Generalized Finitary Real-Time Calculus

Kai Lampka*†, Steffen Bondorf‡, Jens B. Schmitt‡, Nan Guan§, Wang Yi*
*Real-Time Systems Group, Department of Information Technology, Uppsala University, Sweden
†Elektrobit Automotive, Germany
‡Distributed Computer Systems (DISCO) Lab, University of Kaiserslautern, Germany
§Department of Computing, Hong Kong Polytechnic University, Hong Kong

***Abstract*—Real-time Calculus (RTC) is a non-stochastic queuing theory to the worst-case performance analysis of distributed real-time systems. Workload as well as resources are modelled as piece-wise linear, pseudo-periodic curves and the system under investigation is modelled as a sequence of algebraic operations over these curves. The memory footprint of computed curves increases exponentially with the sequence of operations and RTC may become computationally infeasible fast. Recently, Finitary RTC has been proposed to counteract this problem. Finitary RTC restricts curves to finite input domains and thereby counteracts the memory demand explosion seen with pseudo periodic curves of common RTC implementations. However, the proof to the correctness of Finitary RTC specifically exploits the operational semantic of the greed processing component (GPC) model and is tied to the maximum busy window size. This is an inherent limitation, which prevents a straight-forward generalization. In this paper, we provide a generalized Finitary RTC that abstracts from the operational semantic of a specific component model and reduces the finite input domains of curves even further. The novel approach allows for faster computations and the extension of the Finitary RTC idea to a much wider range of RTC models.**

I. Introduction

Real-time Calculus (RTC) is a non-stochastic queuing theory. It is based on (min,+) and (max,+) algebra and models the data entry to the network and the service of resources by bounding curves in the interval time domain. This modelling promises to achieve accurate bounds on an event stream's end-to-end delay and components' buffer requirement whilst keeping the runtime of analysis relatively small; especially opposed to stochastic queueing theories, state-based model-checking, and discrete event simulation techniques. The efficiency is of great importance when the design of a system is optimized with a design space exploration.

Increasing the accuracy of RTC comes at a price: the curves bounding data and service are implemented as piece-wise linear, pseudo-periodic functions. They allow accurate descriptions of complex data arrival and processing patterns, e. g., sporadic arrival with bursts or time-triggered resource arbitration. An RTC analysis is a sequence of (min,+)- and (max,+)-algebraic operations on these curves. When successively stepping forward in the analysis, the output curve of an operation usually possesses a periodic tail of the size of the hyperperiod of the input curves. This yields a significant increase in the computation's memory demand run-time of down-streamed RTC-operations.

Fig. 1 illustrates an example to this on the simple plus operation of two curves. The two input curves consist of

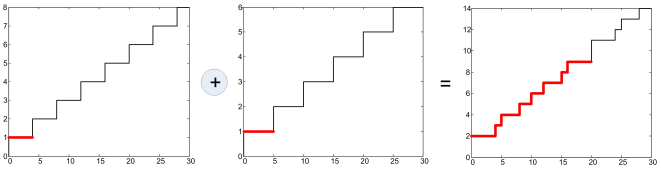


Fig. 1. Red segments of each curve need to be stored to fully characterize the whole curve. This part can increase vastly by the plus operation.

periodic parts only, with period 4 and 5, respectively. Hence, the output curve has period 20, defined by a total of 8 segments. An analysis usually consists of many components that, in turn, define a large sequence of operations. Then, the number of segments of the involved curves increases exponentially, as does the analysis run-time. This phenomenon is called *period explosion*. It is the major reason why the analysis of large-scale systems in RTC can become prohibitive. An industrial example can be found in [13] where an avionic onboard communication system is analyzed with RTC and this limitations become apparent.

The idea of Finitary RTC [4] is to overcome this obstacle by making complete tail descriptions of curves obsolete. Yet, it still guarantees the same analysis results as with the original curves. To do so, [4] formally derives prefix bounds for the so called Greedy Processing Component (GPC) model of RTC. The bounds rely on the busy period concept which the authors of [4] borrowed from classical real-time scheduling theory [9]. Therefore, we call this approach *component-finitary RTC*.

For many other commonly used RTC component models, the greedy processing semantic does not apply or the busy period is not well-defined. Therefore, we develop a generalized form of Finitary RTC in this paper. We provide rules for bounding the length of input curve prefixes for each individual (min,+)- and (max,+)-algebraic operator rather than an entire component model. We call this novel concept the generalized *operator-finitary RTC*. It yields the following advantages:

- **Universality**. Finitary RTC can be made accessible to a general class of component models that are ascribed by RTC operators. Particularly, we demonstrate how our theorems can be applied to the *AND* component models [5], Greedy Traffic Shapers [16], [17], as well as the end-to-end analysis principles Pay Burst Only Once (PBOO) [8] and Pay Multiplexing Only Once (PMOO) [12].
- **Methodological Closure.** Bringing fintiary RTC to the operator level makes it independent of the GPC's op-

erational semantic. This eliminates the need to borrow concepts from real-time feasibility theory. We exploit geometric properties of RTC's stair-case curves and their linear approximations. Therefore, the solution is closed within the RTC modeling and analysis framework and we do not need to refer to scheduling theory.

- **Efficiency Improvements**. Unlike the prefix bounds of [4] resembling the maximum busy period size (MBS) of a component, our bounds are based on two novel concepts: (a) under- and overapproximating functions that bound the stair-case input curves with much simpler curves (b) a lower bound on their maximum vertical distance. Hence, our prefix bounds are commonly smaller than the MBS and the analysis efficiency is further improved.

The remainder of this paper is structured as follows: Sec. II presents the related work, in Sec. III we present standard RTC and Sec. IV presents finitary RTC together with its open issues. Sec. V presents our generalization of finitary RTC, followed by application to the important analysis principles and component models in Sec. VI. Sec. VII evaluates our contributions and Sec. VIII concludes the paper.

II. Related Work

Complementing traditional queuing theoretic approaches, there exists a branch of analysis techniques which base their reasoning on the time-interval domain. These techniques use different notions of Network Calculus [8] and exploit operations rooted in the (min,+) and (max,+) algebra. The most prominent methods and tools of this branch of real-time system theory are Real-time Calculus [15] and its MATLAB-based implementation "Modular Performance Analysis" (MPA) toolbox [17], the "SymTA/S toolsuite" [6] and the "Disco Deterministic Network Calculator" (DiscoDNC) [1].

Although the different tools are based on the same dioid algebra, they strongly differ in the component-level of abstraction. On the one hand, there are tools like the MPA toolbox or SymTA/S which handle each component of a system explicitly. On the other hand, there is the DiscoDNC which puts its focus on a specific data or communication flow within a system. In its flow-oriented analysis, it exploits principles like Pay Bursts Only Once (PBOO) or Pay Multiplexing Only Once (PMOO) in order to obtain tighter end-to-end delay bounds.

The technique of this paper aims at simplifying periodic tail description of curves of the RTC. The initial idea to this is presented in the technical report [14], but lacked the definition and proofs of the tightness preserving bound on the lengths of prefixes as presented here. In consequence, [14] stayed at the level of an approximation technique and can be seen as preliminary work.

The prefix bound for the input curves of finitary RTC [4] relates to the bound on the busy periods of a greedy processing component [9]. In consequence, the proofs provided in [4] are tied to the operational semantic of a greedy processing component (GPC) of RTC [17] or component models alike. This dependency limits the applicability of finitary RTC; it excludes flow-oriented analyses and models with synchronization primitives and traffic shaping elements entirely. In this paper, we tackle this shortcoming by making finitary RTC accessible for these types of analyses and principles.

III. Background Theory: RTC-based Analysis

A. Event Model and Resource Model

With RTC, event streams or task activations are abstractly described by a pair $\alpha(\Delta) = [\alpha^u(\Delta), \alpha^l(\Delta)]$ of curves. The curve providing an upper bound on arrivals is $\alpha^u(\Delta)$ and $\alpha^l(\Delta)$ denotes the lower arrival curve bound. Let $R(s,t)$ be a cumulative counting function associated with an event stream. Upper and lower curve bound the number of events in any *time interval* of length T: $\alpha^l(t-s) \leq R(s,t) \leq \alpha^u(t-s)$, with $s < t$ and $\alpha^l(0) = \alpha^u(0) = 0$.

By scaling the event related arrival curves with the worst and best-case resource demand of the associated events, they can be transferred into workload-based arrival curves.

With RTC, the availability of a processing or communication resource units is modelled in a similar way, namely by a pair of upper an lower strict service curves $\beta(\Delta) = [\beta^u(\Delta), \beta^l(\Delta)]$. These curves bound the availability of resource units for any time interval $[s,t)$: $\beta^l(t-s) \leq C(s,t) \leq \beta^u(t-s)$, where function $C(s,t)$ gives the number of service units available from a resource with $s < t$ and $\beta^l(0) = \beta^u(0) = 0$.

The presented curves cumulatively count data and service, i.e., it holds that $\forall \Delta \geq 0 : \alpha^{\{u,l\}}(\Delta) \geq 0$ and $\beta^{\{u,l\}}(\Delta) \geq 0$.

B. RTC-based System Analysis

With RTC one describes a (distributed) real-time system as a network of asynchronously interacting components $\mathcal{C} := \{C_1, ..., C_n\}$. Components are assumed to be connected via unbounded FIFO channels. In a modelled system, directly connected component C_i and C_j interact via the forwarding of events or the left-over service curve.

With RTC this relation is captured by piping the arrival curve $\alpha_{i,j}$ (or service curve $\beta_{k,i}$) as input curve(s) into the flow equations of the directly downstreamed component. In this paper, we annotate curves with pairs of lower indices: the first position addresses the source component, i. e., the component which produces the respective curve as output. The second element of the pair addresses the component which consumes the respective curve as input. For $\alpha_{i,j}$, component C_i is the source putting out an event stream bounded by $\alpha_{i,j}$ and component C_j is the receiver of it. Accordingly, $\beta_{k,i}$ denotes the left-over service curve after component C_k served the event stream crossing it. This service is then available at the downstream component C_i.

For being generic, we replace individual indices with an asterisk if suitable, e. g., $\alpha_{*,j}$ refers to some input curve fed into component C_j and service curve $\beta_{k,*}$ refers to the left-over service curve of component C_k.

In line with the above discussion, RTC components can be understood as transformers of arrival or service curves. The respective transformation either precisely reflects or at least overapproximates the operational semantic of the modelled

entity, e. g., a synchronization element could be modelled by an *AND*-component [5].

Curve transformations of a component model are expressed as flow equations that employ (min,+) and (max,+) operators.

1) RTC-operators: RTC-operators are common (min,+) and (max,+) operators like minimum, sum and difference of curves, together with more involved operators like the (min,+) and (max,+) convolution and deconvolution. Whereas the common operators are defined as expected, the convolution and deconvolution operators are more evolved.

$$\begin{array}{l}(a \otimes b)(\Delta) = \inf_{0\le\lambda\le\Delta}\{a(\Delta-\lambda)+b(\lambda)\}\\ (a \oslash b)(\Delta) = \sup_{\lambda\ge 0}\{a(\Delta+\lambda)-b(\lambda)\}\\ (a \overline{\otimes} b)(\Delta) = \sup_{0\le\lambda\le\Delta}\{a(\Delta-\lambda)+b(\lambda)\}\\ (a \overline{\oslash} b)(\Delta) = \inf_{\lambda\ge 0}\{a(\Delta+\lambda)-b(\lambda)\}\end{array} \quad (1)$$

The symbols $\otimes$ and $\oslash$ denote the (min,+) convolution, deconvolution operator and the symbols $\overline{\otimes}$ and $\overline{\oslash}$ denote the (max,+) convolution, deconvolution operator respectively.

With piece-wise linear curves, which are pseduo-periodic, the potentially infinite number of function pairings from input curve a and b is reduced to a finite number of pairs, namely one value for each linear segment and their combinations. With the deconvolution operators, the number of relevant pairs might become very large, as for the computation of a specific function value one needs to iterate over all possible pairs of linear segments $a(\Delta+\lambda)$ and $b(\lambda)$ [3]. This follows from the above definitions of $\oslash$ and $\overline{\oslash}$. It makes a restriction of input curves to a finite domain and w. r. t. these operators non-trivial.

The above RTC-operators are used in flow equations for computing a bound on the outgoing events of a component or the left-over service of a component.

2) Flow-equations of the GPC: The *Greedy Processing Component* (GPC) model [17] is the major component model of RTC. It is characterized by the following flow equations:

$$\begin{array}{l}\alpha^u_{i,*} = \min\{(\alpha^u_{*,i} \otimes \beta^u_{*,i}) \oslash \beta^l_{*,i}, \beta^u_{i,*}\}\\ \alpha^l_{i,*} = \min\{(\alpha^l_{*,i} \oslash \beta^u_{*,i}) \otimes \beta^l_{*,i}, \beta^l_{*,i}\}\\ \beta^u_{i,*} = (\beta^u_{*,i} - \alpha^l_{*,i}) \overline{\oslash} 0\\ \beta^l_{i,*} = (\beta^l_{*,i} - \alpha^u_{*,i}) \overline{\otimes} 0\end{array} \quad (2)$$

The GPC-oriented analysis is a straight forward mapping of the network's structure to a network of GPCs. It demands to compute a set of input curves for each component on the basis of the above flow equations. They enable RTC to compute bounds on the delay and the backlog experienced at that component. By summing up delay bounds over a sequence of GPCs, the end-to-end bound for the events of a specific stream is obtained. Delay and backlog bounds of each component are calculated as the least upper bound on the vertical and horizontal distance of the upper arrival and lower service input curves to a GPC [17]. The exact definitions follow below.

3) Computing bounds on delays and buffer sizes: For convenience, we assume the presence of a function $v\_dist(a,b)$ which returns the least upper bound on the vertical distance of two curves a and b:

$$v\_dist(a,b) = \sup_{\Delta\ge 0}\{a(\Delta)-b(\Delta)\} \quad (3)$$

This translates into the backlog bound of a GPC C_i as follows:

$$backlog_i \le v\_dist(\alpha^u_{*,i}, \beta^l_{*,i}) \quad (4)$$

For convenience, we already define the pseudo-inverse of the above function and refer to it with $v\_dist^{-1}(a,b,v)$. We define it as follows:

$$v\_dist^{-1}(a,b,v) = \sup\{\Delta \ge 0 \,:\, a(\Delta)-b(\Delta) > v\} \quad (5)$$

This pseudo-inverse is the largest value λ such that the vertical distance of a and b and $\forall\lambda' > \lambda$ is smaller than v.

In analogy to the vertical distance, one may define functions $h\_dist(a,b)$ and $h\_dist^{-1}(a,b,h)$ as follows:

$$\begin{array}{l}h\_dist(a,b) = \sup_{\lambda\ge 0}\{\inf\{h \ge 0 \,:\, a(\lambda) \le \beta(\lambda+h)\}\}\\ h\_dist^{-1}(a,b,h) = \sup\{\Delta \ge 0 \,:\, a(\Delta) \le b(\Delta+h)\}.\end{array} \quad (6)$$

This gives the upper bound on the (positive) horizontal distance h of curve a and b and its pseudo-inverse.

For an upper arrival curve $\alpha^u_{*,i}$ and a lower service curve $\beta^l_{*,i}$ serving as input to a GPC C_i, the above definition of $h\_dist(a,b)$ directly translates into the GPC's delay bound:

$$delay_i \le h\_dist(\alpha^u_{*,i}, \beta^l_{*,i}) \quad (7)$$

IV. Component-Finitary RTC and its Limitation

A. Component-Finitary RTC

Finitary RTC [4] tackled the scalability problem of original RTC for one particular RTC-component type, the GPC. It borrowed the *busy period* concept from real-time feasibility theory [9]. A busy period is a maximum interval in time during which the processor / server is continuously busy. Instead of analyzing the scheduling behavior along the infinite time line, it is possible to restrict the analysis to individual busy periods. What has happened before the last idle time instant, does not affect the current scheduling behavior. For any system where the long-term overall resource requirement rate is smaller than 1, the maximum length of the busy period is pseudo-polynomially bounded. In practice, this bound is much smaller than the curves' hyperperiods that arise during the analysis which is the insight to the efficiency improvement for RTC.

Component-Finitary RTC applies the idea of a busy period to the GPC. Thereby it is limited to GPC those long-term resource requirement rate is smaller than 1. In this setting it shows the following key points:

- Delay bound and backlog bound of a single GPC can be safely derived within the maximum busy period size (MBS) of this component. The involved arrival curve and service curve can both be reduced to a prefix of size MBS.
- For the output curves of a GPC, namely the output arrival curve and the left-over service curve, the maximally computable prefix depends on the input curves. Let them have prefixes of size x, then we can compute output curves of prefix size up to $x - \text{MBS}$.

Analyzing a GPC network with prefixed initial input curves thus reduces the computable output prefix by each crossed component's MBS. However, the curves fed into the last component must still possess a prefix of this GPC's MBS.

Otherwise, the delay bound and the backlog bound cannot be safely computed. Deriving sufficiently large initial prefixes is one of the main problems for finitary RTC. In [4], this is solved with two-step procedure:

1) An entire analysis is executed with overapproximated curves. At the end of this step, each GPC can upper bound its MBS.
2) The network is traversed in opposite direction of the paths taken by event streams and resources. During this backtracking, the upper bounds on all crossed GPCs' MBS are summed up.

The backtracking terminates as soon as all initial input curves to the system are reached. Each initial curve can be safely prefixed to the sum of MBS bounds reaching it. In a final step, the RTC analysis is executed with the prefixed curves in order to attain the accurate bounds of original RTC.

B. Limitation of Component-Finitary RTC

The procedure consisting of an approximate analysis and backtracking of prefix requirements is generally applicable to any component model. However, with prefix sizes based on the sum of MSBs for each traversed component, this step with component-Finitary RTC may still lead to initial input curves of significant size.

The core of current Component-Finitary RTC prevents a generalization. The MBS derivation, based on the *busy period* concept from real-time feasibility theory, is natural to the GPC as it essentially models preemptive fixed-priority scheduling. The proofs given in [4], therefore, only apply to the GPC, i. e., a different component model requires novel proofs considering its operational semantics. However, these need not guarantee for a bounded busy period. The foremost example to this is the AND-component. At each point in time, at least one of its two input port buffers is not empty. Therefore, the busy period cannot be bounded and neither can the history to consider. In summary, current Component-Finitary RTC only applies to the resource consuming component models that follow a greedy processing, fixed-priority semantic.

In RTC, components are ascribed with flow equations – sequences of (min,+) and (max,+) operations. We generalize finitary RTC by relating it to these RTC-operators rather than providing proofs for individual component models. Without borrowing concepts from related theories, we achieve a generic solution within the RTC methodology. The provided proofs are independent of any operational semantic and thus the properties they establish are universally valid for an operator. Moreover, our novel Operator-Finitary RTC also achieves tighter prefix bounds and thus faster computation run-times.

V. Generalized, Operator-Finitary RTC

A. Preliminaries

1) Fast but approximate RTC-based System Analysis: Backlog and delay bounds of components are obtained by evaluating Eq. (4) and (7) for each resource consuming component and w. r. t. some input curves $\alpha^u_{*,i}$ and $\beta^l_{*,i}$. In case of complex curves fed from the environment into the outer components and long sequences of computations for determining all relevant curves, computational effort may significantly increase. This stems from the fact, that output curve tails are defined by the hyperperiod of input curve tails.

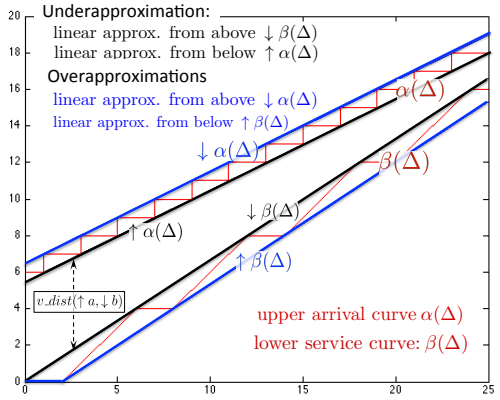


Fig. 2. Arrival and service curve and their linear approximations

The efficiency of the computation within RTC can drastically be improved if the curves are simplified. Such simplifications, however, need to be safe, i. e., computed delay and backlog bounds need to be larger than with the original curves.

Conservativeness of delay and backlog bounds is guaranteed by overapproximation: Upper curves are replaced by upper bounds and lower curves are replaced by lower bounds. The combination of upper and lower bounding of complex arrival and service curves introduces a larger variety of behaviours compared to the original set of curves. Underapproximation refers to lower bounding of arrival curves and upper bounding of service curves, reducing covered behaviours.

In this work, we use curve simplifications for efficiently computing upper and lower bounds on the maximum vertical distances of pairs of curves fed into RTC-operators. These allow us to ultimately derive bounds on the relevant domains of input curves w. r. t. the required domain size of an operation's output curve. We use the following definitions:

- Curve $\downarrow\gamma$ is an upper bound on curve γ if $\forall\Delta \geq 0 : \gamma(\Delta) \leq \downarrow\gamma(\Delta)$ holds.
- Curve $\uparrow\gamma$ is a lower bound on curve γ if $\forall\Delta \geq 0 : \gamma(\Delta) \geq \uparrow\gamma(\Delta)$ holds.

The construction of linear upper or lower bounds that consist of a single linear function appears straightforward: RTC curves are either sub- or super-additive. According to Fekete's lemma for sub- and superadditive sequences, there exists a finite long-term rate $\rho = \lim_{\Delta\to+\infty} \frac{\gamma(\Delta)}{\Delta}$. This allows to derive the following linear approximations for a curve γ:

- $\downarrow\gamma = \max(0, N_u + \rho \cdot \Delta)$ and
- $\uparrow\gamma = \max(0, N_l + \rho \cdot \Delta)$ such that
- $\uparrow\gamma(\Delta) \leq \gamma(\Delta) \leq \downarrow\gamma(\Delta)$ holds.

These curves are also known as affine curves. Fig. 2 illustrates how they approximate from above or below.

When executing approximate system analysis, the exact definition of an upper or lower bound becomes irrelevant, e. g., instead of linear overapproximations, one could also exploit approximations based on specific arrival patterns like

pjd [11] or minimum and maximum combinations of linear curves [7] etc. As far as the domain bounding is concerned, we rely on linear approximations of the above kind. Similar to Component-Finitary RTC, we stress the use of linear functions which have the lowest feasible slopes in case of arrival curves and the largest feasible slope in case of service curves.

Using (linear) upper and lower bounding on curves α and β allows us to (efficiently) compute an upper and lower bound on their maximum vertical and horizontal distance. The following relations apply:

$$v\_dist(\downarrow\alpha, \uparrow\beta) \geq v\_dist(\alpha, \beta) \geq v\_dist(\uparrow\alpha, \downarrow\beta)$$

and

$$v\_dist^{-1}(\downarrow\alpha, \uparrow\beta, v) \geq v\_dist^{-1}(\alpha, \beta, v) \geq v\_dist^{-1}(\uparrow\alpha, \downarrow\beta, v)$$

With α being an upper input arrival curve and β being a lower input service curve, the vertical distance directly translates into the backlog bound of the respective component. Thereby, $v\_dist(.)$ applied to the overapproximations delivers an upper bound on the maximum backlog and its application to the underapproximations yields the lower bound. This observation and the above properties build the basis of our proofs.

For the delay bound, a respective reasoning holds w. r. t. function h_dist. Details are omitted for brevity.

B. Technical problem description

In the following, we rely on the these definitions.

- We call the fragment of a curve defined on the interval $[0, k]$ the (k-)prefix of a curve. Parameter k gives the size or length of the prefix. The part of the curve defined on the interval $(k, +\infty)$ is called tail. Note, that tools like the MPA-toolbox implicitly support this definition.
- Let curves a and b be standard RTC curves defined on the interval $[0, +\infty)$.
- Let curves a' and b' be their prefixed counterparts, i. e., they are defined on the finite interval $[0, k_a]$ and $[0, k_b]$, where for $\Delta \in [0, k_a] : a(\Delta) = a'(\Delta)$ and for $\Delta \in [0, k_b] : b(\Delta) = b'(\Delta)$ holds.
- Let $\odot$ be an operator relevant for RTC, i. e.,

$$\odot \in \{\otimes, \overline{\otimes}, \min, +, -, \oslash, \overline{\oslash}\}$$

For a finite constant k we need to clarify the condition on the size of k_a and k_b w. r. t. to k and operator $\odot$ such that

$$\forall \Delta \in [0, k] : \quad (a \odot b)(\Delta) = (a' \odot b')(\Delta)$$

holds. Requesting that k_a and k_b are sufficiently large solves the subtle problem that the operation $(a' \odot b')(\Delta)$ is defined only for $\Delta \in [0, k]$ and not for $\Delta \in (k, \infty)$. Relating prefix bounds to RTC operators allows us to ignore indices to components, as well as the differentiation between upper and lower input curves. The type of curves and the exploited features will be made explicit upon need.

In the following we proceed by groups of RTC-operators and employ a combination of upper and lower linear bounding of curves as introduced in Sec. V-A1.

C. Operator-Finitary RTC for Common Operators

Operator group I is defined by the set $\{\otimes, \overline{\otimes}, \min, +, -\}$.

Lemma 1: *Domain bounds with common RTC operators.* Satisfaction of the relation $k \leq \min(k_a, k_b)$ yields that for $\odot \in \{\otimes, \overline{\otimes}, \min, +, -\}$ and $\Delta \in [0, k] : (a \odot b)(\Delta) = (a' \odot b')(\Delta)$ holds.

The above lemma directly follows from the definition of the operators of group I: when constructing output curve $c(\Delta) = (a \odot b)(\Delta)$ for $\Delta \in [0, k = \min(k_a, k_b)]$ one only needs to use function values of curve a and b up to k_a and k_b.

For example, let $c(\Delta) = a(\Delta) \otimes b(\Delta)$ and let, $c'(\Delta) = a'(\Delta) \otimes b'(\Delta)$. For $0 \leq \Delta \leq \min(k_a, k_b)$ we truly have $a(\Delta) = a'(\Delta)$ and $b(\Delta) = b'(\Delta)$. Therefore, for $\Delta \in [0, \min(k_a, k_b)]$: $c(\Delta) = c'(\Delta)$ has to hold.

Guideline for Prefix Construction 1: Assume that a component's flow equation only consists of operators of group I, e. g., the OR-component of [5]. If it is required to deliver output curves defined on $[0, k]$, then the above lemma implies that its input curves must have prefixes of at least k as well.

D. Operator-Finitary RTC for the (min,+)-Deconvolution

Handling of the (min,+) deconvolution is more complex and restriction of curves to a finite prefix can only be achieved by exploiting structural assumptions made on the input curves.

In the following, let $c(\Delta) = a \oslash b(\Delta) = \sup_{\lambda \geq 0}\{a(\Delta + \lambda) - b(\lambda)\}$ and we exploit the following definitions.

1) Let $\downarrow x$ be a *linear* upper bound and let $\uparrow x$ be a linear lower bound on curve $x \in \{a, b\}$, where we assume that $\downarrow x$ and $\uparrow x$ have the same slope. This is justified as we pick approximations with the lowest possible slope for arrival curves and with the largest possible slope for service curves.
2) We use the symbol ρ when referring to the slope of linear approximations of curve a and we use the symbol σ in case of curve b.
3) Component-Finitary RTC requests that the long-term overall maximum resource utilization is smaller than 1, i. e., , $\rho < \sigma$. With Operator-Finitary RTC we extend to the case $\rho \leq \sigma$. For $\rho = \sigma$, i. e., when facing components with a long-term utilization of 1, we require that the pseudo-inverse of $v\_dist(\uparrow a, \downarrow b)$ but w. r. t. the overapproximations of input curve a and b exists. This means $\downarrow a - \uparrow b \leq v\_dist(\uparrow a, \downarrow b)$ has to hold; see also discussion below.
4) Let Λ be the parameter to bound the prefix of curves a' and b' serving as input to a (min,+) deconvolution. Note, if Λ is a valid prefix bound w. r. t. the (min,+) deconvolution, the definition of the latter already implies that with $k_b \leq \Lambda$ the relation $k_a \leq \Lambda + k$ has to hold in order to obtain the property $a \oslash b(\Delta) = a' \oslash b'(\Delta)$ $\forall \Delta \in [0, k]$.

We define a domain bound Λ as follows:

$$\Lambda = \max(0, \sup(\lambda : \downarrow a(\lambda) - \uparrow b(\lambda) \leq v\_dist(\uparrow a, \downarrow b))) \quad (8)$$

The above definition can be interpreted as follows: the right-hand side of the above inequality, which is $v\_dist(\uparrow a, \downarrow b)$,

delivers the lower bound on the least upper bound of the vertical distance of curve a and b, see Eq. (3). The left-hand side defines the pseudo-inverse of this value, but w. r. t. the overapproximations of curve a and b, see Eq. (5). The maximum computation in the above definition solves the unusual case that the second operand is negative.

Value Λ exists as a side effect of the requirement stated above (number 3 of the listing). It yields that for all input values $\lambda > \Lambda$, we know that $\downarrow a(\lambda) - \uparrow b(\lambda) \leq v\_dist(\uparrow a, \downarrow b)$, because, either (a) curve $\downarrow a$ grows slower than curve $\uparrow b$, which is the setting of components which are have a long-term utilization below 1 or (b) the curves have the same slope, but their tails have a vertical distance below $v\_dist(\uparrow a, \downarrow b)$.

In case requirement 3 form above does not apply, the pseudo-inverse Λ does not exist and the following theorem does not apply.

Theorem 1: *Domain bounds with (min,+) deconvolution* Given a domain bound k, we guarantee $a \oslash b(\Delta) = a' \oslash b'(\Delta)$ if $k_a = \Lambda + k$ and $k_b = \Lambda$.

Proof 1: Let $\underline{v} = v\_dist(\uparrow a, \downarrow b)$. The definition of the $\oslash$ operator, put together with the (linear) underapproximations, yields $\uparrow a \oslash \downarrow b(\Delta) = \underline{v} + (\sigma - \rho)\Delta$.

Moreover, $\bar{v} = v\_dist(\downarrow a, \uparrow b)$ provides a lower bound on $\downarrow a \oslash \uparrow b(\Delta)$ and in analogy to the above equation, we have $\downarrow a \oslash \uparrow b(\Delta) = \bar{v} + (\sigma - \rho)\Delta$. This yields the following property $\downarrow a \oslash \uparrow b(\Delta) - \uparrow a \oslash \downarrow b(\Delta) = \bar{v} - \underline{v}$, which we exploit as follows.

Let $\Lambda = v\_dist^{-1}(\downarrow a, \uparrow b, \underline{v})$, which is equivalent to the definition provided in Eq. (8). Together with the above property, we get:

$$\sup_{\lambda' > \Lambda} (\downarrow a(\Delta + \lambda') - \uparrow b(\lambda')) < \uparrow a \oslash \downarrow b(\Delta) \tag{9}$$

This holds because the rates of the over- and underapproximations are pairwise identical and for $\lambda' > \Lambda : \downarrow a(\lambda') - \uparrow b(\lambda') < \underline{v}$.

Due to the nature of underapproximation, we also have: $\uparrow a \oslash \downarrow b(\Delta) \leq a \oslash b(\Delta)$.

This can be put together with the property of Eq. (9), which allows us to simply exchange the right-hand side of the above equation. We obtain:

$$\sup_{\lambda' > \Lambda} (\downarrow a(\Delta + \lambda') - \uparrow b(\lambda')) \leq a \oslash b(\Delta)$$

This immediately implies that $\sup_{\lambda \geq 0}(a(\Delta + \lambda) - b(\lambda))$ is found on the stretch $\lambda \in [0, \Lambda]$, as the operator values found beyond Λ are already smaller with the overapproximations. Thus, for $k_a = \Lambda + \Delta$ and $k_b = \Lambda$: $a \oslash b(\Delta) = a' \oslash b'(\Delta)$ has to hold. ■

The above theorem affects the computation of a component-global prefix bound as follows.

Guideline for Prefix Construction 2: The Λ computed for a component C_i and w. r. t. to its output bound k yields that the component directly upstream of C_i needs to provide a prefixed arrival curve $\alpha'_{*,i}$ of length $\Lambda + k$. Analogously, at the component one-step upstream of C_i, w. r. t. the sharing of a resource, we need to generate a left-over service curve $\beta'_{*,i}$ of length Λ.

It is also easy to see that for $v\_dist(\uparrow a, \downarrow b)) > 0$ we produce curves which have smaller prefixes than the MBS. Hence our Operator-Finitary RTC is not only more general, it is also likely to be more efficient.

Above, we required that $\rho < \sigma$. This is not a decisive limitation: (a) $\rho > \sigma$ implies that $a \oslash b(\Delta) = \infty$ and thus the respective computation produces a result which is irrelevant for the analysis of a component. (b) The case $\rho = \sigma$ appears to either follow the handling of $\rho < \sigma$, or needs to be manually eliminated. Such an elimination is, however, only necessary in the rare case that the pseudo-inverse $v\_dist^{-1}(\downarrow a, \uparrow b, v\_dist(\uparrow a, \downarrow b))$ does not exist.

E. Operator-Finitary RTC for the (max,+)-Deconvolution

By swapping the arguments we obtain $c(\Delta) = b \overline{\oslash} a(\Delta)$. This allows us to exploit the definitions and properties from above and derive a bound Λ beyond which the value $b(\lambda' + \Delta) - a(\lambda')$ grows. For conciseness and as the (max,+) deconvolution is of less importance for standard RTC-analysis, we omit details.

F. Ensuring Enclosure of Backlogs and Delays

So far, we clarified the size of domains for input curves a' and b' w. r. t. a given domain bound k of an output curve c and w. r. t. a given RTC-operator.

Assuming that at the very last component of a sequence of components, input curve a' and b' are used for bounding the delay and backlog, we also need to clarify the initial value which we require for k_a and k_b at that component.

In the following we develop a new and tighter bound R for guaranteeing backlog enclosure and a new and tighter bound U for guaranteeing delay enclosure.

To formalize this idea, we use the following definitions.

- Let $a = \alpha^u$ be an upper arrival curve, let $b = \beta^l$ be a lower service curve at a component C_i and let a' and b' be their prefixed variants, with k_a and k_b as the domain bounds.
- Let $delay$ and $backlog$ be the performance measures computed with Eq. (4) and (7) and w. r. t. curve a and b.
- Let $delay'$ and $backlog'$ be the performance measures obtained from the prefixed curves a' and b' and let ϕ' be the value of Δ such that Eq. (4) is satisfied. Also, let λ' and h' be the values such that relation Eq. (7) is satisfied.

For the above setting, we give the following lemma.

Lemma 2: *Backlog and Delay enclosure*

$$\lambda' + h' \leq \min(k_a, k_b) \text{ and } \phi' \leq \min(k_a, k_b)$$
$$\Rightarrow delay = delay' \text{ and } backlog = backlog'$$

Proof 2: Curve a' coincides with the non-prefixed curve a on $[0, k_a]$ and b' coincides with the non-prefixed curve b' on $[0, k_b]$. Thus, delay and backlog are identical to those of the original curve model if they are contained in the prefixes. ■

The challenge is to find k_a and k_b such that the above lemma applies. To do so, we introduce variables U and R.

- Let $R := v\_dist^{-1}(\downarrow a, \uparrow b, v\_dist(\uparrow a, \downarrow b))$ and
- Let $U := h\_dist^{-1}(\downarrow a, \uparrow b, h\_dist(\uparrow a, \downarrow b))$.

These are the pseudo-inverses of the lower bounds on the delay and backlog bound, but w. r. t. the overapproximations of a and b.

Theorem 2: For $k_a \geq \max(U, R)$ and $k_b \geq \max(R, U + h\_dist(\downarrow a, \uparrow b))$, the domain bounds k_a and k_b are sufficiently large such that delay and backlog bounds can be found in the intervals $[0, k_a]$ and $[0, k_b]$, i. e., the above lemma applies.

Proof 3: In the following, we re-use the pseudo-inverse of the delay and backlog bound, which we define as follows:

$$u^* := v\_dist^{-1}(a, b, v\_dist(a, b))$$
$$r^* := h\_dist^{-1}(a, b, h\_dist(a, b))$$

We proceed by a case distinction.

a) Delay bound enclosure: The definition of U gives that $\forall \Delta > U : \downarrow a(\Delta) < \uparrow b(\Delta + h\_dist(\uparrow a, \downarrow b))$. As $\downarrow a$ and $\uparrow b$ are linear overapproximations of a and b, this implies that $\forall \Delta > U : a(\Delta) < b(\Delta + h\_dist(\uparrow a, \downarrow b))$ has to hold as well. Exploiting the relation $h\_dist(\uparrow a, \downarrow b) \leq h\_dist(a, b) \leq h\_dist(\downarrow a, \uparrow b)$ yields that $u^* \leq U$ holds. Thus, U is a safe upper bound for the domain of a and $U + h\_dist(\downarrow a, \uparrow b)$ is truly an upper bound for the domain of b. Enclosure of the effective delay bound in their prefixes is guaranteed.

b) Backlog bound enclosure: This case follows the same line of reasoning, but function h_dist is replaced by function v_dist. For brevity we omit further details.

With r^* being bounded by R and u^* being bounded by U, we obtain that delay bound and backlog bound can both be found with a and b on the interval $[0, \max(U, R)]$, resp. $[0, \max(R, U + h\_dist(\downarrow a, \uparrow b))]$ – the definition provided in the above theorem. ■

VI. Generalized, Operator-Finitary RTC for Standard Component Models

Breaking up ties between finitary RTC and specific component models has various practical advantages: universality, simplicity and improved efficiency. In this section, we demonstrate how the former two result in finitary RTC's straightforward applicability for different purposes. Their common prerequisite is the use of (min,+) and (max,+) operators of Eq. (1) outside the context of flow equations ascribing the GPC of Eq. (2). With the generalized finitary, proofs stay valid for any combination of these operations in any context. First, we discuss analysis principles that tighten the end-to-end delay bound of event streams and then we demonstrate how to extend finitary RTC to other component models.

A. The PBOO and PMOO analysis principles

Generalized finitary RTC allows for more accurate delay bounds. This can be achieved by implementing the following analysis principles that capture behavior observable in realistic systems.

Pay Burst Only Once (PBOO, [8]): Analyzing a sequence of GPCs crossed by an event stream demands the derivation of its arrivals, $\alpha^u_{i,*}$ and $\alpha^l_{i,*}$, at the input of each component. Both derivations within the GPC model apply the (min,+) deconvolution to determine the previous component's worst-case output burstiness. The analyzed event stream's burstiness increases with every component it crosses and crucially impacts delay bounding (Eq. (7)). An independent worst-case setting is assumed at every component, i. e., the order of events within the stream that determines the worst-case delay can differ between subsequent components. Considering the entire path of an event stream as a whole, however, shows that the order of events is retained across subsequent hops. The Pay Bursts Only Once (PBOO) analysis principle suggests to convolve the service curves on a stream's path before bounding the delay with h_dist. In contrast to the GPC analysis, burstiness only appears once in the PBOO derivation, guaranteeing for more accurate delay bounds.

Pay Multiplexing Only Once (PMOO, [12]): The PBOO analysis principle solely reduces occurrences of the analyzed event stream's burst term to a minimum. The burstiness of other event streams might still be considered multiple times. They are found in the derivations of $\beta^u_{i,*}$ and $\beta^l_{i,*}$, the service available to the analyzed event stream. From its point of view, multiplexing is paid for more than once. Accordingly, the analysis principle counteracting this situation is known as Pay Multiplexing Only Once (PMOO). It states that service curves should be convolved into as little curves as possible before deriving the required $\beta^u_{i,*}$ and $\beta^l_{i,*}$ from them.

B. The AND-component model

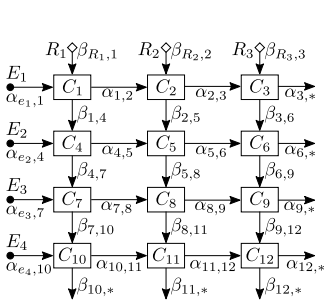
The AND-component model is discussed in [5]. Its operational semantic is as follows.

The component solely produces and output event if there is at least one input event in each input buffer. Upon output event generation, the AND-component consumes exactly one event from each input buffer and produces one output event, where this activity is instantaneous. Therefore, the AND-component models a synchronization primitive as commonly found in distributed computing environments.

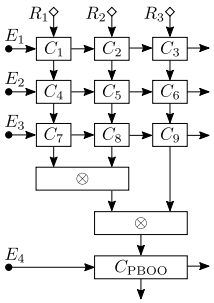
At an abstract level, the flow operation of the AND-component is of the following form: $\alpha^u = \max(\min(a \oslash b, d), \min(\ldots))$; the lower output curve is constructed but by swapping $\min$ and $\max$. We skip the derivation for brevity.

With the upper arrival output curve, we see two min-plus deconvolutions at the inner most level and each embedded into a $\min$ computation. This is completed by an outer $\max$ computation over the aforementioned $\min$ computations.

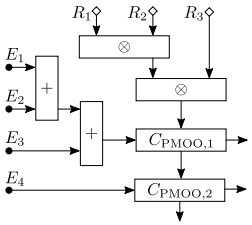
For the outer maximum and the inner minimum computation, Lemma 1 applies, i. e., for producing output curve c of length k we need to ensure that the input curves a, b and d are at least of length k. However, for the inner deconvolution, this may not bet enough, i. e., the prefixes of curve a and b may need to be longer. For deriving the prefix length of a and b for obtaining a prefix of length k for curve $c = a \oslash b$ we can apply a case distinction.



(A) The 4 × 3 mesh of GPCs.



(B) PBOO analysis.



(C) PMOO analysis.

Delays

	Linear	RTC	C-finit	$\odot$-finit
GPC	71.424	21	21	21
PBOO	56.960	19	–	19
PMOO	22.044	15	–	15

Run-times

	Linear	RTC	C-finit	$\odot$-finit
GPC	0.137s	315.97s	0.240s	0.228s
PBOO	0.140s	35.67s	–	0.148s
PMOO	0.035s	17.67s	–	0.035s

(D) Delay bounds and run-times.

Fig. 3. Sample grid network [4] and analysis of event stream E_4.

Let ρ and σ be the long-term rates of curve a, b respectively.

- $\rho < \sigma$. On the basis of Theorem 2, we are able to compute a Λ and thereby bound the value of k_a and k_b
- $\rho > \sigma$. The result of the min-plus deconvolution becomes irrelevant as $a \oslash b = \infty$. It is absorbed by the inner minimum computation.
- $\rho = \sigma$. In case the pseudo-inverse $v\_dist^{-1}(\downarrow a, \uparrow b$, $v\_dist(\uparrow a, \downarrow b))$ exists, Theorem 2 applies. Otherwise, k_a and k_b are unknown and our generalization is not applicable, presumingly to be resolved manually.

This is a clear extension of the component-finitary RTC, since, our operator-finitary RTC features the handling of non-resource consuming component models, e. g., , the AND-component. In addition, it also covers the case $\rho = \sigma$, given that the pseudo-inverse of $v\_dist(\uparrow a, \downarrow b))$ w. r. t. the overapproximations exists.

C. Greedy Traffic Shapers

Traffic shaping is a technique for coping with the non-determinism introduced by bursty workloads. The Greedy Traffic Shaping (GTS) component model for RTC is discussed in [16]. Its operational semantic is as follows.

A greedy shaper with a shaping curve s delays events of an input event stream such that the output event stream has s as an upper arrival curve. Its flow equations are based on a min-plus convolution and a max-pus deconvolution. The convolution is covered by Lemma 1. The max-plus deconvolution is applied to the shaping curve s of the component: $s \overline{\oslash} s$. This convolution targets at a transformation of the shaping curve into a super-additive service curve. Assuming that the shaping curve is not a left-over service curve from an upstreamed component, yields that it is specified on the complete domain of $\mathbb{R}^+$. Hence, a restriction to a subset $[0, k_s]$ when computing $\beta = s \overline{\oslash} s$ is not necessary and for the outer min-plus convolution, we have a prefix available which is as precise as the original curve model.

VII. Evaluation

A. Operator-Finitary RTC with PBOO and PMOO Principles

With a GPC-oriented style of analysis, the computation of an end-to-end delay for a specific event stream is only attainable by summing up the component-local delay bounds along the stream's path. We illustrate this on the sample network from [4] in Fig. 3(A). The employed stair-case curves allow for a tight linear overapproximation. It allows for a tight linear overapproximation of the employed stair-case curves , i.e., the added pessimism has least possible impact. In our evaluation, we are mainly concerned with the delay derivation: the end-to-end delay experienced by any event of stream E_4 is bound by $delay_{10} + delay_{11} + delay_{12}$.

It is easy to show that a component-oriented computation of the end-to-end delay is in general not tight. E. g., $delay_{10}$ is derived from the arrival curve $\alpha^u_{e4,10}$ and $delay_{11}$ is derived from $\alpha^u_{e_1,11} = \min\left\{\left(\alpha^u_{e_4,10} \otimes \beta^u_{7,10}\right) \oslash \beta^l_{7,10}, \beta^u_{7,10}\right\}$. Both local delay bound derivations consider arrival curves $\alpha^u_{e_4,10}$ to its full extent. Using the (min,+)-operations of RTC directly, we can, however, create a *flow-oriented* RTC analysis that counteracts this problem. The flow equation for the analyzed flow, event stream E_4, does not reveal the individual components crossed by it anymore. Their service curves are convolved to the end-to-end service $(\beta_{7,10} \otimes \beta_{8,11}) \otimes \beta_{9,12}$ before E_4's delay is bounded in the (virtual) PBOO-component C_{PBOO}. This last component does not correspond to a real component of the network, and we call it *virtual*.

The PBOO principle solely concerns the analyzed event stream's burstiness, i. e., Fig. 3(A) and Fig. 3(B) only differ in the derivation on E_4's path. Thus, the same problem still applies to the other streams, E_1, E_2 and E_3. Their arrivals appear multiple times in E_4's delay bound derivation. Applying the convolution as early as possible changes the analysis proceeding to Fig. 3(C). The analysis makes heavy use of (min,+) operations outside the context of components: service curves are convolved to $\beta_{R_1,1} \otimes \beta_{R_2,2} \otimes \beta_{R_3,3}$ before subtracting the aggregated arrivals of the higher prioritized event streams. Subtraction is provided by the virtual component $C_{\text{PMOO},1}$ and delay bounding is provided by $C_{\text{PMOO},2}$.

1) Delay Bounds: End-to-end delay bounds of event stream E_4 are shown in Table 3(D), Delays. Delay bounds obtained with any analysis that employs linearly approximated curves are strictly larger than the RTC bounds; even the linear PMOO analysis bound exceeds the GPC analysis result with stair-case curves. As expected, C-finitary RTC preserves the delay bound of the GPC analysis with stair-case curves in standard format. Our operator-finitary RTC covers PBOO and PMOO analysis and thereby delivers tighter delay bounds.

2) Analysis Run-times: For the evaluation of the analysis run-times, we used the MPA toolbox [10] and Matlab $R2015b$

on a 2.7 GHz Intel dual core processor model i5, with $3MB$ shared level 3 cache, $8GB$ of onboard $1866MHz$ LPDDR3 SDRAM and running Mac OS 10.11.3. Table 3(D), Run-times, depicts the time required to compute the according delay bound. We can observe two distinct sources of improvements that counteract the problem of growing hyperperiods:

1) PBOO and PMOO reduce the amount of RTC-operators.
2) Finitary RTC makes tail descriptions of curves obsolete.

The former is depicted in the RTC column of Table 3(D), Run-times. With the original grid network of 12 GPCs and the standard curve layout, the analysis takes 315.97s to complete. The PBOO analysis reduces the run-time by nearly one order of magnitude, to 35.67s, and PMOO computations finish after 17.67s. However, a running time of almost 20s for a model of trivial size clearly indicates the limitations in the efficiency when reducing the numbers of RTC-operators only. It seems that reasonably sized networks of distributed real-time systems with complex communication patterns and activities at different orders of the time scale remain out of the reach of standard RTC-implementations. In such scenarios, the components crossed by the analyzed event stream only constitute a small part of the network. Then, efficiency gains will be smaller [2].

Therefore, improvements of Operator-Finitary RTC are highly desirable for PBOO and PMOO analysis. For both, we achieve a reduction of run-times by more than two orders of magnitude compared to standard RTC. The more interesting fact is, however, that computing delay bounds with linear approximated curves and with Operator-Finitary RTC takes nearly the same time: PBOO finishes in 0.140s vs. 0.148s and the PMOO delay bound computations both required 0.035s – a difference could not be measured.

Finally, let us briefly address the difference in run-times between Component-Finitary RTC and Operator-Finitary RTC. In the GPC-based analysis scheme, run-times are nearly identical. This is for the following reason: for comparability, we computed GPC-global prefixes with Operator-Finitary RTC instead of a distinct prefix for every curve in Eq. (2). This is, in fact, another aspect where our solution can excel the previous one in order to better scale to larger networks.

B. Operator-Finitary RTC with the AND Component Model

Last, we evaluate a network that uses the AND-component model as a synchronization primitive, the model can be found in [5]. The arrival of jobs is modelled as before by pjd-arrival curve, where the service follows once again a TDMA scb-service pattern. Component-Finitary RTC does currently not provide insight on the computable length of the output curve's prefix of the AND-components of this model.

The original RTC analysis with linearly approximated curves required 0.0806s and with stair-case curves, it took 0.3313s – already four times as long in this small example. We prefixed the curves with our Operator-Finitary RTC and observed a run-time of only 0.1205s. Not only can we derive prefixes for networks with AND components, they are also vastly reducing the analysis run-time.

VIII. Conclusion

We eliminated the dependency of Finitary RTC on the greedy processing component and the maximum busy period. To do so, we bring Finitary RTC to the level of RTC-operators. With this generalization, Finitary RTC can be applied to common component models such as the PBOO, PMOO, AND and the GTS-component. These models allow for tighter delay and backlog bounds if compared with a pure GPC-driven analysis approach.

With the extension of the Finitary RTC idea to a much wider range of RTC models, we also developed alternative bounds on required curve prefixes. Deriving these prefix sizes from combinations of under and overapproximations of curves, results in potentially smaller prefixes. They reduce the run-time of RTC-based system analysis even further, making this theory a good candidate for design space exploration.

Acknowledgements

Kai Lampka and Wang Yi are partially supported by UPMARC. Steffen Bondorf is supported by the Carl Zeiss Foundation. Gratitude goes to Simon Perathoner, Urban Suppiger and Lothar Thiele. The joint work [13], [14] on the case study and the heuristic based on linear tail approximations was the starting point for the presented work.

References

[1] S. Bondorf and J. B. Schmitt. The DiscoDNC v2 – A Comprehensive Tool for Deterministic Network Calculus. In *Proc. ValueTools*, 2014.
[2] S. Bondorf and J. B. Schmitt. Calculating Accurate End-to-End Delay Bounds – You Better Know Your Cross-Traffic. In *ValueTools*, 2015.
[3] A. Bouillard and E. Thierry. An Algorithmic Toolbox for Network Calculus. *Journal of Discrete Event Dynamic Systems*, 2008.
[4] N. Guan and W. Yi. Finitary Real-Time Calculus: Efficient Performance Analysis of Distributed Embedded Systems. In *Proc. IEEE RTSS*, 2013.
[5] W. Haid and L. Thiele. Complex task activation schemes in system level performance analysis. In *Proc. CODES+ISSS*, 2007.
[6] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst. *System Level Performance Analysis - the SymTA/S Approach*. The Institution of Electrical Engineers, London, UK, 2006.
[7] K. Lampka, S. Perathoner, and L. Thiele. Analytic real-time analysis and timed automata: A hybrid methodology for the performance analysis of embedded real-time systems. *Design Automation for Embedded Systems*, 14(3):193–227, 2010.
[8] J.-Y. Le Boudec and P. Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer, 2001.
[9] J. P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Proc. IEEE RTSS*, 1990.
[10] Modular Performance Analysis Framework. www.mpa.ethz.ch.
[11] S. Perathoner, K. Lampka, and L. Thiele. Composing Heterogeneous Components for System-wide Performance Analysis. In *DATE*, 2011.
[12] J. B. Schmitt, F. A. Zdarsky, and I. Martinovic. Improving Performance Bounds in Feed-Forward Networks by Paying Multiplexing Only Once. In *Proc. of GI/ITG MMB*, 2008.
[13] U. Suppiger, S. Perathoner, K. Lampka, and L. Thiele. Modular performance analysis of large-scale distributed embedded systems: An industrial case study. Technical Report 330, ETH Zurich, 2010.
[14] U. Suppiger, S. Perathoner, K. Lampka, and L. Thiele. A simple approximation method for reducing the complexity of modular performance analysis. Technical Report 329, ETH Zurich, Aug 2010.
[15] L. Thiele, S. Chakraborty, and M. Naedele. Real-Time Calculus for Scheduling Hard Real-Time Systems. In *Proc. IEEE ISCAS*, 2000.
[16] E. Wandeler, A. Maxiaguine, and L. Thiele. On the use of greedy shapers in real-time embedded systems. *ACM TECS*, 2012.
[17] E. Wandeler, L. Thiele, M. Verhoef, and P. Lieverse. System Architecture Evaluation Using Modular Performance Analysis - A Case Study. *International Journal on Software Tools for Technology Transfer*, 2006.