Timing Analysis of Process Graphs with Finite Communication Buffers

Chung-Wei Lin[†], Marco Di Natale[‡], Haibo Zeng[§], Linh Thi Xuan Phan[*], and Alberto Sangiovanni-Vincentelli[†]
U. of California, Berkeley[†], Scuola Superiore Sant'Anna[‡], McGill University[§], U. of Pennsylvania[*]
E-Mails: {cwlin,alberto}@eecs.berkeley.edu, marco@sssup.it, haibo.zeng@mcgill.ca, linhphan@cis.upenn.edu

***Abstract*—Real-Time Calculus (RTC) is a modular performance analysis framework for real-time embedded systems. It can be used to compute the worst-case and best-case response times of tasks with general activation patterns and configurations, such as pipelines of tasks that are connected via finite buffers. In this paper, we extend the existing RTC framework to analyze arbitrary graph configurations of tasks and messages, with mixed periodic and event-based activation models and finite buffers between any pair of nodes. Our extension also improves upon several sources of pessimism in the existing analysis. We present an application of the extended RTC to the Loosely Time-Triggered Architecture (LTTA) implementation of synchronous models, commonly used in the development of embedded automotive, avionics and control systems. We show how our method can be used to model scheduling and communication delays in an LTTA mapping, which gives tighter analysis bounds on the output rate and the latency compared to existing techniques. The evaluation on automotive workloads shows that our approach is scalable and outperforms existing techniques in terms of analysis accuracy.**

I. Introduction

Originated from the Network Calculus [3], [7] in the networks domain, the Real-Time Calculus (RTC) [6] has been proposed to model and analyze timing properties of software task systems. RTC allows for the modeling of several task activation patterns, such as sporadic events (with a minimum and maximum inter-arrival time), periodic models with jitter, and transactions in which tasks (messages) are activated by the completion of their predecessors or new message arrivals. RTC can also be used to analyze the system utilization [17] and the timing behavior of tasks with dependencies, including systems with feedback controls [4] and loops [13].

In this paper, we extend the existing RTC framework to analyze general system configurations that consist of arbitrary graphs of processes (representing tasks and messages). In such systems, tasks and messages are scheduled based on their priorities, and they can be activated in a mixed time- and event-driven manner. Each root task (at the sensor end of the system graph) is activated according to a sporadic or periodic with jitter model, whereas an intermediate message or task is activated by either a periodic clock or the completion (arrival) of its predecessor, or in a mixed mode, as specified next. In addition, any two tasks or messages can be connected via a finite-length buffer (queue), which stores the activation information or tokens. Such system configurations fit the standard requirements in many complex distributed real-time systems, e.g., in the automotive domain where electronic control units (ECUs) and networks often employ priority-based scheduling, and they are typically not synchronized.

We present an application of our method to the analysis of Loosely Time-Triggered Architecture (LTTA) [1] implementations. LTTA is of practical relevance, not only due to its prominence in many automotive and control systems but because of the existence of LTTA implementations that preserve the correctness of synchronous reactive (SR) models. We provide a method for checking the assumptions required by previous rate analysis, applied to the mapping of SR models onto LTTA, such as the schedulability of all tasks within the deadline or bounded communication delays. In addition, we present a method for computing end-to-end latencies and communication and scheduling delays. Unlike the work in [2], which assumes the existence of an upper bound on communication delays (without addressing the related analysis) and enforces a corresponding waiting time in the execution of tasks, our method provides a way to compute safe bounds on task and message delays and does not require modifications to the task execution model.

In summary, the main contributions of this work are:

- We correct a flaw in the existing RTC formula for computing the lower bounds on performance [6], [12], which is important as the existing analysis may lead to buffer overflows and unsafe timing behavior of the system (Section III).
- We improve the RTC analysis for pipelines of tasks with feedback controls [4] to enable more efficient analysis and tighter results. We further propose an extension to fork-and-merge topologies, thereby enabling the analysis of general-type task graph configurations (Section IV).
- We present an application of our analysis method to the analysis of synchronous models implemented in LTTA, which improves upon the method in [14] in both efficiency and accuracy as well as modeling flexibility (Section V).
- We illustrate the benefits of our method via evaluations using both synthetic and automotive workloads. The results show that our method outperforms existing RTC methods in both efficiency and accuracy (Section VI).

II. System Model

Our system model consists of an arbitrary *acyclic* graph of statically allocated processes (representing tasks and messages). Tasks $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ are scheduled on CPUs

based on their priorities. Messages $\mathcal{M} = \{\mu_1, \mu_2, \ldots, \mu_m\}$ are sent over communication (broadcast) buses and are associated with a priority. When a bus is idle or at the end of a message frame, an arbitration protocol ensures that the highest priority message is transmitted next. Examples of standards supporting this model are the OSEK/AUTOSAR OS and the controller area network (CAN). Source tasks (in the graph) are activated by *triggers* according to a sporadic or periodic (with jitter) event model. Each intermediate task or message may be activated by triggers, or by the completion of a predecessor task or the arrival of an incoming message (modeled as an *activation token*, possibly associated with information content passed from the predecessor in the graph). If Θ_i is a reference period for a task τ_i (or message μ_i), and δ_i^m and δ_i^M are positive relative bounds, with $0 \leq \delta_i^m < 1$ and $\delta_i^M \geq 0$, the sequence of activation events $\theta_i(n)$ can be modeled as:

$$\Theta_i^m = \Theta_i(1-\delta_i^m) \leq \theta_i(n+1) - \theta_i(n) \leq \Theta_i(1+\delta_i^M) = \Theta_i^M. \quad (1)$$

The trigger events of tasks are modeled by a stream $t_i(n)$:

$$t_i(n) \in [\theta_i(n), \theta_i(n) + J_i], \quad (2)$$

where J_i is the jitter. Equation (1) captures the effects of the drift of a clock, or sporadic activations. Equation (2) captures the effect of a periodic activation with jitter. In addition, a pair of tasks, (τ_i, τ_j), or a task and a message, (μ_i, τ_j) or (τ_i, μ_j), may be connected by a communication link $l_{i,j}$. Each communication link $l_{i,j}$ carries a limited number of activation and information tokens, and is further characterized by a buffer size $B_{i,j}$.

The general form of a task body is shown in the following code section. Line 2 (optional for intermediate tasks and messages) represents a possible trigger-based activation. The **`wait until`** condition of Line 3 is the union of an activation model based on the completion of the predecessor task or the arrival of a predecessor message (the first condition) and a condition that ensures that communication buffers do not overflow (the second condition in the conjunction). When both Lines 2 and 3 are present, the activation model is mixed: the task is activated by a trigger and then waits for tokens and queue space availability.

```
1  while (true) {
2    wait trigger;
3    wait until (all input queues are non-empty
      and all output queues are non-full);
4    read input from input queues;
5    execute body;
6    write output to output queues;
7  }
```

III. Existing Analysis using RTC

In this section, we present a background on RTC and present a correction to a flaw in the existing analysis.

A. RTC Background

The RTC [6], [12] uses the min-plus algebra or max-plus algebra [3], [7] to analyze the timing performance of computation and communication systems.

Definition 1: Let $\mathcal{R}^+ = [0, \infty)$ and $f, g : \mathcal{R}^+ \to \mathcal{R}$. The common **min-plus operators** are:

- Minimum: $(f \oplus g)(t) = \min(f(t), g(t))$.
- Convolution: $(f \otimes g)(t) = \inf_{0 \leq s \leq t}(f(s) + g(t-s))$.
- Deconvolution: $(f \oslash g)(t) = \sup_{u \geq 0}(f(t+u) - g(u))$.
- Zero element: $f(t) = \infty, \forall t$.
- Unitary element: $f^0(0) = 0$ and $f^0(t) = \infty, \forall t \neq 0$.
- Sub-additive closure:

$$f^n = \bigotimes_{i=1}^{n} f; f^* = \bigoplus_{i \geq 0} f^i; f^+ = \bigoplus_{i > 0} f^i.$$

The **max-plus convolution** and *deconvolution* are:

- Convolution: $(f \overline{\otimes} g)(t) = \sup_{0 \leq s \leq t}(f(s) + g(t-s))$.
- Deconvolution: $(f \overline{\oslash} g)(t) = \inf_{u \geq 0}(f(t+u) - g(u))$.

RTC is based on the concepts of arrival and service functions and curves. They are formally defined as follows:

Definition 2: An **input arrival function** $R[s,t)$ is the amount of load requests that arrive over the time interval $[s,t)$.

Definition 3: An **output arrival function** $R'[s,t)$ is the amount of load that leaves the system (tasks completed or messages transmitted) over the time interval $[s,t)$.

Definition 4: An **input service function** $C[s,t)$ is the amount of available service (from a resource) over the time interval $[s,t)$.

Definition 5: An **output service function** $C'[s,t)$ is the amount of remaining service over the time interval $[s,t)$.

Since we are interested in bounding the worst-case behavior, we need to extract the worst-case *arrival curves* from the arrival functions $R[s,t)$.

Definition 6: The increasing functions $\alpha^u(t)$ and $\alpha^l(t)$ are the **upper** and **lower arrival curves** of $R[s,t)$, respectively, if $\forall 0 \leq s \leq t$,

$$\alpha^l(0) \leq 0; \;\; \alpha^u(0) \leq 0; \;\; \alpha^l(t-s) \leq R[s,t) \leq \alpha^u(t-s).$$

The output is produced by the system based on the input request and the availability of processing/transmission time, represented by the *service curves*.

Definition 7: The increasing functions $\beta^u(t)$ and $\beta^l(t)$ are the **upper** and **lower service curves** of $C[s,t)$, respectively, if $\forall 0 \leq s \leq t$,

$$\beta^l(0) \leq 0; \;\; \beta^u(0) \leq 0; \;\; \beta^l(t-s) \leq C[s,t) \leq \beta^u(t-s).$$

Given a system with its (input) arrival curves α^u and α^l and (input) service curves β^u and β^l, the following theorem is provided in [6] to compute the *output arrival curves* and *output service curves* (or remaining service curves).

Theorem 1: Given a system with α^u, α^l, β^u, and β^l, the **upper/lower output arrival/service curves** are given by:

$$\alpha'^u = ((\alpha^u \otimes \beta^u) \oslash \beta^l) \oplus \beta^u; \quad (3)$$

$$\alpha'^l = ((\alpha^l \oslash \beta^u) \otimes \beta^l) \oplus \beta^l; \quad (*) \quad (4)$$

$$\beta'^u = \max\{(\beta^u - \alpha^l) \overline{\oslash}\, 0, 0\}; \quad (5)$$

$$\beta'^l = (\beta^l - \alpha^u) \overline{\otimes}\, 0. \quad (6)$$

These concepts are illustrated in Figure 1. A process is triggered by incoming events, and events are processed in a FIFO order subjected to resource availability. Causality can

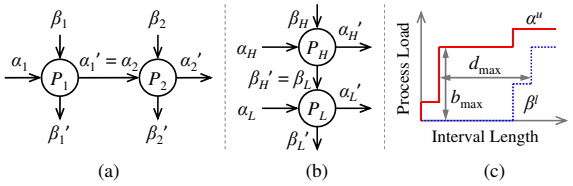


Fig. 1. (a) Causal dependency, (b) priority-based execution, and (c) backlog and delay.

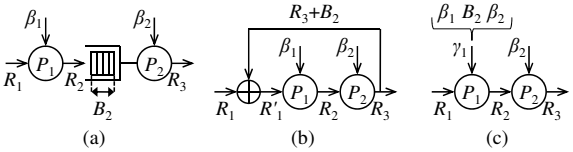


Fig. 2. (a) The system has two processes. (b) The feedback control edge is added to limit the services of the first process. (c) The resulting effective service curve of the first process guarantees that the second buffer never overflows.

be easily represented by a chain (Figure 1 (a)), where the completion of each request by the subsystem P_1 triggers a request to P_2. The vertical stacking along the line of the service curve flow can be used to represent the priority-based execution of requests on the same resource (Figure 1 (b)) with α_H processed at priority higher than α_L. Knowledge of the arrival and service curves for a processing element allows computing the *maximal backlog* and *maximal delay* [6].

***Theorem** 2:* Given a system with α^u and β^l, the **maximal backlog** and the **maximal delay** are:

$$b_{\max}(\alpha^u, \beta^l) = \sup\{\alpha^u(t) - \beta^l(t) \mid t \geq 0\}; \quad (7)$$

$$d_{\max}(\alpha^u, \beta^l) = \sup_{t \geq 0} \left(\inf_{s \geq 0, \alpha^u(t) \leq \beta^l(t+s)} s \right). \quad (8)$$

As shown in Figure 1 (c), $b_{\max}$ is the maximal vertical distance from β^l to α^u, whereas $d_{\max}$ is the maximal horizontal distance from α^u to β^l. Unfortunately, Equation (4) is flawed. We show why and provide a fix in the following.

B. Analysis by Feedback Control

In this section, we recall the analysis method for finite buffers using the feedback control technique presented in [4] with the approximations $\alpha^l(t) = 0$ and $\beta^u(t) = \infty$ for all $t \geq 0$. Following the same notation as in [4], in this section, α and β refer to α^u and β^l, respectively.

Figure 2 (a) illustrates the basic feedback mechanism for two processes P_1 and P_2, connected by a finite queue of length B_2. To ensure that the backlog of the second process does not exceed B_2, $R_2 - R_3 \leq B_2$ or, $R_2 \leq B_2 + R_3$. The first process P_1 must be stalled when the queue is full. This can be obtained by limiting the incoming processed load to the minimum between R_1 and $B_2 + R_3$ and is equivalent to a feedback control before P_1, as shown in Figure 2 (b). The effective arrival of P_1 therefore becomes $R'_1 = \min(R_1, B_2 + R_3)$ (the queue is not shown since its effect is modeled by the feedback). Given that R_3 is computed based on β_2 and R_2, and the latter is a function of the application of the service curve β_1 to R_1, it seems reasonable to model the need for stalling P_1 by replacing the service β_1 with a modified or *effective service curve* γ_1 function of β_1, β_2 and B_2 [4]:

***Theorem** 3:* Given a system with two processes and a buffer between them, the effective service curve of the first process is:

$$\gamma_1 = \beta_1 \otimes [\beta_1 \otimes (\beta_2 + B_2)]^*. \quad (9)$$

The authors proceed to determine an upper bound for the effective service curve of the first process (and the following) in a pipeline configuration as a closed-form formula [4]:

***Theorem** 4:* Given a system of a chain with n processes, the effective service curve of the first process is:

$$\gamma_1 = \beta_1 \otimes \left(\bigoplus_{i=1}^{n} \bigotimes_{j=1}^{i} [\beta_{j-1} \otimes (\beta_j + B_j)]^+ \right). \quad (10)$$

The above analysis only considers the upper bound of an arrival curve and the lower bound of a service curve, and it only considers a pipeline of processes. We are interested in a more general formulation that allows for the treatment of general graphs. Also, we observe that the analysis can be simplified and made tighter. Specifically, the closed form of Equation (10) is obtained by inductive reasoning using Equation (9) on pairs of communicating processes. When extending a system from n to $n+1$ processes, the effective service curve γ_1 is computed by recursively using the effective service curve $\gamma_n = \beta_n \otimes [\beta_n \otimes (\beta_{n+1} + B_{n+1})]^*$ as in Equation (9). However, instead of considering the effective values $\gamma_2, \ldots, \gamma_{n-1}$ of the middle processes, only $\beta_2, \ldots, \beta_{n-1}$ are used in the equation. Hence, although the closed form solution given by Equation 10 is sound, it may be pessimistic. In this paper, we will take into account the effects of $\gamma_2, \ldots, \gamma_{n-1}$ on β_1 via an Effect-Propagation algorithm based on a compact and elegant closed-form of Equation (9), as detailed in Section IV-B.

C. Analysis by Token Control

Another possible method for the analysis of systems with buffered communication is provided in [13], where an equivalent model is defined, in which processes are represented as nodes, edges represent communication flows and tokens on edges are activation events. Given a marked graph in which the service curves β_i^u and β_i^l define the amount of service time on vertex i and the number of tokens on edge (i, j) is denoted as m_{ij}, the system matrices S^u and S^l are defined as follows:

$$S_{ij}^{u,l} = \begin{cases} \beta_i^{u,l} + m_{ji}, & \text{if edge } (j,i) \text{ exists;} \\ \infty, & \text{otherwise.} \end{cases}$$

Then, the upper and lower bounds on the arrival function at each edge are computed as $R^u = (S^u)^* \otimes \beta^u$ and $R^l = (S^l)^* \otimes \beta^l$, where matrix S^* is the min-closure [13] of matrix S. Correspondingly, the upper and lower bound for the arrival curve at each edge are computed as $\alpha^u = R^u \oslash R^u$ and $\alpha^l = R^l \overline{\oslash} R^l$. The method is more general than the analysis of task pipelines in [4], but can still be pessimistic, as shown in Section VI.

D. Fixing the Lower Bound of the Output Arrival Curve

In several research works (including [4]) the lower bound on the output arrival curve is computed using the formula in Equation (4), first developed and demonstrated in [15], which is unfortunately flawed. This is a critical issue because an

overestimated lower bound is not safe, *i.e.*, the real performance may be slower than the computed lower bound and lead to buffer overflows. *We identify the problem with the demonstration in [15] and fix it by providing a safe lower bound.*

We first recall the following two lemmas. The first is a result from [15]. The second is trivially true, since the amount of processing/transmission time that completes (and leaves the system) over a time interval is no more than the amount of available service time over the same time interval.

***Lemma** 1:* If the buffer is empty at time p, then [15]

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

***Lemma** 2:* $R'[s,t) \le C[s,t)$.

Theorem 5 provides a safe bound for the output arrival function (the fix to [15]).

***Theorem** 5:* A safe lower output arrival curve is:

$$\alpha'^l = (\alpha^l \otimes \beta^l) \oplus \beta^l. \tag{11}$$

Note that $\alpha'^l = \alpha^l \otimes \beta^l$ if $\alpha^l(0) = 0$.

Proof: Assuming the buffer is empty at an arbitrarily small time p (same as the demonstration in [15]),

$$\begin{aligned}
&R'[s,t) = R'[p,t) - R'[p,s)\\
&= C[p,t) - \sup_{p\le a\le t}\{C[p,a) - R[p,a)\} - C[p,s) + \sup_{p\le b\le s}\{C[p,b) - R[p,b)\}\\
&= \inf_{p\le a\le t}\left\{\sup_{p\le b\le s}\{R[b,a) - C[b,s) + C[a,t)\}\right\}\\
&= \inf_{\substack{p-s\le\mu\\ \mu\le t-s}}\left\{\sup_{0\le\lambda\le s-p}\{R[s-\lambda,\mu+s) - C[s-\lambda,s) + C[\mu+s,t)\}\right\}\\
&= \min\left\{\inf_{\substack{p-s\le\mu\\ \mu\le t-s}}\left\{\sup_{\substack{0\le\lambda\\ \lambda\le s-p}}\{R[s-\lambda,\mu+s) - C[s-\lambda,s) + C[\mu+s,t)\}\right\}, C[s,t)\right\}.
\end{aligned}$$

At this point, the proof in [15] incorrectly derives a lower bound by extending the evaluation of the superior to $\lambda \ge 0$ (as opposed to the original interval $0 \le \lambda \le s-p$):

$$\begin{aligned}
&\min\left\{\inf_{\substack{p-s\le\mu\\ \mu\le t-s}}\left\{\sup_{\substack{0\le\lambda\\ \lambda\le s-p}}\{R[s-\lambda,\mu+s) - C[s-\lambda,s) + C[\mu+s,t)\}\right\}, C[s,t)\right\}\\
&\ge \min\left\{\inf_{\substack{0\le\mu\\ \mu\le t-s}}\left\{\sup_{\lambda\ge 0}\{R[s-\lambda,\mu+s) - C[s-\lambda,s) + C[\mu+s,t)\}\right\}, C[s,t)\right\}.
\end{aligned}$$

This is not correct since the superior could be higher if evaluated on a larger interval. A correct lower bound can instead be obtained by restricting the evaluation of the superior to the case $\lambda = 0$:

$$\begin{aligned}
&\min\left\{\inf_{\substack{p-s\le\mu\\ \mu\le t-s}}\left\{\sup_{\substack{0\le\lambda\\ \lambda\le s-p}}\{R[s-\lambda,\mu+s) - C[s-\lambda,s) + C[\mu+s,t)\}\right\}, C[s,t)\right\}\\
&\ge \min\left\{\inf_{\substack{0\le\mu\\ \mu\le t-s}}\left\{\sup_{\lambda=0}\{R[s-\lambda,\mu+s) - C[s-\lambda,s) + C[\mu+s,t)\}\right\}, C[s,t)\right\}\\
&= \min\left\{\inf_{0\le\mu\le t-s}\{R[s,\mu+s) + C[\mu+s,t)\}, C[s,t)\right\}\\
&= (R\otimes C)\oplus C \ge (\alpha^l\otimes\beta^l)\oplus\beta^l.
\end{aligned}$$

■

IV. RTC Analysis of Graphs with Finite Buffers

We now extend the existing RTC analysis to consider systems with general graph configurations and limited buffer sizes.

A. Problem Description

A system can be modeled as a graph (V, E). If the in-degree of a process (task or message) $v_i \in V$ is 0, then the input arrival curve of the process is given; otherwise, the effective input arrival curve of the process can be computed based on the output arrival curves of its sources.

Problem statement: given a graph (V, E) with corresponding input service curves, input arrival curves, and buffer sizes $B_{i,j}$ between v_i and v_j, compute the upper bound and the lower bound of the effective output arrival curve for each v_i such that buffers never overflow.

***Definition** 8:* A solution (set of pair of bounds) is **correct** if any process request function outside the defined upper and lower bounds results in a buffer overflow or underflow.

Finding a correct solution is clearly not enough, given that ∞ as an upper bound and 0 as a lower bound are always feasible but do not allow any progress of the network. We are interested in tight bounds, possibly optimal, according to the definition below.

***Definition** 9:* A correct solution is **optimal** if any decrease in its upper bound or increase in its lower bound makes it incorrect.

B. Effect Propagation

We first focus on a chain topology and improve on the bound in Equation (10) from [4]. Our Effect-Propagation algorithm follows the reverse topological order and propagates the effects from the last process to the first. In each step, it uses the service curve of the next process to update the service curve of the current process:

$$\gamma_i^u = (\gamma_{i+1}^l + B_{i,i+1}) \oplus \beta_i^u; \tag{12}$$

$$\gamma_i^l = (\gamma_{i+1}^l + B_{i,i+1}) \oplus \beta_i^u, \tag{13}$$

where $B_{i,i+1}$ is the buffer size between the i-th process and $(i+1)$-th process in the chain. Using $(\gamma_{i+1}^l + B_{i,i+1})$ as a constraint in the algorithm has several advantages:

- Guarantees that buffers never overflow (see Theorem 6).
- Provides a tighter bound on the effective service curves, compared with Equations (9) and (10) in [4].
- Requires fewer operations, compared with Equations (9) and (10) in [4].
- Applies to any acyclic graph, unlike Equation (10) which can only be applied to a chain topology.

***Theorem** 6:* The Effect-Propagation algorithm guarantees that buffers never overflow.

Proof: By Equation (12) and the definition of $\oplus$ (min operator), $\gamma_i^u \le (\gamma_{i+1}^l + B_{i,i+1})$. Similarly, considering the last term of Equation (3) is β_i^u, replaced by γ_i^u (effective service), it is $\alpha_i'^u \le (\gamma_{i+1}^l + B_{i,i+1})$. Also, $\alpha_i'^u = \alpha_{i+1}$ in the pipeline configuration. Last, replacing β with γ in Equation (7), the theorem is proved. ■

C. Fork and Merge Topologies

Handling fork and merge topologies is necessary for the analysis of more general graph configurations. A fork topology is shown in Figure 3 (a). When process P_1 finishes its

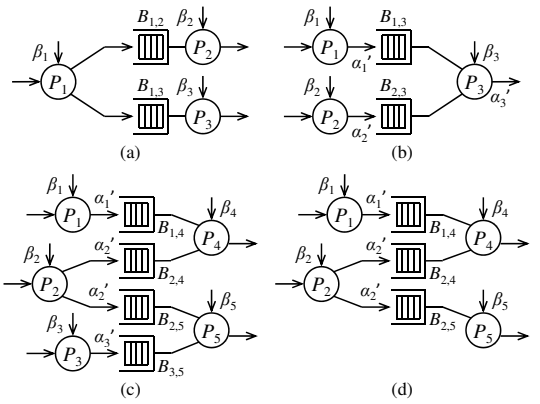


Fig. 3. Fork and merge topologies.

computation, it writes the result to both output buffers. The effective service curve of P_1 computed for the first branch is an upper bound of the service for P_1 that prevents overflow on the corresponding buffer. The same is true for the curve for the second branch. Therefore, the effective service curve of P_1 can be obtained by considering the minimum service resulting from the constraints on both sides, similar to Equations (12) and (13):

$$\gamma_1^u = (\gamma_2^l + B_{1,2}) \oplus (\gamma_3^l + B_{1,3}) \oplus \beta_1^u;$$
$$\gamma_1^l = (\gamma_2^l + B_{1,2}) \oplus (\gamma_3^l + B_{1,3}) \oplus \beta_1^u.$$

A merge topology is shown in Figure 3 (b). The successor process P_3 can only start its computation when both its input buffers are non-empty. In this case, the problem also comes from the effects of other input processes. For example, if P_2 is slow, P_1 needs to wait for P_2 to prevent its output buffer from overflowing. In this case, to guarantee that all buffers do not overflow, we could use $(\alpha_1'^l \oplus \alpha_2'^l \oplus \gamma_3 + B_{1,3})$ and $(\alpha_1'^l \oplus \alpha_2'^l \oplus \gamma_3 + B_{2,3})$ to compute the effective service curves of P_1 and P_2. Although this is correct, there is an opportunity for computing tighter upper and lower bounds. As shown in Figure 3 (b), if any input arrival curve is the minimum of all arrival curves of P_3, *i.e.*, $a_1'^l = a_1'^l \oplus a_2'^l$ or $a_2'^l = a_1'^l \oplus a_2'^l$, we can use $(a_2'^l \oplus \gamma_3^l + B_{1,3})$ or $(a_1'^l \oplus \gamma_3^l + B_{2,3})$ to update β_1 or β_2:

$$\gamma_2^u = \begin{cases} (a_1'^l \oplus \gamma_3^l + B_{2,3}) \oplus \beta_2^u, & \text{if } a_2'^l = a_1'^l \oplus a_2'^l; \\ (\alpha_1'^l \oplus \alpha_2'^l \oplus \gamma_3^l + B_{2,3}) \oplus \beta_2^u, & \text{otherwise;} \end{cases}$$

$$\gamma_2^l = \begin{cases} (a_1'^l \oplus \gamma_3^l + B_{2,3}) \oplus \beta_2^l, & \text{if } a_2'^l = a_1'^l \oplus a_2'^l; \\ (\alpha_1'^l \oplus \alpha_2'^l \oplus \gamma_3^l + B_{2,3}) \oplus \beta_2^l, & \text{otherwise;} \end{cases}$$

$$\gamma_1^u = \begin{cases} (a_2'^l \oplus \gamma_3^l + B_{1,3}) \oplus \beta_1^u, & \text{if } a_1'^l = a_1'^l \oplus a_2'^l; \\ (\alpha_1'^l \oplus \alpha_2'^l \oplus \gamma_3^l + B_{1,3}) \oplus \beta_1^u, & \text{otherwise;} \end{cases}$$

$$\gamma_1^l = \begin{cases} (a_2'^l \oplus \gamma_3^l + B_{1,3}) \oplus \beta_1^l, & \text{if } a_1'^l = a_1'^l \oplus a_2'^l; \\ (\alpha_1'^l \oplus \alpha_2'^l \oplus \gamma_3^l + B_{1,3}) \oplus \beta_1^l, & \text{otherwise.} \end{cases}$$

The minimum does not always exist because $a_1'^l \oplus a_2'^l$ may not be the same as either $a_1'^l$ or $a_2'^l$. When applicable, this bound is correct because the process with minimum arrival curve never waits for itself, while the other processes wait for it. Theorem 7 will demonstrate that buffers never overflow.

In more general graph configurations, cross-dependencies between fork and join nodes require the evaluation of the

```
Algorithm: Fork-Merge-Extension(V, E, α^u, α^l, β^u, β^l, B)
01  Sort V by the topological order;
02  while 1
03    for i = 1 to |V|
04      if in-degree(v_i) > 0
05        α_i^u = ⊕_{j∈S_i} α_j'^u;
06        α_i^l = ⊕_{j∈S_i} α_j^l;
07      α_i'^u = ((α_i^u ⊗ β_i^u) ⊘ β_i^l) ⊕ β_i^u;
08      α_i'^l = (α_i^l ⊗ β_i^l) ⊕ β_i^l;
09    (γ^u, γ^l) = (β^u, β^l);
10    if all buffers are not overflowed
11      return γ^u and γ^l;
12    for i = |V| to 1
13      if out-degree(v_i) > 0
14        V' = {v | v ∈ V, (v_i, v) ∈ E};
15        for each v ∈ V'
16          j = the index of v in V;
17          if in-degree(v_j) > 1 and α_i'^l ≠ α_j^l
18            γ_i^u = (α_j^l ⊕ γ_j^l + B_{i,j}) ⊕ γ_i^u;
19            γ_i^l = (α_j^l ⊕ γ_j^l + B_{i,j}) ⊕ γ_i^l;
20          else if in-degree(v_j) > 1 and α_i'^l = α_j^l
21            α̂_j^l = ⊕_{k∈S_j\{i}} α_k'^l;
22            γ_i^u = (α̂_j^l ⊕ γ_j^l + B_{i,j}) ⊕ γ_i^u;
23            γ_i^l = (α̂_j^l ⊕ γ_j^l + B_{i,j}) ⊕ γ_i^l;
24          else
25            γ_i^u = (γ_j^l + B_{i,j}) ⊕ γ_i^u;
26            γ_i^l = (γ_j^l + B_{i,j}) ⊕ γ_i^l;
27    if (β^u, β^l) = (γ^u, γ^l)
28      return (γ^u, γ^l);
29    (β^u, β^l) = (γ^u, γ^l);
```

Fig. 4. The Fork-Merge-Extension algorithm.

propagation of the corresponding conditions on flows across nodes. Consider, for example, Figure 3 (c). In this case, if P_2 is faster than P_3, then P_2 will be stalled at some point, because P_5 cannot go faster than P_3 and, in turn, in order not to overflow its output buffer, P_2 will need to match the slower rate of P_5. Also, if P_1 is faster than P_2, then P_1 needs to wait for both P_2 and P_3, *i.e.*, the effect of our bounded buffers on processing speeds will propagate from P_3 to P_1. The generalized Fork-Merge-Extension algorithm is shown in Figure 4. Before introducing the algorithm, it is useful to introduce some definitions on the graph (V, E) with $n = |V|$ processes and $m = |E|$ edges.

- S_i: the index set of the sources of P_i.
- T_i^-: the index set of the non-merging targets of P_i.
- $T_i^>$: the index set of the merging targets of P_i.

The Fork-Merge-Extension algorithm is summarized as follows:

- In Phase 1 (Lines 3–8), scan all processes following the topological order:
  - For any process with in-degree larger than 0, set the input arrival curve bounds as

$$(\alpha_i^u, \alpha_i^l) = \left(\bigoplus_{j \in S_i} \alpha_j'^u, \bigoplus_{j \in S_i} \alpha_j'^l \right). \quad (14)$$

  - For any process, set the output arrival curve bounds as

$$\alpha_i'^u = ((\alpha_i^u \otimes \beta_i^u) \oslash \beta_i^l) \oplus \beta_i^u; \quad (15)$$
$$\alpha_i'^l = (\alpha_i^l \otimes \beta_i^l) \oplus \beta_i^l. \quad (16)$$

- In Phase 2 (Lines 12–26), scan all processes, following a reverse topological order:
  - For any process with out-degree equal to 0, set the effective service curve bounds as $(\gamma_i^u, \gamma_i^l) = (\beta_i^u, \beta_i^l)$.
  - For any other process, define

$$T_i^* = \{j \in T_i^> | a_i'^l = a_j^l\}; \quad (17)$$

$$\widehat{\alpha}_j^l = \bigoplus_{k \in S_j \setminus \{i\}} \alpha_k'^l, \quad (18)$$

  and set the effective service curve bounds as

$$\gamma_i^u = \bigoplus_{j \in T_i^-} \left(\gamma_j^l + B_{i,j}\right) \oplus \bigoplus_{j \in T_i^*} \left(\widehat{\alpha}_j^l \oplus \gamma_j^l + B_{i,j}\right) \oplus \bigoplus_{j \in T_i^> \setminus T_i^*} \left(\alpha_j^l \oplus \gamma_j^l + B_{i,j}\right) \oplus \beta_i^u; \quad (19)$$

$$\gamma_i^l = \bigoplus_{j \in T_i^-} \left(\gamma_j^l + B_{i,j}\right) \oplus \bigoplus_{j \in T_i^*} \left(\widehat{\alpha}_j^l \oplus \gamma_j^l + B_{i,j}\right) \oplus \bigoplus_{j \in T_i^> \setminus T_i^*} \left(\alpha_j^l \oplus \gamma_j^l + B_{i,j}\right) \oplus \beta_i^l. \quad (20)$$

- After Phase 2, update all service curves
  - For any process, $(\beta_i^u, \beta_i^l) = (\gamma_i^u, \gamma_i^l)$.

  and iterate the two phases until no buffers overflow.

During the computation of the propagation effects, there can be three cases. An example of the first case (Lines 17–19) is shown in Figure 3 (c), where the algorithm considers the input arrival curve of P_5 (since P_5 may be slowed down by both P_2 and P_3) and then propagates the effects to both P_2 and P_3. An example of the second case (Lines 20–23) is also shown in Figure 3 (c) if $a_2'^l = a_2'^l \oplus a_3'^l$. where the algorithm does not need to consider the output arrival curve of P_2 (since P_2 does not need to wait itself). An example of the third case (Lines 24–26) is shown in Figure 3 (d), where the algorithm does not need to consider the input arrival curve of P_5 (since P_2 only needs to know the lower service curve of P_5), and the algorithm can directly propagate the effects to P_2.

D. Feasibility and Convergence

We have the following three lemmas for the three cases:

Lemma 3: The buffer between P_i and P_j does not overflow if $j \in T_i^-$ and

$$\sup_t(\alpha_i'^u(t) - \beta_j^l(t)) \le B_{i,j}. \quad (21)$$

Proof: Since $j \in T_i^-$, P_j is non-merging with only one input buffer, and $\alpha_j^u = \alpha_i'^u$. By Theorem 2, the maximal backlog is $\sup_t(\alpha_j^u(t) - \beta_j^l(t)) = \sup_t(\alpha_i'^u(t) - \beta_j^l(t))$. Hence, the buffer does not overflow if $\sup_t(\alpha_i'^u(t) - \beta_j^l(t)) \le B_{i,j}$. ■

Lemma 4: The buffer between P_i and P_j does not overflow if $j \in T_i^*$ and

$$\sup_t\left(\alpha_i'^u(t) - \widehat{\alpha}_j^l(t) \oplus \beta_j^l(t)\right) \le B_{i,j}. \quad (22)$$

Proof: The process can at least provide $\widehat{\alpha}_j^l \oplus \beta_j^l$ service, and the following is similar to Lemma 3. ■

Lemma 5: The buffer between P_i and P_j does not overflow if $j \in T_i^>$ and

$$\sup_t\left(\alpha_i'^u(t) - \alpha_j^l(t) \oplus \beta_j^l(t)\right) \le B_{i,j}. \quad (23)$$

Proof: The process can at least provide $\alpha_j^l \oplus \beta_j^l$ service, and the following is similar to Lemma 3. ■

Note that $\alpha_j^l(t) \oplus \beta_j^l(t) \le \widehat{\alpha}_j^l \oplus \beta_j^l \le \beta_j^l(t)$, hence Equation (23) is stricter than Equation (22), and Equation (22) is stricter than Equation (21). This also means that a stricter condition is required for a merging process than a non-merging process.

We now demonstrate the correctness of the Fork-Merge-Extension algorithm (always computes a correct solution). We first introduce a lemma:

Lemma 6: For any process, $\alpha_i'^u \le \beta_i^u$.

Proof: $\alpha_i'^u = ((\alpha_i^u \otimes \beta_i^u) \oslash \beta_i^l) \oplus \beta_i^u \Longrightarrow \alpha_i'^u \le \beta_i^u$. ■

Theorem 7: The Fork-Merge-Extension algorithm returns a feasible solution which guarantees the buffers do not overflow.

Proof: If the service curves of all processes are not changed (Lines 26–27), *i.e.*, $\forall i$, $(\beta_i^u, \beta_i^l) = (\gamma_i^u, \gamma_i^l)$, then, $\forall i$,

$$\beta_i^u = \bigoplus_{j \in T_i^-} \left(\beta_j^l + B_{i,j}\right) \oplus \bigoplus_{j \in T_i^*} \left(\widehat{\alpha}_j^l \oplus \beta_j^l + B_{i,j}\right) \oplus \bigoplus_{j \in T_i^> \setminus T_i^*} \left(\alpha_j^l \oplus \beta_j^l + B_{i,j}\right) \oplus \beta_i^u,$$

which implies

$$\begin{cases} \beta_i^u \le (\beta_j^l + B_{i,j}), & \forall j \in T_i^-; \\ \beta_i^u \le (\widehat{\alpha}_j^l \oplus \beta_j^l + B_{i,j}), & \forall j \in T_i^*; \\ \beta_i^u \le (\alpha_j^l \oplus \beta_j^l + B_{i,j}), & \forall j \in T_i^>, \end{cases} \quad (24)$$

and

$$\begin{cases} \sup(\alpha_i'^u - \beta_j^l) \le \sup(\beta_i^u - \beta_j^l) \\ \le \sup(\beta_j^l + B_{i,j} - \beta_j^l) = B_{i,j}, & \forall j \in T_i^-; \\ \hline \sup(\alpha_i'^u - \widehat{\alpha}_j^l \oplus \beta_j^l) \le \sup(\beta_i^u - \widehat{\alpha}_j^l \oplus \beta_j^l) \\ \le \sup(\widehat{\alpha}_j^l \oplus \beta_j^l + B_{i,j} - \widehat{\alpha}_j^l \oplus \beta_j^l) = B_{i,j}, & \forall j \in T_i^*; \\ \hline \sup(\alpha_i'^u - \alpha_j^l \oplus \beta_j^l) \le \sup(\beta_i^u - \alpha_j^l \oplus \beta_j^l) \\ \le \sup(\alpha_j^l \oplus \beta_j^l + B_{i,j} - \alpha_j^l \oplus \beta_j^l) = B_{i,j}, & \forall j \in T_i^>. \end{cases}$$

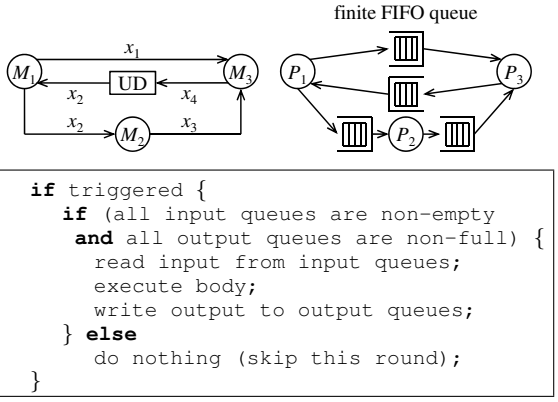
The first "$\le$" for each case is implied by Lemma 6, and the second "$\le$" for each case is implied by Equation (24). Then, by Lemmas 3, 4, and 5, all buffers do not overflow. This is a sufficient but not necessary condition for the algorithm to terminate, since we have another terminating check (Lines 13–14), and it also guarantees that the buffers do not overflow. ■

Theorem 8: If all curves can be defined by *finite number of segments and periods*, the Fork-Merge-Extension algorithm converges (terminates) in a finite number of iterations, and all buffers do not overflow or underflow.

Proof: Due to space limit, the detailed proof of convergence in a finite number of iterations is not provided here. By Theorem 7, the algorithm keeps updating service curves until it computes a feasible solution. Service curves are decreasing functions lower bounded by zero. At each update step a new service curve is produced, which differs from the previous by at least one value at one time point that is lower than the previous one by a non-infinitesimal amount. Hence, the number of possible updates is upper bound and the algorithm converges (terminates) in a finite number of iterations. ■

V. RTC Analysis of LTTA

This section describes the application of our analysis method to the mapping of synchronous models into LTTA. We first summarize LTTA and their existing performance analysis methods [14].




```
1   if triggered {
2     if (all input queues are non-empty
       and all output queues are non-full) {
3       read input from input queues;
4       execute body;
5       write output to output queues;
6     } else
7       do nothing (skip this round);
8   }
```

Fig. 5. From a synchronous network to an FFP and FFP process behavior.

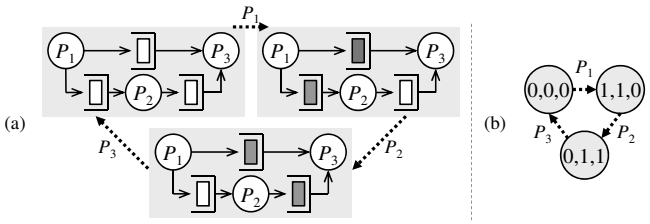


Fig. 6. (a) The example system where all buffer sizes are 1 and (b) its corresponding reachability graph.

A. Implementing SR on LTTA

In [14], it is demonstrated how a generic network of communicating synchronous state machines can be implemented by a network of processes communicating through bounded FIFO queues *with assumptions on the activation times of the processes*. The target model of the mapping is called FFP (Finite FIFO Platform) and consists of a network of processes implementing the state machines and using blocking read and write access to the queues (Figure 5). The implementation is correct with respect to the preservation of the communication flows (the signal values). LTTA processes are triggered by their clock, but only executed if all their input buffers are non-empty, and their output buffers are non-full; otherwise, they skip (in our model they are stalled, which is functionally equivalent). In the LTTA network model of [14], activation clocks are required to have a lower bound for the time interval between any two ticks.

B. Performance Analysis of LTTA

We discuss the performance analysis in [14] and compare it with a more recent work about LTTA [2].

- First, a logical clock is defined such that there is at least one activation event for any FFP process between any two logical clock ticks (it provides a lower bound on the rate of progression of each process).
- Then, a reachability graph is computed, indicating the state of the buffer queues after the firing of the processes.
- Finally, the computation of the worst-case triggering order of the FFP processes is determined.

Figure 6 (a) shows an example system, and Figure 6 (b) is its reachability graph. If the three buffers are initially empty, and the triggering order is $< P_1, P_2, P_3 >$ in one clock cycle, the system can complete all of the three steps in Figure 6 (a). On the contrary, if the triggering order is $< P_3, P_2, P_1 >$ in one clock cycle, P_3 and P_2 will be skipped due to their empty input buffers, and the system can only complete one step. The analysis in [14] has two sources of pessimism.

The first is the use of a logical clock lower bounding the rates of execution of the clocks of all processes. The second pessimism is in the slow triggering policy which assumes that, at each logical-clock tick, the triggering order of processes is the worst-case (resulting in the slowest progression of the network computations). In the example, for the first tick, the triggering order is $< P_2, P_3, P_1 >$ or $< P_3, P_2, P_1 >$, so that only P_1 can be executed; at the second tick, the triggering order is $< P_1, P_3, P_2 >$ or $< P_3, P_1, P_2 >$, and only P_2 can be executed; in the third tick, the triggering order is $< P_1, P_2, P_3 >$ or $< P_2, P_1, P_3 >$, and only P_3 can be executed. As a result, it takes three logical clock ticks to complete the network flow (the worst-case logical-time throughput is $\frac{1}{3}$). In general (and especially when the processes are activated according to a periodic model with jitter), the triggering order cannot be assumed as arbitrary at each tick, *i.e.*, *there are dependencies between activations, but these dependencies are ignored by the slow triggering policy.*

Besides the problem of pessimism, another problem is that the computation of the reachability graph and the slow triggering policy may require the evaluation of an exponential numbers of vertices in the graph and possible orders. In [2] two variants of LTTA, token-based (or back-pressure) LTTA and time-based LTTA, are introduced. The time-based LTTA is not discussed here because it assumes preliminary knowledge of task and message response times, causing a chicken-and-egg problem. For a token-based LTTA, the worst-case throughput is $4T_{\max} + 2\tau_{\max}$, where $T_{\max}$ is the upper bound for the time interval between any two successive ticks, and $\tau_{\max}$ is the upper bound of communication delays. In the analysis, a maximum interarrival $T_{\max}$ is used to bound the distance between any two clock ticks and the throughput is computed by assuming that every time a token reaches a place it just misses the tick of the destination process (both assumptions are sources of pessimism). As a result, the long-term throughput is underestimated.

C. LTTA Model

Compared with the clock model in [14], we assume a more realistic and less pessimistic activation model of periodic clocks with drift and jitter, but the RTC analysis can deal with a general activation pattern. A periodic request with clock period Θ can be modeled by the upper and lower stair functions in Figure 7 (a). A clock drift bounded in the range δ_m, δ_M can be represented by the upper and lower curves in Figure 7 (b), and a jitter J can be modeled by the upper and lower curves in Figure 7 (c). By the definition of the RTC, an arrival event is like a token, *i.e.*, if it is not served, then it will stay in a queue; on the contrary, an unused service will be lost. Therefore, the periodic activation of a process is modeled as an arrival curve curve in Figure 7 (d) along with other

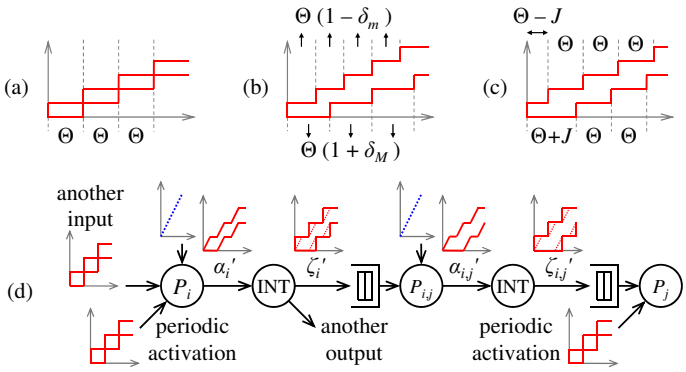


Fig. 7. (a) Upper and lower curves with period Θ, with (b) clock drifts bounded by δ_m, δ_M and (c) jitter J. (d) The model for a process and one of its output edge (if communication is considered).

arrival curves. The service curve used to model the resource availability is the linear function in Figure 7 (d) where the slope represents the execution rate of the processor. Another change for an LTTA process is the need of ceiling and the floor functions to reshape the upper and lower bounds of the output arrival curve from α_i' to ζ_i', *i.e.*, $\zeta_i'^u = \lceil \alpha_i'^u \rceil$ to $\zeta_i'^l = \lfloor \alpha_i'^l \rfloor$, because an LTTA process writes to its output buffers only after it finishes computing the new state and output values.

If communication delay is considered, for each edge between processes P_i and P_j, we add another network process $P_{i,j}$ to represent the execution of the communication. The model is shown in Figure 7 (d), and the service curve of $P_{i,j}$ is also a linear function whose slope represents the transmission rate.

D. Improved Performance Analysis of LTTA

To apply the analysis in Section IV to LTTA systems, further developments are required:

- As shown in Figure 7 (d), the arrival curve representing the periodic activation of P_i is α_i^*. To match the problem statement in Section IV, for each process P_i (not including communication processes $P_{i,j}$), we add another process P_i^*, with input arrival curve α_i^*, and service curve f_0 (the unitary element). We then remove α_i^* from P_i and add a connection from P_i^* to P_i with buffer size ∞. As a result, there is always an arrival curve α_i^* from P_{n+i}, which is not changed by any effect propagation (of course, P_i always becomes a merge case).
- As shown in Figure 7 (d), the effective output of a process is ζ_i'. Therefore, we first replace Equations 15 and 16 (Lines 7–8 in Figure 4) by

$$\zeta_i'^u = \lceil ((\alpha_i^u \otimes \beta_i^u) \oslash \beta_i^l) \oplus \beta_i^u \rceil; \quad (25)$$
$$\zeta_i'^l = \lfloor (\alpha_i^l \otimes \beta_i^l) \oplus \beta_i^l \rfloor, \quad (26)$$

and then replace $\alpha_i'^u$ and $\alpha_i'^l$ by $\zeta_i'^u$ and $\zeta_i'^l$, respectively, in Equation (14) (Lines 5–6 in Figure 4), Equation (17) (Lines 17 and 20 in Figure 4), and Equation (18) (Line 21 in Figure 4). Besides, when we check if a buffer overflowed by Lemmas 3, Lemmas 4 and 5, we also replace $\alpha_i'^u$ by $\zeta_i'^u$.

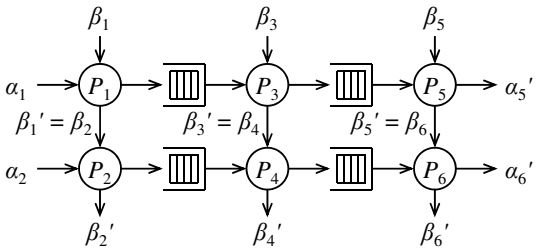


Fig. 8. An example with remaining service curves.

The only thing we need to take care of is Theorem 7, because Lemma 6 only makes sure $\alpha_i'^u \le \beta_i^u$, not $\zeta_i'^u \le \beta_i^u$ ($\alpha_j^u = \zeta_i'^u$ in Equation (7) now). However, in the following lemma, we fix this problem by proving that $\sup_t(\alpha_i'^u(t) - \beta_j^l(t)) \le B_{i,j}$ can still guarantee that the buffer between P_i and P_j does not overflow. The lemma applies to the first case in Theorem 7, but the other two cases can be proved similarly.

***Lemma** 7:* The condition $\sup_t(\alpha_i'^u(t) - \beta_j^l(t)) \le B_{i,j}$ still guarantees that the buffer between P_i and P_j does not overflow when output arrival curves are modified by applying the ceiling function to the upper bound and the floor to the lower bound.

Proof:

$$\begin{aligned}
& \sup_t(\alpha_i'^u(t) - \beta_j^l(t)) \le B_{i,j} \\
\Rightarrow\ & \forall t, \alpha_i'^u(t) - \beta_j^l(t) \le B_{i,j} \\
\Rightarrow\ & \forall t, \lceil \alpha_i'^u(t) \rceil - \lceil \beta_j^l(t) \rceil \le B_{i,j} \ (B_{i,j} \text{ is an integer}) \\
\Rightarrow\ & \sup_t(\lceil \alpha_i'^u(t) \rceil - \lceil \beta_j^l(t)) \rceil) \le B_{i,j} \\
\Rightarrow\ & \sup_t(\zeta_i'^u(t) - \lceil \beta_j^l(t)) \rceil) \le B_{i,j}.
\end{aligned}$$

When P_j follows the execution body of an LTTA process and starts its computation, the corresponding token (the message) has left the buffer, so, from the view of the buffer, the lower bound of the service provided by P_j is $\lceil \beta_j^l(t)) \rceil$. Since $\zeta_i'^u$ is the arrival curve of the buffer, by Equation (7), the lemma is proved. ■

E. Remaining Service Curve

Given a system with its (input) arrival curves α^u and α^l and (input) service curves β^u and β^l, the remaining service curves can be computed by Equations (5) and (6) [6]. As an example in Figure 8 (the integer operation and communication delay are neglected), if P_1 and P_3 are slowed down due to the effect from P_5, it is possible that P_2 and P_4 execute more often because the remaining service curves of P_1 and P_3 are larger. Therefore, a service curve may now increase, and convergence cannot be guaranteed. To deal with this problem, we observe that, in an LTTA case, the output arrival curve is usually dominated by its arrival curves (probably the activation one) or the bounds provided by feedback control edges. Therefore, if an input service curve increases, its output arrival curve usually does not increase. Based on the observation, in Phase 1 of the first iteration, we compute the remaining service curves, assign them as the input service curves of the following processes, and remove the corresponding edges. After that, we follow the Fork-Merge-Extension algorithm (Figure 4) without considering the remaining service curves. Although the approach is pessimistic (we do not increase a service curve which could be larger), it guarantees convergence as the service curve never becomes larger.

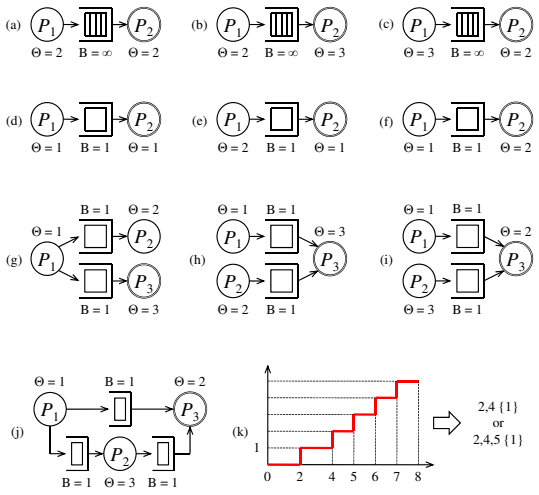


Fig. 9. (a) Case A1, (b) Case A2, (c) Case A3, (d) Case B1, (e) Case B2, (f) Case B3, (g) Case B4, (h) Case B5, (i) Case B6, (j) Case B7, and (k) the curve can be represented as "2,4 {1}" or "2,4,5 {1}".

VI. Experimental Results

We use several simple graph configurations to compare the different methods as shown in Figure 9. For each case, one process (marked with a double line) is selected as representative to demonstrate the differences in performance among the analysis methods (all processes show similar characteristics). Θ represents the period of a process, and B represents the buffer size. The service curves of a process are stair functions $(\beta^u(t), \beta^l(t)) = (\lceil \frac{t}{\Theta} \rceil, \lfloor \frac{t}{\Theta} \rfloor)$, and the input arrival curves of an initial process are $(\alpha^u(t), \alpha^l(t)) = (\infty, \infty)$, which means that the output arrival curves of the root process are equal to its service curve. This choice of functions allows to better match the model with execution skips in [14]. We used the RTC Toolbox [16] and implemented our algorithm for computing the effective arrival and service curves in MATLAB. We also implemented the analysis methods in [6], [13], and [4] for comparison. In the following paragraphs, a sequence of numbers which is almost linear periodic (in the definitions of min-plus algebra) represents the output arrival curve of a target processes. The first part of the sequence, outside of the braces, represents the arrival times of initial events; the number inside braces represents the steady-state period of the events (or linear period in min-plus algebra). For example, the function shown in Figure 9 (k) can be represented as "2,4 {1}" or "2,4,5 {1}". If the function is the lower bound of an output arrival curve, the first number represents the longest delay, and the number in the braces represents the long-term processing period.

RTC Analysis with Reduced Pessimism

The results of our fixes and improvements to the existing RTC analysis are listed below. For Case A2, the lower bound of [6] is larger than the optimal lower bound, meaning that the analysis is infeasible and not safe. The other approaches are feasible, but [14] is very pessimistic, and so is [13] in the A3 case.

Test Cases	Upper Bound				
	[6]	[14]	[13]	[4]	Ours/**Optimal**
A1	0,2,4 {2}	—	0,2,4 {2}	—	0,2,4 {2}
A2	0,3,6 {3}	—	0,3,6 {3}	—	0,3,6 {3}
A3	0,2,4 {3}	—	0,3,6 {3}	—	0,2,4 {3}

Test Cases	Lower Bound		
	[6]	[14]	[13], [4], Ours, **Optimal**
A1	4,6,8 {2}	4,6,8 {2}	4,6,8 {2}
A2	3,6,9 {3}	6,9,12 {3}	5,8,11 {3}
A3	5,8,11 {3}	6,9,12 {3}	5,8,11 {3}

Effect Propagation

The results of the effect propagation are shown in the below table, where [6] is not considered because its analysis may be infeasible. Our approach computes exactly the optimal best/worst performance. All other approaches, especially [14], are pessimistic in at least some of the cases, including the evaluation of the long-term processing rate.

Test Cases	Upper Bound			
	[14]	[13]	[4]	Ours/**Optimal**
B1	—	0,1,2 {1}	—	0,1,2 {1}
B2	—	0,2,4 {2}	—	0,1,3 {2}
B3	—	0,2,4 {2}	—	0,2,4 {2}

Test Cases	Lower Bound			
	[14]	[13]	[4]	Ours/**Optimal**
B1	2,4,6 {2}	2,4,6 {2}	3,4,5 {1}	2,3,4 {1}
B2	4,8,12 {4}	3,6,9 {3}	5,7,9 {2}	3,5,7 {2}
B3	4,8,12 {4}	3,6,9 {3}	4,6,8 {2}	3,5,7 {2}

Fork and Merge Topologies

The results of the analysis of fork and merge topologies are listed below. [4] is not considered because it is not applicable to fork and merge topologies (the method in [4] was the only one to compute the steady-state period as accurately as ours). In all the cases we tried, our approach performed better than the other competing analysis methods even if it did not always compute the optimal performance bounds.

By comparison, we also tried a linear service curve $(\beta_i^u(t), \beta_i^l(t)) = (it, it)$ and stair arrival curve. In this case, the methods (in [13] and [6]) that support the analysis compute the same results as our method in terms of processing rates, but longer delays (slope of the output arrival and service curves).

Test Cases	Upper Bound			
	[14]	[13]	Ours	**Optimal**
B4	—	0,3,6 {3}	0,3,6 {3}	0,3,6 {3}
B5	—	0,3,6 {3}	0,3,6 {3}	0,3,6 {3}
B6	—	0,3,6 {3}	0,2,4,7 {3}	0,2,4,8,10,14,...
B7	—	0,3,6 {3}	0,2,4,7 {3}	0,2,4,8,10,14,...

Test Cases	Lower Bound			
	[14]	[13]	Ours	**Optimal**
B4	6,12,18 {6}	4,8,12 {4}	4,7,10 {3}	4,7,10 {3}
B5	6,12,18 {6}	5,10,15 {5}	5,8,11 {3}	5,8,11 {3}
B6	6,12,18 {6}	5,10,15 {5}	5,8,11 {3}	5,8,11 {3}
B7	9,18,27 {9}	6,12,18 {6}	6,9,12 {3}	6,9,12 {3}

Analysis of an Automotive System

We used a subset of the functions of a comprehensive safety vehicle as a case study. The architecture consists of 29 ECUs (Electronic Control Units) connected with 4 CAN (Controller Area Network) buses, with speeds ranging from 25kb/s to 500kb/s. The vehicle supports advanced distributed functions collecting data from 360-degree sensors to the actuators, consisting of the throttle, brake and steering subsystems and

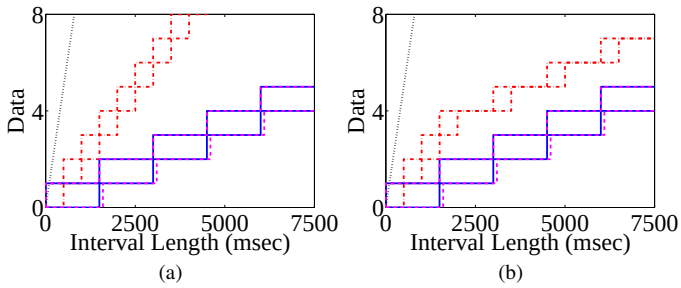


Fig. 10. Case study: (a) without buffer limitation (b) with buffer limitation.

of Human-Machine Interface (HMI) devices. A total of 92 tasks are executed on the ECU nodes, and 196 messages are exchanged over the four buses. Worst-case execution time estimates have been obtained for all tasks. Message length and bus speed is used to calculate the maximum transmission times. Each ECU executes between 1 and 22 tasks and each CAN bus transmits between 14 and 105 messages. The system graph contains a total of 604 links.

The experiments are done with two different settings: (1) with unbounded buffers and (2) with buffer constraints. Both of them use the LTTA model (stair arrival curves for activations and linear service curves) and consider integer operations and communication delay mentioned in Section V. Figure 10 shows one of the links in the example, with typical results (dash-dotted red line, the input arrival; solid blue line, the activation; dotted black line, the service; dashed magenta line, the output arrival). After considering the buffer limitation, since processes are slowed down to guarantee that buffers never overflow, we can see that the input arrival in Figure 10 (b) is smaller than that in Figure 10 (a). The long-term processing period of the lower (worst-case) output arrival curve of a process always matches that of the slowest process which has a path to the process. There is also a delay for the lower (worst-case) output arrival curve, which is generated by the processing delay (integer operations) and the communication delay. In the example in Figure 10, the long-term processing period is 1,500 msec, and the delay is 125 msec. The runtimes on these two settings are 12 and 121 seconds, respectively, on an Xeon E5630 2.4G machine. The algorithm converges in 1 iteration for the first setting because there is no buffer limitation. On the other hand, it converges in 4 iterations for the second setting. We observe that it takes less time in the first iteration and more time in the following iterations, which is because the updates make the aperiodic parts of service curves having more line segments in the data structure of the RTC Toolbox [16]. These results show that our algorithm converges in a reasonable number of iterations in practice, with the approximations discussed in the previous section that may introduce pessimism in the output service curves.

VII. Conclusion and Future Work

We extended the Real-Time Calculus (RTC) as a general model for the analysis of task graphs and the mapping of synchronous models on the Loosely Time Triggered Architectures (LTTA). Several fixes were also required to the original RTC theory. Experimental results have shown that our approach can indeed provide a performance analysis that is safe and more accurate than existing methods. Possible future directions are to consider cyclic topologies and to find an algorithm for updating service curves that can guarantee convergence without conservative assumptions.

Acknowledgement

This work was supported by the Multiscale Systems Center (MuSyC), the Industrial Cyber-Physical Systems Center (iCyPhy), ARO Grant No. W911NF1110403, and NSF Grant No's. CPS 1135630, CNS 1117185.

References

[1] A. Benveniste, P. Caspi, P. L. Guernic, H. Marchand, J.-P. Talpin, and S. Tripakis, "A protocol for loosely time-triggered architectures," *Proc. ACM International Conference on Embedded Software*, pp. 252–265, 2002.

[2] A. Benveniste, A. Bouillard, and P. Caspi, "A unifying view of loosely time-triggered architectures," *Proc. ACM International Conference on Embedded Software*, pp. 189–198, 2010.

[3] J.-Y. Le Boudec and P. Thiran, "Network calculus: a theory of deterministic queuing systems for the internet," Springer-Verlag, 2001.

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

[5] P. Caspi, A. Curic, A. Maignan, C. Sofronis, S. Tripakis, and P. Niebert, "From Simulink to SCADE/Lustre to TTA: a layered approach for distributed embedded applications," *Proc. ACM SIGPLAN Conference on Language, Compiler, and Tool for Embedded Systems*, pp. 153–162, 2003.

[6] S. Chakraborty, S. Kunzli, and L. Thiele, "A general framework for analysing system properties in platform-based embedded system designs," *Proc. IEEE/ACM Conference on Design, Automation and Test in Europe*, pp. 190–195, 2003.

[7] C.-S Chang, "Performance guarantees in communication networks," Springer-Verlag, 2000.

[8] H. Kopetz, "Real-time systems: design principles for distributed embedded applications," Kluwer Academic, 1997.

[9] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Comm. ACM*, vol. 21, no. 7, pp. 558–565, 1978.

[10] N. Lynch, "Distributed algorithms," Morgan Kaufmann, 1996.

[11] A. Sangiovanni-Vincentelli, "Quo vadis, SLD? Reasoning about the trends and challenges of system level design," *Proc. IEEE*, vol. 95, no. 3, pp. 467–506, 2007.

[12] L. Thiele, S. Chakraborty, and M. Naedele, "Real-time calculus for scheduling hard real-time systems," *Proc. IEEE International Symposium on Circuits and Systems*, pp. 101–104, 2000.

[13] L. Thiele and N. Stoimenov, "Modular performance analysis of cyclic dataflow graphs," *Proc. ACM International Conference on Embedded Software*, pp. 127–136, 2009.

[14] S. Tripakis, C. Pinello, A. Benveniste, A. Sangiovanni-Vincentelli, P. Caspi, and M. Di Natale, "Implementing synchronous models on loosely time-triggered architectures," *IEEE Trans. on Computers*, vol. 57, no. 10, pp. 1300–1314, 2008.

[15] E. Wandeler, "Modular performance analysis and interface-based design for embedded real-time systems," Ph.D. Dissertation, ETH Zurich, 2006.

[16] E. Wandeler and L. Thiele, Real-Time Calculus (RTC) Toolbox, `http://www.mpa.ethz.ch/Rtctoolbox`, 2006.

[17] J. Wu, J.-C. Liu, and W. Zhao, "A general framework for parameterized schedulability bound analysis of real-time systems," *IEEE Trans. on Computers*, vol. 59, no. 6, pp. 776–783, 2010.