Finitary Real-Time Calculus: Efficient Performance Analysis of Distributed Embedded Systems

Nan Guan and Wang Yi
Uppsala University, Sweden
Email: {nan.guan|yi}@it.uu.se

***Abstract*—Real-Time Calculus (RTC) is a powerful framework to analyze real-time performance of distributed embedded systems. However, RTC may run into serious analysis efficiency problems when applied to systems of large scale and/or with complex timing parameter characteristics. The main reason is that many RTC operations generate curves with periods equal to the hyper-period of the input curves. Therefore, the analysis in RTC has exponential complexity. In practise the curve periods may explode rapidly when several components are serially connected, which leads to low analysis efficiency.**

In this work, we propose *Finitary RTC* to solve the above problem. Finitary RTC only maintains and operates on a limited part of each curve that is relevant to the final analysis results, which results in pseudo-polynomial computational complexity. Experiments show that Finitary RTC can drastically improve the analysis efficiency over the original RTC. The original RTC may take hours or even days to analyze systems with complex timing characteristics, but Finitary RTC typically can complete the analysis in seconds. Even for simple systems, Finitary RTC also typically speeds up the analysis procedure by hundreds of times. While getting better efficiency, Finitary RTC does *not* introduce any extra pessimism, i.e., it yields analysis results as precise as the original RTC.

I. INTRODUCTION

Real-Time Calculus (RTC) [18] is a framework for performance analysis of distributed embedded systems rooted in the Network Calculus theory [4]. RTC uses *variability characterization curves* [12] to model workload and resource, and analyzes workload flows through a network of computation and communication resources to derive performance characteristics of the system. RTC has proved to be one of the most powerful methods for real-time embedded system performance analysis, and has drawn considerable attention from both academia and industry in recent years.

Although one of RTC's advantages is avoiding the state-space explosion problem in state-based verification techniques such as timed automata [3], it may still encounter scalability problems when applied to systems with complex timing properties. The major reason of RTC's efficiency problem is due to the fact that the workload/resource representations and computations in the RTC framework are defined for the infinite range of time intervals. Although in practise it is possible to compactly represent the workload and resource information of infinite time intervals by finite data structures [21], it may still have to maintain a great amount of information in the analysis procedure and take very long time to complete the computations when the timing characteristics of the system are complex. More specifically, many RTC operations generate curves with periods equal to the LCM (least common multiple) of the input curve periods. Therefore, the curve periods may explode rapidly when several components with complex timing characteristics, (e.g., co-prime periods) are connected in a row. We call this phenomenon *period explosion* (Section III discusses the period explosion problem in detail). In general, the analysis in RTC has *exponential* complexity and in practise the efficiency may be extremely low for large systems with complex timing characteristics.

In this paper, we propose *Finitary Real-Time Calculus*, a refinement of the RTC framework, to solve the above problem. Finitary RTC has *pseudo-polynomial* complexity, and in practise can drastically improve the analysis efficiency of complex distributed embedded systems. The key idea is that only the system behavior in time intervals up to a certain length is relevant to the final analysis results, hence we can safely chop off the part of a workload and resource curve beyond that length limit and only work with a (small) piece of the original curve during the analysis procedure.

While getting better efficiency, Finitary RTC does not introduce any extra pessimism, i.e., the analysis results obtained by Finitary RTC are as precise as using the original RTC approach. This is a fundamental difference between Finitary RTC and other methods that approximate the workload and resource information to simplify the problem but lead to pessimistic analysis results (e.g., adjusting task periods for a smaller hyper-period).

We conduct experiments to evaluate the efficiency improvement of Finitary RTC over the original RTC. Experiments show that Finitary RTC can drastically speed up the analysis. For systems with complex timing characteristics, the analysis procedure in the original RTC may take hours or even days, but Finitary RTC typically can complete the analysis in seconds. Even for simple systems, Finitary RTC also typically speeds up the analysis procedure by hundreds of times.

A. Related Work

Real-Time Calculus (RTC) is based on Network Calculus (NC) [4]. There are several significant differences between RTC and NC. First, RTC can model and compute the remaining service of each component, which is not explicitly considered in NC. Second, while NC mainly uses the upper arrival curves and lower service curves, RTC uses both lower and upper curves for both events and resource, which supports a tighter computation of the output curves. Furthermore, RTC has been extended to model and analyze common problems in the real-time embedded system domain, such as structured event streams [14], workload correlations [6], interface-based design [19], mode switches [15], and cyclic systems [9].

The classical real-time scheduling theory is extended to distributed systems by *holistic analysis* [20]. MAST [8] is a well-known example of this approach. MAST supports offset-based holistic schedulability analysis to guarantee various real-time performance constraints such as local deadlines, global deadlines and maximal jitters. Different from the holistic analysis approach in MAST, RTC supports a compositional analysis framework, where local analysis is performed for each component to derive the output event/resource models and afterwards the calculated output event models are propagated to the subsequent components. SymTA/S [7] is another well-known compositional real-time performance analysis framework, which uses a similar event and resource model with RTC. The local analysis in SymTA/S is based on the classical busy period technique in real-time scheduling theory [11]. We refer to [17] for comparisons among these frameworks.

State-based verification techniques such as Timed Automata [3] provide extremely powerful expressiveness to model complex real-time systems. However, this approach usually suffers from the state-space explosion problem. The analytical (stateless) method of RTC depends on solutions of closed-form expressions, which in general yields a much better scalability than the state-based approach. Hybrid methods combining RTC and timed automata [10] are used to balance the expressiveness and scalability in the design of complex systems. However, as pointed out in this paper, although RTC is significantly more scalable than state-based verification techniques, it still may run into serious efficiency problems due to the *period explosion* phenomenon.

II. RTC Basics

A. Arrival and Service Curves

RTC uses *variability characterization curves* (*curves* for short) to describe timing properties of event streams and available resource:

Definition 1 (Arrival Curve)**.** *Let $R[s,t)$ denote the total amount of requested capacity to process in time interval $[s,t)$. Then, the corresponding upper and lower arrival curves are denoted as α^u and α^l, respectively, and satisfy:*

$$\forall s < t, \quad \alpha^l(t-s) \le R[s,t) \le \alpha^u(t-s) \tag{1}$$

where $\alpha^u(0) = \alpha^l(0) = 0$.

Definition 2 (Service Curve)**.** *Let $C[s,t)$ denote the number of events that a resource can process in time interval $[s,t)$. Then, the corresponding upper and lower service curves are denoted as β^u and β^l, respectively, and satisfy:*

$$\forall s < t, \quad \beta^l(t-s) \le C[s,t) \le \beta^u(t-s) \tag{2}$$

where $\beta^u(0) = \beta^l(0) = 0$.

The arrival and service curves are monotonically non-decreasing. Further, we only consider curves that are piece-wise linear and the length of each linear segment is lower-bounded by a constant. Therefore, we exclude curves that are "infinitely complex". The number of linear segments contained by a curve in an interval is polynomial with respect to the interval length. To simplify the presentation, we sometimes use a curve pair α (β) to represent both the upper curve α^u (β^u) and the lower curve α^l (β^l).

The RTC framework intensively uses the min-plus/max-plus convolution/deconvolution operations:

Definition 3 (Convolution and Deconvolution)**.** *Let f, g be two curves, the min-plus convolution $\otimes$, max-plus convolution $\overline{\otimes}$, min-plus deconvolution $\oslash$ and max-plus deconvolution $\overline{\oslash}$ are defined as:*

$$(f \otimes g)(\Delta) \triangleq \inf_{0\le\lambda\le\Delta}\{f(\Delta-\lambda)+g(\lambda)\}$$
$$(f \overline{\otimes} g)(\Delta) \triangleq \sup_{0\le\lambda\le\Delta}\{f(\Delta-\lambda)+g(\lambda)\}$$
$$(f \oslash g)(\Delta) \triangleq \sup_{\lambda\ge0}\{f(\Delta+\lambda)-g(\lambda)\}$$
$$(f \overline{\oslash} g)(\Delta) \triangleq \inf_{\lambda\ge0}\{f(\Delta+\lambda)-g(\lambda)\}$$

In this paper, we assume an upper (lower) bound curve to be *sub-additive* (*super-additive*) [21]:

Definition 4 (Sub-Additivity and Super-Additivity)**.** *A curve f is sub-additive iff*

$$\forall x,y \ge 0 : f(x) + f(y) \ge f(x+y)$$

A curve f is super-additive iff

$$\forall x,y \ge 0 : f(x) + f(y) \le f(x+y)$$

Moreover, we assume the arrival curve $\alpha = (\alpha^u, \alpha^l)$ and service curves $\beta = (\beta^u, \beta^l)$ to be *causal* [13]:

Definition 5 (Causality)**.** *Given a sub-additive upper bound curve f^u and a super-additive lower bound curve f^l, curve pair $f = (f^u, f^l)$ is* causal *iff*

$$f^u = f^u \overline{\oslash} f^l \quad \textit{and} \quad f^l = f^l \oslash f^u$$

The arrival and service curves are *well-defined* if they comply with the monotonicity, sub/super-additivity and causality constraints. Arrival and service curves that are not well-defined contain inconsistent information in modeling realistic system timing behaviors, and can be transferred into their monotonicity, sub/super-additivity and causality closures [21], [13]. The computations in RTC generate well-defined arrival and service curves if the inputs are well-defined. We assume that all arrival and service curves are well-defined, which is necessary to establish the Finitary RTC approach in this paper. Finally, we define the *long-term slope* of a curve f as

$$s(f) \triangleq \lim_{\Delta\to+\infty}(f(\Delta)/\Delta)$$

B. Greedy Processing Component (GPC)

In this paper, we focus on the most widely used abstract component in RTC called *Greedy Processing Component* (GPC) [5]. A GPC processes events from the input event stream (described by arrival curve α) in a greedy fashion, as long as it complies with the availability of resources (described by the service curve β). GPC produces an output event stream, described by arrival curve $\alpha' = (\alpha^{u'}, \alpha^{l'})$, and an output remaining service, described by service curve $\beta' = (\beta^{u'}, \beta^{l'})$:

$$\alpha^{u'} \triangleq [(\alpha^u \otimes \beta^u) \oslash \beta^l] \wedge \beta^u \tag{3}$$
$$\alpha^{l'} \triangleq [(\alpha^l \oslash \beta^u) \otimes \beta^l] \wedge \beta^l \tag{4}$$
$$\beta^{u'} \triangleq (\beta^u - \alpha^l)\ \overline{\oslash}\ 0 \tag{5}$$
$$\beta^{l'} \triangleq (\beta^l - \alpha^u)\ \overline{\otimes}\ 0 \tag{6}$$

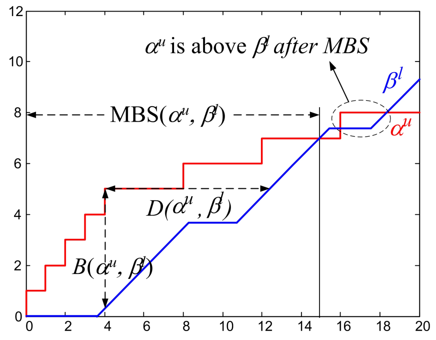

Figure 1. Illustration of $B(\alpha^u, \beta^l)$, $D(\alpha^u, \beta^l)$ and $\mathsf{MBS}(\alpha^u, \beta^l)$.

where $(f \wedge g)(\Delta) = \min(f(\Delta), g(\Delta))$. We use the following form to compactly represent the computation in (3) ~ (6):

$$(\alpha^{'}, \beta^{'}) = \mathsf{GPC}(\alpha, \beta)$$

The number of events in the input queue, i.e., the *backlog*, and the *delay* of each event can be bounded by $B(\alpha^u, \beta^l)$ and $D(\alpha^u, \beta^l)$ respectively:

$$B(\alpha^u, \beta^l) \triangleq \sup_{\lambda \geq 0} \left\{ \alpha^u(\lambda) - \beta^l(\lambda) \right\} \quad (7)$$

$$D(\alpha^u, \beta^l) \triangleq \sup_{\lambda \geq 0} \left\{ \inf\{\tau \in [0, \lambda] : \alpha^u(\lambda - \tau) \leq \beta^l(\lambda)\} \right\} \quad (8)$$

Intuitively, $B(\alpha^u, \beta^l)$ and $D(\alpha^u, \beta^l)$ are the maximal vertical and horizontal distance from α^u to β^l, as illustrated in Figure 1. In this paper, we intensively use another notation *maximal busy-period size*:

$$\mathsf{MBS}(\alpha^u, \beta^l) = \min\{\lambda > 0 : \alpha^u(\lambda) = \beta^l(\lambda)\}$$

Intuitively, $\mathsf{MBS}(\alpha^u, \beta^l)$ is the maximal length of the time interval in which α^u is above β^l, i.e., the maximal size of the so-called *busy periods* [1] (formally defined in the appendix). Note that in general α^u may goes above β^l again after they intersect at $\mathsf{MBS}(\alpha^u, \beta^l)$, as shown in Figure 1. We use MBS as the abbreviation of $\mathsf{MBS}(\alpha^u, \beta^l)$ when α^u and β^l are clear from the context.

In this paper we assume for each component $s(\alpha^u)/s(\beta^l)$ is bounded by a constant ε that is strictly smaller than 1, and thus $\mathsf{MBS}(\alpha^u, \beta^l)$ is bounded by a number that is *pseudo-polynomially* large. This is essentially the same as the common constraint in real-time scheduling theory that the system *utilization* is strictly smaller than 1 [2].

C. Performance Analysis Network

The RTC framework connects multiple components into a network to model systems with resource sharing and networked structures, as illustrated in Figure 2. As in most literature on RTC we assume the performance analysis networks are *acyclic*. Therefore, one can conduct the analysis of the whole network following the event and resource flows: it starts with a number of *initial input curves*, to generate *intermediate curves* step by step, and eventually generate the *final output curves*. The components whose inputs are all initial input curves are called *start components*, and the ones whose outputs are all final output curves are called *end components*.

For example, in Figure 2 the analysis starts with the start component 1, using the initial input curves α_1, β_1 to derive its backlog bound $B(\alpha_1^u, \beta_1^l)$ and delay bound $D(\alpha_1^u, \beta_1^l)$, and generate the output curves $\alpha_1^{'}, \beta_1^{'}$. Then $\alpha_2, \beta_1^{'}$ are used to analyze component 2, and $\alpha_2, \beta_1^{'}$ are used to analyze component 3, and finally the resulting curves $\alpha_2^{'}, \beta_2^{'}$ from these two components are used to analyze the end component 4.

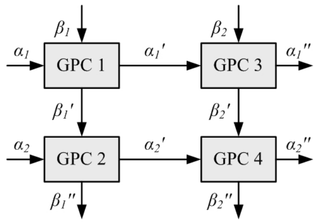

Figure 2. An example of analysis network in RTC.

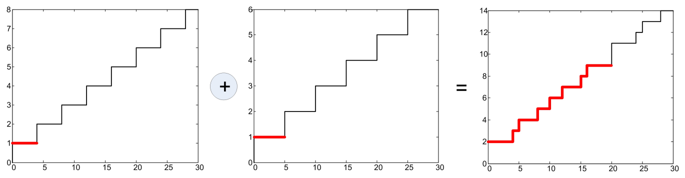

Figure 3. Illustration of the segment number increase caused by a plus operation, where the red part of each curve are the segments that need to be stored to represent the whole curve.

Besides backlog bound $B(\alpha^u, \beta^l)$ and delay bound $D(\alpha^u, \beta^l)$, another important performance metric is the *end-to-end delay* of an event stream, denoted by D_{end}. This can be obtained by summing up the individual delay bound of each component along the event stream flow. However, this may lead to pessimistic end-to-end delay bound estimation. One can "group" the components along the event flow to get a more precise end-to-end delay bound (the so-called "pay-burst-only-once") [4]:

$$D_{end} = D(\alpha^u, \ \beta_1^l \otimes \beta_2^l \otimes \cdots \otimes \beta_n^l) \quad (9)$$

where $\beta_1^l, \beta_2^l, \cdots, \beta_n^l$ are the lower service curves of the components along the event stream flow with initial input α^u.

III. Efficiency Bottleneck of RTC

In RTC, the arrival/service curves are defined for the infinite range of positive real numbers $\Delta \in \mathbb{R}_{\geq 0}$. For a practical implementation, we need a finite representation of curves and the curve operations should be completed in a finite time.

To solve this problem, RTC Toolbox [22] restricts to a class of *regular* curves [21], which can be efficiently represented by *finite* data structures but are still expressive enough to model most realistic problems. A regular curve consists of an aperiodic part, followed by a periodic part. Each part is represented by a concatenation of linear *segments*. Generally, the computation time and memory requirement of an operation between two curves is proportional to the number of segments contained by the curves.

Many RTC operations (e.g., plus, minus and convolution) generate output curves with much longer periods than the periods of the input curves. Typically, the period of the output curve equals to the LCM (least common multiple) of the input periods. Figure 3 shows an example of the plus operation of two curves. Both input curves are strictly periodic (i.e., the aperidoc parts are empty). The first input curve's period is 4 and the second input curve's period is 5. Each of them only contains one segment. Applying the plus operation to these two curves, the result is a periodic curve with period 20, which is the LCM of 4 and 5, and containing 8 segments.

In general, when many components are serially connected in a row, the number of segments contained by the curves increases *exponentially*, and thus the time cost of the analysis increases *exponentially* as it steps forward along the analysis flow. We call this phenomenon *period explosion*, which is the major reason why the analysis of large-scale systems in RTC is problematic.

IV. Overview of Finitary RTC

We start with a simple example of only one component. As introduced in Section II-B, the backlog bound $B(\alpha^u, \beta^l)$ and delay bound $D(\alpha^u, \beta^l)$ of this system is calculated by measuring the maximal horizontal and vertical distance between α^u and β^l (in the part where α^u is above β^l).

Actually, to calculate $B(\alpha^u, \beta^l)$ and $D(\alpha^u, \beta^l)$, one only needs to check both curves up to $\mathsf{MBS}(\alpha^u, \beta^l)$ on the x-axis:

Theorem 1. *Given a sub-additive upper arrival curve α^u and a super-additive service curve β^l,*

$$B(\alpha^u, \beta^l) = \sup_{\mathsf{MBS} \geq \lambda \geq 0} \left\{ \alpha^u(\lambda) - \beta^l(\lambda) \right\}$$
$$D(\alpha^u, \beta^l) = \sup_{\mathsf{MBS} \geq \lambda \geq 0} \left\{ \inf\{\tau \in [0, \lambda] : \alpha^u(\lambda - \tau) \leq \beta^l(\lambda)\} \right\}$$

Theorem 1 is proved in the appendix.

Note that α^u may be above β^l again after MBS, as shown in Figure 1. Nevertheless, Theorem 1 guarantees that the maximal backlog and delay occurs in time intervals of size up to $\mathsf{MBS}(\alpha^u, \beta^l)$. Therefore, the analysis can "chop off" the parts beyond $\mathsf{MBS}(\alpha^u, \beta^l)$ of both curves, and only use the remaining *finitary curves* to obtain exactly the same $B(\alpha^u, \beta^l)$ and $D(\alpha^u, \beta^l)$ results as before. This represents the basic idea of Finitary RTC:

Main Idea: *Instead of working with complete curves, we only work with the part of each curve that is relevant to the analysis results.*

This is similar to the standard analysis techniques based on *busy periods* in classical real-time scheduling theory. For example, in the schedulability analysis of EDF based on *demand bound functions* [1], one only needs to check what happens in a busy period. The analysis of time intervals beyond the maximal busy period size is irrelevant to the system schedulability and thereby can be ignored.

However, it is difficult to apply this idea to the analysis of networked systems in the RTC framework. For example, suppose we want to analyze the backlog and delay bound of component 4 in Figure 2. By the discussions above, we only need to keep the input curves of component 4 up to $\mathsf{MBS}(\alpha_2^{u'}, \beta_2^{l'})$ on the x-axis to obtain the desired results. However, the value of $\mathsf{MBS}(\alpha_2^{u'}, \beta_2^{l'})$ is not revealed until we have actually obtained $\alpha_2^{u'}$ and $\beta_2^{l'}$. To calculate $\alpha_2^{u'}$ and $\beta_2^{l'}$, we need to first accomplish the analysis of component 2 and 3, and component 1 at the first place. Therefore, we still have to conduct the expensive analysis for the preceding components of component 4 with complete curves, although we only need a small part of the input curves of component 4 to calculate its backlog and delay bound. In this way, the efficiency improvement is trivial since the analysis procedure is as expensive as in the original RTC except for the very last step to analyze component 4.

The target of Finitary RTC is to work with finitary curves (the parts of curves up to certain limits) through the whole analysis network. In other words, we should already use finitary curves at the initial inputs, and generate finitary curves at outputs from the input finitary curves at each component. By this, the overall analysis procedure is significantly more efficient than the original RTC approach. To achieve this, we first need to solve the following problem:

Problem 1: *How to compute the output finitary curves from the input finitary curves for each component?*

Recall that the computation in $(\alpha^{'}, \beta^{'}) = \mathsf{GPC}(\alpha, \beta)$ uses the min-plus and max-plus deconvolution operations:

$$(f \oslash g)(\Delta) = \sup_{\lambda \geq 0}\{f(\Delta + \lambda) - g(\lambda)\}$$
$$(f \overline{\oslash} g)(\Delta) = \inf_{\lambda \geq 0}\{f(\Delta + \lambda) - g(\lambda)\}$$

In order to calculate $(f \oslash g)(\Delta)$ or $(f \overline{\oslash} g)(\Delta)$ for a particular Δ, it is required to check the value of $f(\Delta + \lambda) - g(\lambda)$ for *all* $\lambda \geq 0$, i.e., slide over curve f from Δ to above and slide over the whole curve g, to get the suprema (infima). Therefore, even if we only want to calculate a small piece of a curve at the output, we still need to know the *complete* input curves defined in the infinite range. Section V is dedicated to the solution of this problem, where we use a "finitary" version of the deconvolution operations in the computation of the output arrival and service curves. We prove that to calculate the output curve up to interval size x, it is enough to only visit the part of input curves up to interval length $x + \mathsf{MBS}$.

If we somehow know the MBS value for each component, then we can "backtrack" the whole analysis network to decide the size of the input finitary curves for each component, and eventually decide the size of the curves we need to keep at the initial inputs. However, the MBS value for each component is not revealed until its input arrival and service curves are actually known, and we do not want to actually perform the expensive analysis with complete curves to obtain the MBS information. Therefore, we need to solve the following problem:

Problem 2: *How to efficiently estimate the* MBS *value of each component in the network?*

We address this problem in Section VI, using safe approximations of the input curves to quickly "pre-analyze" the whole analysis network and obtain safe estimation of the MBS value for each component. The key point is that as long as we always use *over-approximations* (formally defined in Section VI) of the curves, the obtained MBS estimation is guaranteed to be no smaller than its real value. Note that the *over-approximations* of input curves are used merely for the purpose of estimating MBS. After we have obtained the

MBS estimation of each component, the following analysis procedure does not introduce any extra pessimism, and the analysis results we finally obtained are as precise as using the original RTC.

V. Analyzing GPC with Finitary Deconvolution

This section addresses the first problem, i.e., how to compute the output finitary curves from the input finitary curves for each component. We first define the finitary version of the min-plus and max-plus deconvolutions operations:

Definition 6 (Finitary Deconvolution)**.** *The* finitary min-plus deconvolution *and* finitary max-plus deconvolution *regarding a non-negative real number T, denoted by $\oslash_T$ and $\overline{\oslash}_T$ respectively, are defined as:*

$$(f \oslash_T g)(\Delta) \triangleq \sup_{T \geq \lambda \geq 0} \{f(\Delta + \lambda) - g(\lambda)\}$$
$$(f \overline{\oslash}_T g)(\Delta) \triangleq \inf_{T \geq \lambda \geq 0} \{f(\Delta + \lambda) - g(\lambda)\}$$

The result of $(f \oslash_T g)(\Delta)$ and $(f \overline{\oslash}_T g)(\Delta)$ for a particular Δ only depends on f in $[\Delta, \Delta + T]$ and g in $[0, T]$.

Now we can refine the computation $(\alpha', \beta') = \mathsf{GPC}(\alpha, \beta)$, using $\oslash_T$ and $\overline{\oslash}_T$ to replace $\oslash$ and $\overline{\oslash}$, and still safely bound the output event stream and remaining service:

Theorem 2. *Given an event stream described by the arrival curves α^u, α^l and a resource described by the service curves β^u, β^l. If T is a real number with*

$$T \geq \mathsf{MBS}(\alpha^u, \beta^l)$$

then the processed event stream and remaining service are bounded from above and below by $(\alpha', \beta') = \mathsf{GPC}_T(\alpha, \beta)$:

$$\alpha^{u'} \triangleq ((\alpha^u \otimes \beta^u) \oslash_T \beta^l \wedge \beta^u) \quad (10)$$
$$\alpha^{l'} \triangleq ((\alpha^l \oslash_T \beta^u) \otimes \beta^l \wedge \beta^l) \quad (11)$$
$$\beta^{u'} \triangleq (\beta^u - \alpha^l) \overline{\oslash}_T 0 \quad (12)$$
$$\beta^{l'} \triangleq (\beta^l - \alpha^u) \overline{\otimes} 0 \quad (13)$$

Moreover, GPC_T does *not* sacrifice any analysis precision comparing with GPC that uses the original deconvolution operations:

Theorem 3. *The output event arrival curves and remaining service curves obtained by GPC_T in Theorem 2 is at least as precise as that obtained by GPC.*

The proofs of Theorem 2 and 3 are presented in the appendix. Note that these proofs are *not* simple reproductions of their counterparts in the original RTC. Particularly, sophisticated techniques are developed to utilize the busy period concept in the construction of the desired bounds in Theorem 2, and Theorem 3 relies on the sub/super-additivity and causality property of the input arrival and service curves.

The original deconvolution in GPC is replaced by the finitary deconvolution operations in GPC_T, so the computation of a particular point on the output curve only requires to slide over a limited range of the input curves. Therefore, we can establish the information dependency between the input curves and output curves in GPC_T:

Theorem 4. *Let $(\alpha', \beta') = \mathsf{GPC}_T(\alpha, \beta)$. The computation of α' or β' in the range of $[0, x]$ on the x-axis only depends on the input curves in the range of $[0, x + T]$ on the x-axis.*

Theorem 4 can be easily proved by rewriting (10) ~ (13) with the definition of the convolution and finitary deconvolution operations. Proof details are omitted due to space limit.

We use $|f|$ to denote the *upper limit* on the x-axis to which we want to keep the curve (pair) f. Then according to Theorem 4, we know the following constraint between the upper limits of the input and output curves in GPC_T:

$$|\alpha| = |\beta| \geq T + \max(|\alpha'|, |\beta'|) \quad (14)$$

For example, if we want to use GPC_6 to generate output curves with upper limit of 5, then the upper limit of the input curves should be at least $5 + 6 = 11$, i.e., the input curves should be defined at least in the range $[0, 11]$.

VI. Analysis Network in Finitary RTC

The GPC networks considered in this paper are *acyclic*. Therefore, as soon as we have chosen the T value in GPC_T for each component, we can traverse the network backwards and use the relation in (14) to iteratively decide the upper limits of the input finitary curves of each component. The choice of T for each component is subject to the constraint $T \geq \mathsf{MBS}$. Therefore, it is sufficient to derive an upper bound of MBS, and use this upper bound as the value of T in GPC_T at each component. In the following, we first introduce how to efficiently compute a safe upper bound of MBS_i for each component i, then introduce how to decide the upper limits of input finitary curves of each component.

A. Bounding Individual MBS

We first define the over-approximation of curves:

Definition 7 (Over-approximation)**.** *For two upper bound curves c^{u*} and c^u, if $\forall \Delta \geq 0 : c^{u*}(\Delta) \geq c^u(\Delta)$, then c^{u*} is an over-approximation of c^u. For two lower bound curves c^{l*} and c^l, if $\forall \Delta \geq 0 : c^{l*}(\Delta) \leq c^l(\Delta)$, then c^{l*} is an over-approximation of c^l. Curve pair $c^* = (c^{u*}, c^{l*})$ is an over-approximation of curve pair $c = (c^u, c^l)$ if c^{u*} and c^{l*} are over-approximations of c^u and c^l respectively. We use $a \succeq b$ to denote that a is an over-approximation of b.*

To compute a safe upper bound of MBS_i for each component, we use over-approximations of the initial input curves to "pre-analyze" the whole network. First we have the following property by examining the computation rules of GPC:

Property 1. *If $\alpha_1 \succeq \alpha_2$ and $\beta_1 \succeq \beta_2$, then $\alpha_1' \succeq \alpha_2'$ and $\beta_1' \succeq \beta_2'$, where $(\alpha_1', \beta_1') = \mathsf{GPC}(\alpha_1, \beta_1)$ and $(\alpha_2', \beta_2') = \mathsf{GPC}(\alpha_2, \beta_2)$.*

So we know that if we start with over-approximations of the initial input curves, then during the whole "pre-analysis" procedure all the resulting curves are over-approximations of their correspondences. Further, we know

Property 2. *If $\alpha_1^u \succeq \alpha_2^u$ and $\beta_1^l \succeq \beta_2^l$, then $\mathsf{MBS}(\alpha_1^u, \beta_1^l) \geq \mathsf{MBS}(\alpha_2^u, \beta_2^l)$.*

Therefore, if we use over-approximations of the initial input curves to conduct the analysis of the system, then the resulted maximal busy-period size of each component i, denoted by MBS_i^*, is an safe upper bound of the real MBS_i of that component with the original curves.

There are infinitely many possibilities to over-approximate the initial input curves to conduct the pre-analysis procedure. In this paper we choose to use the "tightest" linear functions as their over-approximations: each initial input curve pair $f = (f^u, f^l)$ is over-approximated by $\overline{f} = (\overline{f^u}, \overline{f^l})$ where $\overline{f^u}(\Delta) = a^u \times \Delta + b^u$ and $\overline{f^l}(\Delta) = a^l \times \Delta + b^l$ with

$$a^u = s(f^u) \quad b^u = \inf\{\ b \mid \forall\Delta : a^u \times \Delta + b \ge f^u(\Delta)\}$$
$$a^l = s(f^l) \quad b^l = \sup\{\ b \mid \forall\Delta : a^l \times \Delta + b \le f^l(\Delta)\}$$

Using these linear functions as the initial inputs, the generated curves in the pre-analysis procedure only contain polynomially many segments and the computations are very efficient.

B. Decision of Curve Upper Limits

The decision of the upper limit of each finitary curve starts with the end components. Since the output curves of an end component are not used by other components, it is enough to set the upper limit of the input finitary curves of each end component i to be MBS_i^*. Then we can traverse the whole analysis network backwards, using (14) to iteratively compute the upper limits of the finitary input curves of each component. The pseudo-code of this procedure is shown in Algorithm 1.

```
1: Mark all component as *unfinished*.
2: **for** each end component $i$ **do**
3:     $|\alpha_i^{in}| = |\beta_i^{in}| \leftarrow \mathsf{MBS}_i^*$
4:     Mark component $i$ as *finished*
5: **end for**
6: **while** exist *unfinished* components **do**
7:     Select an *unfinished* component $j$ whose successors are
       both *finished* (i.e., $|\alpha_j^{out}|$ and $|\beta_j^{out}|$ are known).
8:     $|\alpha_j^{in}| = |\beta_j^{in}| \leftarrow \mathsf{MBS}_j^* + \max(|\alpha_j^{out}|, |\beta_j^{out}|)$
9:     Mark component $j$ as *finished*
10: **end while**
```

Algorithm 1: Pseudo-code of algorithm computing the upper limit of each finitary curve.

Example 1. *We use linear functions to over-approximations the initial inputs $\alpha_1, \alpha_2, \beta_1, \beta_2$ to pre-analyze the system in Figure 2. Suppose the resulting estimated maximal busy-period sizes are $\mathsf{MBS}_1^* = 10$, $\mathsf{MBS}_2^* = 12$, $\mathsf{MBS}_3^* = 14$ and $\mathsf{MBS}_4^* = 20$. Then we iteratively compute the size of the finitary curves:*

$$|\alpha_2^{'}| = |\beta_2^{'}| = \mathsf{MBS}_4^* = 20$$
$$|\alpha_1^{'}| = |\beta_2| = \mathsf{MBS}_3^* + \max(|\beta_2^{'}|, 0) = 14 + 20 = 34$$
$$|\alpha_2| = |\beta_1^{'}| = \mathsf{MBS}_2^* + \max(|\alpha_2^{'}|, 0) = 12 + 20 = 32$$
$$|\alpha_1| = |\beta_1| = \mathsf{MBS}_1^* + \max(|\alpha_1^{'}|, |\beta_1^{'}|) = 10 + 34 = 44$$

To use (9) to analyze the end-to-end delay D_{end} of an event stream, we can use the over-approximations of the initial input arrival curve and the related service curves to decide the upper limits to which these curves are required to be kept:

$$|\alpha| = |\beta_1| = \cdots = |\beta_n| = \mathsf{MBS}(\alpha^{u*}, \beta_1^{l*} \otimes \beta_2^{l*} \otimes \cdots \otimes \beta_n^{l*})$$

where $\alpha^{u*}, \beta_1^{l*}, \cdots, \beta_n^{l*}$ are the over-approximations of the initial input upper arrival curve α^u and the related lower service curves $\beta_1^l, \cdots, \beta_n^l$.

C. Complexity

In summary, the analysis of a network by Finitary RTC consists of three steps: (1) Using linear over-approximations of initial input curves to analyze the network "forwards" and get an estimated MBS^* for each component. (2) Using the estimated MBS^* values to traverse the analysis network "backwards" to decide the upper limits of the input finitary curves of each component. (3) Conduct the analysis of the network "forwards" with the finitary curves.

Finding the "tightest" linear over-approximation of a curve is of pseudo-polynomial complexity (polynomial with respect to the number of segments contained by the curve). With the linear over-approximations of the initial input curves, the maximal number of segments of each curve generated in the pre-analysis is polynomial with respect to network size. So the overall complexity of step (1) is pseudo-polynomial. It is easy to see that the complexity of step (2) is polynomial. Since for each component $s(\alpha^u)/s(\beta^l)$ is bounded by a constant ε strictly smaller than 1, the estimated MBS^* with the linear curve over-approximations is bounded by a number that is pseudo-polynomially large. Therefore, each finitary curve obtained by Algorithm 1 has a pseudo-polynomial upper limit. Since the complexity of GPC_T is polynomial with respect to the curve upper limits and the value of T (i.e., the estimated MBS^*), the complexity of the analysis for each component is pseudo-polynomial, and the overall complexity of step (3) is also pseudo-polynomial. In summary, the overall complexity of the whole analysis procedure is pseudo-polynomial.

VII. Evaluation

In this section we use synthetic systems to evaluate the efficiency improvement of the Finitary RTC approach. We use the metric *speedup ratio* to represent the efficiency improvement of Finitary RTC over the original RTC:

$$\text{speedup ratio} = \frac{\text{time cost of analysis by original RTC}}{\text{time cost of analysis by Finitary RTC}}$$

In Section VII-A we present a case study to give a general picture of the efficiency improvement by Finitary RTC. Then in Section VII-B we adjust the parameters of this system to discuss different factors that affect the speedup ratio.

We implement Finitary RTC in RTC Toolbox [22]. RTC Toolbox consists of two major software components: a Java-implemented kernel of basic RTC operations and a set of Matlab libraries to provide high-level modeling capability. Since the Java kernel is not open-source, we implement Finitary RTC only using the open-source Matlab libraries: For each finitary curve f, we use a linear curve to replace the part beyond its upper limit $|f|$ instead of actually removing this part. Therefore, we can reuse the Java kernel of RTC Toolbox to implement the idea of Finitary RTC. All experiments are conducted on a desktop computer with a 3.4GHZ Intel Core i7 processor.

A. Case Study

We consider a fairly small and simple system of a 4×3 2D-mesh analysis network, as shown in Figure 4. All initial input

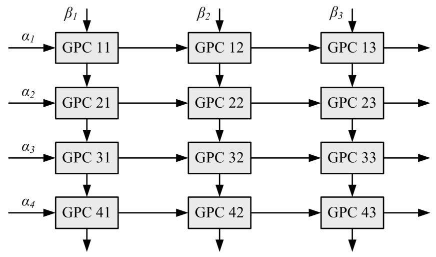

Figure 4. Analysis network of the case study.

	α_1	α_2	α_3	α_4
p	10	14	18	22
j	2	3	5	6
d	4	6	8	4

	β_1	β_2	β_3
s	4	6	8
c	6	8	10
b	1	1	1

Table I
PARAMETERS OF THE INITIAL INPUT ARRIVAL AND SERVICE CURVES

event curves are specified by the parameter triple (p, j, d), where p denotes the period, j the jitter and d the minimum inter-arrival distance of events [16]. The arrival curves of a (p, j, d)-specified stream are

$$\alpha^u(\Delta) = \min\left(\left\lceil \frac{\Delta + j}{p} \right\rceil, \left\lceil \frac{\Delta}{d} \right\rceil\right), \quad \alpha^l(\Delta) = \left\lfloor \frac{\Delta - j}{p} \right\rfloor$$

All initial input service curves correspond to TDMA resource. Each resource is specified by a triple (s, c, b), where a slot of length s is allocated with very TDMA cycle c, on a resource with total bandwidth b [22]. The service curves of a (s, c, b)-specified resource are

$$\beta^u(\Delta) = \left(\left\lfloor \frac{\Delta}{c} \right\rfloor \cdot s + \min(\Delta \bmod c, s)\right) \cdot b$$
$$\beta^l(\Delta) = \left(\left\lfloor \frac{\Delta'}{c} \right\rfloor \cdot s + \min(\Delta' \bmod c, s)\right) \cdot b$$

where $\Delta' = \max(\Delta - c + s, 0)$. The parameters of the input arrival and service curves are shown in Table I.

Using the original RTC approach, the analysis of the whole network completes in 415 seconds, while using the Finitary RTC approach proposed in this paper, the analysis only takes 0.14 seconds. The Finitary RTC approach leads to a speedup ratio of about 3000 in this case study.

If we modify the system parameters by changing the period p of the third and fourth event stream to 19 and 23 respectively, which leads to a larger hyper-period of the four event streams, then the original RTC approach cannot accomplish the analysis since an overflow error occurs in the Java kernel of the RTC-Toolbox, while the analysis by Finitary RTC still completes within 0.14 seconds.

B. Factors that Affect the Speedup Ratio

The speedup ratio between the Finitary RTC and original RTC heavily depends on the system parameters. In this section we adjust the parameters of the example system in the above case study to discuss factors that affect the speedup ratio.

As introduced in Section III, in the original RTC the period of the output curves typically equals the LCM of the periods of the input curves. For example, a system with initial input curves having co-prime periods may lead to very large curve periods. To evaluate the analysis efficiency of the original RTC with different parameter complexity degree, we adjust the period parameter p of each initial input arrival curve to get systems with different hyper-periods. We randomly generate period p of each curve (within a certain scope), to construct systems with different hyper-periods of input event streams, then calculate the average analysis time cost in the original RTC and Finitary RTC of systems with hyper-periods in a certain scope, as shown in Table II. For example the column starting with "1 ~ 2" in Table II reports the average analysis time cost in the original RTC and Finitary RTC of systems with hyper-period between 1000 and 2000. From Table II we can see that the analysis time cost in the original RTC grows rapidly as the hyper-period increases, while Finitary RTC is not sensitive to the hyper-period changes and keeps very low time cost all the time. The speedup ratio is more significant for systems with more complex timing characteristics.

The analysis efficiency of Finitary RTC depends on the curve upper limits. In general, the closer is the long-term ratio of α^u to β^l, the bigger is $\mathsf{MBS}(\alpha^u, \beta^l)$ and thus the bigger is the upper limits of the finitary curves. We define the maximal *utilization* among all the components in the analysis network:

$$U_{max} = \max_{\text{each component } i} \left\{ \frac{s(\alpha_i^u)}{s(\beta_i^l)} \right\}$$

where α_i^u and β_i^l are the input upper arrival and lower service curve of component i. We adjust the bandwidth b for all of the three (s, c, b)-specified TDMA resource in Table I to construct systems with different U_{max}. We are only interested in systems with $U_{max} \leq 1$, since otherwise some components in the system deem to have unbounded backlog and delay, and the system is considered to be a failure immediately. From Table III we can see that the time cost of the Finitary RTC approach increases as we increase U_{max}. Nevertheless, Finitary RTC can speed up the analysis by 1000 times with systems of relatively high U_{max} (up to 0.8). Even for systems with U_{max} that are pretty close to 1 (e.g., $U_{max} = 0.96$), Finitary RTC still can significantly improve the analysis efficiency (speed up the analysis by 20 times).

The above experiments show that Finitary RTC drastically speeds up the analysis under different parameter configurations. The speedup is more significantly for systems with more complex timing characteristics and lighter workloads.

VIII. CONCLUSIONS AND FUTURE WORK

In this paper we propose Finitary RTC to drastically improve the analysis efficiency of the RTC framework. The cental idea is to use a (small) piece of each infinite curve in the analysis procedure, which avoids the "period explosion" problem in the original RTC and results in a pseudo-polynomial complexity (the original RTC has exponential complexity). In practise, Finitary RTC can drastically improve the analysis efficiency, especially for systems with complex timing characteristics. Finitary RTC does not introduce any extra pessimism, i.e., its analysis results are as precise as using the original RTC.

Although we present Finitary RTC in the context of a particular type of component GPC, the same idea can be also extended to components modeling different scheduling policies like FIFO and EDF. In this paper we assume the analysis networks to be *acyclic*. It is not clear yet how to

hyper-period ($\times 10^3$)	1 ~ 2	2 ~ 3	3 ~ 4	4 ~ 5	5 ~ 6	6 ~ 7	7 ~ 8	8 ~ 9	9 ~ 10	10 ~ 11	11 ~ 12	12 ~ 13
original RTC (second)	2.15	14.51	108.94	195.77	362.95	487.65	643.097	797.32	841.75	1043.68	1342.16	1696.53
Finitary RTC (second)	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14
speedup ratio	15	112	778	1398	2592	3483	4592	5695	6012	7454	9586	12118

Table II
EXPERIMENT RESULTS OF SYSTEMS WITH DIFFERENT HYPER-PERIODS.

b (bandwidth)	1	0.9	0.8	0.7	0.6	0.5	0.45	0.425
U_{max}	0.41	0.45	0.51	0.58	0.68	0.82	0.91	0.96
original RTC (second)	415.58	487.24	522.38	419.58	388.64	271.01	322.17	305.77
Finitary RTC (second)	0.14	0.14	0.15	0.19	0.20	0.40	14.25	12.74
speedup ratio	2968	3478	3480	2747	2092	970	19	24

Table III
EXPERIMENT RESULTS OF SYSTEMS WITH DIFFERENT MAXIMAL UTILIZATIONS

generalize Finitary RTC to the analysis of systems with cyclic event and resource dependencies. A straightforward extension of Finitary RTC to cyclic systems is to bound the number of iterations after which the analysis is guaranteed to converge, and extend the curve upper limits following the analysis iterations, which may lead to very large curve upper limits and less significant efficiency improvement. In the future we will study efficient application of Finitary RTC to cyclic systems.

REFERENCES

[1] S. K. Baruah, A. Mok, and L. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the 11th Real-Time Systems Symposium (RTSS)*, 1990.

[2] Sanjoy K. Baruah, Deji Chen, Sergey Gorinsky, and Aloysius K. Mok:. Generalized multiframe tasks. *Real-Time Systems*, 1999.

[3] Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In *Lectures on Concurrency and Petri Nets*, 2003.

[4] J. L. Boudec and P. Thiran. Network calculus - a theory of deterministic queuing systems for the internet. In *LNCS 2050*. Springer Verlag, 2001.

[5] Samarjit Chakraborty, Simon Künzli, and Lothar Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *DATE*, 2003.

[6] Lothar Thiele Ernesto Wandeler. Characterizing workload correlations in multi processor hard real-time systems. In *RTAS*, 2005.

[7] Arne Hamann, Marek Jersak, Kai Richter, and Rolf Ernst. Design space exploration and system optimization with symta/s-symbolic timing analysis for systems. In *RTSS*, 2004.

[8] Michael Gonzalez Harbour, J. J. Gutierrez Garcia, Jose C. Palencia Gutierrez, and J. M. Drake Moyano. Mast: Modeling and analysis suite for real time applications. In *ECRTS*, 2001.

[9] Bengt Jonsson, Simon Perathoner, Lothar Thiele, and Wang Yi. Cyclic dependencies in modular performance analysis. In *EMSOFT*, 2008.

[10] Kai Lampka, Simon Perathoner, and Lothar Thiele. Analytic real-time analysis and timed automata: a hybrid method for analyzing embedded real-time systems. In *EMSOFT*, 2009.

[11] J. P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *RTSS*, 1990.

[12] A. Maxiaguine, Yongxin Zhu, S. Chakraborty, and Weng-Fai Wong. Tuning soc platforms for multimedia processing: identifying limits and tradeoffs. In *CODES+ISSS*, 2004.

[13] Matthieu Moy and Karine Altisen. Arrival Curves for Real-Time Calculus: The Causality Problem and Its Solutions. In *TACAS*, 2010.

[14] Simon Perathoner, Tobias Rein, Lothar Thiele, Kai Lampka, and Jonas Rox. Modeling structured event streams in system level performance analysis. In *LCTES*, 2010.

[15] Linh T. X. Phan, Samarjit Chakraborty, and P. S. Thiagarajan. A multi-mode real-time calculus. In *RTSS*, 2008.

[16] K. Richter. Compositional scheduling analysis using standard event models. In *Ph.D. Thesis, Technical University Carolo-Wilhelmina of Braunschweig*, 2005.

[17] Lothar Thiele Arne Hamann Simon Schliecker Rafik Henia Razvan Racu Rolf Ernst Michael Gonzalez Harbour Simon Perathoner, Ernesto Wandeler. Influence of different abstractions on the performance analysis of distributed hard real-time systems. In *Design Autom. for Emb. Sys.*, 2009.

[18] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *Proc. Inti. Symposium on Circuits and Systems*, 2000.

[19] Lothar Thiele, Ernesto Wandeler, and Nikolay Stoimenov. Real-time interfaces for composing real-time systems. In *EMSOFT*, 2006.

[20] Ken Tindell and John Clark. Holistic schedulability analysis for distributed hard real-time systems. In *Microprocessing and Microprogramming*, 1994.

[21] Ernesto Wandeler. Modular performance analysis and interface-based design for embedded real-time systems. In *PhD thesis, ETHZ*, 2006.

[22] Ernesto Wandeler and Lothar Thiele. Real-Time Calculus (RTC) Toolbox. `http://www.mpa.ethz.ch/Rtctoolbox`.

APPENDIX A: PROOF OF THEOREM 1

We first prove that in order to compute $B(\alpha^u, \beta^l)$ it is sufficient to only check α^u and β^l up to $\mathsf{MBS}(\alpha^u, \beta^l)$. More precisely, we prove

$$\sup_{\lambda \ge 0} \left\{ \alpha^u(\lambda) - \beta^l(\lambda) \right\} = \sup_{\mathsf{MBS} \ge \lambda \ge 0} \left\{ \alpha^u(\lambda) - \beta^l(\lambda) \right\} \tag{15}$$

Let $m = \mathsf{MBS}(\alpha^u, \beta^l)$, so $\alpha^u(m) = \beta^l(m)$. For any non-negative λ, let $q = \lfloor \frac{\lambda}{m} \rfloor$ and $r = \lambda - q \cdot m$, and by the sub-additivity of α^u and super-additivity of β^l we have

$$\alpha^u(\lambda) = \alpha^u(q \cdot m + r) \le \alpha^u(q \cdot m) + \alpha^u(r) \le q \cdot \alpha^u(m) + \alpha^u(r)$$
$$\beta^l(\lambda) = \beta^l(q \cdot m + r) \ge \beta^l(q \cdot m) + \beta^l(r) \ge q \cdot \beta^l(m) + \beta^l(r)$$

by which we have

$$\begin{aligned} & \alpha^u(\lambda) - \beta^l(\lambda) \le q \cdot (\alpha^u(m) - \beta^l(m)) + \alpha^u(r) - \beta^l(r) \\ \Leftrightarrow \quad & \alpha^u(\lambda) - \beta^l(\lambda) \le \alpha^u(r) - \beta^l(r) \quad (\because \alpha^u(m) = \beta^l(m)) \end{aligned}$$

In other words, for any $\lambda \ge 0$, we can always find a corresponding r in the range of $[0, m]$ ($m = \mathsf{MBS}(\alpha^u, \beta^l)$) such that $\alpha^u(\lambda) - \beta^l(\lambda) \le \alpha^u(r) - \beta^l(r)$, which proves (15).

To prove the theorem for $D(\alpha^u, \beta^l)$, we can use the same reasoning as above to the “inverse functions” of $\alpha^u(\lambda)$ and $\beta^l(\lambda)$: $D(\alpha^u, \beta^l)$ is the horizontal distance between the “inverse functions” of $\alpha^u(\lambda)$ and $\beta^l(\lambda)$, and the “inverse function” of $\alpha^u(\lambda)$ is super-additive and the “inverse function” of $\beta^l(\lambda)$ is sub-additive. We omit the proof details for $D(\alpha^u, \beta^l)$ due to space limit.

APPENDIX B: PROOF OF THEOREM 2

We first introduce some useful concepts and notations [21]: $R[s,t)$ denotes the number of events arrived in time interval $[s,t)$, and $R'[s,t)$ denotes the number of *processed* events in $[s,t)$. $C[s,t)$ denotes the amount of available resource in $[s,t)$, and $C'[s,t)$ denotes the amount of remaining resource in $[s,t)$. Further, the following relation is known [21]:

$$R'[s,t) = C[s,t) - C'[s,t) \tag{16}$$

$B(t)$ denotes the backlog at time t. Moreover, we use p to denote an arbitrarily early time point with $B(p) = 0$. The RTC framework assumes there always exists such a time point p.

Definition 8 (Busy Period). *A time interval (s,t) is a* busy period *iff both of the following conditions hold: (i) $\forall x \in (s,t)$: $B(x) \neq 0$ and, (ii) $B(s) = B(t) = 0$. Moreover, we call s the* start point *of the busy period (s,t), and t its* end point.

Lemma 1. *Given an event stream restricted by upper arrival curve α^u and resource restricted by lower service curves β^l, the size of any busy period is bounded by $\mathsf{MBS}(\alpha^u, \beta^l)$.*

Proof: Follows the definitions of MBS and busy period. ■

Then we introduce another important auxiliary lemma:

Lemma 2. *Let x_2 be an arbitrary time point with $B(x_2) = 0$. Then $\forall x_1 : p \leq x_1 \leq x_2$:*

$$\sup_{x_1 \leq x \leq x_2} \{C[p,x) - R[p,x)\} = C'[p,x_2) \tag{17}$$

Proof:

Since $B(x_2)$, all the events arrived before x_2 have been processed before x_2, so we have

$$\begin{aligned} C'[p,x_2) &= C[p,x_2) - R[p,x_2) - B(p) \\ &= C[p,x_2) - R[p,x_2) \quad (\because B(p) = 0) \end{aligned}$$

So in the following we only need to prove

$$C[p,x_2) - R[p,x_2) = \sup_{x_1 \leq x \leq x_2} \{C[p,x) - R[p,x)\} \tag{18}$$

It is easy to see that the LHS of (18) is no larger than its RHS. In the following we only need to prove that it holds LHS ≥ RHS as well. We do this by contradiction, assuming

$$C[p,x_2) - R[p,x_2) < \sup_{x_1 \leq x \leq x_2} \{C[p,x) - R[p,x)\}$$

So there must exist a time point $x' \in [x_1, x_2)$ such that:

$$\begin{aligned} C[p,x_2) - R[p,x_2) &< C[p,x') - R[p,x') \qquad (19) \\ \Leftrightarrow C[x',x_2) &< R[x',x_2) \qquad (20) \end{aligned}$$

i.e., the total service provided in $[x', x_2)$ is strictly smaller than the events arrived in the same time interval, which leads to a contradiction with $B(x_2) = 0$. ■

In the following we prove that the upper/lower bounds in Theorem 2 are safe. More specifically, for any two time points s, t with $t - s = \Delta \geq 0$, we prove the following inequalities:

$$\begin{aligned} \alpha^{u'}: \quad & R'[s,t) \leq \min\Big(\sup_{0 \leq \lambda \leq T}\Big\{\inf_{0 \leq \mu \leq \lambda+\Delta}\{\alpha^u(\mu) \\ & \qquad + \beta^u(\lambda + \Delta - \mu)\} - \beta^l(\lambda)\Big\}, \beta^u(\Delta)\Big) \\ \alpha^{l'}: \quad & R'[s,t) \geq \min\Big(\inf_{0 \leq \mu \leq \Delta}\Big\{\sup_{0 \leq \lambda \leq T}\{\alpha^l(\mu) \\ & \qquad - \beta^u(\lambda + \Delta - \mu)\} + \beta^l(\lambda)\Big\}, \beta^l(\Delta)\Big) \\ \beta^{u'}: \quad & C'[s,t) \leq \inf_{\Delta \leq \lambda \leq T}\{\beta^u(\lambda) - \alpha^l(\lambda)\}^+ \\ \beta^{l'}: \quad & C'[s,t) \geq \sup_{0 \leq \lambda \leq \Delta}\{\beta^u(\lambda) - \alpha^l(\lambda)\} \end{aligned}$$

Proof of $\alpha^{u'}$:

First of all, by $R'[s,t) = R'[p,t) - R'[p,s)$ and (16) we have

$$R'[s,t) = C[p,t) - C'[p,t) - C[p,s) + C'[p,s) \tag{21}$$

We define a time point s' as follows:

- If $B(s) = 0$, then let $s' = s$.
- If $B(s) \neq 0$, then let s' be the start point of the busy period containing s. By $T \geq \mathsf{MBS}$ and Lemma 1 we know $s - T \leq s'$.

Then we can apply Lemma 2 ($x_1 = s - T$ and $x_2 = s'$) to get

$$C'[p,s') = \sup_{s-T \leq b \leq s'} \{C[p,b) - R[p,b)\}$$

Therefore, we can rewrite (21) as

$$R'[s,t) = \sup_{s-T \leq b \leq s'} \{C[s,t) - C'[p,t) + C[p,b) - R[p,b)\} \tag{22}$$

Now we focus on the expression inside the sup operation in the above equation, with an arbitrary b satisfying $s-T \leq b \leq s'$.

By the same way as defining s', we define t' with respect to t, and apply Lemma 2 ($x_1 = b$ and $x_2 = t'$) to get

$$C'[p,t') = \sup_{b \leq a \leq t'} \{C[p,a) - R[p,a)\} \tag{23}$$

We discuss in two cases

- If $B(t) = 0$, i.e., $t = t'$, we can rewrite (23) as
$$C'[p,t') = \sup_{b \leq a \leq t} \{C[p,a) - R[p,a)\}$$
- If $B(t) \neq 0$, i.e., t' is the start point of the busy period containing t. Therefore, for any time point $c \in (t', t]$, the available resource in time interval $[t', c)$ is strictly smaller than the request in that interval (otherwise the busy period terminates at c), i.e.,
$$\begin{aligned} & \forall c \in (t',t] : C[t',c) - R[t',c) < 0 \\ \Leftrightarrow & \forall c \in (t',t] : C[p,c) - R[p,c) < C[p,t') - R[p,t') \\ \Rightarrow & \sup_{b \leq a \leq t}\{C[p,a) - R[p,a)\} = \sup_{b \leq a \leq t'}\{C[p,a) - R[p,a)\} \end{aligned}$$
Combining this with (23) we get
$$C'[p,t') = \sup_{b \leq a \leq t} \{C[p,a) - R[p,a)\}$$

In summary, no matter whether $B(t)$ equals 0 or not, we have

$$C'[p,t') = \sup_{b \leq a \leq t} \{C[p,a) - R[p,a)\}$$

And by the definition of t' we know $C'[t',t) = 0$, we have

$$C'[p,t) = C'[p,t') = \sup_{b \leq a \leq t} \{C[p,a) - R[p,a)\}$$

So we can rewrite (22) as

$$\begin{aligned} R'[s,t) &= \sup_{s-T \leq b \leq s'}\{C[s,t) - \sup_{b \leq a \leq t}\{C[p,a) - R[p,a)\} \\ & \qquad + C[p,b) - R[p,b)\} \\ &= \sup_{s-T \leq b \leq s'}\{\inf_{b \leq a \leq t}\{C[s,t) + C[a,b) - R[a,b)\}\} \\ &\leq \sup_{s-T \leq b \leq s}\{\inf_{b \leq a \leq t}\{C[s,t) + C[a,b) - R[a,b)\}\} \end{aligned}$$

We define $\lambda = s - b$ and $\mu = a + \lambda - s$. Since $a \geq b$, we also know $\mu \geq 0$. Further, $\Delta = t - s$. Applying these substitutions to above we have

$$R'[s,t) \leq \sup_{0\leq\lambda\leq T} \{ \inf_{0\leq\mu\leq\lambda+\Delta} \{R[s-\lambda, \mu-\lambda+s) + C[\mu-\lambda+s, t) - C[s-\lambda, s)\}$$

Use the upper and lower curves to substitute R and C in above:

$$R'[s,t) \leq \sup_{0\leq\lambda\leq T} \{ \inf_{0\leq\mu\leq\lambda+\Delta} \{\alpha^u(\mu) + \beta^u(\lambda+\Delta-\mu)\} - \beta^l(\lambda)\}$$

Further, it is obvious that $R'[s,t)$ is also bounded by $\beta^u(\Delta)$, so finally we have

$$R'[s,t) \leq \min(\sup_{0\leq\lambda\leq T} \{ \inf_{0\leq\mu\leq\lambda+\Delta} \{\alpha^u(\mu) + \beta^u(\lambda+\Delta-\mu)\} - \beta^l(\lambda)\}, \beta^u(\Delta))$$

Proof of $\alpha^{l'}$:

By the computation of $\alpha^{l'}$ in the original GPC we have:

$$R'[s,t) \geq \min(\inf_{0\leq\mu\leq t-s} \{ \sup_{0\leq\lambda} \{\alpha^l(\mu) - \beta^u(\lambda+t-s-\mu)\} + \beta^l(\lambda)\}, \beta^l(t-s))$$
$$\Rightarrow R'[s,t) \geq \min(\inf_{0\leq\mu\leq t-s} \{ \sup_{0\leq\lambda\leq T} \{\alpha^l(\mu) - \beta^u(\lambda+t-s-\mu)\} + \beta^l(\lambda)\}, \beta^l(t-s))$$

Proof of $\beta^{u'}$:

By the computation of $\beta^{u'}$ in the original GPC we have:

$$C'[s,t) \leq \inf_{t-s\leq\lambda} \{\beta^u(\lambda) - \alpha^l(\lambda)\}^+$$
$$\Rightarrow C'[s,t) \leq \inf_{t-s\leq\lambda\leq T} \{\beta^u(\lambda) - \alpha^l(\lambda)\}^+$$

Proof of $\beta^{l'}$:

The proof is trivial since the computation of $\beta^{l'}$ in GPC$_T$ is exactly the same as in the original GPC.

Appendix C: Proof of Theorem 3

We first introduce an important lemma:

Lemma 3. *Given well-defined arrival and service curves α and β, and two arbitrary non-negative real numbers x and y,*

$$\alpha^l(x) \geq \alpha^l(x+y) - \alpha^u(y) \quad (24)$$
$$\beta^u(x) \leq \beta^u(x+y) - \beta^l(y) \quad (25)$$

Proof: Since $\alpha = (\alpha^u, \alpha^l)$ is a well-defined (thus *causal*) arrival curve pair, we know $\alpha^l = \alpha^l \oslash \alpha^u$ (see Definition 5), i.e.,

$$\alpha^l(x) = \sup_{\lambda\geq 0} \{\alpha^l(x+\lambda) - \alpha^u(\lambda)\}$$
$$\Rightarrow \quad \alpha^l(x) \geq \alpha^l(x+y) - \alpha^u(y)$$

Since $\beta = (\beta^u, \beta^l)$ is also a causal service curve pair, we know $\beta^u = \beta^u \overline{\oslash} \beta^l$, i.e.,

$$\beta^u(x) = \inf_{\lambda\geq 0} \{\beta^u(x+\lambda) - \beta^l(\lambda)\}$$
$$\Rightarrow \quad \beta^u(x) \leq \beta^u(x+y) - \beta^l(y)$$

■

Now we start to prove Theorem 3. More specifically, we shall prove that the output arrival/service upper curves obtained by GPC$_T$ are no larger than its counterpart obtained by GPC, and the output arrival/service lower curves are no smaller than that obtained by GPC.

Proof of $\alpha^{u'}$:

Let $F(\Delta, \lambda) = \inf_{0\leq\mu\leq\lambda+\Delta} \{\alpha^u(\mu) + \beta^u(\lambda+\Delta-\mu)\} - \beta^l(\lambda)$. To prove that the $\alpha^{u'}$ obtained by GPC$_T$ is no larger than that obtained by GPC, we need to show that $\forall \Delta \geq 0$:

$$\min\{ \sup_{0\leq\lambda\leq T} \{F(\Delta,\lambda)\}, \beta^l(\Delta)\} \leq \min\{ \sup_{0\leq\lambda} \{F(\Delta,\lambda)\}, \beta^l(\Delta)\}$$

which is obviously true.

Proof of $\alpha^{l'}$:

The main task of this proof is to show

$$\sup_{0\leq\lambda\leq T} \{\alpha^l(\mu+\lambda) - \beta^u(\lambda)\} = \sup_{0\leq\lambda} \{\alpha^l(\mu+\lambda) - \beta^u(\lambda)\} \quad (26)$$

As soon as (26) is proved, it is easy to see that the $\alpha^{l'}$ obtained by GPC$_T$ is no smaller than its counterpart obtained in GPC. In the following we prove (26).

For simplicity of presentation, let m = MBS(α^u, β^l), so $\alpha^u(m) = \beta^l(m)$. For an arbitrary λ with $\lambda \geq 0$, let $q = \lfloor \frac{\lambda}{m} \rfloor$ and $r = \lambda - q \cdot m$. Then by Lemma 3 and the sub-additivity of α^u we know

$$\alpha^l(\mu+\lambda) - \alpha^l(\mu+r) \leq \alpha^u(q\cdot m) \leq q \cdot \alpha^u(m) \quad (27)$$

On the other hand, by Lemma 3 and the supper-additivity of β^l we also know

$$\beta^u(\mu+\lambda) - \beta^u(\mu+r) \geq \beta^l(q\cdot m) \geq q \cdot \beta^l(m) \quad (28)$$

By (27), (28) and $\alpha^u(m) = \beta^l(m)$ we can get

$$\alpha^l(\mu+\lambda) - \beta^u(\mu+\lambda) \leq \alpha^l(\mu+r) - \beta^u(\mu+r)$$

In other words, for any $\lambda \geq 0$, we can always find a corresponding r in the range of $[0, m]$ such that $\alpha^l(\mu+\lambda) - \beta^u(\mu+\lambda) \leq \alpha^l(\mu+r) - \beta^u(\mu+r)$, and by $m \leq T$ we finally get (26).

Proof of $\beta^{u'}$:

We want to prove

$$\inf_{0\leq\lambda\leq T} \{\beta^u(\lambda) - \alpha^l(\lambda)\}^+ = \inf_{0\leq\lambda} \{\beta^u(\lambda) - \alpha^l(\lambda)\}^+ \quad (29)$$

Similar with the above proof, let m = MBS(α^u, β^l), so $\alpha^u(m) = \beta^l(m)$. Let $q = \lfloor \frac{\lambda}{m} \rfloor$ and $r = \lambda - q \cdot m$. Then by Lemma 3 and the sub-additivity of α^u we know

$$\alpha^l(\lambda) - \alpha^l(r) \leq \alpha^u(q\cdot m) \leq q \cdot \alpha^u(m) \quad (30)$$

On the other hand, by Lemma 3 and the super-additivity of β^l we can get

$$\beta^u(\lambda) - \beta^u(r) \geq \beta^l(q\cdot m) \geq q \cdot \beta^l(m) \quad (31)$$

By (30), (31) and $\alpha^u(m) = \beta^l(m)$ we have

$$\beta^u(\lambda) - \alpha^l(\lambda) \geq \beta^u(r) - \alpha^l(r)$$

In other words, for any $\lambda \geq 0$, we can always find a corresponding r in the range of $[0, m]$ such that $\beta^u(\lambda) - \alpha^l(\lambda) \geq \beta^u(r) - \alpha^l(r)$, and since $m \leq T$, finally we get (29).

Proof of $\beta^{l'}$:

The proof is trivial since the computation of $\beta^{l'}$ in GPC$_T$ is exactly the same as in the original GPC.