PUBLICATION
INTERNE
N° 1854

LOOSELY TIME-TRIGGERED ARCHITECTURES BASED ON COMMUNICATION-BY-SAMPLING

A. BENVENISTE , P. CASPI , M. DI NATALE , C. PINELLO ,
A. SANGIOVANNI VINCENTELLI , S. TRIPAKIS

ISSN 1166-8687

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES
Campus de Beaulieu – 35042 Rennes Cedex – France
Tél. : (33) 02 99 84 71 00 – Fax : (33) 02 99 84 71 71
http://www.irisa.fr

Loosely Time-Triggered Architectures based on Communication-by-Sampling*

A. Benveniste , P. Caspi , M. Di Natale , C. Pinello , A. Sangiovanni Vincentelli , S. Tripakis **

Systèmes communicants
Projet S4

Publication interne n1854 — June 2007 — 41 pages

Abstract:

We address the problem of mapping a set of processes which communicate synchronously on a distributed platform. The Time Triggered Architecture (TTA) proposed by Kopetz for the communication mechanism of a distributed platform offers a direct mapping that would preserve the semantics of the specification. However, its exact implementation may, at times, be problematic as it requires the distributed platform to have the clocks of its components perfectly synchronized. We propose as implementation architecture a relaxation of TTA called Loosely Time-Triggered Architecture (LTTA), in which computing units perform writes into and reads from the communication medium independently, triggered by local, quasi-periodic but non synchronized, clocks. LTTA offers some of the advantages of TTA with lower hardware cost and greater flexibility. So far LTTA was studied for single directional two-users communications over an LTT bus. General topology was not studied. In this paper we propose a design flow that ensures semantics preservation for an LTT communication network with arbitrary topology. Key elements are two new protocols for clock regeneration and predictive traffic shaping. Our approach relies on a mathematical Model of Communication (MoC) that we describe in detail.

Key-words: real-time systems, distributed control, time-triggered, loosely time-triggered, MoCC

* This research was supported in part by the European Commission under the projects IST-2001-34820 ARTIST and IST-004527 ARTIST2 Networks of Excellence on Embedded Systems Design, by the NSF under the project ITR (CCR-0225610), and by the GSRC.

** A.B. is with IRISA/INRIA, Campus de Beaulieu, 35042 Rennes cedex, France; P.C. is with VERIMAG/CNRS, Centre Equation, 2 avenue de Vignate, 38610 Gières, France; M.d.N. is with Comput. Eng. & Sci. Dept., ReTiS Lab., Scuola Superiore S. Anna, Pisa, Italy; C.P. and S.T. are with Cadence Berkeley Labs, 1995 University Ave, Suite 460, Berkeley, CA 94704, USA; A.SV. is with EECS Dept., U.C. Berkeley, Berkeley, CA, USA.

Centre National de la Recherche Scientifique Institut National de Recherche en Informatique
(UMR 6074) Université de Rennes 1 – Insa de Rennes et en Automatique – unité de recherche de Rennes

(Résumé : tsvp)

Architectures Quasi-Synchrones et Communication par Echantillonnage

Résumé : On étudie une nouvelle architecture répartie pour les applications temps-réel, dont l'usage est en vigueur dans plusieurs secteurs industriels, dont l'avionique. Elle reprend de l'architecture TTA de Hermann Kopetz l'idée d'un assemblage de composants de calcul et de communication activés par le temps. Mais elle abandonne l'exigence de synchronisation stricte des horloges, qui fait le fondement de TTA. Il suffit que les horloges soient à peu près synchronisées, d'où le nom LTTA (Loosely TTA). On étudie comment, lors du déploiement, préserver la sémantique des données comme du temps.

Mots clés : systèmes temps-réel, systèmes répartis, synchrone, quasi-synchrone, TTA, LTTA

Contents

1 Introduction

In automotive and avionics applications, the propagation of information from one end to the other of a functional chain is typically implemented by a set of periodically activated tasks and messages. The execution platform is a distributed architecture consisting of several ECUs connected by buses or an interconnection network. Some of the communications between tasks must guarantee no loss of data. We refer to this constraint as *data (or stream) semantics preservation*. Furthermore, real-time constraints may be defined on the computation and communication latencies. We address the problem of mapping the functional requirements onto an implementation platform so that stream semantics preservation constraints are satisfied, timing constraints are met, and appropriate cost functions can be optimized.

In this paper, following the basic philosophy of Platform Based Design [36], we consider the following design flow:

1. We start from
   - a set of (possibly interacting) functions. Each function obeys the following model: it proceeds by a (possibly nonterminating) sequence of successive logical reactions composed of a finite set of actions; in addition, timing constraints such as periodicity and/or deadlines can be attached to each action.
   - an implementation platform consisting of interconnected ECUs, protocols for accessing the interconnection network and middleware for each ECU. The platform is characterized by its performance in terms of timing, capacity, power and cost.
2. We perform mapping of the functions onto the implementation architecture so that the constraints are verified and cost functions defined on the implementation architecture can be minimized.

The mapping problem is complex since the limitation of the implementation platform may make stream semantics preservation difficult to achieve. Often, when the implementation platform is event-based, guaranteeing that the logical and timing constraints are satisfied is an unsolved problem or requires a large overhead. A possible solution to this problem is to select the implementation architecture so that some of the constraints are satisfied by construction and the analysis can be carried out using formal methods.

The Time Triggered Architecture (TTA) proposed by Kopetz in [19] for the communication mechanism of a distributed platform offers a direct mapping that would preserve the semantics of the specification. It consists in implementing, on a distributed hardware, the real-time periodic synchronous model. Using this approach, correctness of a distributed implementation can be analyzed rigorously with formal techniques, see also [26]. However, this approach carries cost and timing penalties (see, for instance, [22, 24]) that may not be acceptable for some applications. In particular, TTA is not easily implementable for long wires (such as in systems where control intelligence is widely distributed) or for wireless communications.

Hence, there has been growing interest in less constrained architectures such as the Loosely Time-Triggered Architecture (LTTA) used in the aerospace industry and studied in [9, 35, 20, 21, 8, 4]. LTTA is characterized by the following features:

- access to the communication medium occurs quasi-periodically, using the different local clocks; while not synchronized, these clocks are bound to deviate from each other with limited drift and jitter. We call such clocks *quasi-periodic*.
- writings and readings are performed independently at all nodes connected to the medium in synchrony with the above mentioned local clocks;
- the communication medium behaves like a shared memory, i.e., values are sustained and are periodically refreshed, based on a local clock owned by the medium; how multi-user access to the communication medium is performed, is left unspecified.

This architecture prescribes the communication scheme but leaves several parts of an implementation unspecified offering designers further flexibility in the final implementation. In fact, the LTTA mechanisms can be either implemented in customized hardware, or built on top of existing distributed execution infrastructures — for example, CAN based networks as shown in [14].

The LTTA mechanism has not been fully characterized nor a complete design flow with potential design space exploration has been offered. In [37, 41, 38, 8] protocols for time-sensitive and LTT architectures were proposed to compensate for a change in latencies, from specification to implementation. In [9], sufficient conditions were given to ensure preservation of stream semantics. However, the results of [9] are specific to single-user case and provide no basis for a systematic extension to multi-user, multi-bus, communication. Recent work [14] extends the work of [9] by showing how cascade LTT communication can be implemented on top of a CAN based architecture.

In this paper, we propose a comprehensive design flow that maps functional requirements onto an LTTA with arbitrary topology that can be implemented in a variety of ways including a set of LTT buses. The design flow is guaranteed to produce a semantic preserving implementation if appropriate assumptions are satisfied.

Under the framework of Platform-based design, we define a process in which a functional model is mapped into a distributed LTTA architecture by means of intermediate abstractions (see Figure 1). The functional model is characterized by requirements of flow preservation on communications and execution rates on both computations and communications (top of the Figure). In the LTTA physical platform, nodes are characterized by clocks with limited jitter and drift. Furthermore, the execution platform does not provide clock synchronization (bottom of the Figure).

Our mapping uses a hierarchy of platforms, with the corresponding mappings *from* the functional layer and *to* the LTTA architecture. The platforms are defined based on the *communication by sampling* or *CbS* abstract model of computation. CbS is a quite general communication model and, by itself, cannot provide flow preservation, nor rate guarantees. A mapping on CbS into an LTTA architecture with the above described properties, can be

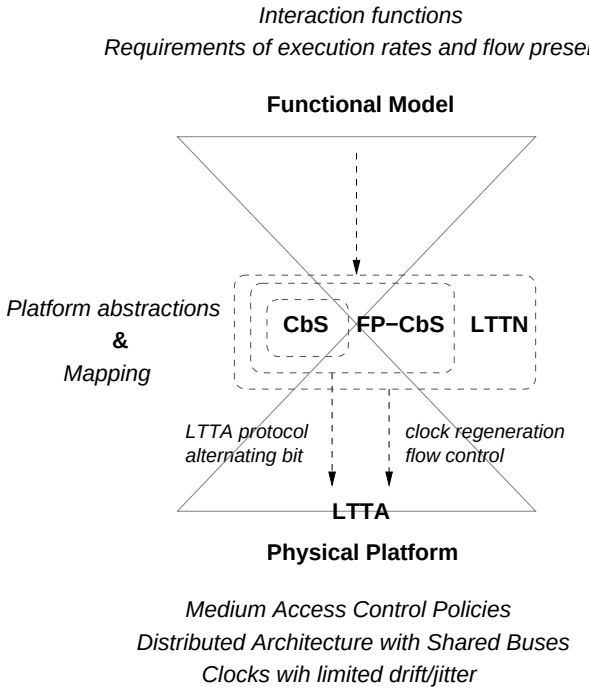

Figure 1: Mapping of a functional model into an LTTA architecture: Platforms and protocols.

defined so that a refined version of flow-preserving CbS (FP-CbS in the Figure) is obtained. At this stage, flow preservation is guaranteed on single writer-to-reader links. This mapping leverages a suitably defined LTTA protocol and refines CbS by restricting its semantics (in the Figure CbS is actually included meaning that an additional protocol layer is built on top of it.)

Another intermediate layer of abstraction called *Loosely Time-Triggered Network (LTTN)* is useful to extend our approach to general topologies. LTTN uses pure communication-by-sampling mechanisms. To guarantee that the mapping of the functional model on LTTN is semantic preserving for general end-to-end flows, we introduce two protocols: the first protocol performs *clock regeneration* to avoid cumulative slow down of clocks along directed communication paths; and the second protocol performs *predictive traffic shaping* to avoid writers to overload readers.

Our flow is completed by providing mappings of LTTN into several medium access control schemes for shared buses that may be available at the physical level.

The paper is organized as follows: we first present a mathematical toolkit in Section 2 that details the model of computation used throughout the paper. In particular, all

the needed protocols use a unique, fundamental, two-counter mechanism developed in this section, which enlightens the elegance of our LTT Architecture. Communication by Sampling is studied in detail in Section 3. Following this study, we propose a PBD approach to map a set of functional requirements on LTTA potentially implemented with a set of LTT busses. In particular, in Section 5 we present the LTTN and two protocols that guarantee semantics preservation on the mapping of the requirements onto the network. How to replace the ideal Communication by Sampling by an LTT bus and is analyzed in Section 7. Section 8 is devoted to multiple access; we study both Time Division and Priority Based Multiple Access. Finally, fundamental mechanisms ensuring fault tolerance are proposed in Section 9.

2 Mathematical toolkit

In this section we develop the toolkit we shall use throughout this study. This toolkit collects the building blocks of the LTTA MoCC, thus revealing its elegance by the small number of needed mechanisms.

2.1 An algebra of flows, daters, and counters

A mathematical model for CbS must handle flows, defined as successive dated occurrences of valued events. We first present the corresponding material, by building upon the pioneering work [11]. Symbol $\mathbb{N}$ denotes the positive integers: $\mathbb{N} = 1, 2, 3, \ldots$. Formally, *flows* are infinite sequences $e = (e_n)_{n\in\mathbb{N}}$ of *valued and dated events*.

The *value* of event e_n is denoted by ν^e_n. Unless ambiguity can result from such an overloading, we shall simply write e_n instead of ν^e_n. We call *stream of* e the sequence of values $(\nu^e_n)_{n\geq 0}$, where, by default, $\nu^e_0 = \star$, where symbol "$\star$" means *undefined*. A *clock* is a flow with values in the singleton set {tick}.

The *dater* of e is a sequence t^e such that $t^e_n \in \mathbb{R}_+$ is the date of e_n; denote by T^e the set of all dates of events of flow e. When the considered flow is a clock κ, by abuse of notation we also denote by κ its set of dates, instead of T^κ. Accordingly,

$$\text{we shall write } \kappa \subseteq \kappa' \text{ to mean } T^\kappa \subseteq T^{\kappa'}. \tag{1}$$

The *counter* of e is a non decreasing function $\mathbb{R}_+ \mapsto \mathbb{N}$ defined by

$$c^e_t \ =_{\text{def}}\ \text{Card}\{\ n \mid t^e_n \leq t\ \} \tag{2}$$

and we define the *strict counter* of e by

$$c^{e^-}_t \ =_{\text{def}}\ \lim_{s\nearrow t} c^e_s \ =\ \text{Card}\{\ n \mid t^e_n < t\ \} \tag{3}$$

The dater and the counter of a flow carry the same information. The counter can be obtained from the dater as shown above. Alternatively, the dater can be obtained from the counter as follows.

$$t^e_n = \min\{t \mid c^e_t = n\} \tag{4}$$

To be able to refer to "the date of the nth event", or "the index of the last event before t", or "the last value before t", etc, we will need in the sequel to compose the above operators. The following generic notation will be used for this purpose: the composition $t^e \circ c^e$ is defined by

$$(t^e \circ c^e)_t \ =_{\text{def}} \ t^e_{c^e_t}$$

Formally, $t^e \circ c^e$ is the composition of the two functions $t \mapsto c^e_t$ and $n \mapsto t^e_n$. To simplify the notations,

$$\text{we shall write } t^e \circ c^e_t \text{ instead of } (t^e \circ c^e)_t.$$

Thus, $t^e \circ c^e_t$ delivers, for flow e, the date of the event whose index is the last before or including t.

It will be at times needed to reason about delayed flows. Flows can be delayed, both logically (by passing them through a k-step shift register) and physically (by delaying the date of event occurrences). Delaying flow e logically by an amount of k is modeled by considering that date t^e_n of the nth occurrence of e is in fact the date of the $(n-k)$th event of the delayed flow e'; that is, e' is characterized by the delayed dater $t^{e'} = t^e \circ (Id + k)$, where Id is the identity function and the context allows to determine whether it operates on times or on indices. Thus, $t^{e'}_n = t^e_{n+k}$.

Similarly, delaying flow e physically by t means that the new flow e' is characterized by the dater $t^{e'}_n = t^e_n + t$. This can be modeled by considering that the number c^e_s of occurrences of e before time s is in fact the number of events of e' at time $s + t$; that is, e' can be characterized by the delayed counter $c^{e'} = c^e \circ (Id - t)$.

So far we discussed only daters and counters. The following macros are useful when considering values.

Interpolating flows. It will be useful to interpolate values between the occurrences of successive events. This is simply achieved by overloading the "value" operator ν^e_n:

$$\forall t \in \mathbb{R}_+ : \nu^e_t \ =_{\text{def}} \ \nu^e \circ c^e_t$$

The following *current* operator[1] is a variant of the former one. It sustains the last value seen in the strict past:

$$\forall t \in \mathbb{R}_+ : \nu^{e^-}_t \ =_{\text{def}} \ \nu^e \circ c^{e^-}_t$$

[1] The names "current" and "when" (used later in this text) are taken from similar operators found in the Lustre language [12].

For $t = t^e_n$, we get $\nu^{e^-}_t = \nu^e_{n-1}$ and $\nu^e_t = \nu^e_n$, whereas, for $t \notin T^e$, $\nu^{e^-}_t = \nu^e_t$ holds. When no confusion can result, we shall again use the

simplified notations e_t and e^-_t, instead of ν^e_t and $\nu^{e^-}_t$.

Regarding dates, the operator *last* provides at any time the last occurrence time of the flow:

$$l^e_t \ =_{\text{def}} \ t^e \circ c^e_t \tag{5}$$

Combining flows. The operator *when* filters the occurrence of flow e according to some predicate b provided synchronously with e (i.e., $T^b = T^e$):

$$\begin{array}{rcl} e \text{ when } b & = & \widehat{e}, \text{ where} \\ T^{\widehat{e}} & = & \{t^e_n \mid n \in \mathbb{N} \text{ and } \nu^b_n = true\} \\ \forall t \in T^{\widehat{e}} : \nu^{\widehat{e}}_t & = & \nu^e_t \end{array} \tag{6}$$

The operator *at* delivers, at each occurrence of some flow κ, the current value of flow e:

$$\begin{array}{rcl} e \text{ at } \kappa & = & \widehat{e}, \text{ where} \\ T^{\widehat{e}} & = & T^\kappa \\ \forall n \in \mathbb{N} : \ \nu^{\widehat{e}}_n & = & \nu^e \circ c^e \circ t^\kappa_n \end{array} \tag{7}$$

2.2 Semantics preservation

Consider two flows e_1 and e_2. We say that *flow* e_2 *stream preserves flow* e_1 if the following holds:

$$\forall n \in \mathbb{N} : \nu^{e_1}_n \ = \ \nu^{e_2}_n. \tag{8}$$

Stream preservation between computing units guarantees that the considered distributed architecture is GALS (Globally Asynchronous, Locally Synchronous), so that techniques from [31, 32, 17] can be used to ensure correct-by-construction deployment of synchronous (or polychronous) specifications.

The so defined stream preservation does not account for timing issues. Strict preservation of timing is too strong and irrelevant for LTTA. Instead, we shall complement stream preservation with bounds on the relative periods and jitters, for the considered flows, in each case.

3 Communication by Sampling

Communication by Sampling (CbS) is the only basic building block of our LTT Architecture. CbS involves pairs of the form {writer, reader}. It is formalized using a composition operator $\triangleright$ between flows. First we discuss this operator informally and the define it formally and provide some of its properties. Denote by w and r the flows of writings and readings. Then,

$$w \triangleright r$$

is the flow collecting the successive deliveries, by the reader, of the successive writes. That is:

- if w performs a writing that is overwritten by a subsequent writing before being read by r, then no corresponding event for flow $w \triangleright r$ is produced;
- if r performs a reading that follows a previous reading without having a writing occurring between the two, then no corresponding event for flow $w \triangleright r$ is produced;
- if w performs a writing that is followed by a corresponding reading of r prior to a next writing, then an event for flow $w \triangleright r$ is issued at the time of that reading; equivalently, if r performs a reading that follows a corresponding writing of w, then an event for flow $w \triangleright r$ is issued at the time of this reading.

Operator $w \triangleright r$ is illustrated on Figure 2. Note that, despite the notation, $w \triangleright r$ depends on the flow w of writings, but it depends on r only through its clock κ_r. So, we have

$$w \triangleright r \;=\; w \triangleright \kappa_r, \tag{9}$$

which enlightens the causal dependency, from the pair (w, κ_r), to the output stream ν^r.

We shall now formalize this description. To define $w \triangleright r$, we need to define two things: its "timing aspect", that is, its dater; and its "value aspect", that is, its sequence of values.

Timing aspect. The dater of flow $w \triangleright r$ is characterized by its counter:

$$\begin{aligned} & c_t^{w \triangleright r} \\ =\; & \mathrm{Card}\left\{ t_k^r \;\middle|\; \left[t_k^r \le t\right] \wedge \left[c_{t_k^r}^{w^-} - c_{t_{k-1}^r}^{w^-} \ge 1\right] \right\} && (10) \\ =\; & \mathrm{Card}\left\{ t_k^w \;\middle|\; \left[t_k^w \le t\right] \wedge \left[c_{\min(t, t_{k+1}^w)}^{r} - c_{t_k^w}^{r} \ge 1\right] \right\} && (11) \end{aligned}$$

Formula (10) consists in counting the number of reads that are preceded by at least one write. Dual formula (11) consists in counting the number of writes that are followed by at least one read before being overwritten. These two formulas are illustrated in Figure 2.

Notice that the dater $t_n^{w \triangleright r}$ can be derived from the counter of $w \triangleright r$ as shown in Equation (4).

Value aspect. Regarding values, the nth value read by the reader is the currently written value at the time of the nth read:

$$\nu_n^{w \triangleright r} \;=\; w^- \circ t_n^{w \triangleright r} \tag{12}$$

This completes the definition of flow $w \triangleright r$.

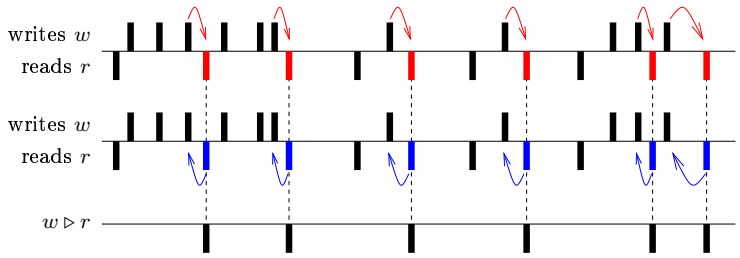

Figure 2: Illustrating: $w \triangleright r$–bottom; formula (10)–middle; and formula (11)–top. The origin of each arrow points: for formula (10), to a read event (in blue) satisfying the associated predicate, and, for formula (11), to a write event satisfying the associated predicate. The end of each arrow points to the event that realizes satisfaction of the associated predicate.

3.1 Effective procedure for computing $c^{w \triangleright r}$, and properties

Using formulas (10) and (11), the following precise description of $c^{w \triangleright r}$ can be given, by switching between these two formulas at appropriate times. To get such a formula, key remarks are:

- Suppose that, for some $t > 0$, condition

$$c^{w^-}_{t^r_k} - c^{w^-}_{t^r_{k-1}} \geq 1 \tag{13}$$

  is satisfied for every $t^r_k \leq t$. Then, applying formula (10) simply yields $c^{w \triangleright r}_t = c^r_t$. The set of t's satisfying this property is an interval of the form $[0, t_{(10)})$, where $t_{(10)} \geq 0$.

- Alternatively, suppose that, for some $t > 0$, condition

$$c^r_{\min(t, t^w_{k+1})} - c^r_{t^w_k} \geq 1 \tag{14}$$

  is satisfied for every $t^w_k \leq t$. Then, applying formula (11) simply yields $c^{w \triangleright r}_t = c^w_t$. The set of t's satisfying this property is an interval of the form $[0, t_{(11)})$, where $t_{(11)} \geq 0$.

When a read occurs, we know that no current write is pending, and thus we can regard this read event as if it was the origin of times: $t = 0$. Thus, we can repeat the above reasoning starting from any read event. Read events of interest for doing this are obviously those where one switches between the above two cases.

More precisely: call *switch flow of* $w \triangleright r$ the flow ℓ consisting of the minimal (for set inclusion) subset of events of r such that, within the interval $(t^\ell_{m-1}, t^\ell_m]$ between two successive events of ℓ:

- either Condition (13) is satisfied for every interval

$$(t^r_{k-1}, t^r_k] \subseteq (t^\ell_{m-1}, t^\ell_m],$$

expressing that there are more writes than reads in the considered interval. In such an interval, we have

$$\forall t \in T^r \cap (t^\ell_{m-1}, t^\ell_m] \ : \ c^{w \triangleright r}_t - c^{w \triangleright r}_{t^\ell_{m-1}} = c^r_t - c^r_{t^\ell_{m-1}}$$

Hence we say that such an interval is *of type read.*

- or Condition (14) is satisfied for every interval

$$(t^w_k, t^r \circ c^r \circ t^w_{k+1}] \subseteq (t^\ell_{m-1}, t^\ell_m],$$

expressing that there are more reads than writes in the considered interval. By definition of function composition, we have

$$t^r \circ c^r \circ t^w_{k+1} \ = \ t^r_{c^r_{t^w_{k+1}}} \ = \ \max\{t^r_m \mid t^r_m \leq t^w_{k+1}\}. \tag{15}$$

In such an interval, we have

$$\forall t \in T^r \cap (t^\ell_{m-1}, t^\ell_m] \ : \ c^{w \triangleright r}_t - c^{w \triangleright r}_{t^\ell_{m-1}} = c^w_t - c^w_{t^\ell_{m-1}}$$

Hence we say that such an interval is *of type write.*

Intervals of type read and of type write alternate, and the evaluation of $c^{w \triangleright r}_t$ is performed accordingly. This mechanism is illustrated on Figure 3. The switch from type write to type

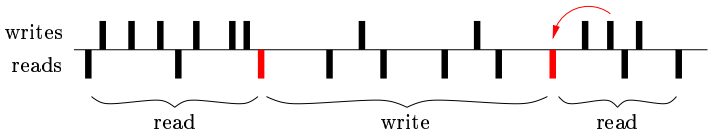

Figure 3: Illustrating the evaluation of $c^{w \triangleright r}_t$. The switching times between the two modes are shown in red.
read is discovered at events of w. However it must be implemented at an event of r, whence the backtracking shown by the backward pointing arrow in the figure, which actually realizes formula (15).

From this analysis, a number of consequences can be drawn:

Property 1 (Communication by sampling)

1. *If $c^{w \triangleright r}_t = c^w_t$ holds, then $w \triangleright r$ stream preserves w, meaning that $\nu^{w \triangleright r} = \nu^w$.*
2. *If writes are more frequent than reads (formally, type read always holds), then, for every $t \in T^r$, $c^{w \triangleright r}_t = c^r_t$ holds, i.e., no read is superfluous.*
3. *If reads are more frequent than writes (formally, type write always holds), then, for every $t \in T^r$, $c^{w \triangleright r}_t = c^w_t$ holds. Thus, in this case, $w \triangleright r$ stream preserves w.*

4. *We always have* $c_t^{w\triangleright r} \leq \min(c_t^w, c_t^r)$. *Furthermore, this condition refines as follows:*

$$\begin{aligned} & \forall t \in T^r \ : \ c_t^{w\triangleright r} \\ = & \max_{0=t_0<t_1<\cdots<t_{m-1}<t_m=t} \\ & \sum_{k=1}^{m} \min\left(c_{t_k}^w - c_{t_{k-1}}^w, c_{t_k}^r - c_{t_{k-1}}^r\right), \end{aligned} \tag{16}$$

where the maximum ranges over the set of all finite partitions of interval $(0,t]$. *Equality is reached if* $0 = t_0 < t_1 < \cdots < t_{m-1} < t_m = t$ *are the successive dates of the switch flow of* $w \triangleright r$.

5. *We have*

$$\begin{aligned} \inf_{k>0}(t_k^r - t_{k-1}^r) \ \leq \ & \inf_{k>0}(t_k^{w\triangleright r} - t_{k-1}^{w\triangleright r}) \ \leq \ \sup_{k>0}(t_k^{w\triangleright r} - t_{k-1}^{w\triangleright r}) \\ & \leq \ 2\sup\big(\sup_{k>0}(t_k^w - t_{k-1}^w)\,,\, \sup_{k>0}(t_k^r - t_{k-1}^r)\big) \end{aligned} \tag{17}$$

Statement 1 expresses that the preservation of counters guarantees stream preservation. Statements 2 and 3 describe the behaviour of communication by sampling in case of slow/fast and fast/slow modes for the pair {writer, reader}. Formula (16) provides exact evaluation of $w\triangleright r$. Finally, formula (17) shows how lower and upper bounds for delays between successive events are preserved by communication by sampling.

Proof: Statements 1 – 4 are immediate. So we focus on statement 5. Assume that successive writes and successive reads are separated by at least duration δ. Then, flow $w \triangleright r$ cannot have two successive events in less than δ. This shows the first inequality of (17).

To prove the second inequality of (17), assume that two successive reads always occur within less than Δ, and the same holds for two successive writes. Then, assume that there exists some $k > 0$ such that $t_k^{w\triangleright r} - t_{k-1}^{w\triangleright r} > 2\Delta$. By assumption on w, we must have at least two writes and two reads in the open interval $(t_{k-1}^{w\triangleright r}, t_k^{w\triangleright r})$. Suppose that one of these writes occurs before one of these reads. Then, this would have resulted in an extra occurrence of flow $w \triangleright r$ in the considered interval; a contradiction. Therefore, no read can follow a write in the considered open interval. Let w_{m_k} be the first write and r_{n_k} be the last read in the interval $(t_{k-1}^{w\triangleright r}, t_k^{w\triangleright r})$. Then, one of the two inequalities must hold, namely: either $t_{n_k+1}^r - t_{n_k}^r > \Delta$, or $t_{m_k}^w - t_{m_k-1}^w > \Delta$. This contradicts the assumption and proves the second inequality of (17).

3.2 Taking latencies into account

So far we have ignored delay in the operator $\triangleright$: we have assumed that what is written is immediately available for reading. Clearly, this is rarely the case in practice, where various types of latencies are introduced between a writer and a reader, including program and

operating system execution, communication, etc. To account for latencies, we consider the physically-delayed flow w', related to w by $\forall n : \nu_n^{w'} = \nu_n^w$ and

$$t_k^{w'} \;=\; t_k^w + \delta_k, \tag{18}$$

where δ_k is a (possibly variable) positive latency. Since latencies δ_k vary, it is possible that the order of events is reverted in w': for instance, if $\delta_k - \delta_{k+1} > t_{k+1}^w - t_k^w$, then we have $t_k^{w'} = t_k^w + \delta_k > t_{k+1}^w + \delta_{k+1} = t_{k+1}^{w'}$. We wish to forbid this, therefore we make the following assumption.

Assumption 1 *Latencies δ_k do not revert data:*

$$\forall k \in \mathbb{N} : t_{k+1}^{w'} \;\geq\; t_k^{w'} \tag{19}$$

Using (19), (18) can be rewritten as $\forall t \in \mathbb{R}_+ : c^{w'}_{t+\delta_{c_t^w}} = c_t^w$, also written as

$$c^{w'} \circ (Id + \delta_{c^w}) \;=\; c^w \tag{20}$$

We denote by δ the flow of latencies $\delta_k, k \in \mathbb{N}$, and by

$$\delta\,[w] \tag{21}$$

the flow w' related to w via (18) or (20). Whenever needed, all the results we provide in the sequel can be adapted to handle latencies, by replacing a considered flow e by its delayed version $\delta\,[e]$.

3.3 Buffered Communication by Sampling

Up to now, we have considered basic CbS, where the communication medium behaves like a shared memory. It is of interest to extend this mechanism with bounded buffers, as follows. We assume that the reader is equipped with a buffer of size M. The buffering mechanism, illustrated in Figure 4, is as follows:

- When the buffer is full, an additional writing puts the fresh data in place 1 of the buffer, and shifts by 1 place all data previously sitting in the buffer. This causes the loss of the oldest data, sitting in place M.
- Readings from the buffer get the oldest data from it, i.e., the data sitting at place $N_t^{w,r}$, where $N_t^{w,r}$ is the buffer level at instant t.
- When the buffer level is > 1, then readings consume the data. Alternatively, when the buffer level equals 1, then readings do not consume the data.
- The buffer is initially non-empty.

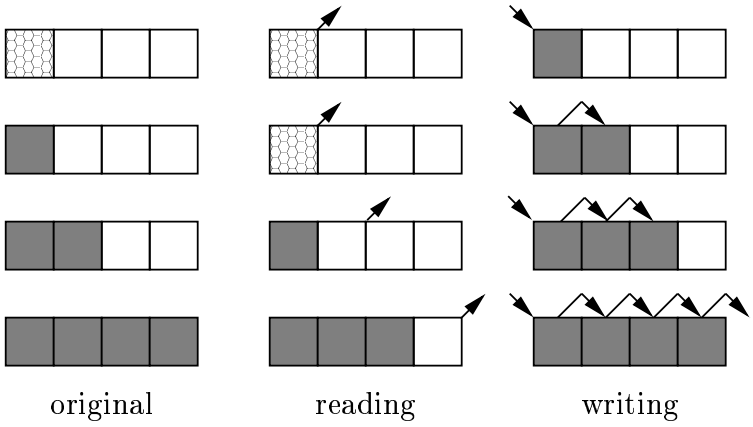

Figure 4: Buffered Communication by Sampling. Places in the buffer are indexed, from left to right, by $1, 2, 3, 4$. Four scenarios are shown in the figure (from top to bottom). Each scenario illustrates how the original buffer is transformed by a read or a write operation (the write operation is applied to the original buffer and not to the buffer resulting after the read). Grey boxes indicate places filled with data; white boxes are empty; and paved white boxes indicate that the data is removed except when this would make the buffer empty. Arrows indicate the move of data. When writing, a paved box is regarded as empty, as depicted in the first row.

This mechanism is non blocking, for both writings and readings, and reduces to basic CbS when $M = 1$. Note the first row in Figure 4, which illustrates that the data is not consumed when buffer level is 1. This mechanism is denoted by

$$w \triangleright_M r \tag{22}$$

It is formalized next. For x and y two flows, set

$$\begin{aligned} c^x_{s,t} &= c^x_t - c^x_s \\ N^{x,y}_{s,t} &= (c^x_{s,t} - c^y_{s,t}) - \min_{u\in[s,t]} (c^x_{s,u} - c^y_{s,u}) \end{aligned} \tag{23}$$

(Note that (23) generalizes the two-counter mechanism (34) to the case where the initial instant is s instead of 0.)

Let w and r be the input and output flows of $w \triangleright_M r$, respectively. The clocks of w and r are the writer's and reader's clocks, respectively. In the following equations, vector flow $w[0, \ldots, M-1]$ is a buffer of size M, whose components are denoted by $w[k]$ for

$k = 0, \ldots, M-1$. For every $m = 0, \ldots, M-1$ and $s \in \mathbb{R}_+$, set

$$\begin{array}{rcl} t^+[m,s] & = & \min\left\{t \in T^w \mid t > s \text{ and } m + N_{s,t}^{w,r} \geq M-1\right\} \\ t^-[m,s] & = & \min\left\{t \in T^r \mid t > s \text{ and } m + N_{s,t}^{w,r} \leq 0\right\} \\ t[m,s] & = & \min(t^+[m,s]\,,\, t^-[m,s]) \\ p[m,s] & = & \text{if } t[m,s] = t^-[m,s] \text{ then } 0 \text{ else } M-1 \end{array} \tag{24}$$

where, by convention, the minimum of the empty set is $+\infty$. Instant $t^+[m,s]$ is the first instant where the cumulated excess of writings over readings starting from level m at time s, reaches $M-1$. Symmetrically, $t^-[m,s]$ is the first instant where the cumulated excess of writings over readings starting from level m at time s, reaches 0. Finally, if the buffer level is m at time s, $t[m,s]$ is the first instant where the buffer reaches one of the two boundaries 0 or $M-1$, and $p[m,s]$ gives the buffer level reached at that time. Define inductively the following sequence of instants and buffer levels:

$$\begin{array}{rcl@{\quad,\quad}rcl} t_0 & = & 0 & m_0 & = & 0 \\ t_{n+1} & = & t[m_n, t_n] & m_{n+1} & = & p[m_n, t_n] \end{array} \tag{25}$$

If $t_n = +\infty$, then the subsequent terms of the sequence are also infinite. Then, for every $t \in \mathbb{R}_+$, the buffer level at time t is equal to:

$$t_n \leq t \leq t_{n+1} \;\Rightarrow\; N_t^{x,y} = m_n + N_{t_n,t}^{w,r} \tag{26}$$

And, finally, the buffer contents and output of the mechanism are given by:

$$\begin{array}{rcl} w[0]_t & = & w_t \\ \forall k = 1, \ldots, N_t^{w,r} \;\Rightarrow\; w[k]_t & = & w[k-1]_t^- \\ r_t & = & w[N_t^{w,r}]_t \end{array} \tag{27}$$

Equations (24)–(27) formalize buffered communication $w \triangleright_M r$. Observe that these equations are easily implemented in an on-line form. When

$$t_1 \;=\; +\infty \tag{28}$$

holds, i.e., the buffer never gets full, Equations (24)–(27) simplify as follows:

$$\begin{array}{rcl} N_t^{w,r} & = & N_{0,t}^{w,r} \\ w[0]_t & = & w_t \\ \forall k = 1, \ldots, N_t^{w,r} \;\Rightarrow\; w[k]_t & = & w[k-1]_t^- \\ r_t & = & w[N_t^{w,r}]_t \end{array} \tag{29}$$

4 Single write/read communication over an LTT bus

The next step in our design flow is to map the functional requirements and the communication-by-sampling mechanism over an LTTA. We proceed to solve the general problem by first solving the problem of mapping a single write/read communication by sampling mechanism over an LTT bus. That is, in this section, we consider a triple

$$\{\text{writer } w,\ \text{bus } b,\ \text{reader } r\}$$

communicating by sampling, in this order. The overall model of this communication medium is described by

$$(w \triangleright b) \triangleright r \tag{30}$$

Not surprisingly, due to (9), $(w \triangleright b) \triangleright r \neq w \triangleright (b \triangleright r)$ in general. Thus we prefer keeping explicit bracketing when cascading CbS communications.

Using notation (21), LTT bus with latency is modeled by

$$(\delta\,[w \triangleright b]) \triangleright r \tag{31}$$

Suppose that, in communication model (30), bus b implements M-buffered CbS communication, whence the overall communication, from writer to bus to reader, is modelled as follows, see (22):

$$(w \triangleright_M b) \triangleright r$$

Then, sufficient conditions for stream preservation are the following:

Property 2 (LTT bus with buffer) *Assume that the writing, bus transmission, and reading times satisfy the following conditions:*

1. *For every* $k > 0$, $\quad c^b \circ t^w_k - c^b \circ t^w_{k-M} \geq M$ *holds.*
2. *Type* write *always holds for* $(w \circ (Id - N^{w,b}_t)) \triangleright b$, *where* $N^{w,b}_t$ *is defined in (34).*

Then, $(w \triangleright_M b) \triangleright r$ *stream preserves* w.

Proof: Condition 1 is just Hypothesis $\mathbf{H}_2$ of Section 6.1, reformulated for the pair $(x, y) = (w, b)$. Thus, if flow w puts tokens in the bus buffer for (possibly immediate) consumption, then this buffer will never overflow. Equivalently, Condition 1 ensures that no write will be missed by the bus. Note that this condition is equivalently reformulated as $\forall t \in \mathbb{R}_+, N^{w,b}_t \leq M - 1$. Thus we are in the case (28) where formulas (29) are valid for describing $w \triangleright_M b$.

Let us now focus on Condition 2. Note that, for every t positive, $\nu^w \circ (Id - N^{w,b})_t$ is the most recent relevant value input by the writer, that is output by the bus at or before time t. Hence, by Property 1.3, Condition 2 ensures that the reader will not miss any such data. $\diamond$

More effective conditions can be stated that imply Conditions 1 and 2 of Property 2 and refine the conditions of Theorem 1 of [9], for the case of buffered bus communications with variable periods:

Corollary 1 (LTT bus with buffer) *Assume that the writing, bus and reading times satisfy the following equations, where δ^b, Δ^b, etc. are positive finite constants:*

$$\begin{array}{rrcl} \forall n: & \delta^w & \leq & t^w_{n+M} - t^w_n \\ \forall p: & \delta^b & \leq & t^b_{p+M} - t^b_p \leq \Delta^b \\ \forall m: & & & t^r_{m+M} - t^r_m \leq \Delta^r \end{array} \tag{32}$$

If

$$\delta^w \geq \Delta^b \text{ and } \left\lfloor \frac{\delta^w}{\Delta^b} \right\rfloor \geq \frac{\Delta^r}{\delta^b} \tag{33}$$

where $\lfloor x \rfloor$ denotes the largest integer $\leq x$, then $(w \triangleright_M b) \triangleright r$ stream preserves w.

Proof: First, note that condition $\delta^w \geq \Delta^b$ implies Condition 1 of Property 2. Next, $\lfloor \frac{\delta^w}{\Delta^b} \rfloor$ gives the minimal number of ticks of the bus clock that separate two successive writes (see the proof of Theorem 1 of [9] for the details of this argument). Finally, $\lfloor \frac{\Delta^r}{\delta^b} \rfloor$ is the maximal number of ticks of the bus clock that separate two successive reads. Thus, the second condition of (33) implies condition 2 of Property 2. This proves the corollary. ◇

Comment. Discrepancy between lower and upper bounds for the time-varying periods in (32) is due to a combination of reasons. First, relative drift between physical clock exists as soon as clocks are not strictly synchronized. Next, low level mechanisms of the write, read, and fetch tasks in the communication medium causes jitter, that is, fluctuations in the period that do not accumulate. Jitter amplitude is typically much larger than drift amplitude. On one time period, both variations are subsumed by the single mechanism of lower and upper bounds of (32). However, this no longer holds if several successive periods are considered: while drifts accumulate, jitter does not. Therefore, in the long run, accumulated drift will dominate jitter. To address this, some studies introduce explicitly drift and jitter in the time model of the communication medium, e.g., see [14]. This is particularly important if buffered transmission is considered. ◇

5 The LTT communication Network (LTTN)

The general case of arbitrary communication topology deployed on top of possibly several buses requires additional work, because the conditions of Corollary 1 essentially require that the writer must be slower than the reader.

To cope with this problem, as a next step, we consider mapping over a LTT Network, that is an ideal communication network in which every communication occurs according to the exact (unbuffered) CbS scheme modeled in Section 3.

Formally, we consider a network involving computing units $C_i, i \in I$, where set I of sites is finite. Each site C_i is equipped with a quasi-periodic clock κ_i. Clocks κ_i are loosely, not strictly, synchronized (this is formalized later). The network is modeled as a directed graph $\mathcal{G}$ having $C_i, i \in I$ as set of vertices. Having a branch $C_i \rightarrow C_j$ in $\mathcal{G}$ means that C_i acts as a writer w_i and C_j acts as a reader r_j in a point to point CbS communication. Bi-directional and ring communications are allowed, thus $\mathcal{G}$ can have loops.

Communication $C_i \rightarrow C_j$ is by sampling, with the following characteristics (a) – (d), where $\kappa \downarrow 2$ denotes clock κ, down sampled by a factor of 2, i.e.,

$$\forall n \geq 0 : t_n^{\kappa \downarrow 2} = t_{2n}^{\kappa}$$

(a) Loose synchronization: for evey pair (i, j) of sites such that $C_i \rightarrow C_j$ is a branch of $\mathcal{G}$, pair $(\kappa_i \downarrow 2, \kappa_j)$ satisfies the assumption of Property 1.3, i.e., clock κ_j is more frequent than downsampled clock $\kappa_i \downarrow 2$.

(b) Writes w_i are tentatively triggered by clock $\kappa_i \downarrow 2$. Effective writes occur at a clock κ_i^w, downsampled from $\kappa_i \downarrow 2$ by traffic shaping [23], see below for a detailed description. A key feature of our traffic shaping policy is that it only depends on the effective writer's clocks for the different sites, not on the messages transferred.

(c) Reads r_j are triggered by clock κ_j. To cope with upsampling, from writer's clock κ_i^w to reader's clock κ_j, each message written by w_i comes equipped with an additional one-bit stamp that alternates between values 0 and 1. Alternations in the messages read by r_j, from 0 to 1 and from 1 to 0, indicate a fresh value. Other occurrences of w_i are repetitions. Together with the loose synchronization property (a), this one-bit stamp is in charge of maintaining data semantics, in the Kahn process network sense (see later).

(d) Only fresh data are used in performing writes w_j at rate $\kappa_j^w \subseteq \kappa_j \downarrow 2$. This operation is called "clock regeneration" and is studied next.

Communication by sampling, from writer w_i to reader r_j, is illustrated on Figure 5.

We now have to link the LTTN to the underlying LTTA and to optimize the use of LTTA resources. In LTTA, being loosely synchronized, the physical clocks may suffer from relative drift and jitter. If not compensated for, clock drifts cause buffer overflows, thus resulting in loss of data and lack of semantical preservation. Thus, a set of protocols will be developed to ensure that the set of distributed logical clocks built on top of the loosely synchronized physical clocks exhibit no relative drift, but only bounded jitter.

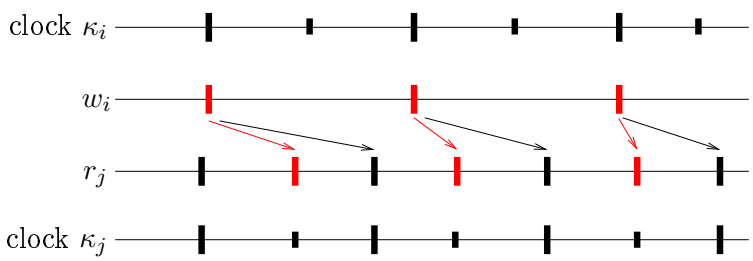

Figure 5: Illustrating communication in LTT network. Ticks of clock $\kappa \downarrow 2$ are indicated by large ticks of clock κ. Red bars indicate fresh data. Directed arrows indicate copying values.

6 Protocols to adapt LTTA to LTTN semantics

The two protocols introduced here (Clock Regeneration and Predictive Traffic Shaping) use a simple two-counter mechanism that we introduce first.

6.1 A generic two-counter monitoring protocol

All the protocols needed for LTTA will rely on a unique mechanism that we describe now. Consider two flows x and y, together with one of the following two hypotheses:

$\mathbf{H}_1$: there exists a real number $D > 0$ such that, for every $k > 0$, $t^y_k - t^y_{k-1} < D < t^x_k - t^x_{k-1}$ holds.

$\mathbf{H}_2$: there exists an integer $M > 0$ such that, for every $k > M$, $c^y \circ t^x_k - c^y \circ t^x_{k-M} \geq M$ holds.

Hypothesis $\mathbf{H}_1$ expresses that the (possibly time varying) period of flow y possesses an upper bound that is also a lower bound for the period of flow x. Hypothesis $\mathbf{H}_2$ expresses that, if flow x puts tokens in a buffer for (possibly immediate) consumption by flow y, then this buffer will never overflow provided that it has size at least M. Define the following quantity, for $t \in \mathbb{R}_+$:

$$N^{x,y}_t =_{\text{def}} (c^x_t - c^y_t) - \min_{s \in [0,t]} (c^x_s - c^y_s) = c^x_{s_t,t} - c^y_{s_t,t} \tag{34}$$

where

$$c^x_{s,t} =_{\text{def}} c^x_t - c^x_s$$

and s_t is the last instant where the minimum over $s \in [0,t]$ is reached in (34) — s_t exists since counters are right continuous. Then, under $\mathbf{H}_1$ we have $\forall t \in \mathbb{R}_+$, $N^{x,y}_t \leq 1$ and, under $\mathbf{H}_2$ we have $\forall t \in \mathbb{R}_+$, $N^{x,y}_t \leq M$. So, monitoring the violation of either hypothesis is performed by maintaining counter $N^{x,y}$ and comparing it with the appropriate threshold.

However, formula (34) defining $N^{x,y}$ is not suitable for on-line evaluation. We shall thus reformulate (34) in an on-line form. First, note that, although it is integer valued, $N^{x,y}$ is not a counter associated to some flow in the sense of Section 2.1. We must regard it as a flow itself. Seen as a flow, $N^{x,y}$ has set $T^{N^{x,y}}$ of dates of occurences, and sequence of values $\nu_n^{N^{x,y}}$, for $n = 1, 2, \ldots$, which we shall denote by $N_n^{x,y}$, with the abuse of notation proposed in the beginning of Section 2.1. This being said, note that

$$T^{N^{x,y}} \;=\; T^x \cup T^y. \tag{35}$$

Then, let z be the flow with values in the set $\{-1, 0, +1\}$, such that $T^z = T^x \cup T^y$, and whose value at a given occurence is $+1$ if x occurred alone, -1 if y occurred alone, and 0 if both x and y occurred simultaneously. Formally:

$$\begin{aligned} \nu^z \;&\in\; \{-1, 0, +1\} \\ T^z \;&=\; T^x \cup T^y \\ \forall t \in T^z : \nu_t^z \;&=\; \begin{cases} +1 \text{ if } x \text{ occurred alone at } t \\ -1 \text{ if } y \text{ occurred alone at } t \\ \;\;0 \text{ if otherwise} \end{cases} \end{aligned} \tag{36}$$

Then, the following recursive formula for evaluating the successive values $N_1^{x,y}, N_2^{x,y}, \ldots$ of flow $N^{x,y}$ holds: $N_0^{x,y} = 0$, and, for every $n > 0$:

$$N_n^{x,y} \;=\; \max\big(\, N_{n-1}^{x,y} + z_n \,,\, 0 \,\big) \tag{37}$$

This mechanism (35–37) for monitoring the violation of an hypothesis of the form $\mathbf{H}_1$ or $\mathbf{H}_2$ will be used in several contexts to develop our LTT architecture.

Comment: link with statistics. On-line algorithm (37) for monitoring buffer level possesses some interesting robustness properties. We introduced it for the on-line monitoring of buffer level in a deterministic setting, with hard bounds specified by hypotheses $\mathbf{H}_1$ or $\mathbf{H}_2$.

Now, suppose we are instead in an adaptive, Quality of Service oriented, context [16] where we do not seek for zero loss in buffers, but with "low probability" loss in situations where reads are more frequent than writes in the average. This is formalized by equipping flow z defined in (36) with a probability law making the successive values of z probabilistically independent with $\mathbb{P}(\nu^z = -1, 0, +1) = \alpha, \beta, \gamma$, where $\alpha + \beta + \gamma = 1$. Expressing that reads are probabilistically more frequent than writes is then formalized by the condition $\alpha > \gamma$. As long as this condition holds, then the probability of buffer overflow can be made small by proper sizing of the buffer.

Problems arise if condition $\alpha > \gamma$ gets violated, say, if $\alpha < \gamma$ becomes true at some time for some reason. Suppose, for simplicity, that $\alpha - \gamma$ changes, from η to $-\eta$ for some positive parameter η (excess read becomes symmetric excess write). Then, it turns out that comparing $N_n^{x,y}$ computed as in (37) to a certain threshold is optimal for detecting this

change, in a statistical sense [27]—in statistics, this procedure is known as the Page-Hinkley Cumulative Sum test [6].

To summarize, mechanism (35–37) is good for deterministic buffer level monitoring and possesses additional virtues regarding robustness with respect to uncertainties.

6.2 Clock Regeneration

The preservation of stream semantics is formalized by condition (8). To guarantee (8), statement 3 of Property 1 essentially requires that the writer shall be slower than the reader—what "slower" means is quantified precisely in the referred statement. Compensating for the ever decreasing sampling period in cascade communications is performed by a clock regeneration protocol, implemented on each computing unit C, see Figure 6. This protocol

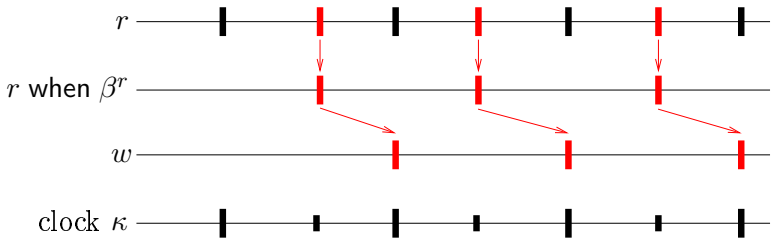

Figure 6: Illustrating clock regeneration Protocol 1. See Figure 5 for graphical conventions.

performs consistent downsampling of the input data read at rate κ, for re-emission at a rate (possibly downsampled from) $\kappa\downarrow 2$. The function performed by this protocol is to discard replicates, while properly emitting fresh data with no loss. This protocol is formalized next.

Protocol 1 (Clock Regeneration protocol) *C possesses a quasi-periodic clock κ and hosts a reader r and a writer w. Reader r and writer w are driven by κ and κ^w, respectively. Clock κ^w is some clock downsampled from $\kappa\downarrow 2$, i.e., $\kappa^w \subseteq \kappa\downarrow 2$, see (1), resulting from the traffic shaping policy described later. Read flow r has value of the following type*

$$\forall n \geq 0 : r_n \in D \times \{0,1\}$$

meaning that, for every n, the nth value of flow r is marked by a bit, denoted by β^r_n.

Then, site C prepares flow w for its output by selecting the fresh reads and discarding repetitions, i.e., w is the output of 2-buffered CbS communication $\widehat{r} \triangleright_2 \kappa^w$, where

$$\widehat{r} \;=\; r \text{ when } [\beta^r \neq \beta^r \circ (Id - 1)]$$

Buffered CbS communication is defined in Section 3.3 and operator " when " is defined in Section 2.1. The relevant information transmitted by our LTT network is captured by the set of reads "r when β^r" and writes "w", attached to each site C. Protocol 1 is illustrated on Figure 7, obtained by "merging" Figures 5 and 6.

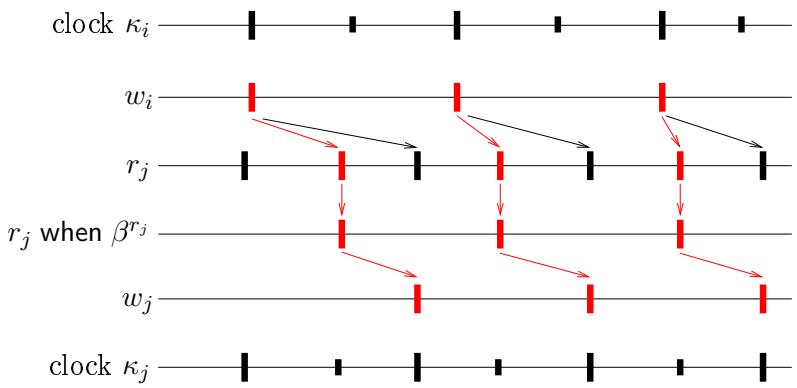

Figure 7: Combining $C_i \rightarrow C_j$ communication with Clock Regeneration Protocol at C_j.

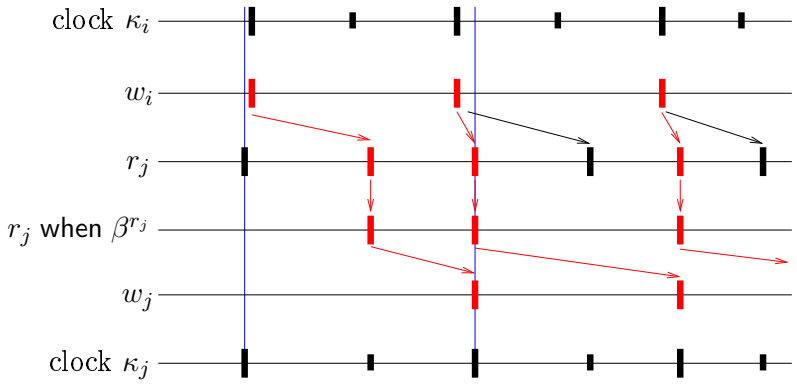

Figure 8: Combining $C_i \rightarrow C_j$ communication with Clock Regeneration Protocol at C_j. A situation where buffering is needed, compare with Figure 7. The problem is that two ticks of $\kappa_i \downarrow 2$ occur between the first two ticks of $\kappa_j \downarrow 2$.

So far Figure 7 shows the favorable case, where stream semantics is preserved without the need for a 2-buffer. However, Figure 8 shows that problems can occur if a 1-buffer only is used, despite our technique of downsampling, when clock κ_j is slower than clock κ_i. In this figure, the buffer gets filled at the second blue thin bar. It may never get empty, and may eventually overflow later, thus violating the preservation of stream semantics.

To overcome this problem, we use *predictive traffic shaping* at the writer. The idea is that the writer would implement some LTT based monitor detecting the risk of overwhelming its corresponding reader—see below. When approaching a risky situation, the writer would

then anticipate and decide to skip a clock cycle and delay its transmission to the next cycle. This is formalized next.

6.3 Predictive traffic shaping

Predictive traffic shaping is implemented at site i in the following way. Assume for a while the following regarding network $\mathcal{G}$:

Assumption 2 *For each direct link $C_i \rightarrow C_j$ of $\mathcal{G}$, the reverse link $C_j \rightarrow C_i$, also exists in $\mathcal{G}$.*

Using this reverse communication link, site i can observe counters $c^{\kappa_i^w}$ and $c^{\kappa_j^w}$, so it can compare them by maintaining the counter

$$N_t^{ij} =_{\text{def}} N_t^{\kappa_i^w, \kappa_j^w} \tag{38}$$

If transmission exhibits zero latency, then, at any instant t, N_t^{ij} yields the level of the input buffer attached, at site j, to the communication link $C_i \rightarrow C_j$. Note that there is no circular definition in doing this, since traffic shaping will be performed in a predictive way, as we shall see. (The more realistic case of non-zero latency is discussed in Section 6.6.) Maintaining counter N^{ij} on-line is performed by using the mechanism of Section 6.1.

For example, the situation depicted in Figure 8 occurs at the first instant t_* where $N_t^{ij} \geq 2$ holds. More precisely, the second blue thin bar would occur at that instant. Assume a 2-buffer is available at site j, as indicated in the figure. Then, the buffer gets full at instant t_* for the first time. If bounds exist for the relative drift and jitter between the two clocks at sites i and j, then the buffer will not overflow immediately, but some safe period exists, such that $N_t^{ij} \leq 2$ is guaranteed during this period. Now, two cases can occur:

- The buffer is emptied before the end of the safe period, thus bringing the communication link back to its safe mode, i.e., $N_t^{ij} \leq 1$.
- The end of the safe period is reached while the buffer is still full. Then, writer w_i decides to postpone sending its data to site j.

Since traffic shaping delays emissions, effective writes at site i are not according to nominal clock $\kappa_i \downarrow 2$, but are indeed possibly downsampled, with clock $\kappa_i^w \subseteq \kappa_i \downarrow 2$, see (1). This predictive traffic shaping policy is formalized next. Recall the notation $c_{s,t}^{\kappa} = c_t^{\kappa} - c_s^{\kappa}$, for $s < t$.

Assumption 3

1. *There exists $K > 0$ such that, $\forall\ s, t$ with $s < t$*

$$\max\left(1\,, \min_i c_{s,t}^{\kappa_i}\right) \geq K(t-s) \tag{39}$$

Constant K is called a pessimistic rate *for network $\mathcal{G}$.*

2. *There exists a* safe period $\tau_{\rm safe} \geq 0$ *such that, for any two* s *and* t *such that* $0 \leq t-s \leq \tau_{\rm safe}$,

$$\max_{i,j} \left(c^{\kappa_i}_{s,t} - c^{\kappa_j}_{s,t} \right) \ \leq \ 2$$

Condition 1 expresses that the rates of all clocks are uniformly bounded from below—i.e., no clock can be "possibly infinitely slow".

Protocol 2 (Traffic shaping policy)

- *As part of running link* $C_i \rightarrow C_j$, *site* i *maintains counter* N^{ij} *defined in (38). Define the* safe mode of link $C_i \rightarrow C_j$ *by the condition* $N^{ij}_t \leq 1$. *Say that* site i is in safe mode *if all its outgoing links* $C_i \rightarrow C_j$ *for each* $j \neq i$ *are safe.*
- *When safe mode is left at site* i *(i.e.,* $N^{ij}_t = 2$ *occurs for some* $j \neq i$*) a timer is started by site* i, *with timeout value equal to* $\tau_{\rm safe}$. *This timer is killed as soon as the site returns to its safe mode.*
- *Alternatively, if timeout occurs, then site* i *delays its next emission to all sites it communicates with, until return to the safe mode occurs.*

Note that chosing $\tau_{\rm safe} = 0$ amounts to using no timer.

6.4 Semantics preserving properties

Theorem 1 *Assumptions 2 and 3 are in force.*

1. *A sufficient condition ensuring stream preserving for each communication over network* $\mathcal{G}$ *is that each link* $C_i \rightarrow C_j$ *comes equipped with Protocols 1 and 2.*
2. *In this case, we have, for each site* i, *and every pair* (s,t) *such that* $s < t$*:*

$$\max \left(1 \, , \, c^{\kappa^w_i}_{s,t} \right) \ \geq \ \tfrac{K}{2}(t-s)$$

Comments: The first statement says that our architecture is a Kahn process network [18]. The second statement says that, although some slow down results from applying traffic shaping, the overall network rate is at least $K/2$.

Proof of Theorem 1. The proof proceeds by a few lemmas.

Lemma 1 *At any instant* t, N^{ij}_t *yields the level of the input buffer attached, at site* j, *to the communication link* $C_i \rightarrow C_j$.

Proof: Direct consequence of (38). ◇

Lemma 2 *The network is stream preserving.*

Proof: For each site i, writes at this site occur according to clock κ_i^w. By Lemma 1, for each link $C_i \to C_j$, counter N_t^{ij} yields the level of the input buffer for this link. Protocol 2 avoids the buffer to reach level 3. Since a 2-buffer is used in Protocol 1, Protocols 1 and 2 together ensure that no input buffer will overflow. Since Protocol 2 delays emissions (but doest not cancel them), the network preserves stream semantics. In other words, it behaves as a *Kahn process network* [18]. ◇

So far we proved statement 1 of Theorem 1. This statement is not enough since stream semantics might be satisfied at the price of uncontrolled slow down of the communication pace of the network. Statement 2 addresses this. We prove this statement by using a graphical analysis bearing similarities with the techniques of Network Calculus [23, 39].

A graphical analysis of Protocol 2. To prove Statement 2, we shall first develop a graphical analysis of how Protocol 2 performs, for a pair (w, r) of writer and reader of read type, meaning that reads are less frequent than writes. In this case Protocol 2 must be applied by the writer, otherwise data will get lost.

The reader is referred to Figure 9-top for a graphical analysis of this situation. The diagram of this figure displays counter level versus time. The horizontal grid depicts the successive values $1, 2, \ldots$, for the counter. Let us first focus on the red straight line and staircase. The red staircase depicts the increase of counter c^r for a flow r that we interpret as a reader. To simplify the drawing of this example, we have taken a strictly periodic event. We can equally well recover this staircase by considering the red straight line that links the points on this plane with coordinates (t_n^r, n), for $n = 1, 2, \ldots$ (if the flow r is not strictly periodic, this yields a piecewise straight line). Shifting this red straight line to the top by 2 yields the red dashed line, which defines the boundary of the safe mode, for a writer. Call *safe zone* the zone between these two lines.

Next, a writer w is shown in blue. Let us first focus on the small pattern on top left of the diagram. This pattern shows the evolution of the counter c^w for 4 successive events of w, when this writer skips writing at ticks $2, 3$, and 4 of its clock. As a result, no writing occurs at these instants and the final value for the counter is 1. This staircase is equally well encoded by the piecewise straight line in blue. For our following discussion, we interpret this piecewise straight line as a continuous time "clock" subject to resets consisting of decrements by an amount of 1. These resets are indicated by the downgoing arrows.

How the writer's counter c^w behaves under Protocol 2 is depicted using the above introduced continuous time clock (depicted by the piecewise straight line) sitting between the two straight red lines. The duty of the protocol is to reset the continuous time clock so that it stays within the safe zone. This is always possible provided that the vertical thickness of the safe zone is at least two—whence the size two for the buffer we use in this protocol.

So far we have captured graphically the safe zone for a site not subject to downsampling due to Protocol 2. What is the safe zone around w, for a new site willing to communicate messages to w? This safe zone (in magenta) is shown in combination with the copy of the blue clock sitting in the middle of the diagram. The two zones, for the reader r (in red)

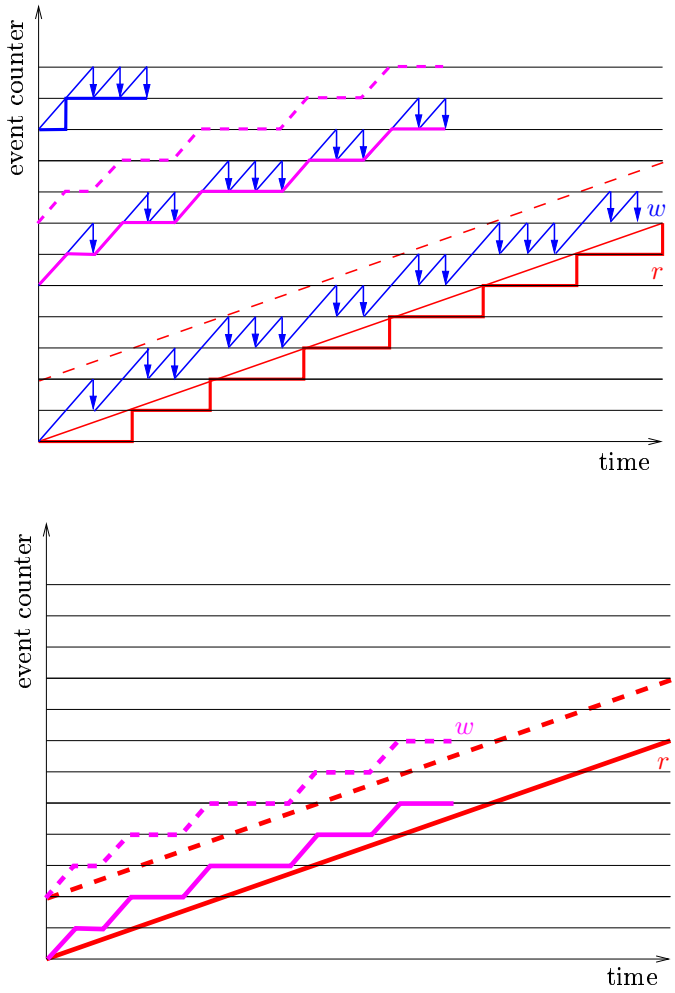

Figure 9: Top: Illustrating Protocol 2 for a fast writer w (in blue) and a slow reader r (in red). Bottom: showing the safe zones for the pair (w, r) when r is slower than w.

and the writer w (in magenta) are jointly depicted, with their exact relative positions, in Figure 9-bottom This figure shows that the floor of the magenta safe zone for w possesses the following property: shifting this floor to the right by one tick of the magenta clock yields a line that sits at a vertical distance strictly larger than 1 below the ceiling of the safe zone for r. This graphical property reflects the fact that the buffer level, from w to r, will never exceed 2.

Study of Protocol 2 for a star shaped network of sites having a slowest center site. Now, assume that we have a star-shaped network of sites $C_i, i \in I$ with center C_{i_*} for some $i_* \in I$. Each site $C_i, i \neq i_*$, is in bi-directional communication with the center of the network. In addition, there are additional direct bi-directional communications $C_i \leftrightarrow C_j$, for some pairs (i, j) such that $i, j \neq i_*$. We assume that the center site has slowest clock, that is, for each $i \neq i_*$, $w_i \triangleright r_{i_*}$ has type *read*, i.e., communications $C_i \rightarrow C_{i_*}$ require traffic shaping, whereas opposite communications don't. We can attach to each CbS communication $w_i \triangleright r_{i_*}$ two diagrams such as in Figures 9 (showing the exact controlled buffer level) and **??** (showing the safe zones, for w_i and r_{i_*}). Superimposing the different safe zones for each $w_i, i \neq i_*$ reveals the following:

1. For each link $C_i \rightarrow C_{i_*}$, traffic shaping must be applied.
2. For each reverse link $C_i \leftarrow C_{i_*}$, traffic shaping is not needed. The reason is that the level of the buffer associated to this communication will never exceed 1.
3. For each link $C_i \rightarrow C_j$, where $i, j \neq i_*$, the two safe zones overlap by a vertical amount strictly greater than 1. Consequently, no additional traffic shaping is required by these lateral communications since their associated buffer, for each direction, will never exceed level 2.

Study of Protocol 2 for a general network satisfying Assumption 2. It remains to show Statement 2 without assuming that the network is star shaped with a slowest site at its center. To this end, create an extra, dummy site i_0, not belonging to I. Equip site i_0 with a clock κ_{i_0} satisfying (39) and making this site slower that all pre-existing sites—these two conditions are not contradictory. Create a bi-directional link between i_0 and every $i \in I$. First, we can apply the above argument and prove Statement 2 for this augmented network. And the proof is now complete if we observe that adding one more clock can only slow down, not accelerate, the pre-existing clocks. This finishes the proof of the theorem. ◇

Applying repeatedly Theorem 1 shows that:

Corollary 2 *LTT networks, when combined with nodes implementing Protocols 1 and 2, preserve stream semantics.*

6.5 Relaxing Assumption 2 on network topology

So far we assumed that, for each direct link $C_i \rightarrow C_j$ of $\mathcal{G}$, the reverse link $C_j \rightarrow C_i$, also exists in $\mathcal{G}$. Assume now the following:

Assumption 4 *Network $\mathcal{G}$ is strongly connected.*

Site i in general has no direct knowledge of clock κ_j^w so it cannot compute counter N^{ij} defined in (38). Let $C_i \rightarrow C_j$ be a link of $\mathcal{G}$, and $C_j = C_{j_1} \rightarrow \ldots \rightarrow C_{j_M} = C_i$ be a path of shortest length, from site j back to site i. The idea is that site i will use $\kappa_{j_{M-1}}^w$ as an estimate of κ_j^w. We claim that

Theorem 2 *The combination of Protocols 1 and 2 preserves stream semantics, provided that a M-buffer is used instead of a 2-buffer in Protocol 1.*

Proof: The theorem is a direct consequence of the following lemma:

Lemma 3 *For $2 \leq m \leq M$, and for any two s and t such that $0 \leq t - s \leq \tau_{\text{safe}}$,*

$$c_{s,t}^{\kappa_{j_0}} - c_{s,t}^{\kappa_{j_m}} \;\leq\; m$$

Proof: The proof is easy, by induction over m. ◇

When $\mathcal{G}$ is not strongly connected, we apply Theorem 2 to each connected component. The different connected components are partially ordered and form a cascade communication, for which it is enough to have clocks of decreasing nominal periods, adjusted to ensure type write for communications between successive connected components.

6.6 Discussion

Taking latencies into account. So far our network model assumes zero-delay transmission. If we need to account for transmission delay, then the latter is simply absorbed by subtracting, from the safe period τ_{safe}, a global upper bound of the transmission latency. This of course assumes that the latter upper bound is small enough so that such a reduction still leaves room for a non empty safe period.

Relaxing perfect real-time synchrony at each given site. Protocol 1 assumes that both r and w are bound to the same clock. This requires perfect sampling for two different signals. However, Theorem 1 can in fact tolerate a small bounded jitter between sampling times, for reads and writes on the same site.

Handling Event Triggered applications. So far we have assumed that all communications are performed according to a single logical clock (provided by the successive ticks of the effective writers clocks at each site). This, however, does not capture the case in which some tasks are triggered by alarm signals and otherwise silent. A relaxed assumption encompassing such cases is to consider that there exists an overall logical clock, from which all logical communication clocks are derived by (possibly data dependent) logical downsampling. Our architecture allows this by "stuffing" idle communication periods by one-bit fake messages aimed at driving traffic shaping.

Traffic Shaping vs. Flow Control. Under Assumption 2, each site C_i sends a message also to its sources. In this case, our predictive traffic shaping protocol (PTSP) resembles *flow control algorithms* (FCA). Typical FCA's use an explicit handshaking protocol whereby the receiver can slow down transmissions from the sender using special flow-control messages. In contrast, the PTSP proposed in this paper, estimates (really measures) the level of the receiver buffer by observing the effective clocks (writes) of both the sender and the receiver. This approach is particularly effective in our systems: we can use any message written by

the receiver and reaching the source, regardless of its content. The PTSP amounts to logical clock synchronization, whereby the faster sites will skip clock cycles to realign with the slower sites.

There are also FCA's that rely on counting buffer levels, similarly to our PTSP, one such example is the FCA in the PCI Express bus [30]. It is based on "credits", that is:

1. at initialization, the receiver declares its credits limits (in our case the 2 buffer places in Section 5).[2]
2. before transmitting, the sender checks a local counter (same as N^{ij}). The transmission is delayed if it would bring the counter beyond the declared credit limit.
3. the local counter is decremented when the receiver sends a "flow control update" (FCU) frame back to the source on the same physical link.

Note that, in addition to counter decrements, the FCU frame can declare a new credit limit when resources (buffers) are allocated dynamically on the receiver.

Practical use of LTT network. Stream semantics preserving is a questionable feature for real-time systems. This assumes that emissions can be postponed, i.e., data can be delayed. This also assumes that input data to the network are independent from network pace. Of course, if input data are provided by polling sensors in real-time, then inputs become timing dependent and the relevance of preserving input/output stream semantics may be questionable. Still, this feature is important regarding discrete control for mode changes and protection. Thus, in addition to providing Theorem 1, we feel it useful to explain how our LTT network works in practice, when Protocols 1 and 2 are applied.

The different nodes $i \in I$ of the network are equipped with quasi-periodic clocks κ_i for message reading, and with effective clocks κ_i^w for message writing. Clocks κ_i are subject to both relative jitter and drift. In contrast, *the protocol-controlled effective clocks κ_i^w are only subject to jitter and suffer from no drift.* Writes at each site use the most recently available input data to the overall system, as well as read messages sent by other sites from a 2-buffer. This bounds the overall latency of the system. The bottom line is that LTTA offers interesting features for distributed real-time computer controlled systems.

Are requirements met? The first sentence of this paper states: "All modern real-time distributed architectures share the viewpoint that communications should be non blocking." So far our present LTTN does not satisfy this: if a site clock fails silently, then all sites to which the faulty site sends messages, directly or indirectly, will be blocked. However, prior knowledge of nominal bounds for the periods allows detecting and isolating the suspect. Fault tolerance for LTTA is adressed in Section 9.

[2] PCIE supports different kinds of message traffic. The receiver has different credit limits associated to each kind.

7 Mapping LTTN on LTT busses

So far we have investigated single communication over an LTT bus and then LTT overlay networks, by considering that the communication medium directly provides communication-by-sampling services for point-to-point communication between sites. In this section we investigate how to deploy an LTT overlay network over an LTT bus. Combining Corollary 1 and 2 yields:

Corollary 3 *LTTN, when implemented over an LTT bus satisfying the conditions of Corollary 1, and combined with nodes implementing Protocols 1 and 2, preserve stream semantics.*

Suppose that clocks κ_i are equal to a same clock κ for all i's. One simple way of chosing the bus b is to have its clock such that $\kappa = \kappa_b \downarrow 2$. In this case, the clock of the writes (where information is actually transmitted) is equal to $\kappa_b \downarrow 4$. This is exactly the oversampling required by the robustifying protocol called "consolidation robuste", proposed in [20, 21]. Our theorem allows for less oversampling.

Bus based LTTN: From the above analysis, the natural implementation of a bus based LTTN can be roughly characterized as follows:

- Bus has quasi-periodic clock with "nominal" period δ_b. This means that phase can be arbitrary and actual period can vary somewhere around δ_b.
- Sites have quasi-periodic clocks with "nominal" period $2\delta_b$, with the same interpretation.
- Read at each site is triggered by the site clock, whereas writes are at least twice as slow, and subject to traffic shaping. Different operations at the same site need not be bound to the same clock, strictly. Small jitter can be tolerated.

8 Multiple Access

So far we have assumed that a single LTT bus is dedicated to each LTT link. In this section, we study the multiplexing of several communication trails

$$(w[k] \triangleright b[k]) \triangleright r[k], \text{ for } k = 1, \ldots, K \tag{40}$$

on the same bus. The flow b associated with the bus is then the multiplexing of the flows $b[k], k = 1, \ldots, K$ associated with the different trails. This relation is written

$$b = \biguplus_{k=1}^{K} b[k]$$

and the *multiplexing* operator $\uplus$ is characterized by the following equations regarding counters, dates, and values:

$$c_t^b = \sum_{k=1}^{K} c_t^{b[k]} \tag{41}$$

$$T^b = \bigcup_{k=1}^{K} T^{b[k]},\ k \neq j \Rightarrow T^{b[k]} \cap T^{b[j]} = \emptyset \tag{42}$$

$$\nu_t^b = \nu_t^{b[k_*(t)]} \tag{43}$$

where

$$k_*(t) = \arg \max_{k \in \{1,\ldots,K\}} l_t^{b[k]}$$

Formula (41) is self-explanatory; formula (42) expresses that the different trails interleave on the bus; finally, (43) indicates that the bus carries, at a given instant, the value written by the last trail that has written to the bus.

The implementation of the LTTA abstraction on selected bus platforms requires the consideration of the bus scheduling policies, which define the sharing of the available bus bandwidth. In this implementation stage, a quantitative analysis is necessary to check if the provided bandwidth can be allocated by the medium access control policy so that the delay and rate requirements can be guaranteed. Such a quantitative analysis makes use of information resulting from the deployment onto the physical platform. Each flow $b[k]$ needs to be annotated with the amount of transmission time $\gamma[k]$ that it requires in the worst case, for the communication of a writer-reader instance pair.

In the following, we outline the mapping process of the LTTA model into bus platforms implementing Time Division, Priority-Based and Resource Reservation access mechanisms for sharing the physical bus among several writer-reader pairs. Furthermore, we discuss the mapping onto platforms capable of adaptive bandwidth allocation.

8.1 Time Division Multiple Access (TDMA)

TDMA is the method of choice for multi-user access in Time-Triggered architectures. The LTTA abstraction can be mapped into a TDMA bus platform with a suitable choice of a static scheduling policy that allocates transmission slots among the different flows $1, \ldots, K$. Following the general principles of Platform Based Design, the scheduling policy must provide an implementation that guarantees the timing requirements of the LTTA Model. When a single TDMA bus is used to implement the LTTA abstraction of a communication among several writers-readers, the relevant timing characteristics of the TDMA access policy are:

- Bounds $\delta^{w_i}, \delta^b, \Delta^b, \Delta^{r_i}$ and buffer capacity M occurring in Property 2, where i ranges over the set of writer-reader pairs.

- Pessimistic rate K and safe period τ_{safe} occurring in Assumption 3.

The implementation of LTTA on a time division bus is an interesting exercise, but probably of limited practical value. A time division bus requires a global clock synchronization scheme that is typically used to enforce communication models with a more restrictive semantics than the LTTA. However, understanding the conditions for an LTTA implementation may still be useful in cases when the bus utilization is high and the application does not require strict time determinism. In this case, LTTA can relax the slot allocation constraints and alleviate the scheduling problem.

Examples of existing time-triggered platforms are the TTP protocol [19] and the FlexRay standard for automotive communication [15].

8.2 Priority Based Multiple Access (PBMA)

PBMA policies are preferred for Event-Triggered mode of computation and communication since they are more flexible. Event-Triggered communication is generally preferred for sparse flows, where effective communication occurs only at a subset of the ticks of the writer site's clock, see the third item regarding Event Triggered application, in Discussion section 6.6. Accordingly, we assume that the mechanism described in that session is used: idle communication periods are stuffed with one bit fake communications aimed at ensuring Protocol 2 of traffic shaping. In contrast, communication of effective data occurs according to a sparse, event driven, mode. Since stuffing bits require less bandwitdth than actual data, we can consider that, in Protocol 2, postponing communication consists in delaying the emission of actual data and replacing the emission of actual data by stuffing bits.

The mapping relation of a flow $b[k]$ into a priority-based platform requires that the flow is annotated with an additional parameter consisting of its priority $\pi[k]$. A priority-based bus platform typically includes a scheduler that can guarantee a worst case latency to each flow, defined in the communication model abstraction by its rate and the worst case time required for each transmission.

The worst case latency, and the corresponding bounds δ^b and Δ^b are typically obtained by applying worst case latency analysis, such as the one defined in [40] for the case of a CAN bus platform [10], in which messages are scheduled by priority, according to their identifier.

The reader is referred to [14] for the complete discussion of a CAN-based implementation of an LTTA model, including the determination of the remaining bounds δ^{w_i} and Δ^{r_i}, the rate K, and the safe period τ_{safe}.

8.3 Resource Reservation Multiple Access (RRMA)

Resource Reservation is a general class of scheduling policies that allows the definition of a virtual channel allocated to each writer-reader pair. Each channel is guaranteed a transmission rate corresponding worst case delays to each periodic transmission instance. Resource reservation schemes are also typically characterized by the possibility of guaranteeing tim-

ing protection among the different message streams that share the bus, so that an excessive bandwidth request from any one of them does not impact the timing properties of the others.

Resource Reservation access policies have been known for some time in the networking community. Relevant examples are the Weighted Fair Queuing policies, or WFQ [13], and their successor WFQ^2 [7]. In the processor scheduling domain, the possible policies are the Generalized Processor Sharing [29], the Constant Bandwidth Server [1] and the PFair schemes [5].

All of them are characterized by the following general property: each stream is assigned to a virtual channel with a nominal share of the resource (bus or processor) bandwidth. The implementation of the virtual channel and the sharing of the resource time by the scheduler typically cannot provide a fluid allocation. Each channel is allowed to deviate from its nominal share by the definition of worst case lead and lag times at which the actual assigned bandwidth matches the ideal one.

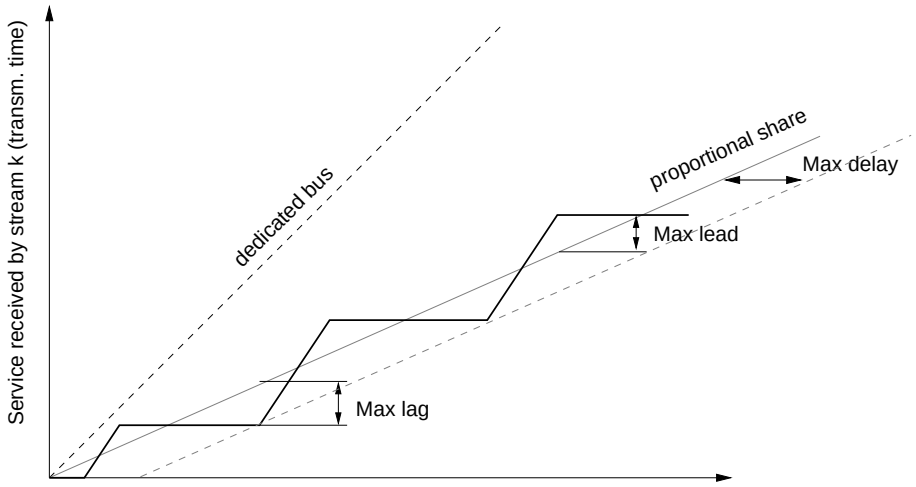

Figure 10: Allocation by resource reservation, nominal share, lead and lag times.

The share is often defined as a reservation for a transmission time $\gamma'[k]$ at each time interval $t'[k]$ for flow $b[k]$. The worst case lead and lag times can be respectively identified by $\alpha[k]$ and $\beta[k]$. This means that, starting from time 0, at time $nt'[k]$, the flow is assigned a transmission time in the range $[n\gamma'[k]-\beta[k], n\gamma'[k]+\alpha[k]]$ (see Figure 10). These guarantees can be transformed into the delay bounds δ^b and Δ^b and the rate bounds that must be applied to the verification of the LTTA conditions.

Resource reservation schemes can be implemented as native protocols, but also on top of priority based policies (as is the case for the CBS policy, implemented on top of CAN in [28].

8.4 Bus Access with Adaptation (BAA)

One additional possibility is the use of a bus platform that allows on-line adaptation of the flow relative rates $b[k]$, subject to the capacity constraint of the bus (41).

Adaptation can be used to tune the rate of critical flows $b[k]$ in order to adjust to the rate of flow $w[k]$. Correspondingly, for other trails, identified as subject to best effort service, the rate of their flow $b[k]$ can be dynamically decreased.

A generic adaptive policy uses the mechanisms of Section 6.1. Assume that our LTT bus is equipped with a buffer of capacity M. Thanks to Property 2, ensuring that trail k has enough bus bandwidth is formalized by enforcing Hypothesis $\mathbf{H}_2$ of Section 6.1, for the pair

$$(x, y) \;=\; (w[k], b[k]) \tag{44}$$

As detailed in Section 6.1, this is achieved by: 1/ computing, on-line, the counter $N_n^{x,y}$ associated via formula (37) to the pair (x, y) defined in (44), and 2/ guaranteing that

$$N_n^{x,y} \;\leq\; M - 1 \tag{45}$$

always holds. Preventing from a violation of (45) is achieved by allocating a slot of the bus to trail k when its writer $w[k]$ is asking for it. This is possible as long as the condition (41) regarding total buffer capacity is satisfied.

If granting access to trail k leads to the violation of condition 45, then access to all trails cannot be granted and conflict must be resolved. This is achieved by performing traffic shaping on the *effective communications of actual data* performed by trail k. More precisely, suppose writer $w[k]$ is asking for access. Then, if granting access may lead to the violation of condition 45 and some trail also asks for access, then access is refused to trail k and the communication of the corresponding actual data is postponed. Instead, trail k emits a stuffing bit.

Note that, in contrast to the traffic shaping of Protocol 2, this traffic shaping, aimed at controlling multiple access, relates to the application layer. Therefore, performing this application level traffic shaping requires that the application layer shall be adaptive, by supporting smooth degradation.

9 Fault tolerance

Fault tolerance mechanisms are needed to address the following types of faults—note that issues related to data are orthogonal to issues related to synchronization and clocks:

- *Corrupted data* arising from perturbations to the bus. Provision for error correcting codes and associated due oversampling is a first classical means to handle this; this is not specific to LTTA and thus we do not discuss it here. In contrast, bus redundancy in LTTA deserves specific study.
- *Overloading of the bus by a writer,* due to a failure of this writer to obey the traffic shaping protocol. This will be handled by means of a *writer guardian* described later.

Note that the same problem arising to a reader raises no specific problem (unless the writer of the same site is also affected).

- *Network starvation* due to the failure of a site to communicate properly with the other sites. As we already said, the traffic shaping protocol causes slow down of the entire network to the slowest clock; this blocks the entire system if some site fails silent.

In fact, the same generic technique can be used for the last two cases, therefore we examine this first.

9.1 Counter based guardians

Counter based guardians are a generic mechanism that takes advantage of the time-triggered nature of LTT architectures—very much the same way that strict TT architectures do—to compensate for undesirable features of Kahn network types of architectures regarding fault tolerance.

Bus overloading can be prevented by monitoring that the rate of writing does not exceed a pre-specified level. Symmetrically, network starvation due to site failure can be prevented from by monitoring that the clock rate of every site does not fall below a pre-specified threshold. Thus both services are variations of the same generic mechanism we describe now.

Consider a flow e for which the following prior bounds are available, where d and D are two constants such that $0 < d < D < \infty$:

$$\forall k > 0 \ : \ d \ < \ t^e_k - t^e_{k-1} \ < \ D \tag{46}$$

holds. We wish to detect when the period of e possibly deviates from these bounds. To this end, we associate to e its upper/lower *monitor* $\mu^{\pm}$, whose prior bounds are set as follows:

$$\forall k > 0 \ : \ D \ < \ t^{\mu^+}_k - t^{\mu^+}_{k-1} \ < \ D' \tag{47}$$

$$\forall k > 0 \ : \ d' \ < \ t^{\mu^-}_k - t^{\mu^-}_{k-1} \ < \ d \tag{48}$$

In these formulas, d and D are as in (46), and the other constants are selected such that $D' > D$, $d' < d$, and the differences $D' - D$ and $d - d'$ are small in a way compatible with technology costs and detection needs. Combining (46) and (47) shows that, for every $t > 0$,

$$N^{\mu^+,e}_t \leq 1 \tag{49}$$

Similarly, Combining (46) and (48) shows that, for every $t > 0$,

$$N^{e,\mu^-}_t \leq 1 \tag{50}$$

Thus we need to detect the first instant where one of the two conditions (49) or (50) gets violated and then emit an alarm. Computing on-line these two counters is performed by means of the mechanism of Section 6.1.

Application to writer guardian. The above technique is applied to the writer w in consideration, and the lower monitor μ_w^- associated to flow w is considered. When an alarm is emitted, this means that either the writer or its associated monitor is faulty. Protection is ensured by shutting this writer down and disconnecting the associated site from the LTT network. This site is then removed from graph $\mathcal{G}$. If another site needs data from the removed site, it simply reads a constant default value.

Application to network starvation. This is the symmetric case. The upper monitor μ_w^+ associated to flow w is considered. When an alarm is emitted, this means that either the writer or its associated monitor is faulty. Protection is again ensured by shutting this writer down and disconnecting the associated site from the LTT network. This site is then removed from graph $\mathcal{G}$. If another site needs data from the removed site, it simply reads a constant default value.

9.2 Redundant buses

Redundancy can be given to the bus by providing a backup for it. This requires having a *voter* to detect deviations between the two buses. For LTT buses, we need to account for uncertainties in the timing of the successive rounds of the bus. Therefore, direct comparison of transmitted data by the two buses cannot be used—this problem has been an important focus of [20, 21]. The voter we propose here has two stages. In the first stage, clocks are compared. If the two clocks agree, then the second stage is activated, which consists in comparing the data transmitted by the two buses. It is possible to rely on the first stage alone. We detail the two stages next.

Clock based voter. To maintain the clocks of the two buses b_1 and b_2 within prespecified bounds, we need to compute the two counters $N_t^{12} =_{\text{def}} N_t^{b_1,b_2}$ and $N_t^{21} =_{\text{def}} N_t^{b_2,b_1}$. The clock based voter follows, by using the same trick again and again.

Data based voter. The data based voter is activated as long as the clock based voter does not emit an alarm. It consists in monitoring the difference

$$\nu_n^{t_1} - \nu_n^{t_2}$$

for successive $n \in \mathbb{N}$, where $t_i = (w \triangleright b_i) \triangleright r$, for $i = 1, 2$, and comparing it to a prespecified tolerance bound on values.

10 Conclusion

We presented a comprehensive design flow from functional requirements to implementation on a distributed system consisting of multiple ECUs and interconnections that leverages the concept of Loosely Time Triggered Architecture. To do so, we introduced a formal

model of computation to capture the communication mechanisms typical of LTTA and an intermediate layer of abstraction called Loosely Time Triggered Network. This layer of abstraction can then be mapped onto a variety of implementation architectures. The design process is guaranteed to preserve stream semantics if appropriate assumptions hold. The implicit assumption we have made in this paper is that the functional requirements are related to discrete control functions that include protection (fault tolerance, voting), mode changes, and some event triggered side functions.

Open problems are the efficient application of the LTTA architecture to implement continuous and hybrid functions such as low level safety critical feedback control loops, whose underlying design is performed based on continuous time models. Typical examples include flight control of aircrafts or combustion control for engines.

Continuous control functions. For such functions, a natural LTTA implementation would require that the sensor input data be periodically sampled and communicated to their associated computer following LTT mode. The (distributed) computers implement the desired Multi Input Multi Output control by communicating according to the LTT mode and send their sampled control values to the actuators following the LTT mode. This is a fairly natural implementation where sampling is performed locally and independently, at the different sites of the distributed architecture. Consistency of the whole is ensured by the underlying continuous time feedback control and its built-in robustness with respect to sampling. This approach is indeed supported by several experimental studies demonstrating that sampling jitter can be accommodated, possibly by proper tuning of the control parameters, see, e.g., [25, 2, 3].

Still, it is unclear whether this implementation technique is actually mathematically and provably supported by robust control design techniques. Classical studies regarding robust control design are either performed in continuous time (which rules out any consideration regarding sampling), or discrete time but with a unique perfectly periodic clock. The only work that we are aware of regarding robust feedback control design with time-varying period for sampling is that of O. Sename et al. [33, 34], where a single approximately periodic clock is considered. Thus, to our knowledge, deploying continuous feedback control loops on LTTA in an important issue that still requires research on robust control design.

It should be however acknowledged that, today, no safety critical low level continuous control feedback loop is deployed on a distributed computing architecture. One control loop is running on a single computer as a single task, by collecting distributed sensor data and sending control values to distributed actuators. But, again, this is expected to change, e.g., for future aircraft with higly distributed sensors and actuators having distributed local intelligence; not to speak about flight formations.

Mixed Continuous/Discrete control functions. So far LTTA and continuous feedback implementation provide a satisfactory answer for continuous and discrete control functions, considered separately. However it does not encompass their interaction. Of course, nothing prevents us from using the above "logically synchronous" layer for deploying continuous control as well. However, this is clearly a waste of energy since building this layer requires protocols, as we have seen. In contrast, the direct implementation we have advocated for

continuous control seems more effective, from practical and performance viewpoint. Thus a comprehensive theory encompassing both continuous and discrete control, as well as their interaction, would be welcome. While attempts are provided in [20] toward a comprehensive theory of continuity for mixed continuous/discrete control systems, this has not resulted yet in an effective set of results and design guidelines.

References

[1] L. Abeni and G. C. Buttazzo. Integrating multimedia applications in hard real-time systems. In *RTSS*, pages 4–13, 1998.

[2] K.-E. Årzén. Timing Analysis and Simulation Tools for Real-Time Control. In *FORMATS*, pages 142–143, 2005.

[3] K.-E. Årzén, A. Cervin, and D. Henriksson. Implementation-Aware Embedded Control Systems. In *Handbook of Networked and Embedded Control Systems*, pages 377–394. 2005.

[4] M. Baleani, A. Ferrari, L. Mangeruca, and A. L. Sangiovanni-Vincentelli. Efficient embedded software design with synchronous models. In *EMSOFT*, pages 187–190, 2005.

[5] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. Proportionate progress: a notion of fairness in resource allocation. pages 345–354, 1993.

[6] M. Basseville and I. Nikiforov. *Detection of abrupt changes – theory and applications*. Prentice Hall, 1993.

[7] J. C. R. Bennett and H. Zhang. WF 2 q: Worst-case fair weighted fair queueing. In *INFOCOM (1)*, pages 120–128, 1996.

[8] A. Benveniste, B. Caillaud, L. P. Carloni, P. Caspi, A. L. Sangiovanni-Vincentelli, and S. Tripakis. Communication by sampling in time-sensitive distributed systems. In *EMSOFT*, pages 152–160, 2006.

[9] A. Benveniste, P. Caspi, P. L. Guernic, H. Marchand, J.-P. Talpin, and S. Tripakis. A protocol for loosely time-triggered architectures. In *EMSOFT*, pages 252–265, 2002.

[10] R. Bosch. Can specification, version 2.0. Stuttgart, 1991.

[11] P. Caspi and N. Halbwachs. A functional model for describing and reasoning about time behaviour of computing systems. *Acta Inf.*, 22(6):595–627, 1986.

[12] P. Caspi, D. Pilaud, N. Halbwachs, and J. Plaice. Lustre: a declarative language for programming synchronous systems. In *14th ACM Symp. POPL*, 1987.

[13] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *ACM SIGCOMM*, pages 1–12, September 1989.

[14] M. Di Natale, A. Benveniste, P. Caspi, C. Pinello, A. Sangiovanni-Vincentelli, and S. Tripakis. An implementation of the Loosely Time-Triggered Architecture on the Controller Area Network: feasibility conditions and system design. Technical report, 2007. submitted.

[15] Flexray. Protocol specification v2.1 rev. a. available at `http://www.flexray.com`, 2006.

[16] G. Fohler and G. C. Buttazzo. Introduction to the special issue on flexible scheduling. *Real-Time Systems*, 22(1-2):5–7, 2002.

[17] A. Girault. A survey of automatic distribution method for synchronous programs. In F. Maraninchi, M. Pouzet, and V. Roy, editors, *International Workshop on Synchronous Languages, Applications and Programs, SLAP'05*, ENTCS, Edinburgh, UK, Apr. 2005. Elsevier Science.

[18] G. Kahn. The semantics of a simple language for parallel programming. In *Information Processing 74, Proceedings of IFIP Congress 74*, Stockholm, Sweden, 1974. North-Holland.

[19] H. Kopetz. *Real-Time Systems*. Kluwer Academic Publishers, 1997.

[20] C. Kossentini. *Implantation robuste de contrôles-commandes hybrides répartis: application aux commandes de vol Airbus.* PhD thesis, Institut National Polytechnique de Grenoble (INPG), 2006.
[21] C. Kossentini and P. Caspi. Approximation, sampling and voting in hybrid computing systems. In *HSCC*, pages 363–376, 2006.
[22] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.
[23] J.-Y. Le Boudec and P. Thiran. *Network Calculus.* Lecture Notes in Computer Science (LNCS). Springer Verlag, 2001.
[24] N. A. Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.
[25] P. Martí, J. M. Fuertes, K. Ramamritham, and G. Fohler. Jitter Compensation for Real-Time Control Systems. In *IEEE Real-Time Systems Symposium*, pages 39–48, 2001.
[26] S. Matic and T. A. Henzinger. Trading end-to-end latency for composability. In *RTSS*, pages 99–110, 2005.
[27] G. Moustakides. Optimal stopping times for detecting changes in distributions. *Annals of Statistics*, 14(4):1379–1387, 1986.
[28] T. Nolte, M. Sjodin, and H. Hansson. Server-based scheduling of the can bus, 2003.
[29] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the singlenode case. volume 1, pages 344–357, June 1993.
[30] PCI Express. Pci express base specification 1.1. http://www.pcisig.com/specifications/pciexpress/base, Mar 2005.
[31] D. Potop-Butucaru, B. Caillaud, and A. Benveniste. Concurrency in synchronous systems. *Formal Methods in System Design*, 28(2):111–130, 2006.
[32] P. Potop-Butucaru and B. Caillaud. Correct-by-construction asynchronous implementation of modular synchronous specifications. *Fundamenta Informaticae*, 2006.
[33] D. Robert. *Contribution à l'interaction commmande/ordonnancement.* PhD thesis, Institut National Polytechnique de Grenoble (INPG), Jan 2007.
[34] D. Robert, O. Sename, and D. Simon. Synthesis of a sampling period dependent controller using lpv approach. In *5th IFAC Symposium on Robust Control Design ROCOND'06*, Toulouse, France, 2006.
[35] J. Romberg and A. B. 0002. Loose synchronization of event-triggered networks for distribution of synchronous programs. In *EMSOFT*, pages 193–202, 2004.
[36] A. Sangiovanni-Vincentelli. Quo Vadis SLD? Reasoning about trends and challenges of System Level Design. *Proceedings of the IEEE*, 95(3):467–506, 2007.
[37] N. Scaife and P. Caspi. Integrating model-based design and preemptive scheduling in mixed time- and event-triggered systems. In *ECRTS*, pages 119–126, 2004.
[38] C. Sofronis, S. Tripakis, and P. Caspi. A memory-optimal buffering protocol for preservation of synchronous semantics under preemptive scheduling. In *EMSOFT*, pages 21–33, 2006.
[39] L. Thiele, E. Wandeler, and N. Stoimenov. Real-time interfaces for composing real-time systems. In *EMSOFT*, pages 34–43, 2006.
[40] K. Tindell, A. Burns, and A. J. Wellings. Calculating controller area network (can) message response times. *Control Eng. Practice*, 3(8):1163–1169, 1995.
[41] S. Tripakis, C. Sofronis, N. Scaife, and P. Caspi. Semantics-preserving and memory-efficient implementation of inter-task communication on static-priority or edf schedulers. In *EMSOFT*, pages 353–360, 2005.