Modeling Buffers with Data Refresh Semantics in Automotive Architectures*

Linh T.X. Phan[1] Reinhard Schneider[2] Samarjit Chakraborty[2] Insup Lee[1]
[1]Department of Computer and Information Science, University of Pennsylvania, USA
[2]Institute for Real-Time Computer Systems, TU Munich, Germany
E-mail: {linhphan,lee}@cis.upenn.edu, reinhard.schneider@rcs.ei.tum.de, samarjit@tum.de

ABSTRACT

Automotive architectures consist of multiple electronic control units (ECUs) which run distributed control applications. Such ECUs are connected to sensors and actuators and communicate via shared buses. Resource arbitration at the ECUs and also in the communication medium, coupled with variabilities in execution requirements of tasks results in jitter in the signal/data streams existing in the system. As a result, buffers are required at the ECUs and bus controllers. However, these buffers often implement different semantics – FIFO queuing, which is the most straightforward buffering scheme, and data refreshing, where stale data is overwritten by freshly sampled data. Traditional timing and schedulability analysis that are used to compute, e.g., end-to-end delays, in such automotive architectures can only model FIFO buffering. As a result, they return pessimistic delay and resource estimates because in reality overwritten data items do not get processed by the system. In this paper we propose an analytical framework for accurately modeling such data refresh semantics. Our model exploits a novel feedback control mechanism and is purely functional in nature. As a result, it is scalable and does not involve any explicit state modeling. Using this model we can estimate various timing and performance metrics for automotive ECU networks consisting of buffers implementing different data handling semantics. We illustrate the utility of this model through three case studies from the automotive electronics domain.

1. INTRODUCTION

Automotive architectures typically consist of a collection of electronic control units (ECUs) that are connected by multiple communication buses implementing various protocols such as CAN and FlexRay. Such a platform is used to execute multiple distributed control applications that obtain their input from various sensors. Hence, there is information processing and propagation in the form of data/message streams, originating from sensors and terminating at actuators and passing through various ECUs and buses. Although the sensors typically sample data at periodic intervals and tasks are also periodically activated, resource arbitration at the ECUs and buses, coupled with variable processing demands of tasks, introduce jitters in the data streams. As a result, buffers need to be placed at various positions in the architecture, e.g., at the bus controllers.

Given such a setup, there are two predominant data handling semantics implemented in the buffers: (i) First In First Out or FIFO, where data should not be lost and the history of transmission or ordering of data is important. Data streams containing *incremental information*, e.g., the *speed increase* of a car, are buffered in this manner. (ii) Data refresh semantics, where buffers are of restricted size and stale data is overwritten by freshly sampled sensor data. Such semantics is used where data with the most *recent values* are of interest, e.g., the *actual speed* of a car. However, modeling such buffer overwrites – while doing timing/performance analysis of the system – turns out to be a challenging problem and has mostly been ignored in the past. Ignoring such overwrites is acceptable in streaming multimedia applications, e.g., where encoded video data is stored in a buffer and existing data is *never* overwritten (in fact overwriting or data loss is not desirable). However, for many control applications, stale data is not useful and is replaced by newly sampled data. Neglecting overwrites in this case leads to pessimistic estimates on computation and communication resource requirements (since the overwritten data is also assumed to be processed in the model).

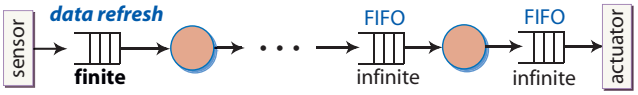

Figure 1: Network having buffers with different semantics.

In this paper we develop an analytical model for ECU networks containing buffers implementing both the above semantics (e.g., see Fig. 1). Our model is motivated by previous work on modeling and analysis of applications processing continuous data streams using the *Real-Time Calculus* (RTC) framework [16]. We extend the RTC framework using a novel feedback control mechanism that enables the modeling of complex "state information", which in this case is the deletion of existing data in the buffer by fresh data. The main challenge in modeling such overwrites stems from the relatively complex overwriting process, viz., the *oldest* data in the buffer is overwritten by freshly sampled data. Typically such data refresh semantics requires explicit state-based modeling, e.g., using timed automata (see [8]). This leads to the well-known state space explosion problem, which has also been reported in the context of state-based modeling of applications processing data streams [6]. Our main contribution is to suitably modify the RTC framework with an appropriate abstraction and a feedback control construct, that avoids any state-based modeling, but nevertheless accurately captures the complex buffer refresh semantics.

The importance of this problem has recently been pointed out in a number of studies. In [11] buffering mechanisms with overwriting have been discussed to bridge the gap between synchronous semantics at the model level and the asynchronous nature of implementation platforms. The semantics of *tag systems* [1] has been used to model systems with buffers implementing the data refresh semantics as we do in this paper [2]. This has been extended to model loosely time-triggered architectures in [3]. However, all of these efforts were directed towards studying the *functional* correct-

*This research was supported in part by NSF CNS-0931239, NSF CNS-0834524, NSF CNS-0721541, NSF CNS-0720703 and the DFG (Germany) through the SFB/TR28 Cognitive Automobiles.

ness of the system. In this paper, we model the data refresh semantics to *quantitatively* capture the load on system resources and the volume of actual data that is processed by the system. Our goal is to factor this into the computation of timing properties of the system, e.g., delays suffered by messages. The need for quantitatively capturing this in performance analysis techniques has also been pointed out in [7].

Developed on top of the Network Calculus [4] theory from the communication networks domain, the RTC framework [5] we use in this paper has been extensively adapted to model and analyze heterogeneous real-time systems in a compositional manner (e.g., see [17]). The central concept in this framework is its use of *arrival functions* to model the timing properties of data streams and *service functions* to capture the availability of resources. Specifically, each data stream is modeled by a pair of arrival functions, $\alpha^u(\Delta)$ and $\alpha^l(\Delta)$, which denote the upper- and lower-bound on the number of data items that may arrive in any time interval of length Δ. Similarly, a resource is modeled by a pair of service functions, $\beta^u(\Delta)$ and $\beta^l(\Delta)$, which specify the maximum and minimum number of items that can be processed by the resource within any time interval of length Δ. Given the arrival functions of an input data stream and the service functions of a resource, one can compute – using purely algebraic techniques – various bounds on system properties such as the maximum backlog of data items at a buffer, the maximum delay suffered by the input stream, the arrival functions of the output stream, and the service functions of the remaining resource. The output arrival functions can then be fed as inputs to the next resource whereas the remaining service functions can be used to process the next data stream.

The arrival and service functions in the RTC framework admit a much richer collection of arrival sequences and resource patterns than the classical event and resource models (e.g., periodic, sporadic, bounded delay) do. Its algebraic feature also enables efficient computation of system's performance in a fully compositional manner. However, the standard RTC formalism assumes an unbounded buffer size and does not model buffering schemes that are dependent on the state of the buffer. As mentioned earlier, this assumption is not only unrealistic but also prevents RTC from being applicable to many common practical systems.

When data refresh semantics is implemented in a buffer, the smaller the buffer size, fewer will be the the number of data items to be processed downstream (e.g., on the ECU next to the actuator in Fig. 1). This is because certain data items will be overwritten and will therefore not have to be processed subsequently (i.e., beyond the buffer which implements data refresh in Fig. 1). In such cases, assuming that no data is lost – as in the standard RTC framework – results in a higher system load and hence pessimistic timing bounds and resource estimates.

Our contributions. In this paper, we extend the existing RTC framework to model and analyze systems with buffers implementing both FIFO as well as the data refresh semantics. The key idea in our technique is to use in combination the concept of a virtual processor to encapsulate the data overwriting scheme and a feedback control mechanism to capture the overflow constraints. Our analysis relies solely on algebraic manipulations and thus can be computed efficiently. The technique we propose here significantly enhances the modeling power of the existing RTC framework while sidestepping the problems associated with other fine-grain state-space models [6, 8]. At the same time, it is modular and fully compositional. Through case studies, we illustrate how our method can be seamlessly integrated into the current RTC framework, and at the same time we show the effects of capturing buffer overwrites on the accuracy of the analysis. We also provide an experimental validation of our analysis method against simulation. It is worth noting, however, that our analytical method is not only faster but also able to provide guaranteed bounds on the system properties, which cannot be achieved using simulation.

Related work. The first line of work targeted towards state-dependent systems comes from the formal methods domain. Timed automata and related automata-theoretic formalisms have been employed to model task scheduling of hard real-time systems [8] as well as systems processing data streams [6]. Although automata-theoretic models are highly expressive, they often suffer from the state explosion problem when applied to realistic settings.

The effect of finite buffer capacities has been studied in the context of data flow graphs [10]. For instance, an algorithm for computing the buffer capacities that satisfy throughput constraints was presented in [18]. Analysis of self-time scheduling for multirate data flow with finite buffer capacities was studied in [12]. Back-pressure was used in [15] as a mechanism to allow a semantics preserving implementation of synchronous models on Loosely Time Triggered Architectures.

Further, as mentioned above, [2, 3, 11] have proposed techniques for modeling systems with data refreshing, although from a functional correctness perspective. The main goals of these frameworks are to investigate communication and clock synchronization protocols that are data semantics preserving in a distributed time triggered platform with asynchronous communication. This is done by means of tag structures [1], which hold information about the freshness levels of the data, and an enforcement of constraints on these tags to ensure correctness of data values. Unlike these techniques, our framework does not deal with the functional aspects of the system and imposes no constraints on the system. Instead, it provides methods to compute timing and workload related performance properties in presence the of data refresh, which was not addressed in [2, 3, 11].

Lastly, various data management mechanisms have also been investigated to handle overflow conditions in bounded buffers. For instance, [14] identifies four different overflow policies – namely *Drop Newest*, *Drop Oldest*, *Drop Random* and *Drop All* – and presents a simulation-based framework for analyzing properties such as the number of dropped data items and the average delay of the processed data. The refresh semantics we consider here is identical to the *Drop Oldest* policy, which is most relevant in automotive architectures that involve transmission of sensor data. We further extend our analysis to other data management mechanisms proposed in [14]. It is worth noting that our method is purely analytical and thus applicable to safety-critical applications (which is not the case with simulation-based approaches such as [14] that fail to provide any guaranteed timing bounds). Our method also works faster compared to simulation, which is time consuming.

Organization of the paper. In the next section we describe the basic concepts of the RTC framework. Section 3 focuses on our analysis technique for the basic data refresh semantics, followed by an extension to other data refresh semantics in Section 4. We present our case studies in Section 5 and conclude in Section 6 by outlining some directions for future work.

2. RTC BACKGROUND

The RTC framework was developed based on (min,+) and (max,+) algebra [4] and models data streams and processing resources using a *count-based abstraction*. Specifically, an arrival pattern of a stream is modeled as a cumulative function $A(t)$ that gives the number of items arriving over the time interval $(0,t]$. The set of all

arrival patterns of a stream is represented by a pair of *arrival functions* $\alpha = (\alpha^u, \alpha^l)$, where $\alpha^u(\Delta)$ and $\alpha^l(\Delta)$ specify the maximum and minimum number of data items that can arrive from this stream over any time interval of length Δ. In other words, for all $A(t)$,

$$\forall \Delta \geq 0,\ \forall t \geq 0:\ \ \alpha^l(\Delta) \leq A(\Delta+t) - A(t) \leq \alpha^u(\Delta).$$

Similarly, a service pattern of a resource is captured by a cumulative function $C(t)$, with $C(t)$ denoting the number of items that can be processed by the resource in $(0,t]$. The set of all service patterns of a resource is modeled by a pair of *service functions* $\beta = (\beta^u, \beta^l)$, where $\beta^u(\Delta)$ and $\beta^l(\Delta)$ give the maximum and minimum number of items that can be processed by the resource over any time interval of length Δ respectively.

Formally, let $\overline{\mathbb{R}} = \mathbb{R} \cup \{+\infty, -\infty\}$ where $\mathbb{R}$ is the set of real numbers. Let $\mathcal{F}$ be the set of monotonic functions, i.e., $\mathcal{F} = \{f : \mathbb{R}^+ \to \overline{\mathbb{R}} \mid \forall s < t,\ 0 \leq f(s) \leq f(t)\}$ where $\mathbb{R}^+$ is the set of non-negative real numbers. The minimum operator in $\mathcal{F}$, denoted by $\oplus$, is defined for all $f, g \in \mathcal{F}$ as usual: $\forall t \in \mathbb{R}^+,\ \big(f \oplus g\big)(t) = \min\big\{f(t), g(t)\big\}$. Similarly, $f \sim g$ iff $f(t) \sim g(t)$ for all $t \in \mathbb{R}^+$, where $\sim \in \{\leq, \geq, =\}$. Further, the *supremum* (sup), if it exists, of a set $S \subseteq \mathcal{F}$ is the smallest $U \in \mathcal{F}$ such that $h \leq U$ for all $h \in S$. Similarly, the *infimum* (inf) of S is the largest $L \in \mathcal{F}$ such that $h \geq L$ for all $h \in S$. The definition of sup and inf can also be similarly defined over the set $\mathbb{R}$.

We can now define the (min,+) convolution $\otimes$ and deconvolution $\oslash$ operators as follows. For all $f, g \in \mathcal{F}$ and for all $t \in \mathbb{R}^+$,

$$\big(f \otimes g\big)(t) = \inf\big\{f(s) + g(t-s) \mid 0 \leq s \leq t\big\},$$
$$\big(f \oslash g\big)(t) = \sup\big\{f(t+u) - g(u) \mid u \geq 0\big\}.$$

One can verify the following results: (i) $f \otimes g = g \otimes f$, (ii) $f \otimes g \leq f \oplus g$, (iii) $\big(f \otimes g\big) + c = (f + c) \otimes g = f \otimes (g + c)$, and (iv) $f \oslash g \sim h$ iff $f \sim h \otimes g$ where $\sim \in \{\leq, \geq\}$.

Let $\varepsilon \in \mathcal{F}$ be such that $\varepsilon(0) = 0$ and $\varepsilon(t) = +\infty$ for all $t > 0$. The sub-additive closure of f is given by $f^* = \min\big\{f^n \mid n \geq 0\big\}$, where $f^0 = \varepsilon$ and $f^{n+1} = f^n \otimes f$ for all $n \in \mathbb{N},\ n \geq 0$.

THEOREM 2.1 ([4], THEOREM 4.3.1). *For any given $f, g \in \mathcal{F}$, the inequality $h \leq g \oplus f(h)$ has one unique maximal solution, given by $h = f^*(g)$.*

We denote by $\circ$ the composition of two operators: $(\mathcal{O}_1 \circ \mathcal{O}_2)(x) = \mathcal{O}_1(\mathcal{O}_2(x))$. The linear idempotent operator $\mathcal{I}_g$ for any fixed $g \in \mathcal{F}$ is defined by $\mathcal{I}_g(f)(t) = \inf\{g(t) - g(s) + f(s) \mid 0 \leq s \leq t\}$. Then, the following holds [4],

$$(f \oplus \mathcal{I}_g)^* = (\mathcal{I}_g \circ f)^* \circ \mathcal{I}_g. \tag{1}$$

The (max,+) convolution $\overline{\otimes}$ and deconvolution $\overline{\oslash}$ operators are defined as: for all $f, g \in \mathcal{F}$ and for all $t \in \mathbb{R}^+$,

$$\big(f \,\overline{\otimes}\, g\big)(t) = \sup\big\{f(s) + g(t-s) \ \mid\ 0 \leq s \leq t\big\},$$
$$\big(f \,\overline{\oslash}\, g\big)(t) = \inf\big\{f(t+u) - g(u) \ \mid\ u \geq 0\big\}.$$

Next, let $\overline{\varepsilon}(0) = 0$ and $\overline{\varepsilon}(t) = -\infty$ for all $t > 0$. The super-additive closure of f is defined by $f^{\overline{*}} = \max\big\{f^n \mid n \geq 0\big\}$, where $f^0 = \overline{\varepsilon}$ and $f^{n+1} = f^n \,\overline{\otimes}\, f$ for all $n \in \mathbb{N}$, where $\mathbb{N}$ is the set of natural numbers.

In the context of RTC, we often assume that upper arrival (service) functions are sub-additive and lower arrival (service) functions are super-additive. A function $f \in \mathcal{F}$ is sub-additive iff $f(x+y) \leq f(x) + f(y)$ for all x and y in $\mathbb{R}^+$. Similarly, f is super-additive iff $f(x+y) \geq f(x) + f(y)$ for all x and y in $\mathbb{R}^+$. A function can be made sub-additive (super-additive) by taking its sub-additive (super-additive) closure. In this paper, we assume that all given upper (lower) functions are refined to satisfy sub-additivity (super-additivity) before the analysis. Further, we require that each pair of upper and lower functions satisfies causality, i.e., it does not include infeasible bounds. Specifically, for any given pair of upper and lower functions (f^u, f^l), we must have for all $t \geq 0$, $f^l(t) \leq f^l(x) + f^u(t-x) \leq f^u(t)$ for all $0 \leq x \leq t$.

Lastly, the maximum vertical and horizontal deviation (distance) between two functions $f, g \in \mathcal{F}$ are given by:

$$\mathsf{vdist}(f,g) \stackrel{\text{def}}{=} \sup\{\ f(t) - g(t)\ \mid\ t \geq 0\ \} \tag{2}$$
$$\mathsf{hdist}(f,g) \stackrel{\text{def}}{=} \sup\big\{\ \inf\big\{\tau \geq 0 \mid f(t) \leq g(t+\tau)\big\}\ \mid\ t \geq 0\big\} \tag{3}$$

Performance bounds with unbounded FIFO buffers. Consider an input data stream with arrival functions $\alpha = (\alpha^u, \alpha^l)$ that is processed by a resource with service functions $\beta = (\beta^u, \beta^l)$. Suppose the buffer that stores the data items from the input stream has infinite capacity. Let $A(t)$ be an input arrival pattern of the stream and $A'(t)$ be the corresponding output arrival pattern. Then, from [5],

$$A \otimes \beta^l \ \leq\ A' \ \leq\ A \otimes \beta^u \tag{4}$$

The maximum backlog at the input buffer and the maximum delay experienced by the input stream are given by $\mathsf{vdist}(\alpha^u, \beta^l)$ and $\mathsf{hdist}(\alpha^u, \beta^l)$, respectively. Further, the output arrival functions α'_{inf} and remaining service functions β'_{inf} are computed as follows.

$$\alpha^{u'}_{\mathsf{inf}} = \min\big\{\big(\alpha^u \otimes \beta^u\big) \oslash \beta^l,\ \beta^u\big\} \tag{5}$$
$$\alpha^{l'}_{\mathsf{inf}} = \min\big\{\big(\alpha^l \oslash \beta^u\big) \otimes \beta^l,\ \beta^l\big\} \tag{6}$$
$$\beta^{u'}_{\mathsf{inf}} = \big(\beta^u - \alpha^l\big)\ \overline{\oslash}\ 0 \tag{7}$$
$$\beta^{l'}_{\mathsf{inf}} = \big(\beta^l - \alpha^u\big)\ \overline{\otimes}\ 0 \tag{8}$$

Terminology. We refer to the conventional RTC for *unbounded FIFO buffers* described above as RTC-INF and the method proposed in the next section for *bounded buffers with data refresh semantics* as RTC-DRF. The subscript "inf" ("drf") stands for the results computed by the RTC-INF (RTC-DRF) method. Lastly, an arrival pattern of an input/output stream is also known as an input/output function, and we use them interchangeably in this paper.

3. MODELING FINITE BUFFERS WITH DATA REFRESH SEMANTICS

We now extend the RTC-INFresults to capture systems containing buffers that implement data refresh semantics. In such systems, buffers have bounded capacities and incoming data items are stored in the buffer in the order of their arrivals. However, if an incoming item arrives at a buffer when the buffer is full, the oldest data item – at the head of the buffer – is discarded/overwritten, and the fresh data item is written to the end of the buffer.

EXAMPLE 1. *Consider a buffer B of size 3. Given $B = [e_1\, e_2\, e_3]$ when item e_4 arrives, where items e_1, e_2, e_3 arrived earlier in that order. Then, e_1 will be overwritten and B will be $[e_2\ e_3\ e_4]$, which contains the three most recent data items.*

Objectives. Given such a system, our goal is to compute the standard performance-related metrics mentioned earlier. Since the RTC-INF assumes infinite FIFO buffers, its analysis results become overly pessimistic in presence of data refresh. We present here an extension of RTC-INF to model and analyze systems with data refresh semantics, including methods for computing:

- The maximum delay experienced by the input stream, considering only items that are not overwritten[1]. (Section 3.1.1)
- The arrival functions of the output stream. (Section 3.1.2)
- The remaining service functions of the PE after processing the stream. (Section 3.1.3)

[1]Overwritten data items are lost and hence have no notion of delay.

We further extend our method to analyze systems with a mixture of FIFO and data refresh semantics (Section 3.2) and other buffer management schemes (Section 4).

Note that the maximum backlog of the buffer that implements data refresh semantics is either the buffer capacity or the maximum backlog computed by RTC-INF, whichever is smaller.

3.1 Systems with a single input stream

Consider a system consisting of a single input stream that is processed by a processing element (PE) given in Fig. 2. As shown in the figure, upon arriving at the system, the stream is written to a buffer B before being processed by the PE. We assume that (i) the input stream is modeled by the arrival functions $\alpha = (\alpha^u, \alpha^l)$, (ii) the resource availability of the PE is modeled by the service functions $\beta = (\beta^u, \beta^l)$, and (iii) data refresh semantics is implemented at buffer B, which has a finite capacity of $B_{\max}$ (items).

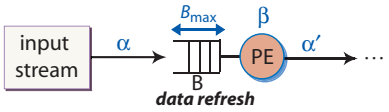

Figure 2: A system with a buffer having data refresh semantics.

Basic modeling ideas. Let A_1 be an arrival pattern of the input stream, C be a service pattern of the PE, and A_3 be the corresponding arrival pattern of the output stream. We denote by A_2 the *effective input function* of A_1, i.e., $A_2(t)$ specifies the number of items of A_1 that arrive in $(0,t]$ and that will not be overwritten. Since all and only the items captured by A_2 will be processed by the PE, A_3 is the actual output function of A_2.

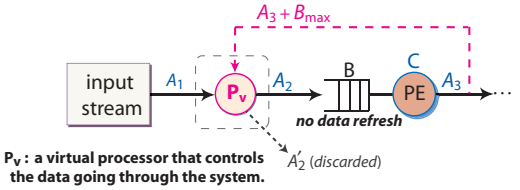

Figure 3: A virtual system equivalent to the one in Fig. 2.

Observe that A_2 is dependent not only on the original arrival pattern A_1 but also on the service pattern C and the size of B. To compute an arrival function that bounds A_2, we employ a feedback control mechanism, where the arrival pattern A_3 of the processed stream is used as feedback information to control the data items going through the system. The original system in Fig. 2 can be recast as an equivalent system that has an additional virtual processor P_v in front of B (see Fig. 3). P_v serves as an admission controller, which splits the original input stream (captured by A_1) into two separate streams:

(i) the former, modeled by the effective input function A_2, consists of all items that will be processed by the PE, and

(ii) the latter, modeled by the input function A'_2, consists of all items that will be overwritten, which will be discarded by P_v.

P_v guarantees that A_2 contains as many items as possible while ensuring that none of these items will be overwritten (i.e., buffer B never overflows). *In essence, the data refresh semantics of the buffer in the original system is now captured completely by the processing semantics of this virtual processor; as a result, the corresponding buffer in the virtual system behaves exactly like an unbounded FIFO buffer.* Note that P_v does not impose any additional delay on the input items as it does not perform any real processing.

EXAMPLE 2. *Fig. 4 shows the effective input function A_2 and the output function A_3 (in solid lines) corresponding to a given input arrival pattern A_1 and a service pattern C, where $B_{\max} = 3$. In the figure, the filled black circles represent the items that go through the system (captured by A_2). The unfilled pink circles represent the items that are discarded by P_v (captured by A'_2). Each blue rectangle corresponds to an item that can be processed (captured by C). The number associated with a blue rectangle denotes the index of the corresponding item in A_1 that is processed by the PE. Note that the second and third rightmost blue rectangles are wasted because there is nothing to process.*

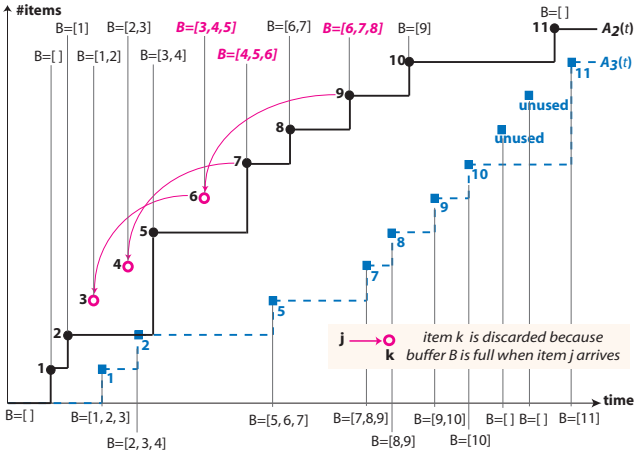

Figure 4: Actual data items that go through the system.

One can verify that, when item 6 arrives, the buffer is full ($B = [3,4,5]$). Therefore, item 3 (the oldest) is overwritten and item 6 is written to the buffer ($B = [4,5,6]$). Similarly, items 4 and 6 will be overwritten when items 7 and 9 arrive, respectively. Thus, in the virtual system, P_v will discard items 3, 4, 6 when they arrive.

The performance-related metrics of the original system can now be analyzed based on this virtual system as outlined in the coming subsections. Fundamentally, the maximum delay is computed based on the conditions for which data refresh occurs. To obtain the remaining service and output arrival functions, we first compute the service function β_v for P_v such that A_2 is the largest effective function possible and B never overflows (cf. Fig. 3). This β_v is then used to derive the arrival function α_v of A_2. From α_v and β, we can apply the RTC-INF to derive the remaining service function of the PE (since B behaves like an infinite FIFO buffer). Further, β_v can also be combined with β to form the overall service function $\widetilde{\beta}$ for the entire system. We then derive the arrival functions of the processed stream based on α and $\widetilde{\beta}$.

3.1.1 Computing maximum delay

Recall the virtual system in Fig. 3. Denote $d(t)$ as the delay experienced by an input item that arrives at time t. Lemma 3.1 states two basic bounds on $d(t)$ due to data refresh. These bounds are shown in Fig. 5.

LEMMA 3.1. *Let $b(t)$ be the number of items in the buffer B at time t, i.e., $b(t) = A_2(t) - A_3(t)$. Then, $d(t) \le \min\{d_1(t), d_2(t)\}$, where*

$$d_1(t) = \min\{\Delta \ge 0 \mid A_1(t+\Delta) - A_1(t) \ge B_{\max}\}, \quad (9)$$

$$d_2(t) = \min\{\Delta \ge 0 \mid C(t+\Delta) - C(t) \ge b(t)\}. \quad (10)$$

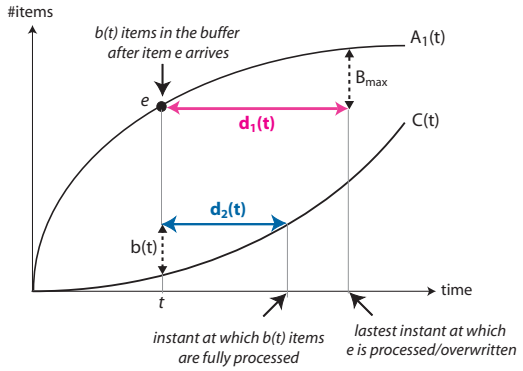

Figure 5: Upper bounds on delay of an output data item.

PROOF. Consider an item of A_1, called e, that arrives at time t. Observe that e is only overwritten when it is the oldest item in B and B is full. Because B contains at most $B_{\max}$ items, e will not be overwritten iff it is processed before the next $B_{\max}$ items of A_1 arrive. (Otherwise, it would be overwritten by the $(k+B_{\max})^{th}$ item). For example, in Fig. 4, item 5 must be processed before item 8 arrives. Thus, the delay $d(t)$ of e satisfies

$$d(t) \le \min\{\Delta \ge 0 \mid A_1(t+\Delta) - A_1(t) \ge B_{\max}\} = d_1(t).$$

Further, at time t, there are $b(t) = A_2(t) - A_3(t)$ items currently in the buffer (with e included). Thus, $d(t)$ will be no more than the amount of time needed to process these $b(t)$ items, i.e.,

$$d(t) \le \min\{\Delta \ge 0 \mid C(t+\Delta) - C(t) \ge b(t)\} = d_2(t).$$

As a result, $d(t) \le \min\{d_1(t),\, d_2(t)\}$. □

Further, since α^u is the upper arrival function of A_1 and β^u is the upper service function of C, $A_1(t) \le \alpha^u(t)$ and $C(t) \le \beta^u(t)$ for all $t \ge 0$. Observe that (i) the buffer can hold at most $B_{\max}$ items, and (ii) at most $\beta^u(t)$ items can be processed over any interval of length t. Hence, the number of items that are not overwritten in $(0,t]$, given by $A_2(t)$, is no more than the minimum of $\alpha^u(t)$ and $\beta^u(t) + B_{\max}$. As a result, the following corollary holds, which in turn implies Lemma 3.3.

COROLLARY 3.2. *Define* $\widetilde{\alpha}^u \stackrel{\text{def}}{=} \min\{\alpha^u, \beta^u + B_{\max}\}$. *Then,* $A_2 \le \widetilde{\alpha}^u$.

LEMMA 3.3. *Let* $\mathsf{del}(f,k) = \min\{t \ge 0 \mid f(t) \ge k\}$ *for all* $f \in \mathcal{F}$ *and for all* $k \ge 0$. *For all* $t \ge 0$,

$$d_1(t) \le \mathsf{del}(\alpha^l, B_{\max}), \tag{11}$$

$$d_2(t) \le \min\{\mathsf{del}(\beta^l, B_{\max}),\, \mathsf{hdist}(\widetilde{\alpha}^u, \beta^l)\}. \tag{12}$$

PROOF. Recall that α^l is the lower arrival function of A_1. Thus, $A_1(t+\Delta) - A_1(t) \ge \alpha^l(\Delta)$ for all $t \ge 0$ and for all $\Delta \ge 0$. By definition of $d_1(t)$, we imply for all $t \ge 0$,

$$d_1(t) \le \min\{\Delta \ge 0 \mid \alpha^l(\Delta) \ge B_{\max}\} = \mathsf{del}(\alpha^l, B_{\max}).$$

Similarly, since β^l is the lower service function of C, we have $C(t+\Delta) - C(t) \ge \beta^l$ for all $t \ge 0$ and for all $\Delta \ge 0$. Hence,

$$\forall t \ge 0 :\ d_2(t) \le \min\{\Delta \ge 0 \mid \beta^l(\Delta) \ge b(t)\}.$$

Further, $b(t) \le B_{\max}$. Hence, $d_2(t) \le \mathsf{del}(\beta^l, B_{\max})$ for all $t \ge 0$. The above two bounds on $d(t)$ are depicted in Fig. 6(i-ii).

We shall now prove that for all $t \ge 0$, $d_2(t) \le \mathsf{hdist}(\widetilde{\alpha}^u, \beta^l)$. Intuitively, this means the delay of an input item that goes through the system is bounded by the maximum horizontal distance between $\widetilde{\alpha}^u$ and β^l (see Fig. 6(iii)). From Corollary 3.2, we have $A_2(t) \le \widetilde{\alpha}^u(t)$. Consider an input item e arriving at time t. Since there are at least $\beta^l(t')$ items that can be processed in $(0,t']$ for any $t' > 0$, item e will be processed latest at the first instant $t' = t + \Delta$ at which $\widetilde{\alpha}^u(t) \ge \beta^l(t')$. In other words, the amount of time required to process e satisfies

$$d_2(t) \le \inf\{\Delta \ge 0 \mid \widetilde{\alpha}^u(t) \ge \beta^l(t+\Delta)\} \le \mathsf{hdist}(\widetilde{\alpha}^u, \beta^l).$$

Thus, Eq. (12) holds and hence the lemma. □

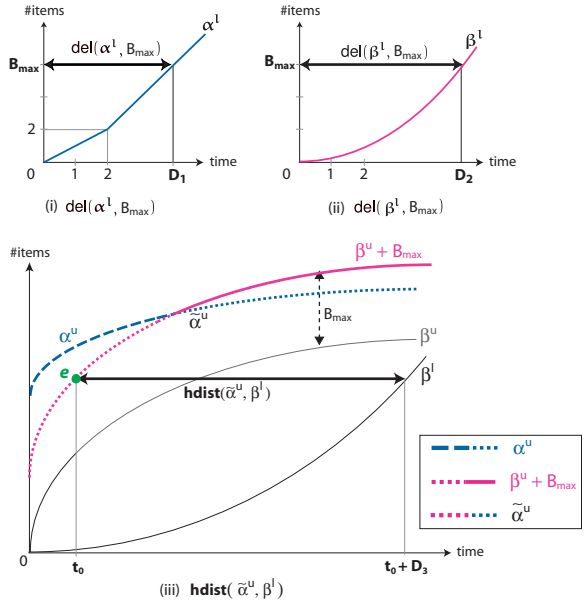

Figure 6: The maximum delay experienced by the input stream is the minimum of $\mathsf{del}(\alpha^l, B_{\max})$, $\mathsf{del}(\beta^l, B_{\max})$, **and** $\mathsf{hdist}(\widetilde{\alpha}^u, \beta^l)$.

From Lemma 3.3, we imply Theorem 3.4, which gives the maximum delay experienced by the input stream.

THEOREM 3.4. *The maximum delay experienced by the input stream is given by*

$$\mathsf{del}_{\mathsf{drf}}(\alpha, \beta, B_{\max}) = \min\{\mathsf{del}(\alpha^l, B_{\max}),\, \mathsf{del}(\beta^l, B_{\max}),\, \mathsf{hdist}(\widetilde{\alpha}^u, \beta^l)\}.$$

Fig. 6 illustrates the delay computation given by Theorem 3.4.

THEOREM 3.5. *The delay bound given by Theorem 3.4 is tight.*

PROOF. Denote $D = \mathsf{del}_{\mathsf{drf}}(\alpha, \beta, B_{\max})$ and $D_3 = \mathsf{hdist}(\widetilde{\alpha}^u, \beta^l)$. Then, $D \le D_3$ and $D \le \mathsf{del}(\alpha^l, B_{\max})$. For any given $f, g \in \mathcal{F}$ and $t \in \mathbb{R}^+$, we define $\mathsf{hdist}(f,g,t)$ to be the horizontal distance between f and g at time t, i.e.,

$$\mathsf{hdist}(f,g,t) \stackrel{\text{def}}{=} \inf\{\Delta \ge 0 \mid f(t) \le g(t+\Delta)\}.$$

We denote by $\Pi_h(f,g,D)$ the first instant t at which $\mathsf{hdist}(f,g,t)$ is at least D, i.e.,

$$\Pi_h(f,g,D) \stackrel{\text{def}}{=} \min\{t \ge 0 \mid \mathsf{hdist}(f,g,t) \ge D\}.$$

We will construct an input arrival pattern $\widehat{A_1}(t)$ constrained by α and a service pattern $\widehat{C}(t)$ constrained by β, such that there is an item of $\widehat{A_1}(t)$ which will be fully processed after D time units. In other words, there exists $T \ge 0$ such that $\mathsf{hdist}(\widehat{A_1}, \widehat{A_3}, T) = D$ and

$\widehat{A_1}(T+D) - \widehat{A_1}(T) \le B_{\max}$, where $\widehat{A_3}(t)$ is the resulting output function of $\widehat{A_1}(t)$ when the PE offers the service pattern $\widehat{C}(t)$. The first condition specifies that the amount of time required to fully process an item arriving at time T is D. The second states that there are no more than $B_{\max}$ items arriving over the interval $(T, T+D]$ (which implies that the item arriving at time T will not be overwritten over this interval).

Since $\widetilde{\alpha}^u = \min\{\alpha^u, \beta^u + B_{\max}\} \le \alpha^u$ and all the given arrival/service functions are non-decreasing,

$$\mathsf{hdist}(\alpha^u, \beta^l) \ge \mathsf{hdist}(\widetilde{\alpha}^u, \beta^l) = D_3 \ge D.$$

Hence, there exists $t \ge 0$ such that $\mathsf{hdist}(\alpha^u, \beta^l, t) \ge D$. Let t_0 be the smallest of such t, i.e., $t_0 = \Pi_h(\alpha^u, \beta^l, D)$. Define $\widehat{A_1}(t) = \alpha^u(t)$ if $t \le t_0$, and $\widehat{A_1}(t) = \alpha^u(t_0) + \alpha^l(t - x_0)$ otherwise. Further, define $\widehat{C}(t) = \beta^l(t)$ for all $t \ge 0$. Since $\alpha^l(t) \le \alpha^u(x) + \alpha^l(t-x) \le \alpha^u(t)$ for all $0 \le x \le t$, $\alpha^l(t) \le \widehat{A_1}(t) \le \alpha^u(t)$ for all $t \ge 0$. Hence, $\widehat{A_1}(t)$ is a valid arrival pattern of the input stream. By construction, $\widehat{C}(t)$ is a valid service pattern of the PE.

Since α^u is sub-additive and β^l is super-additive, $\alpha^u(t) \ge \beta^l(t)$ for all $0 \le t \le t_0$. Indeed, if $\alpha^u(s) < \beta^l(s)$ for some $s < t_0$, then

$$\begin{aligned}\alpha^u(t_0) - \beta^l(t_0 + D) &\le \alpha^u(s) + \alpha^u(t_0 - s) - \big(\beta^l(s) + \beta^l(t_0 + D - s)\big) \\ &< \alpha^u(t_0 - s) - \beta^l(t_0 - s + D).\end{aligned}$$

As a result, $\mathsf{hdist}(\alpha^u, \beta^l, t_0) < \mathsf{hdist}(\alpha^u, \beta^l, t_0 - s)$. Hence,

$$\Pi_h(\alpha^u, \beta^l, D) \le t_0 - s < t_0 = \Pi_h(\alpha^u, \beta^l, D),$$

which is always false.

From the above, we imply $\widehat{A_1}(t) \ge \widehat{C}(t)$ for all $0 \le t \le t_0$. This means all resource offered by $\widehat{C}$ in $[0, t_0 + D]$ will be used to process the items. Thus, the corresponding output function $\widehat{A_3}$ satisfies $\widehat{A_3}(t) = \widehat{C}(t)$ for all $0 \le t \le t_0 + D$. Hence, $\mathsf{hdist}(\widehat{A_1}, \widehat{A_3}, t_0) = D$ (recall that $t_0 = \Pi_h(\alpha^u, \beta^l, D)$. In other words, the delay of an item e arriving at $T = t_0$ is D. Besides, $D \le \mathsf{del}(\alpha^l, B_{\max})$ implies that $\alpha^l(D) \le B_{\max}$. Hence, the number of items arriving in $(t_0, t_0 + D]$ is $\widehat{A_1}(t_0 + D) - \widehat{A_1}(t_0) = \alpha^l(D) \le B_{\max}$, which means e is not overwritten. As a result, the constructed system consisting of $\widehat{A_1}$ and $\widehat{C}$ achieves the delay D given by Theorem 3.4. □

3.1.2 Computing output arrival functions

Recall that $A_2(t)$ is the effective input function of $A_1(t)$, which captures the items that will indeed be processed by the PE (see Fig. 3). Lemma 3.6 states the relationship between these two functions.

LEMMA 3.6. *The effective input function A_2 is bounded by:*

$$A_1 \otimes (\beta^l + B_{\max})^* \le A_2 \le A_1 \otimes \alpha^u \otimes (\alpha^u \otimes \beta^u + B_{\max})^*.$$

PROOF SKETCH. Since none of the items in A_2 is overwritten, for all $t \ge 0$, $b(t) = A_2(t) - A_3(t) \le B_{\max}$, or $A_2 \le A_3 + B_{\max}$. Let f be the function that maps the input A_2 to the output A_3, assuming f is monotonic. Then $A_3 + B_{\max} = f(A_2) + B_{\max} = (f + B_{\max})(A_2)$.

Further, the number of items that pass the admission test at P_v (i.e., not overwritten) over any time interval $(s,t]$ is no more than the number of original items that enter the system over the same interval. In other words,

$$\forall t \ge 0,\ \forall 0 \le s \le t:\ A_2(t) - A_2(s) \le A_1(t) - A_1(s).$$

Recall that $\mathcal{I}_{A_1}(A_2)(t) = \inf\{A_2(s) + A_1(t) - A_1(s) \mid 0 \le s \le t\}$. Then, $A_2 \le \mathcal{I}_{A_1}(A_2)$. Hence,

$$A_2 \le \min\{A_1,\ \mathcal{I}_{A_1}(A_2),\ (f + B_{\max})(A_2)\} \tag{13}$$

$$\Leftrightarrow\ A_2 \le A_1 \oplus \big(\mathcal{I}_{A_1} \oplus (f + B_{\max})\big)(A_2). \tag{14}$$

Hence, the input function of the items that actually go through the system is the maximum solution for Eq. (14). By Theorem 2.1,

$$A_2 = \big(\mathcal{I}_{A_1} \oplus (f + B_{\max})\big)^*(A_1).$$

By applying Eq. (1) (cf. Section 2), the above is equivalent to

$$A_2 = \big(\mathcal{I}_{A_1} \circ (f + B_{\max})\big)^* \circ \mathcal{I}_{A_1}(A_1).$$

Denote $\mathcal{C}_z(x) = x \otimes z$. Since f is the mapping from A_2 to A_3, and β is the service function of the PE, $f(A_2) \le A_2 \otimes \beta^u$, or equivalently, $f \le \mathcal{C}_{\beta^u}$. Similarly, α is the arrival function of A_1 implies that $A_1(t) - A_1(s) \le \alpha^u(t-s)$. Thus, $\mathcal{I}_{A_1}(A_2) \le \alpha^u \otimes A_2$, or $\mathcal{I}_{A_1} \le \mathcal{C}_{\alpha^u}$. Hence,

$$A_2 \le \big(\mathcal{C}_{\alpha^u} \circ (\mathcal{C}_{\beta^u} + B_{\max})\big)^* \circ \mathcal{C}_{\alpha^u}(A_1),$$

which can be rewritten as $A_2 \le A_1 \otimes (\alpha^u \otimes \beta^u + B_{\max})^* \otimes \alpha^u$.

By similar arguments, we can also imply $A_2 \ge A_1 \otimes (\beta^l + B_{\max})^*$. This proves the lemma. □

Lemma 3.7 is derived directly from the bounds established in the above lemma, which holds true due to $A_1 \otimes \beta_v^l \le A_2 \le A_1 \otimes \beta_v^u$.

LEMMA 3.7. *Let $\beta_v^u = \alpha^u \otimes (\alpha^u \otimes \beta^u + B_{\max})^*$ and $\beta_v^l = (\beta^l + B_{\max})^*$. Then, β_v^u and β_v^l are valid upper and lower service functions for P_v.*

By definition, $A_2 \otimes \beta^l \le A_3 \le A_2 \otimes \beta^u$. Thus, $A_1 \otimes \beta^l \otimes \beta_v^l \le A_3 \le A_1 \otimes \beta^u \otimes \beta_v^u$. Hence, $\widetilde{\beta}^l = \beta^l \otimes \beta_v^l$ and $\widetilde{\beta}^u = \beta^u \otimes \beta_v^u$ are the overall service functions given to the input stream when there is data refresh. Based on $\widetilde{\beta}$, we can compute the output arrival functions.

THEOREM 3.8. *The arrival functions of the output stream (A_3) when data refresh semantics is implemented at the input buffer is given by $\alpha' = (\alpha^{u'}, \alpha^{l'})$, where*

$$\alpha^{u'} = \min\big\{\big(\alpha^u \otimes \widetilde{\beta}^u\big) \oslash \widetilde{\beta}^l,\ \widetilde{\beta}^u\big\}, \tag{15}$$

$$\alpha^{l'} = \min\big\{\big(\alpha^l \oslash \widetilde{\beta}^u\big) \otimes \widetilde{\beta}^l,\ \widetilde{\beta}^l\big\}. \tag{16}$$

with $\widetilde{\beta}^l = \beta^l \otimes (\beta^l + B_{\max})^*$ and $\widetilde{\beta}^u = \beta^u \otimes \alpha^u \otimes (\alpha^u \otimes \beta^u + B_{\max})^*$.

We note that when $B_{\max}$ is unbounded, $\widetilde{\beta} = \beta$ and $\alpha' = \alpha'_{\inf}$.

Lemma 3.9 further refines the effective output arrival functions to ensure the sub-additivity property of $\alpha^{u'}$, the super-additivity of $\alpha^{l'}$ and their causal relationship. Its proof can easily be established based on the definition of upper and lower arrival functions. The details are available in [13].

LEMMA 3.9. *Let $\widehat{\alpha}^u = (\alpha^{u'})^*$ and $\widehat{\alpha}^l = (\alpha^{l'})^{\overline{*}}$. Denote*

$$\alpha^{u'}_{\mathsf{drf}}(\Delta) = \min\big\{\widehat{\alpha}^u(\Delta + \tau) - \widehat{\alpha}^l(\tau) \mid \tau \ge 0\big\}, \tag{17}$$

$$\alpha^{l'}_{\mathsf{drf}}(\Delta) = \max\big\{\widehat{\alpha}^l(\Delta + \tau) - \widehat{\alpha}^u(\tau) \mid \tau \ge 0\big\}. \tag{18}$$

Then, $\alpha^{u'}_{\mathsf{drf}}$ ($\alpha^{l'}_{\mathsf{drf}}$) is a valid upper (lower) arrival function for the output stream that is smaller (larger) than the $\alpha^{u'}$ ($\alpha^{l'}$).

3.1.3 Computing remaining service functions

Since β_v^u and β_v^l are the upper and lower service functions of the virtual processor P_v (Lemma 3.7), the effective input function A_2 is bounded by the output arrival functions of P_v, given by

$$\alpha_v^u = \min\big\{\big(\alpha^u \otimes \beta_v^u\big) \oslash \beta_v^l,\ \beta_v^u\big\},$$
$$\alpha_v^l = \min\big\{\big(\alpha^l \oslash \beta_v^u\big) \otimes \beta_v^l,\ \beta_v^l\big\}.$$

Using $\alpha_v = (\alpha_v^u, \alpha_v^l)$ as input arrival functions to the PE, we can derive the remaining service functions of the PE as in the conventional case as below (since there is no buffer overflows).

$$\beta^{u'}_{\mathsf{drf}} = \big(\beta^u - \alpha_v^l\big)\ \overline{\oslash}\ 0 \tag{19}$$

$$\beta^{l'}_{\mathsf{drf}} = \big(\beta^l - \alpha_v^u\big)\ \overline{\otimes}\ 0 \tag{20}$$

Thus, the remaining service function of the PE when the buffer implements data refresh semantics is given by $\beta'_{\mathsf{drf}} = (\beta^{u'}_{\mathsf{drf}}, \beta^{l'}_{\mathsf{drf}})$.

3.2 Heterogeneous systems with a mixture of buffer semantics

In this section, we show how one can apply the RTC-DRF presented in the previous section to analyze heterogeneous systems with different buffer semantics in a compositional manner. Through this, we demonstrate how RTC-DRF can be integrated directly into the conventional RTC-INF while guaranteeing that the overall analysis is at least as tight as the RTC-INF alone.

The systems we consider consist of multiple input streams, namely $s_1, \ldots, s_n$, that are processed by a sequence of PEs under Fixed Priority (FP) scheduling policy. Each buffer in a system can be either an infinite FIFO buffer or a finite buffer with data refresh semantics. An example of such systems is shown in Fig. 7. In this example, B_1 is a finite buffer of size $B_{\max}$ that has data refresh semantics. On the other hand, B'_1 is an unbounded FIFO buffer. Given such a system, we would like to compute the standard performance-related metrics as discussed in the previous sections.

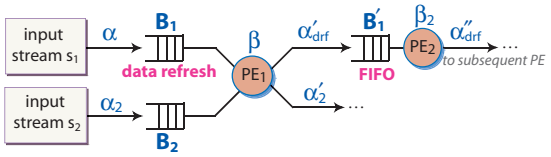

Figure 7: Systems with a mixture of buffer semantics.

Consider the first PE in Fig. 7. Suppose s_i has higher priority than s_j if $i < j$. Then, PE_1's resource will first be given to s_1 and the remaining will be given to s_2, then s_3 and so on. Denote α and β as the arrival function of s_1 and the service function of PE_1, respectively. Applying the RTC-DRF results obtained in the single stream case, we compute the maximum delay using Theorem 3.4 and the output arrival function using Lemma 3.9. The remaining service function β'_{drf} that is used to process the next input stream s_2 can also be computed using Eq. (19) and (20). Based on β'_{drf}, we then analyze s_2, taking into consideration the semantics of B_2. If B_2 is an infinite FIFO buffer, we apply the RTC-INF. However, if B_2 implements data refresh semantics, we analyze using RTC-DRF as done for s_1. The analysis is repeated until we reach s_n.

At the same time, the output arrival function α'_{drf} of s_1 produced by PE_1 is fed as input arrival function to PE_2. At this PE, we repeat the same analysis as above with respect to the semantics of its input buffer B'_1. The computed output arrival function α''_{drf} is then fed as input to the subsequent PEs[2].

Correctness of our compositional analysis. As seen above, the RTC-DRF combined with the RTC-INF enables complex systems with a mixture of different buffer types to be analyzed compositionally. We claim that RTC-DRF does not introduce any loss in terms of analysis accuracy for the overall system. Specifically, it provides a tighter output arrival function than RTC-INF does, and hence ensures accurate analysis at the subsequent PEs (Theorem 3.11). Further, by taking into account data refresh, RTC-DRF is able to capture the service unused by the overwritten items, thereby guarantees more service for the lower priority streams (Theorem 3.12).

[2]A tighter output arrival function for PE_j can be obtained by applying RTC-INF to the overall effective service function for part of the system comprising PE_1 to PE_j (i.e., the convolution of the individual effective service functions).

LEMMA 3.10. *Let $\alpha'_{\mathsf{drf}} = (\alpha^{u'}_{\mathsf{drf}}, \alpha^{l'}_{\mathsf{drf}})$ and $\alpha'_{\mathsf{inf}} = (\alpha^{u'}_{\mathsf{inf}}, \alpha^{l'}_{\mathsf{inf}})$ be the output arrival functions of s_1 at PE_1 (Fig. 7) that are computed using RTC-DRF and RTC-INF, respectively. Then, $\alpha^{u'}_{\mathsf{drf}} \le \alpha^{u'}_{\mathsf{inf}}$.*

PROOF SKETCH. Since $\alpha^{u'}_{\mathsf{drf}} \le \alpha^{u'}$ (due to Lemma 3.9), the theorem holds if $\alpha^{u'} \le \alpha^{u'}_{\mathsf{inf}}$. By Theorem 3.8, $\alpha^{u'} = \min\{(\alpha^u \otimes \widetilde{\beta}^u) \oslash \widetilde{\beta}^l, \widetilde{\beta}^u\}$. Thus, $\alpha^{u'} \le \widetilde{\beta}^u$. We will prove that $\widetilde{\beta}^u \le \alpha^{u'}_{\mathsf{inf}}$.

By definition, we have $\widetilde{\beta}^u = \beta^u \otimes \alpha^u \otimes (\alpha^u \otimes \beta^u + B_{\max})^*$ and $\alpha^{u'}_{\mathsf{inf}} = \min\{(\alpha^u \otimes \beta^u) \oslash \beta^l, \beta^u\}$. Since $\beta^u \otimes g \le \beta^u$ for all $g \in \mathcal{F}$, $\widetilde{\beta}^u \le \beta^u$. Thus, $\widetilde{\beta}^u \le \alpha^{u'}_{\mathsf{inf}}$ iff

$$\beta^u \otimes \alpha^u \otimes (\alpha^u \otimes \beta^u + B_{\max})^* \;\le\; (\alpha^u \otimes \beta^u) \oslash \beta^l \qquad (21)$$

Let $f = \alpha^u \otimes \beta^u = \beta^u \otimes \alpha^u$. Then,

$$(21) \;\Leftrightarrow\; f \otimes (f + B_{\max})^* \le f \oslash \beta^l \;\Leftrightarrow\; f \otimes (f + B_{\max})^* \otimes \beta^l \le f$$
$$\Leftrightarrow\; f \otimes g \le f \quad \text{where } g = (f + B_{\max})^* \otimes \beta^l,$$

which is always true. Hence, $\widetilde{\beta}^u \le \alpha^{u'}_{\mathsf{inf}}$ and thus, $\alpha^{u'} \le \alpha^{u'}_{\mathsf{inf}}$. □

THEOREM 3.11. *Let α'_{drf} and α'_{inf} be defined in Lemma 3.10. Denote by $\mathsf{buf}_{\mathsf{drf}}$, $\mathsf{del}_{\mathsf{drf}}$ and α''_{drf} (resp. $\mathsf{buf}_{\mathsf{inf}}$, $\mathsf{del}_{\mathsf{inf}}$ and α''_{inf}) the maximum backlog, maximum delay and output arrival function at PE_2 where α'_{drf} (resp. α'_{inf}) is used as the input arrival function to PE_2. Then, $\mathsf{buf}_{\mathsf{drf}} \le \mathsf{buf}_{\mathsf{inf}}$, $\mathsf{del}_{\mathsf{drf}} \le \mathsf{del}_{\mathsf{inf}}$ and $\alpha^{u''}_{\mathsf{drf}} \le \alpha^{u''}_{\mathsf{inf}}$.*

PROOF. Since the input buffer of PE_2 is a simple infinite FIFO buffer, we analyze it using RTC-INF. Let β_2 be the service function of PE_2. Following RTC-INF and Lemma 3.10, we have:

$$\begin{aligned}\mathsf{buf}_{\mathsf{drf}} &= \mathsf{vdist}(\alpha'_{\mathsf{drf}}, \beta^l_2) \;=\; \sup\{\alpha^{u'}_{\mathsf{drf}}(\Delta) - \beta^l_2(\Delta) \mid \Delta \ge 0\} \\ &\le \sup\{\alpha^{u'}_{\mathsf{inf}}(\Delta) - \beta^l_2(\Delta) \mid \Delta \ge 0\} \quad (\text{since } \alpha^{u'}_{\mathsf{drf}} \le \alpha^{u'}_{\mathsf{inf}}) \\ &= \mathsf{vdist}(\alpha'_{\mathsf{inf}}, \beta^l_2) \;=\; \mathsf{buf}_{\mathsf{inf}}.\end{aligned}$$

Thus, $\mathsf{buf}_{\mathsf{drf}} \le \mathsf{buf}_{\mathsf{inf}}$. The remaining properties can be proved similarly. □

THEOREM 3.12. *Let β'_{drf} (β'_{inf}) be the remaining service function of PE_1 after processing s_1, which is computed using RTC-DRF (RTC-INF), Then, (i) $\beta^{l'}_{\mathsf{drf}} \ge \beta^{l'}_{\mathsf{inf}}$ and (ii) $\beta^{u'}_{\mathsf{drf}} \ge \beta^{u'}_{\mathsf{inf}}$.*

PROOF. First, for any $f, g \in \mathcal{F}$ such that $f \ge g$, we have:

$$\forall \Delta \ge 0: \;\; (f \overline{\otimes} 0)(\Delta) = \sup_{0 \le x \le \Delta} f(x) \ge \sup_{0 \le x \le \Delta} g(x) = (g \overline{\otimes} 0)(\Delta).$$

Thus, $f \overline{\otimes} 0 \ge g \overline{\otimes} 0$. Similarly, $f \overline{\oslash} 0 \ge g \overline{\oslash} 0$.

(i) Recall that $\beta^{l'}_{\mathsf{drf}} = (\beta^l - \alpha^u_v) \overline{\otimes} 0$ and $\beta^{l'}_{\mathsf{inf}} = (\beta^l - \alpha^u) \overline{\oslash} 0$, with

$$\begin{aligned}\alpha^u_v &= \min\{(\alpha^u \otimes \beta^u_v) \oslash \beta^l_v, \beta^u_v\} \;\le\; \beta^u_v \\ &= \alpha^u \otimes (\alpha^u \otimes \beta^u + B_{\max})^* \;\le\; \alpha^u.\end{aligned}$$

Thus, $\beta^l - \alpha^u_v \ge \beta^l - \alpha^u$. Hence, $(\beta^l - \alpha^u_v) \overline{\otimes} 0 \ge (\beta^l - \alpha^u) \overline{\otimes} 0$. In other words, $\beta^{l'}_{\mathsf{drf}} \ge \beta^{l'}_{\mathsf{inf}}$.

(ii) By definition, $\beta^{u'}_{\mathsf{drf}} = (\beta^u - \alpha^l_v) \overline{\oslash} 0$ and $\beta^{u'}_{\mathsf{inf}} = (\beta^u - \alpha^l) \overline{\oslash} 0$. We will first show that $\alpha^l_v \le \alpha^l$. Indeed,

$$\alpha^l_v = \min\{(\alpha^l \oslash \beta^u_v) \otimes \beta^l_v, \beta^l_v\} \le (\alpha^l \oslash \beta^u_v) \otimes \beta^l_v,$$

which implies that $\alpha^l_v \le \alpha^l$ if $(\alpha^l \oslash \beta^u_v) \otimes \beta^l_v \le \alpha^l$. This is equivalent to $\alpha^l \oslash \beta^u_v \le \alpha^l \oslash \beta^l_v$, which always holds due to $\beta^u_v \ge \beta^l_v$.

From $\alpha^l_v \le \alpha^l$, we imply $\beta^u - \alpha^l_v \ge \beta^u - \alpha^l$. As a result, $(\beta^u - \alpha^l_v) \overline{\oslash} 0 \ge (\beta^u - \alpha^l) \overline{\oslash} 0$. In other words, $\beta^{u'}_{\mathsf{drf}} \ge \beta^{u'}_{\mathsf{inf}}$. □

4. EXTENSIONS TO OTHER BUFFER MANAGEMENT SEMANTICS

In the data refresh semantics considered thus far, if an item arrives when the input buffer is full, the *oldest item in the buffer* is discarded (also known as *Drop Oldest* in [14]). We now extend our analysis method for other buffer management semantics, such as those defined in [14]. We first consider the *Drop Newest (DN)* policy, where incoming items are discarded if the buffer is full.

Observe that the number of items that are discarded in the *DN* and the data refresh semantics are identical. Since only the number of items that are discarded (and not which specific items) affects the number of items that will be processed by the PE, the number of items that are processed over any given time interval in both cases are the same. Hence, the system produces the same number of output items in both semantics. In other words, the output arrival functions and the remaining service functions in both refresh semantics are the same, which are given by Lemma 3.9 and Eq. (19) and Eq. (20).

Further, note that in the *DN* semantics, once an input item is written to the buffer, it will not be overwritten. Hence, the delay experienced by an input item is bounded by the maximum amount of time required to process the item. Using the same argument as in the data refresh semantics case, we imply that the maximum delay experienced by the input stream is

$$\mathsf{del}_{DN} = \min\{\mathsf{del}(\beta^l, B_{\max}), \mathsf{hdist}(\widetilde{\alpha}^u, \beta^l)\}.$$

Based on similar arguments, one could also obtain the analysis results for the *Drop All* and *Drop Random* policies [14] where all items or a random item in the buffer will be discarded when the buffer overflows. Further, systems with multiple PEs and/or multiple input streams under Fixed Priority scheduling which implement a mixture of these semantics can also be analyzed in a compositional manner as detailed in Section 3.2.

5. CASE STUDY

We now present three case studies to demonstrate the applicability of our analysis methods. The first shows how our technique can be used to compute bounds on the amount of data guaranteed to go through the system when the buffer implements the data refresh semantics. The second illustrates the effect of buffer size on the freshness of output data in a traction control application. The last one presents a sensitivity analysis of the variation in the maximum delay experienced by the input stream with respect to changes in the input workload.

5.1 Case study 1: Output guarantees in presence of data refreshing

In this case study, we analyze the bounds on the output stream of the system described in Fig. 2 using our technique in Section 3.1 and a SystemC simulation.

Simulation setup. Using our SystemC event simulator, we generate an arbitrary input event trace $R_{sim}(t)$ that comprises different event types of varying processing cycle requirements. Events from $R_{sim}(t)$ are first kept at a finite buffer that implements data refresh semantics. Here, we are interested in the freshest event and hence the size of the buffer is set to 1. The processor processes the events from the buffer in a greedy fashion, where it is set to run at frequency $f = 5\ MHz$. We then observe the output arrival pattern $R'_{sim}(t)$ of the processed stream.

Obtaining the arrival and service functions. Based on the generated trace $R_{sim}(t)$, we derive the arrival functions $\alpha^u(\Delta)$ and $\alpha^l(\Delta)$ of the input stream by sliding a window of size Δ along the time axis and determine the maximum and minimum number of events generated across all the windows. Further, from the execution requirements of the generated events, we compute the workload functions $\gamma^u(k)$ and $\gamma^l(k)$, which give the maximum and minimum number of processor cycles required to process any k consecutive events. The service functions of the processor can be then obtained from the workload functions and the frequency f using the formulas $\beta^l(\Delta) = \gamma^{-u}(f\Delta)$ and $\beta^u(\Delta) = \gamma^{-l}(f\Delta)$. We then use α and β as input arrival and service functions to compute the output arrival functions using RTC-DRF and RTC-INF.

Simulation vs. analytical results.[3] The upper output arrival function computed by RTC-DRF correctly upper bounds the output simulation trace $R'_{sim}(t)$, and it is closer to $R'_{sim}(t)$ than the upper output arrival function given by RTC-INF. Similarly, the lower output arrival function given by RTC-DRF correctly lower bounds $R'_{sim}(t)$; the RTC-INF, however, gives a wrong bound as its computed value is above the output simulation trace. Hence, by taking into account buffer refresh, our method gives a tighter upper bound than the conventional RTC does, at the same time avoids invalid results faced by the conventional RTC.

Effect of buffer size on the output stream and throughput. Fig. 8 depicts the lower output arrival functions obtained by our technique when varying the input buffer size for the same input stream $\alpha(\Delta)$. As shown in the figure, the lower arrival function corresponding to a lower buffer size is located below the one corresponding to a larger buffer size. This is because, as the buffer size is increased, fewer items are overwritten and thus, more items are processed. It can also be observed from the figure that the output arrival functions for buffer sizes $B = 10$ and $B = 15$ coincide, which happens when all input items are processed.

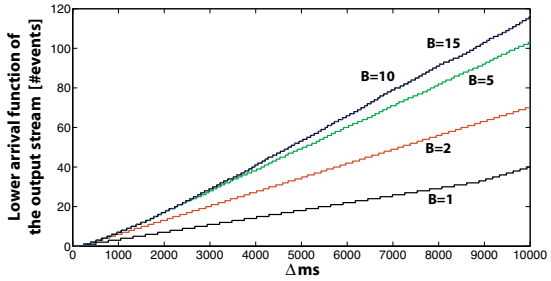

Figure 8: Guarantees on the output stream for different input buffer sizes.

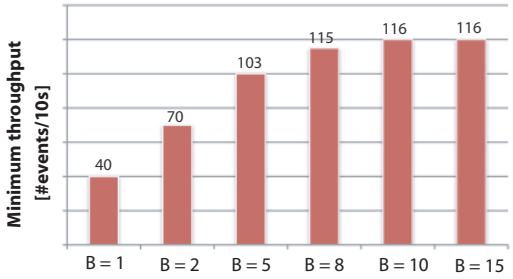

Figure 9: Effect of buffer size on the system's throughput.

The buffer size also has a large impact on the minimum throughput of the system. As illustrated in Fig. 9, initially when we double the size of the buffer, the throughput increases nearly by a factor of two. However, the increasing factor is reduced as we further increase the buffer size, and the throughput will finally converge (at value $B \geq 10$) when the buffer is large enough to avoid overflows.

[3]Due to space constraints, we do not show the detailed results here.

5.2 Case study 2: A traction control system

Strong acceleration can lead to wheel spinning, especially on poorly prepared roads. A traction control system prevents spinning of the driving wheels and provides an optimal traction. Fig. 10 depicts a traction control system application mapped on a CAN architecture we would like to analyze.

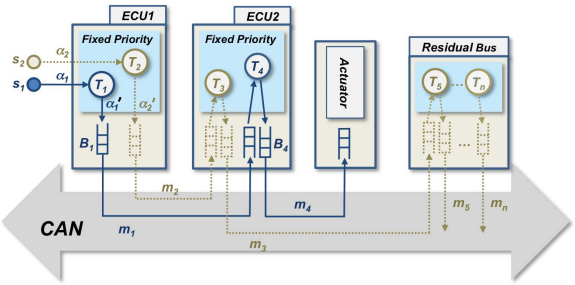

Figure 10: A traction control system on a CAN architecture.

The system consists of a wheel speed sensor cluster s_1, two ECUs for computing the traction control and an actuator performing the wheel braking. ECU1 receives the wheel-speed values from the sensor cluster s_1, and processes the current slip by executing task T_1. The processed slip value is sent via message m_1 to ECU2. Task T_4 is computing the brake force according to the input slip value, especially if a wheel is going to spin. Subsequently, the brake force value is sent via m_4 to the wheel brake actuator which performs the brake application and therefore prevents wheel spinning. As the delay of such a system has to be very short, it is important that the most recent slip value is available for computing the brake forces and that the most recent computed brake force value is sent to the actuator. To achieve this, the buffers B_1 and B_4 are configured to non-queued buffers that allow updating their contents with a new processed value in case the previous value could not be transmitted on the bus according to the CAN scheduling policy. This may happen if too many messages with a higher priority than m_1 and m_4 are transmitted on the CAN bus for a certain period of time (e.g., due to some event triggered higher priority messages which have to be transmitted because of changing system states of other ECUs). Besides, there is a second application running on ECU1 and ECU2. Messages are sent on the CAN bus according to fixed priority non-preemptive scheduling (FPNS).

Given the above system, we are interested in how fresh the wheel-speed values are when they arrive at the actuator. To derive this, we calculate the maximum end-to-end delay of the messages that are transmitted from the sensor s_1 to the actuator through the colored path (in solid blue line).

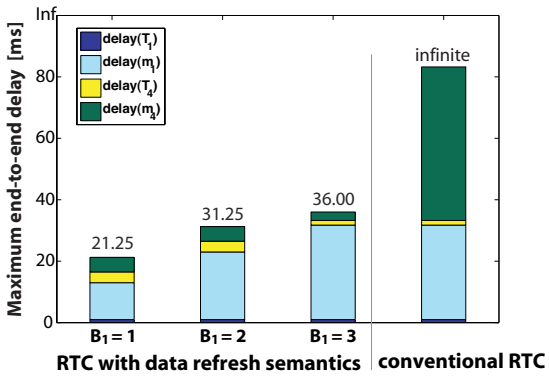

Figure 11: Maximum delay from s_1 to the actuator.

Analysis results. We employ the method in [9] for modeling FPNS policy used by the CAN bus. The residual bus depicted in Fig. 10 consists of n strictly periodic messages with priorities higher than m_1 to m_4. The message priority P_m is defined by $P_{m_1} > P_{m_2} > P_{m_4} > P_{m_3}$ and the task priority P_T is given by $P_{T_1} > P_{T_2}$ and $P_{T_3} > P_{T_4}$. For the analysis, we assume a low speed CAN bus providing a data rate of 125 kbit/s and a fixed frame length for every CAN frame in the system. The sensor task s_1 has a period of 10 ms and an additional jitter of 2 ms. Both B_1 and B_4 are finite buffers with data refresh semantics, where B_4 has a fixed size of 1 and B_1 has a variable size.

Fig. 11 depicts the corresponding maximum delay experienced by a message originated from the sensor s_1 to the actuator when we vary the size of the buffer B_1, computed by RTC-INF (assuming no data refresh) and by RTC-DRF. Here, the longer the delay, the less fresh the data. As illustrated in the figure, according to the RTC-INF, a message may experience unbounded end-to-end delay, which is overly pessimistic. By taking into consideration the buffer size and the data refresh semantics, our RTC-DRF method gives a finite delay. It can also be observed from the figure that, as we increase the buffer size, the delay increases and the data becomes more stale. This is expected because when the buffer is small, it keeps only the most recent data items, which is not the case for a large buffer.

Based on the above observations, it is often appropriate to keep the buffer size small in applications where the freshness of data is critical. On the contrary, applications that require high throughput often need sufficient on-chip memory to maintain the desired level of quality of service.

5.3 Case study 3: Sensitivity analysis

To evaluate the robustness of our method as well as the relationship between input parameters and system performance-related metrics, we study the sensitivity of our analysis with respect to variations in the input stream. Towards this, we consider a single periodic-with-jitter input stream that is processed by a system which implements data refresh semantics, and examine the impact of input jitter variation on the delay of the output stream.

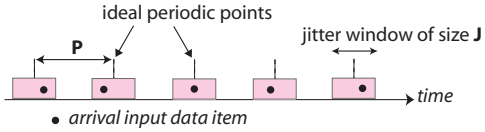

Figure 12: Periodic sensor stream with jitter.

As shown in Fig. 12, such an input stream arrives at a constant period P in average; however, the arrival times of the items may deviate within an interval of length J (called the jitter) surrounding the ideal periodic arrival points. Besides modeling an input source that is not strictly periodic, this jitter is also often used as a means to capture possible errors in the period measurement of an input stream.

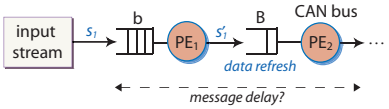

Figure 13: Example system for sensitivity analysis.

System description. Fig. 13 shows the architecture of the system. The periodic sensor stream s_1, with period P and jitter J, upon arriving at the system will be stored in the input buffer b prior to being processed by the processor PE_1. Its output stream s_1' is then written to a transmit buffer B before being transmitted to the CAN bus

(denoting as PE_2). Here, b is an unbounded FIFO buffer; however, B implements data refresh semantics.

We assume that B has a fixed depth of 1 and the CAN bus provides data rate of 125 kbit/s. The input stream s_1 has a period of 10 ms and a variable jitter of J ms. In our experiment, we vary J from 0 to 7 ms in steps of 0.5 ms, and compute the corresponding maximum delay experienced by an input message.

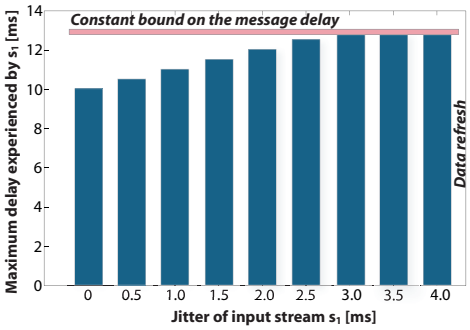

Figure 14: Effect of input jitter on the message delay.

Fig. 14 depicts the maximum delay experienced by an input message corresponding to different input jitter values. As shown in the figure, the maximum message delay increases linearly as we increase the jitter of the input stream. This is expected because when the jitter is increased, more items may arrive in a fixed interval of time, which increases the worst case resource demand of the input stream. As a result, a more jittery stream experiences a larger maximum delay.

On the other hand, the maximum delay stabilizes after the input jitter exceeds a certain value (i.e., $J \geq 3$ in the figure). This convergence of delay is guaranteed due to the enforcement of data refresh semantics. Specifically, since the service functions of the PEs do not change, the maximum workload that can be processed by PE_2 stays constant regardless of the input demand. Further, in a high load scenario (e.g., with high input jitter value), the maximum number of items that wait in the buffer will be limited by the depth of the buffer (since the excessive input items will all be discarded due to the data refresh semantics). Hence, when the input jitter goes beyond a certain threshold, the maximum number of items that will indeed be processed is only limited by the service function and the buffer size. As a result, the maximum delay remains constant as the jitter continues to increase.

From the above sensitivity analysis, one can derive the correlation between the input measurements and the system behavior. In scenarios where jitter is used to accommodate possible input measurement errors, the tightness of the delay results is linearly proportional to the tightness of the input jitter value; however, it is guaranteed to be bounded by a constant accuracy despite how pessimistic the input measurement is.

Lastly, the RTC-DRF results also showcase interesting system behaviors that are not easily visible otherwise. For instance, when data refresh is implemented, the maximum message delay is no longer influenced by the input load once the load is larger than the maximum service provided added with the buffer size. On the contrary, the number of messages that are overwritten is susceptible to input workload, especially when the input load is high. Based on these observation, one can also determine the maximum workload acceptable by the system to guarantee a delay constraint or to minimize the amount of data loss. It is worth noting that such insights into the effects of various design parameters and their trade-offs would have not been possible by using RTC-INF alone.

6. CONCLUDING REMARKS

We have presented an analytical framework to model and analyze systems with buffers implementing data refresh semantics. Our analysis is tight and based solely on algebraic techniques, which can be computed efficiently and compositionally. Further, it can be easily integrated into the existing RTC framework and extended to analyze similar buffer management policies. We have also illustrated the utility of our method using three realistic case studies from the automotive domain. We plan to extend the theoretical results established in this paper to capture more complex system behaviors, such as dynamic scheduling policies, dependencies between input/output streams, and more complex buffer update schemes.

7. REFERENCES

[1] A. Benveniste et al. Heterogeneous reactive systems modeling: capturing causality and the correctness of loosely time-triggered architectures (LTTA). In *EMSOFT*, 2004.

[2] A. Benveniste et al. Communication by sampling in time-sensitive distributed systems. In *EMSOFT*, 2006.

[3] A. Benveniste et al. Loosely time-triggered architectures based on communication-by-sampling. In *EMSOFT*, 2007.

[4] J.-Y. Le Boudec and P. Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*, volume LNCS 2050. Springer, 2001.

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

[6] S. Chakraborty, L. T. X. Phan, and P. S. Thiagarajan. Event count automata: A state-based model for stream processing systems. In *RTSS*, 2005.

[7] N. Feiertag, K. Richter, J. Nordlander, and J. Jonsson. A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics. In *CRTS*, 2008.

[8] E. Fersman, P. Krcál, P. Pettersson, and W. Yi. Task automata: Schedulability, decidability and undecidability. *Information and Computation*, 205(8):1149–1172, 2007.

[9] W. Haid and L. Thiele. Complex task activation schemes in system level performance analysis. In *CODES+ISSS*, 2007.

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

[11] L. Mangeruca, M. Baleani, A. Ferrari, and A. Sangiovanni-Vincentelli. Semantics-preserving design of embedded control software from synchronous models. *IEEE Transactions on Software Engineering*, 33(8), 2007.

[12] O. Moreira and M. Bekooij. Self-timed scheduling analysis for real-time applications. *EURASIP Journal on Advances in Signal Processing*, 2007.

[13] L. T. X. Phan, R. Schneider, S. Chakraborty, and I. Lee. Modeling buffers with data refresh semantics in automotive architectures. `www.cis.upenn.edu/~linhphan/papers/emsoft10TR.pdf`, 2010.

[14] J. Ray and P. Koopman. Data management mechanisms for embedded system gateways. In *DSN*, 2009.

[15] S. Tripakis, C. Pinello, A. Benveniste, A. Sangiovanni-Vincentelli, P. Caspi, and M. Di Natale. Implementing synchronous models on loosely time triggered architectures. *IEEE Transactions on Computers*, 57(10), 2008.

[16] E. Wandeler and L. Thiele. Workload correlations in multi-processor hard real-time systems. *Journal of Computer and System Sciences (JCSS)*, 73(2):207–224, 2007.

[17] W. Wandeler, A. Maxiaguine, and L. Thiele. Quantitative characterization of event streams in analysis of hard real-time applications. *Real-Time Systems*, 29(2-3):205–225, 2005.

[18] M. Wiggers, M. Bekooij, P. Jansen, and G. Smit. Efficient computation of buffer capacities for multi-rate real- time systems with back-pressure. In *CODES+ISSS*, 2006.