda) + \beta^u(t-\lambda)\}$$

$$R^{out}(t) \geq \inf_{0 \leq \lambda \leq t}\{R^{in}(\lambda) + \beta^l(t-\lambda)\}$$

Using the convolution operator

$$(a \otimes b)(\Delta) = \inf_{0 \leq \lambda \leq \Delta}\{a(\lambda) + b(\Delta - \lambda)\} \quad (7)$$

we obtain a more concise notation, see also [7, 9, 15]:

$$R^{in} \otimes \beta^l \leq R^{out} \leq R^{in} \otimes \beta^u \quad (8)$$

In other words, for a single GPC component we can bound the number of tokens 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 tokens 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

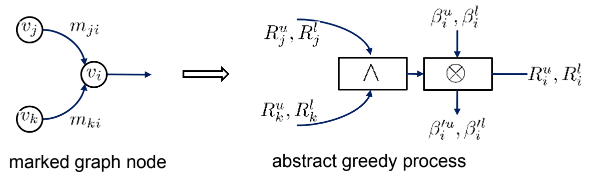

Figure 5: Marked graph node with m_{ji}, m_{ki} initial tokens on edges e_{ji}, e_{ki} and its abstract representation.

the relation holds pointwise or more generally:

$$\Pi^u \geq \Pi \quad \Leftrightarrow \quad \Pi^u \circ R \geq \Pi \circ R \quad \forall R$$

$$\Pi^l \leq \Pi \quad \Leftrightarrow \quad \Pi^l \circ R \leq \Pi \circ R \quad \forall R$$

This leads us to the following result:

Theorem 3.3. *Let $\mathcal{R}$ be a complete lattice. Let Π be a monotone mapping 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 [10], 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 tokens 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 Fig. 3.

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 Fig.5

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

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 Def.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 execution trace of the marked graph with

$$R^u(t) \geq R(t) \geq R^l(t) \tag{13}$$

3.3 Solving the System Equation

Finally, we need to determine solutions to (11, 12) in order to determine bounds on the event sequences between the processes, i.e., tight bounds on the arrival function 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.*

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., $\underline{y} = \inf\{y \ : \ y = S^u \otimes y\}$. Because of lack of space, we will only state the main results:

- The least fixed-point of $y = S^u \otimes y$ with $\underline{y} = \inf\{y : y = S^u \otimes y\}$ is given by $\underline{y} = \lim_{k\to\infty}((S^u)^{(k)} \otimes \perp)$ where $\perp(t) = 0$ for all $t \geq 0$. This result can easily be shown using elementary techniques from lattice theory [10], and by noting that S^u is monotone.
- Given a marked graph where the sum of initial tokens in each directed cycle of the network is strictly larger than 0. Then $\underline{y} = \lim_{k\to\infty}((S^u)^{(k)} \otimes \perp) = \top$ where we have $\perp(t) = 0$ and $\top(t) = \infty$ for all $t \geq 0$.

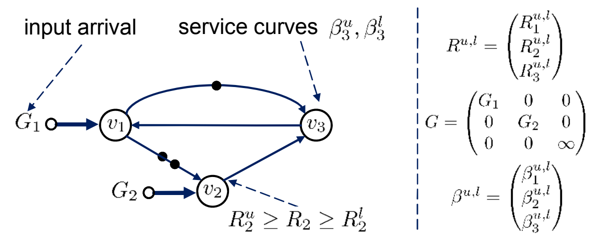

Figure 6: Marked graph with input arrival functions G, service curves β^u, β^l associated to its nodes and the resulting traces characterized by arrival functions R and their bounds R^u, R^l.

These statements lead now to the main results of this section.

Theorem 3.5. *Given a marked graph (V, E, M) and service curves β^u, β^l associated to its nodes. Suppose that the sum of initial tokens in each directed cycle of the network is strictly larger than 0. Then we can determine tight upper and lower bounds on any execution trace of the marked graph $R^u(t) \geq R(t) \geq R^l(t)$ with the arrival functions*

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

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

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

3.4 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 tokens from the environment. Token sources can enter the AND elements like any other channel, see Fig. 5. Therefore, we can start from the elementary system equation (9) and integrate system inputs. Let us define the system input matrix G as shown in Fig. 6. It contains the input arrival functions $G_i(t)$ at nodes v_i. If there is no input at node v_i, we can just set $G_i(t) = \infty$ for all t. 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 (16, 17) in Theorem 3.5 with

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

$$R^u = (S^u)^* \otimes G \otimes \beta^u \tag{19}$$

3.5 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?

Of course, this information is already contained in (18,19). We only have to rewrite the equations such that we (a) only evaluate the destination arrival function $R_d^{u,l}$ and (b) have one explicit input only, 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

$$\begin{aligned} R_d^l &= (\beta_{sd}^l \otimes G_s) \wedge h_{sd}^l & R_d^u &= (\beta_{sd}^u \otimes G_s) \wedge h_{sd}^u \\ \beta_{sd}^l &= (S^l)_{ds}^* \otimes \beta_s^l & \beta_{sd}^u &= (S^u)_{ds}^* \otimes \beta_s^u \\ h_{sd}^l &= \bigwedge_{j \neq s} ((S^l)_{dj}^* \otimes \beta_j^l) & h_{sd}^u &= \bigwedge_{j \neq s} ((S^u)_{dj}^* \otimes \beta_j^u) \end{aligned} \quad (20)$$

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 tokens. Note also that (20) are scalar functions and not matrices or vectors anymore. Figure 7 visualizes the concept of a transfer function for a path in a marked graph.

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 MPA framework, 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. This way, composability in terms of resources is possible which enables 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 tokens 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 Fig. 8.

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 number of tokens 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 deconvolution operators defined as

$$(a \oslash b)(\Delta) = \sup_{\lambda \geq 0}\{a(\Delta + \lambda) - b(\lambda)\} \quad (21)$$

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

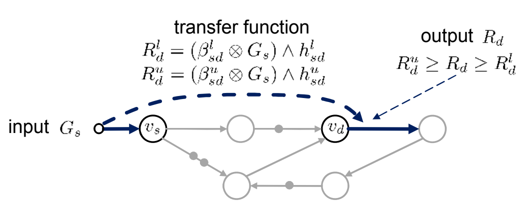

Figure 7: Visualization of transfer functions.

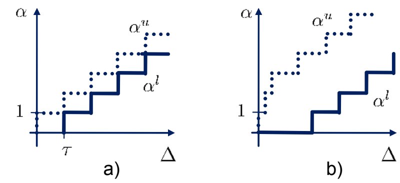

Figure 8: Arrival curves of periodic stream (a) and periodic stream with jitter and limited burst (b).

Now we can make use of Def 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):

$$\alpha^u = R \oslash R, \quad \alpha^l = R \overline{\oslash} R, \quad \gamma^u = G \oslash G, \quad \gamma^l = G \overline{\oslash} G \quad (22)$$

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 tokens 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 tokens 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 tokens.

The above definition of B_{sd} directly yields

$$G_s(t) - R_d(t) \leq G_s(t) - ((\beta_{sd}^l \otimes G_s) \wedge h_{sd}^l)(t) \leq B_{sd}$$

Using the transfer function in (20), the arrival curve corresponding to the input in (22) and the definition of the convolution operator in (7), we obtain

$$\begin{aligned} &G_s(t) - ((\beta_{sd}^l \otimes G_s) \wedge h_{sd}^l)(t) = \\ =&G_s(t) + \max\{-h_{sd}^l(t), \sup_{0 \leq \lambda \leq t}(-G_s(t-\lambda) - \beta_{sd}^l(\lambda))\} \\ \leq& \max\{\gamma_s^u(t) - h_{sd}^l(t), \sup_{0 \leq \lambda \leq t}(\gamma^u(\lambda) - \beta_{sd}^l(\lambda))\} \end{aligned}$$

As a result, we find

$$B_{sd} \leq \max\left\{\sup_{\lambda \geq 0}\{\gamma_s^u(\lambda) - h_{sd}^l(\lambda)\}, \sup_{\lambda \geq 0}\{\gamma_s^u(\lambda) - \beta_{sd}^l(\lambda)\}\right\} \quad (23)$$

In other words, the maximal backlog as defined above can be determined as the maximum of the maximal vertical distances between the functions γ_s^u (the upper arrival curve corresponding to the input stream) and h_{sd}^l as well as between γ_s^u and β_{sd}^l.

Let us now determine a bound D_{sd} on the end-to-end delay of tokens, 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 $\beta_{sd}^l(0) = 0$.

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

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

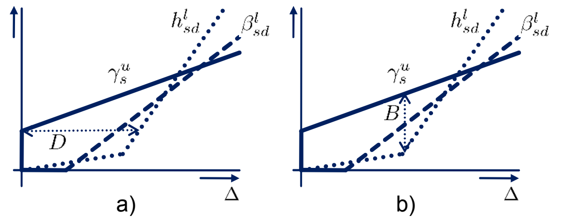

Figure 9: Visualization of delay and backlog bounds in a marked graph.

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

$$D_{sd} \leq \max\{h(\gamma_s^u, \beta_{sd}^l), h(\gamma_s^u, h_{sd}^l)\} \tag{24}$$

In other words, the maximal delay as defined above can be determined as the maximum of the maximal horizontal distances between the functions γ_s^u and h_{sd}^l as well as between γ_s^u and β_{sd}^l. The interpretations of the delay and buffer bounds in marked graphs are visualized in Fig. 9.

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 3.5, these bounds are now given in terms of arrival curves, i.e., not in the time domain but in the time interval domain as is seen when comparing Fig. 7 and Fig. 10. Again, this is necessary to abstract from the concrete time domain and leads to composability in terms of resources and event streams.

The derivation starts from the transfer functions developed in (20). In order to apply the relations known from real-time calculus, see [29], we first approximate the transfer functions from input v_s to node v_d as follows:

$$\begin{aligned} R_d^l &= (\beta_{sd}^l \otimes G_s) \wedge h_{sd}^l \geq (\beta_{sd}^l \wedge h_{sd}^l) \otimes G_s \\ R_d^u &= (\beta_{sd}^u \otimes G_s) \wedge h_{sd}^u \leq \beta_{sd}^u \otimes G_s \end{aligned}$$

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 [29] 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 3.5. 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*

$$\alpha_d^u = \Big((\gamma_s^u \otimes \beta_{sd}^u) \oslash (\beta_{sd}^l \wedge h_{sd}^l) \Big) \wedge \beta_{sd}^u \tag{25}$$

$$\alpha_d^l = \gamma_s^l \otimes (\beta_{sd}^l \wedge h_{sd}^l) \tag{26}$$

where we use $\beta_{sd}^u, \beta_{sd}^l$ from (20).

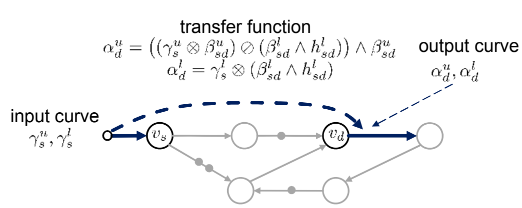

Figure 10: Transfer functions in time interval domain.

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 Fig. 5. This enables us to use the remaining service as an input to some other process and thereby, representing fixed-priority scheduling where the first process, i.e., the one that gets the initial service from the provided resource, has the 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 balance equation of a greedy process component according to Def. 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:

$$\beta_d'^l(\Delta) = \sup_{0 \leq \lambda \leq \Delta} \{\beta_d^l(\lambda) - \alpha_d^u(\lambda)\} \tag{27}$$

$$\beta_d'^u(\Delta) = (\inf_{\Delta \leq \lambda} \{\beta_d^u(\lambda) - \alpha_d^l(\lambda)\})^{\geq 0} \tag{28}$$

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 [6, 31].

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] which is also compositional. 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 Fig. 11a. 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 $T2$

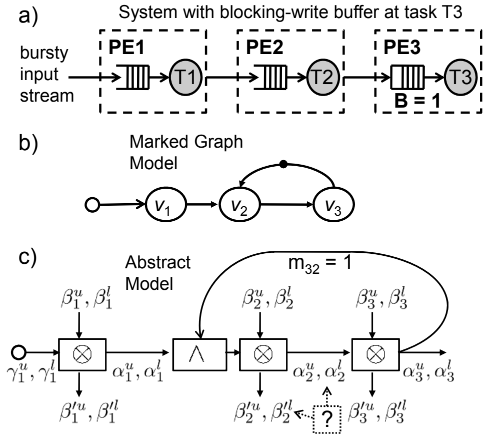

Figure 11: A system with a single finite buffer (a), with its marked graph model (b), and the abstract greedy marked graph processes model (c).

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 Fig. 11b. The abstract model of greedy marked graph processes that is used for performance evaluation of the system with method MG is shown in Fig. 11c.

Scenario. We compare methods MG and FB in terms of tightness of the computed performance metrics for the system in Fig. 11a. 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 Fig. 11c. 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 Fig. 12a. 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 Fig. 12b. 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.

5.2 Validation

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

Table 1: Mapping and cycles for each of the processes in Fig. 13.

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
sim. [ms]	0.02	0.023	0.026	0.031	0.12	0.11
an. [ms]	0.02	0.023	0.027	0.031	0.151	0.13
output	7	8	9	10	11	12
sim. [ms]	0.56	0.688	1.025	0.721	0.691	1.28
an. [ms]	0.56	0.688	1.048	0.726	0.692	1.316

System and models. We use an application from the area of software defined radio. It is adapted from [21]. The Wireless LAN (WLAN) and TD-SCDMA applications run simultaneously. Both of them are modeled as marked graphs as depicted in Fig. 13. In contrast to [21], 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 [21] 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 [6, 31] 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).

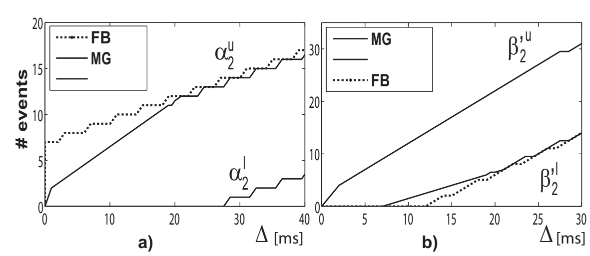

Figure 12: Comparison of methods FB and MG for the output event stream (a), and the remaining service (b). Note that method FB does not compute lower bounds on the output event streams, and upper bounds on the remaining services.

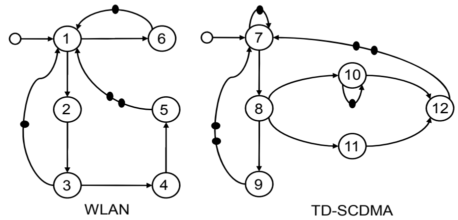

Figure 13: Marked graphs that model WLAN and TD-SCDMA applications.

Scenario. We compare results from analysis computed with inequality (24) 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 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.

7. ACKNOWLEDGMENTS

The work is partially supported by NCCR-MICS, a center supported by the Swiss National Science Foundation under grant number 5005-67322, and by the European Union project COMBEST under grant number 215543.

8. REFERENCES

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

[2] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee, *Synthesis of embedded software from synchronous dataflow specifications*, J. VLSI Signal Process. Syst. **21** (1999), no. 2, 151–166.

[3] A. Bose, X. Jiang, B. Liu, and G. Li, *Analysis of manufacturing blocking systems with network calculus*, Perform. Eval. **63** (2006), no. 12, 1216–1234.

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

[5] S. Chakraborty and D. L. Dill, *Approximate algorithms for time separation of events*, ICCAD '97: Proceedings of the 1997 IEEE/ACM international conference on Computer-aided design, 1997, pp. 190–194.

[6] S. Chakraborty, S. Künzli, and L. Thiele, *A general framework for analysing system properties in platform-based embedded system designs*, DATE '03: Proceedings of the conference on Design, Automation and Test in Europe, 2003, p. 10190.

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

[8] 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** (1985), no. 3, 210–220.

[9] R.L. Cruz, *A calculus for network delay. i. network elements in isolation*, Information Theory, IEEE Transactions on **37** (1991), no. 1, 114–131.

[10] B.A. Davey and H.A. Priestley, *Introduction to lattices and order*, 2nd ed., Cambridge University Press, 2002.

[11] H. Hulgaard and S.M. Burns, *Bounded delay timing analysis of a class of csp programs with choice*, Advanced Research in Asynchronous Circuits and Systems, 1994., Proceedings of the International Symposium on, Nov 1994, pp. 2–11.

[12] M. Jersak and R. Ernst, *Enabling scheduling analysis of heterogeneous systems with multi-rate data dependencies and rate intervals*, DAC '03: Proceedings of the 40th conference on Design automation, 2003, pp. 454–459.

[13] M. Jersak, K. Richter, and R. Ernst, *Performance analysis for complex embedded applications*, International Journal of Embedded Systems **1** (2005), no. 1/2, 33–49.

[14] B. Jonsson, S. Perathoner, L. Thiele, and W. Yi, *Cyclic dependencies in modular performance analysis*, EMSOFT '08: Proceedings of the 8th ACM international conference on Embedded software, 2008, pp. 179–188.

[15] J.-Y. Le Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*, Springer, 2001.

[16] E.A. Lee and D.G. Messerschmitt, *Synchronous data flow*, Proceedings of the IEEE **75** (1987), no. 9, 1235–1245.

[17] E.A. Lee and T.M. Parks, *Dataflow process networks*, Proceedings of the IEEE **83** (1995), no. 5, 773–801.

[18] R. Lu and C.-K. Koh, *Performance analysis of latency-insensitive systems*, Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on **25** (2006), no. 3, 469–483.

[19] A. Maxiaguine, S. Künzli, and L. Thiele, *Workload characterization model for tasks with variable execution demand*, DATE '04: Proceedings of the conference on Design, automation and test in Europe, 2004, p. 21040.

[20] P. B. McGee, S. M. Nowick, and E. G. Coffman, Jr., *Efficient performance analysis of asynchronous systems based on periodicity*, CODES+ISSS '05: Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, 2005, pp. 225–230.

[21] O. Moreira, F. Valente, and M. Bekooij, *Scheduling multiple independent hard-real-time jobs on a heterogeneous multiprocessor*, EMSOFT '07: Proceedings of the 7th ACM & IEEE international conference on Embedded software, 2007, pp. 57–66.

[22] T. Murata, *Petri nets: Properties, analysis and applications*, Proceedings of the IEEE **77** (1989), no. 4, 541–580.

[23] C. D. Nielsen and M. Kishinevsky, *Performance analysis based on timing simulation*, DAC '94: Proceedings of the 31st annual Design Automation Conference, 1994, pp. 70–76.

[24] T. Pop, P. Eles, and Z. Peng, *Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems*, CODES '02: Proceedings of the tenth international symposium on Hardware/software codesign, 2002, pp. 187–192.

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

[26] R. Reiter, *Scheduling parallel computations*, J. ACM **15** (1968), no. 4, 590–599.

[27] S. Schliecker, S. Stein, and R. Ernst, *Performance analysis of complex systems by integration of dataflow graphs and compositional performance analysis*, DATE '07: Proceedings of the conference on Design, automation and test in Europe, 2007, pp. 273–278.

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

[29] L. Thiele, S. Chakraborty, M. Gries, and S. Künzli, *A framework for evaluating design tradeoffs in packet processing architectures*, DAC '02: Proceedings of the 39th conference on Design automation, 2002, pp. 880–885.

[30] K. Tindell and J. Clark, *Holistic schedulability analysis for distributed hard real-time systems*, Microprocess. Microprogram. **40** (1994), no. 2-3, 117–134.

[31] E. Wandeler, L. Thiele, M. Verhoef, and P. Lieverse, *System architecture evaluation using modular performance analysis: a case study*, Int. J. Softw. Tools Technol. Transf. **8** (2006), no. 6, 649–667.

[32] M. Wiggers, M. Bekooij, P. Jansen, and G. Smit, *Efficient computation of buffer capacities for multi-rate real-time systems with back-pressure*, CODES+ISSS '06: Proceedings of the 4th international conference on Hardware/software codesign and system synthesis, 2006, pp. 10–15.

[33] A. Xie, S. Kim, and P. A. Beerel, *Bounding average time separations of events in stochastic timed petri nets with choice*, ASYNC '99: Proceedings of the 5th International Symposium on Advanced Research in Asynchronous Circuits and Systems, 1999, p. 94.