The final publication is available at Springer via https://doi.org/10.1007/s11241-018-9310-2

Uniprocessor scheduling of real-time synchronous dataflow tasks

Abhishek Singh · Pontus Ekberg · Sanjoy Baruah

Abstract The synchronous dataflow graph (SDFG) model is widely used today for modeling real-time applications in safety-critical application domains. Schedulability analysis techniques that are well understood within the real-time scheduling community are applied to the analysis of recurrent real-time workloads that are represented using this model. An enhancement to the standard SDFG model is proposed, which supports the specification of a real-time latency constraint between a specified input and a specified output of an SDFG. A polynomial-time algorithm is derived for representing the computational requirement of each such enhanced SDFG task in terms of the notion of the demand bound function (DBF), which is widely used in real-time scheduling theory for characterizing computational requirements of recurrent processes represented by, e.g., the sporadic task model. By so doing, the extensive DBF-centered machinery that has been developed in real-time scheduling theory for the hard-real-time schedulability analysis of systems of recurrent tasks may be applied to the analysis of systems represented using the SDFG model as well. The applicability of this approach is illustrated by applying prior results from real-time scheduling theory to construct an exact preemptive uniprocessor schedulability test for collections of independent recurrent processes that are each represented using the enhanced SDFG model.

Keywords Real-Time Systems · Synchronous Dataflow (SDF) · Hard Real-Time Streaming Dataflow Applications · Algorithms

Abhishek Singh
The University of North Carolina, Chapel Hill, NC, USA
E-mail: abh@cs.unc.edu

Pontus Ekberg
Uppsala University, Uppsala, Sweden
E-mail: pontus.ekberg@it.uu.se

Sanjoy Baruah
Washington University in St. Louis, St. Louis, MO, USA
E-mail: baruah@wustl.edu

1 Introduction

Although originally developed for describing digital signal processing applications for implementation on parallel hardware, the synchronous dataflow graph (SDFG) model (Lee (1986); Lee and Messerschmitt (1987b)) is widely used today for modeling real-time applications in domains such as telecommunications, automotive systems, and avionics, which require the processing of streaming data under hard-real-time constraints. Run-time scheduling of computational workloads that are represented in the SDFG model has traditionally been done with static methods (e.g., Lee (1986); Lee and Messerschmitt (1987a)), in which scheduling tables are computed off-line and used for making run-time scheduling decisions. More recently, efforts have been made to explore the use of dynamic scheduling approaches, of the kind that are studied in the real-time scheduling theory community, in order to obtain more resource-efficient implementations of systems that are represented using the SDFG model. Examples of such efforts include the following (this is not an exhaustive list):

- Bouakaz et al (2014) apply EDF (earliest deadline first) scheduling in order to implement multiple independent applications each specified in the SDFG model upon a shared (uniprocessor or partitioned multiprocessor) computing platform.
- Bamakhrama and Stefanov (2011, 2012) propose techniques for transforming SDFG specifications of a particular kind (called *acyclic cyclo-static* data flow graphs) to collections of periodic tasks, which can then be scheduled using the methods and algorithms developed in real-time scheduling theory for periodic task scheduling.
- Ali et al (2015) have developed techniques for transforming SDFG specifications of a different kind, called *cyclic homogeneous* SDFG, to collections of periodic tasks.
- Mohaqeqi et al (2016) describe how to represent SDFG specifications in the *digraph* real-time (DRT) task model (Stigge et al 2011).
- Khatib et al (2016); Klikpo and Kordon (2016) describe how periodic tasks with inter-task dependencies may be modeled using SDFGs (and thereby scheduled using approaches that have been developed for the scheduling of SDFGs).

To our knowledge, none of these prior approaches claim optimality from the perspective of run-time resource utilization; indeed, it is fairly easy to construct example instances in which each such approach will result in implementations that make arbitrarily inefficient use of platform computing resources. Other data-flow approaches such as the Processing Graph Method (PGM (1987)) that have been explored in the real-time systems community similarly suffer from an absence of optimality results.

Motivation for this research. The long-term objective of our research is to investigate the applicability of the concepts, techniques, methodologies, and

results of real-time scheduling theory to the analysis of real-time workloads that are represented using the SDFG model.

While real-time scheduling theory has made tremendous progress in recent years, this progress has, by and large, remained focused upon the workload models that are popular within the community, such as Liu & Layland tasks (Liu and Layland 1973) and 3-parameter sporadic tasks (Mok 1983; Baruah et al 1990). Meanwhile, data-flow models such as SDFG are finding increasing use in many embedded application domains, but research on these models has thus far primarily concentrated on ensuring correctness of design rather than enhancing efficiency of implementation. With embedded software becoming increasingly computationally demanding, that obtaining efficient implementations of such software that are often specified using the SDFG model is becoming a primary consideration on par with correctness. This opens up opportunities for the thus-far distinct sub-fields of real-time scheduling and SDFG analysis to collaborate on solving problems of mutual interest and expertise.

This research. This paper addresses the following research question:

> How do we implement a given collection of independent SDFG tasks, each of which is subject to real-time correctness constraints, upon a shared platform in an efficient manner?

The manner in which we propose to solve this problem is by precisely characterizing the cumulative computational requirement of each such SDFG task using the *demand bound function* (DBF) abstraction (Baruah et al (1990)). By obtaining DBF characterizations of the SDFG tasks' computational demand, we are able to deploy the powerful DBF-centered machinery that has been developed in the real-time scheduling theory community to the schedulability analysis and run-time scheduling of systems modeled as collections of independent real-time SDFG tasks; as an illustration, we derive an optimal EDF-based algorithm and an associated schedulability test for implementing such systems upon preemptive uniprocessor platforms.

This paper is an improvement upon our initial efforts in this direction (Singh et al 2017). In the previous paper, we proposed a polynomial-time algorithm for computing the DBF of a subclass of SDFG tasks called *homogeneous* SDFG tasks. The algorithm that was proposed for *general* SDFG tasks, on the other hand, had an exponential-time upper bound. In this paper, we have come up with a simple algorithm for computing the DBF of general SDFG tasks that has polynomial running time. Thus, we are freed from reasoning about homogeneous SDFG tasks separately. While informal arguments were made in the previous paper about the run times, we have a more formal proof for the polynomial run-time bound for general SDFG tasks. Establishing the polynomial run-time bound for DBF computation for general SDFG tasks needed a deeper understanding of the run-time behavior of SDFGs. We consider these insights to be a significant part of the contribution of this paper — they provide us with a better understanding of SDFG executions, and we are currently

engaged in the process of seeking to exploit these insights to obtain some additional schedulability results.

In this paper, we also extend the algorithm for general SDFG tasks to work without the assumption that all enabled actors are maximally fired before run-time. While this extended algorithm is more comprehensive, it is also the first polynomial-time algorithm that can compute the dependency distance (Siyoum 2014) of an SDFG, to the best of our knowledge.

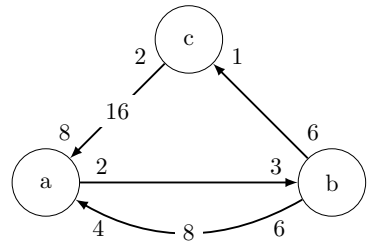
Organization. The remainder of this paper is organized in the following manner. In Section 2 we describe an event-triggered recurrent real-time extension to the SDFG model that has been developed over a number of years, which we will be using in this paper. (In addition to reviewing the model in this section, we define, and prove properties of, the concept of the *reverse* SDFG of a given SDFG; this concept plays a crucial role in enabling us to prove some properties of our algorithms.) In Section 3 we briefly review the 3-parameter sporadic task model (Mok (1983); Baruah et al (1990)) that underpins a large body of research in the real-time scheduling theory community. A significant fraction of this body of research represents the computational requirement of a 3-parameter sporadic task by the demand bound function (DBF) abstraction; in Section 4, we derive an algorithm for determining the DBF for a recurrent real-time SDFG task and prove that this algorithm has running time polynomial in the representation of the task.

2 A real-time SDF model

In this section we describe both the basic SDF model (Lee (1986); Lee and Messerschmitt (1987b)), and several extensions that have been proposed to the model in order to enhance its capabilities to more accurately depict real-time considerations. This introduction is necessarily brief; we refer the interested reader to (Lee and Seshia 2011, Ch 6.3.2) for a text-book description and additional references.

A dataflow graph is a directed graph[1] in which the vertices (known as *actors*) represent computation and edges (known as *channels*) represent FIFO queues that direct data (called *tokens*) from the output of one computation to the input of another. Actors *consume* tokens from their input channels, perform computations upon them (this is referred to as a *firing* of the actor) and *produce* tokens on their output channels. Channels may contain *initial tokens* (also known as *delays*) — these represent data that populate the FIFO queues prior to run time. The number of tokens at each channel determines the *state* or *configuration* of the dataflow graph.

[1] Most SDFG models allow for *multi*graphs, in which there may be multiple edges between the same pair of vertices. This feature is not particularly relevant to determining how they are scheduled and we, therefore, ignore them in this paper. For the same reason, we also ignore edges that are self-loops: lead from a vertex back to itself. We point out that our results are easily extended to deal with multiple edges and self-loops.



$$\Gamma(G) = \begin{array}{c} \\ (a,b) \\ (b,a) \\ (b,c) \\ (c,a) \end{array} \begin{array}{c} \begin{array}{ccc} a & b & c \end{array} \\ \begin{pmatrix} 2 & -3 & 0 \\ -4 & 6 & 0 \\ 0 & 6 & -1 \\ -8 & 0 & 2 \end{pmatrix} \end{array} \qquad \mathbf{q}(G) = \begin{array}{c} \\ a \\ b \\ c \end{array} \begin{array}{c} \\ \begin{pmatrix} 3 \\ 2 \\ 12 \end{pmatrix} \end{array}$$

Fig. 1: An example SDFG G, its topology matrix $\Gamma(G)$, and its repetition vector $\mathbf{q}(G)$. (Rows and columns of the topology matrix are labeled by channel and actor respectively, and rows of the repetition vector are labeled by actor.)

2.1 Synchronous Data Flow Graphs

A *synchronous dataflow graph (SDFG)* is a dataflow graph with the additional constraints that the number of initial tokens on each channel, as well as the number of tokens produced (consumed, respectively) by each actor on each of its outgoing (incoming, resp.) channels upon a firing of the actor, is a known constant.

Definition 1 (SDFG) An SDFG G is represented as a 5-tuple $G = \langle V, E, \text{PROD}, \text{CONS}, \text{DELAY} \rangle$ where

- V denotes the set of actors.
- $E \subset (V \times V)$ is the set of channels. For each channel $e = (u, v)$, we denote u as $\text{TAIL}(e)$ and v as $\text{HEAD}(e)$. For each channel e, $\text{TAIL}(e) \neq \text{HEAD}(e)$, since we ignore self-loops.
- $\text{PROD} : E \to \mathbb{N}_{>0}$. For each $e \in E$, $\text{PROD}(e)$ tokens are added to channel e each time the actor $\text{TAIL}(e)$ fires.
- $\text{CONS} : E \to \mathbb{N}_{>0}$. For each $e \in E$, $\text{CONS}(e)$ tokens are removed from channel e each time the actor $\text{HEAD}(e)$ fires.
- $\text{DELAY} : E \to \mathbb{N}_{\geq 0}$. For each $e \in E$, $\text{DELAY}(e)$ denotes the number of initial tokens (or delays) on channel e.

□

We shall use the SDFG depicted in Figure 1 as our running example. It has three actors a, b and c, denoted by circles containing the actor name. Edges represent channels and are annotated at their ends with production and consumption rates and at their centers with the number of delays if this number is > 0.

As we had stated above, the number of tokens in each channel in E determines the state or configuration of the SDFG $G = \langle V, E, \text{PROD}, \text{CONS}, \text{DELAY}\rangle$. For a particular configuration of G, an actor $v \in V$ is said to be *enabled* if each channel $e \in E$ for which $\text{HEAD}(e) = v$ contains at least $\text{CONS}(e)$ tokens. An enabled actor v may *fire*; doing so changes the configuration of the SDFG in the following manner: $\text{CONS}(e)$ tokens are removed from each channel $e \in E$ for which $\text{HEAD}(e) = v$, and $\text{PROD}(e)$ tokens are added to each channel $e \in E$ for which $\text{TAIL}(e) = v$. The sequence of configurations that an SDFG G goes through via a sequence of firings is sometimes called a *trace* of G; the possible traces of G define its run-time behavior. SDFG analysis techniques and algorithms have been developed (Lee (1986); Lee and Messerschmitt (1987b)) for determining, for a given SDFG, whether traces could lead to *deadlock* — a configuration of tokens on channels such that no actor is enabled, or to *buffer overflow* — the number of tokens in a channel growing without bound[2].

According to the description above, the initial configuration of tokens on channels (as represented by the DELAY function) determines all the future firings of actors; external events play no role in the SDFG's behavior. Lee and Messerschmitt (1987b) state this explicitly: "Connections to the outside world are not considered [...] a node with only inputs from the outside is considered a node with no inputs, which can be scheduled at any time." (Equivalently, it may be assumed that external input is always available when needed by an actor in order for it to fire.) While this assumption may have been reasonable for the original intended use of this model of computation for representing streaming computations, it is inconsistent with our efforts to incorporate real-time considerations; in Section 2.2 below, we discuss how we extend the SDFG model to incorporate timing properties of externally-provided data, which we model as external input tokens.

Definition 2 (Topology matrix (from Lee (1986))) For an SDFG $G = \langle V, E, \text{PROD}, \text{CONS}, \text{DELAY}\rangle$, its *topology matrix*, denoted $\Gamma(G)$, is an $|E| \times |V|$ matrix in which the entry in the i'th row, j'th column, is as follows:

$$\Gamma(G)[i,j] \stackrel{\text{def}}{=} \begin{cases} \text{PROD}(e_i), & \text{if } \text{TAIL}(e_i) = v_j \\ -\text{CONS}(e_i), & \text{if } \text{HEAD}(e_i) = v_j \\ 0, & \text{otherwise} \end{cases}$$

□

Figure 1 depicts an SDFG and its topology matrix. Consider, for instance, its first row corresponding to the channel leading from actor a to actor b. The entry $\Gamma(G)[1,1] = 2$ denotes that each firing of actor a (represented

[2] In addition to requiring that each buffer be of finite size, many SDFG scheduling algorithms seek to minimize the maximum number of tokens each buffer will need to hold. However the algorithms of Singh et al (2017) do not consider buffer-size minimization while seeking to construct schedules in which real-time constraints are met, and we will do likewise here – require that buffers be of finite size, but leave as future work the problem of determining the minimum sizes needed.

by the first column) adds (PRODuces) two tokens to this channel; the entry $\Gamma(G)[1,2] = -3$ denotes that each firing of actor b (represented by the second column) removes (CONSumes) three tokens from this channel.

The following results are from Lee (1986); Lee and Messerschmitt (1987b):

1. Algorithms are known for determining whether a given SDFG G is deadlock-free.
2. A connected[3] SDFG G that is deadlock-free will not suffer from buffer overflow if and only if the rank[4] of its topology matrix $\Gamma(G)$ equals $(|V|-1)$, where $|V|$ denotes the number of actors in G.
   In the remainder of this paper, we will assume that the SDFGs we deal with have been *a priori* verified to possess the properties of being deadlock-free and not subject to buffer overflow.
3. For such an SDFG G, we can efficiently find a positive integer vector $\vec{v} \neq \mathbf{0}$ such that
$$\Gamma(G) \cdot \vec{v} = \mathbf{0}. \tag{1}$$

If we interpret the $|V|$ components of $\vec{v}$ as number of firings of the $|V|$ actors, the reader may verify that satisfying Equation 1 is equivalent to asserting that upon completing the number of firings of each actor represented in $\vec{v}$, the total number of tokens in each channel is unchanged. This observation motivates the following definitions:

Definition 3 (Repetitions vector; Iteration) The *repetitions vector* for an SDFG G is the smallest vector $\vec{v}$ not equal to $\mathbf{0}$ for which Equation 1 holds, and is denoted by $\mathbf{q}(G)$. An *iteration* is a set of actor firings with as many firings as the repetitions vector entry for each actor. □

Henceforth we will often simplify our notation and write Γ and $\mathbf{q}$, rather than $\Gamma(G)$ and $\mathbf{q}(G)$, when the SDFG G under consideration is evident.

Observe that the number of tokens in each channel remains unchanged upon completion of an iteration, during which each actor fires as many times as indicated by its corresponding entry in the repetitions vector. This is formally stated in the following *balance equation*: for each $e \in E$,
$$\text{PROD}(e) \times \mathbf{q}[\text{TAIL}(e)] = \text{CONS}(e) \times \mathbf{q}[\text{HEAD}(e)] \tag{2}$$

In addition to the topology matrix, the repetitions vector for the SDFG depicted in Figure 1 is also shown in the figure. For this example, it is easily verified that Equations 1 and 2 do indeed hold:
$$\Gamma \cdot \mathbf{q} = \begin{pmatrix} 2 & -3 & 0 \\ -4 & 6 & 0 \\ 0 & 6 & -1 \\ -8 & 0 & 2 \end{pmatrix} \cdot \begin{pmatrix} 3 \\ 2 \\ 12 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

[3] An SDFG G is said to be *connected* if the undirected graph formed by taking the actors as vertices and channels as undirected edges is connected.

[4] The *rank* of a matrix is the maximum number of linearly independent columns in it. Efficient polynomial-time algorithms are known for computing rank.

Fig. 2: A simple SDFG and its reverse

An iteration of this example SDFG therefore comprises three firings (i.e., executions) of actor a, two of actor b, and twelve of actor c.

It is sometimes useful to look at the following restricted type of SDFG.

Definition 4 (Homogeneous SDFG (HSDFG)) A *homogeneous* SDFG is one in which all the PROD and CONS rates are equal to one. □

For such homogeneous SDFGs, it follows from the balance equation (Equation 2 above) that the repetitions vector, if one exists (i.e., if the SDFG is deadlock-free and is not subject to buffer overflow) will comprise all ones: $\mathbf{q}(a) = 1$ for all $a \in V$.

2.1.1 Reverse of an SDFG

We now introduce a concept that we will be using later in deriving some of our technical results — that of *reversing* an SDFG. Informally, the reverse G^R of SDFG G is an SDFG with the same set of actors but with the direction of each channel reversed (and with the roles of the production and consumption parameters of each channel reversed). More formally,

Definition 5 (Reverse) The *reverse* of an SDFG $G = \langle V, E, \text{PROD}, \text{CONS}, \text{DELAY} \rangle$ is the SDFG $G^R = \langle V', E', \text{PROD}', \text{CONS}', \text{DELAY}' \rangle$ with

- $V' = V$;
- $E' = \{(v, u) \mid (u, v) \in E\}$;
- $\text{PROD}'(v, u) = \text{CONS}(u, v)$;
- $\text{CONS}'(v, u) = \text{PROD}(u, v)$; and
- $\text{DELAY}'(v, u) = \text{DELAY}(u, v)$.

□

A simple 2-actor SDFG and its reverse are depicted in Figure 2.

Lemma 1 *An SDFG G is deadlock-free and not subject to buffer overflow if and only if its reverse SDFG G^R is deadlock-free and not subject to buffer overflow.*

Proof Sketch As stated previously, a firing of an actor of an SDFG changes the configuration of the SDFG. Suppose that some actor $v \in V$ is enabled at some configuration χ_1 of G, and that the firing of actor v changes the configuration of G from χ_1 to χ_2.

- Since v is enabled in G in configuration χ_1, it must be the case that each channel e in G for which $\text{HEAD}(e) = v$ contains at least $\text{CONS}(e)$ tokens in configuration χ_1.
- Upon firing v in G, $\text{PROD}(e)$ tokens are deposited into each channel e in G for which $\text{TAIL}(e) = v$; hence each such channel will contain at least $\text{PROD}(e)$ tokens in configuration χ_2.

Next, we show v is enabled in an equivalent configuration χ_2 of G^R, and firing v changes the configuration of G^R from χ_2 to χ_1.

- For each channel e in G such that $\text{TAIL}(e) = v$, there is a channel e' in G^R such that $\text{HEAD}'(e') = v$. As we saw above, each such channel e in G has at least $\text{PROD}(e)$ tokens in configuration χ_2. Equivalently, each channel e' in G^R has at least $\text{CONS}'(e')$ tokens in the configuration χ_2. Therefore, actor v is enabled in configuration χ_2 of G^R.
- Firing actor v in configuration χ_2 of G^R removes these tokens, and deposits $\text{PROD}'(e')$ tokens into each channel e' for which $\text{TAIL}'(e') = v$ in G^R. For each such channel e' in G^R, there is a channel e in G such that $\text{HEAD}(e) = v$. Since $\text{PROD}'(e') = \text{CONS}(e)$, $\text{CONS}(e)$ tokens are deposited back into each such channel e in G with $\text{HEAD}(e) = v$. Thus, firing actor v in G^R changes G^R's configuration from χ_2 back to χ_1.

Hence, each trace of G can be "executed in reverse" for G^R. If G is deadlock-free, then a complete iteration of G from the initial configuration leads to exactly the same configuration. We may reverse this entire trace in G^R; thereby establishing that G is deadlock-free and not subject to buffer overflow if and only if G^R is. □

For the SDFG and its reverse that are depicted in Figure 2, the trace for one iteration of both the SDFG and its reverse are depicted in Figure 3. It may be verified that they are mirror images of each other: the configurations and the transitions read top to bottom in the left column are identical to the configurations and the transitions read bottom to top in the right column.

2.2 Incorporating real-time considerations

As initially defined, SDFGs do not deal with the notion of real time: "SDF is an untimed model of computation. All actors under SDF consume input tokens, perform their computation and produce outputs in one atomic operation." (Neuendorffer 2005, page 53). Real-time modeling capabilities were added by incorporating the notions of (i) *latency* between the executions of different actors (Ghamarian et al (2007)); and (ii) the *response time* to external triggering events that may occur recurrently in a sporadic manner (Singh et al (2017)). To account for the execution time of actors, an additional parameter $\left(\text{WCET} : V \to \mathbb{N}_{\geq 0}\right)$ was added to the specification of an SDFG, with the interpretation that for each $v \in V$, $\text{WCET}(v)$ is the worst-case execution

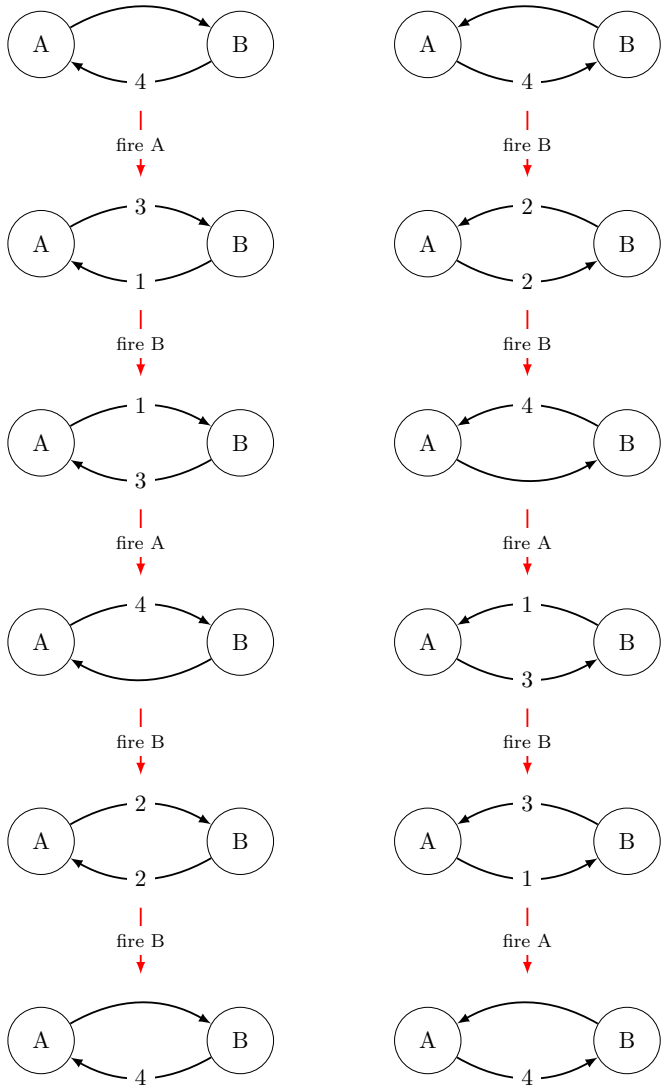
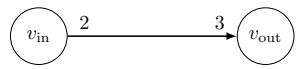


Fig. 3: Illustrating Lemma 1. The left column depicts the trace for one entire iteration of the SDFG depicted in Figure 2; the right column depicts the trace for one entire iteration of the reverse SDFG (also depicted in Figure 2). Observe that the traces are mirror images of each other.

time of a (single) firing of actor v. In order to explicitly represent real-time responsiveness to recurrent external triggering events, the SDFG model was extended as follows: for each SDFG, we are required to additionally specify

1. A single *input actor* v_{in} and a single *output actor* v_{out}. External tokens are assumed to arrive at the input actor v_{in}. That is, we can imagine an additional channel e_{in} with $\text{HEAD}(e_{\text{in}}) = v_{\text{in}}$ and $\text{TAIL}(e_{\text{in}})$ not specified, but rather representing the external environment within which the SDFG is operating. The consume rate $\text{CONS}(e_{\text{in}})$ is one, while no produce rate $\text{PROD}(e_{\text{in}})$ is specified; instead, tokens "appear" on channel e_{in} according to the period parameter (discussed next).
2. A *period* parameter, denoting the minimum duration between successive arrivals of external tokens on the input channel e_{in}.
3. A *relative deadline* parameter, denoting the maximum duration that may elapse between the arrival of an external token on the input channel e_{in} and the completion of the "corresponding" execution of the output actor v_{out} (this notion of correspondence is elaborated upon below — see Definition 6).

Let us now formalize the notion of correspondence alluded to above. Consider a simple real-time SDFG in this extended model comprising just two nodes, one the designated input and the other the designated output, and a single channel with produce rate 2 and consume rate 3 connecting the two:



Suppose that the first external input token arrives at actor v_{in} at some time-instant, thereby causing v_{in} to fire. Observe that since the produce rate of the channel leading from v_{in} to v_{out} is two while the consume rate is three, at least two firings of v_{in} must occur before actor v_{out} may fire for the first time. But since the period of the SDFG denotes only a *lower* bound on the duration between the arrival of successive external input tokens, we cannot provide an upper bound upon the time-instant at which actor v_{out} is enabled – this depends upon when the second external input token arrives at actor v_{in}. It is therefore not particularly meaningful to discuss the latency of the response to the first external input token since the response will be triggered by not the first, but the second external input token.

Ghamarian et al (2007) sidestepped the dilemma that this poses, by arguing that an entire iteration (see Definition 3) of an SDFG should be thought of as representing a single logical chunk of computation: it is not meaningful to consider the arrivals of external input tokens at the input actor, and firings of the output actor, within an iteration; instead, we should only consider the delay between the arrival of the last external input token in the iteration, and the completion of the execution of the last firing of the output actor during that iteration. In the simple SDFG above, the repetitions vector is $(3\ 2)^T$. For this SDFG, latency should be defined as the delay between the arrival of

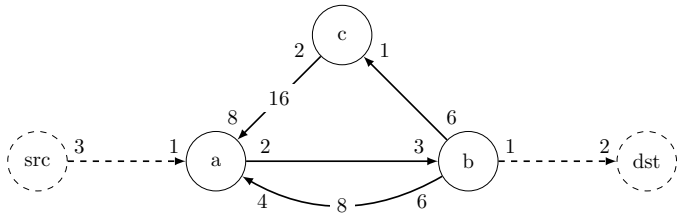


Fig. 4: Applying the transformations of Section 2.2 to the example SDFG shown in Figure 1. Suppose that actors a and b are identified as the input and output actors of the example SDFG of Figure 1. The (dashed) actors and channels designated src and dst are added, and become the designated input and output actors. Recall from Figure 1 that $\mathbf{q}(a) = 3$ and $\mathbf{q}(b) = 2$; hence the production and consumption rates assigned to the channel connecting src to a are 3 and 1 respectively, while the production and consumption rates assigned to the channel connecting b to dst are 1 and 2 respectively. This SDFG is further augmented with a relative deadline and a period parameter, both being positive integers.

the 3rd and last external input token in the iteration, and the completion of execution of the 2nd and last firing of v_{out} in the iteration.

Ghamarian et al (2007) proposed that any SDFG be preprocessed by "add[ing] an explicit source actor to the [input actor] and a destination actor to the [output actor], each of which fires by construction exactly once in every iteration of the graph. If an SDFG already has meaningful input and output actors with repetition vector entries of one, these actors can function as source and destination and no actors need to be added." Such preprocessing makes the following changes (Figure 4 illustrates the result of applying these changes to the example of Figure 1):

- Add two new actors src and dst, with $\text{WCET}(\mathsf{src}) = \text{WCET}(\mathsf{dst}) = 0$, and ensure (see below) that each will execute exactly once per iteration. These now become the designated input and output actors, while the original v_{in} and v_{out} are just "regular" actors.
- Add two new channels: $e_1 = (\mathsf{src}, v_{\text{in}})$, $e_2 = (v_{\text{out}}, \mathsf{dst})$, with
  - $\text{DELAY}(e_1) = \text{DELAY}(e_2) = 0$,
  - $\text{PROD}(e_1) = \mathbf{q}(v_{\text{in}})$, $\text{CONS}(e_1) = 1$, (recall that each actor a executes $\mathbf{q}(a)$ times per iteration), and
  - $\text{PROD}(e_2) = 1$, $\text{CONS}(e_2) = \mathbf{q}(v_{\text{out}})$.

  These assignments of DELAY, PROD, and CONS values to e_1 and e_2 ensure that src and dst both execute exactly once per iteration (i.e., $\mathbf{q}(\mathsf{src}) = \mathbf{q}(\mathsf{dst}) = 1$).

After the transformation, the arrival of $\mathbf{q}(v_{\text{in}})$ external input tokens at actor v_{in} in the original SDFG is modeled as the arrival of one external input

token at src[5]. If the period parameter of the original SDFG had represented the minimum inter-arrival duration of external input tokens at v_{in}, the period parameter of the transformed SDFG should be set equal to this original period multiplied by $\mathbf{q}(v_{\text{in}})$. The interpretation of the relative deadline parameter is ambiguous when the input and output actors may fire multiple times per iteration; we, therefore, assume that the value is actually assigned to this parameter *after* the modifications outlined above have been carried out.

Henceforth, we will assume that our *SDFGs have been pre-processed in this manner*, and that, as a consequence, we have SDFGs with designated input and output actors that are guaranteed to execute exactly once per iteration (for consistency of terminology, we will continue to refer to these input and output actors as v_{in} and v_{out} respectively). We will also assume that each actor is "reachable" via channels from v_{in}, and that v_{out} is reachable from each actor; i.e., each actor is involved in processing and relaying data from the input to the output. (Actors not reachable in this manner will not impact the real-time properties of the SDFG, and may be ignored during pre-processing to be executed in the background during run-time.)

An additional factor that must be taken into account arises from the tokens that populate each channel initially, as specified by the DELAY parameters. There are $\text{DELAY}(e)$ such tokens on each $e \in E$; since each $\text{DELAY}(e)$ is finite and since we require that each actor be reachable from src, any actor can be fired at most a finite number of times prior to v_{in} firing for the first time. The *dependency distance* denotes the maximum number of times the output actor v_{out} can be fired before exhausting the initially-supplied tokens:

Definition 6 (Dependency Distance δ (Siyoum (2014)); Correspondence) Due to the initial distribution of tokens on the channels specified by DELAY, v_{out} can fire some δ times before the first firing of v_{in}. The number δ is called the dependency distance.

For any $k \in \mathbb{N}$, the k-th firing of v_{in} is said to *correspond to* the $(k+\delta)$-th firing of v_{out}, where δ is the dependency distance. □

Suppose that in our example SDFG of Figure 1, appropriately pre-processed to take the form depicted in Figure 4, $\text{DELAY}(a,b)$ were equal to 10 rather than zero (i.e, 10 tokens were initially provided in this channel). Since $\text{CONS}(a,b) = 3$, it is evident that actor b may fire a total of three times prior to the arrival of any external input tokens at src, thereby placing three tokens on the channel connecting actor b to actor dst. Since actor dst needs two tokens on this channel to fire, it may fire once prior to the first arrival of any external input tokens at src; the dependency distance for this SDFG is therefore 1, and for all $k \in \mathbb{N}$

[5] One may choose to think of src as a dummy actor that queues the external input tokens directed at v_{in} until it has accumulated $\mathbf{q}(v_{\text{in}})$ tokens, at which instant it releases them all simultaneously to a; hence, a does not have to deal with the possibility of unbounded durations between the arrivals of the three tokens. (However, an unbounded duration may elapse before the *next* set of three tokens are released to it.) A similar interpretation may be made for dst.

the k'th firing of the input actor src corresponds to the $(k+1)$'th firing of the output actor dst.

There are a variety of semantic reasons as to why channels of an SDFG may be populated with initial tokens. From the perspective of minimizing the amount of execution that must be performed in response to the arrival of an external input token, it is a good strategy to perform as much "*pre-computation*" on the SDFG as possible, and fire as many actors as one can prior to the arrival of the first external input token. (Continuing the example above of having 10 initial tokens on channel (a, b), we could fire actor b thrice beforehand, thus placing 3 tokens on channel (b, dst), $(3 \times 6 =)$ 18 tokens on channel (b, c), and $(3 \times 6 + 8 =)$ 26 tokens on channel (b, a). The tokens on channel (b, dst) would allow actor dst to fire once, while the tokens on channel (b, c) would allow actor c to fire 18 times, placing $(18 \times 2 + 16) = 52$ tokens on channel (c, a). The final state of the channels is then

$$\begin{aligned} &\text{DELAY}(a, b) = 1; \quad \text{DELAY}(b, \mathsf{dst}) = 1; \quad \text{DELAY}(b, a) = 26; \\ &\text{DELAY}(b, c) = 0; \quad \text{DELAY}(c, a) = 52; \quad \text{DELAY}(\mathsf{src}, a) = 0. \end{aligned}$$

In much of the remainder of this paper we will assume that all enabled actors are repeatedly fired prior to run time, so that *there are no enabled actors prior to the arrival of the first external input token*. This immediately implies that the dependency distance $\delta = 0$: the k'th firing of the input actor corresponds to the k'th firing of the output actor for all $k \in \mathbb{N}$.

Firing all enabled actors prior to run time makes sense from the perspective of achieving maximum schedulability — i.e., maximizing the likelihood that all deadlines will be met during run time. If for some reason we are unable to fire all the actors before run time, we discuss, in Section 4.3, how our results may be extended to deal with SDFGs for which all enabled actors have not been fired before run time and the dependency distance δ is potentially a positive integer.

2.3 Summary of, and rationale for, the sporadic real-time SDFG model

We will refer to the recurrent task model obtained by making all the enhancements discussed in Section 2.2 above to the "traditional" SDFG model as the *sporadic real-time SDFG* model. A task in this model is specified as follows:

$$G \stackrel{\text{def}}{=} \Big\langle V, E, \text{PROD}, \text{CONS}, \text{DELAY}, \text{WCET}, v_{\text{in}}, v_{\text{out}}, D, T \Big\rangle \tag{3}$$

with

- V, E, PROD, CONS, and DELAY as specified for traditional SDFGs;
- $\text{WCET} : V \to \mathbb{N}_{\geq 0}$ specifying the worst-case execution times of the actors;
- Actors $v_{\text{in}} \in V$ and $v_{\text{out}} \in V$ being specified as the unique input and output actor, respectively; and
- $D \in \mathbb{N}_{>0}$ and $T \in \mathbb{N}_{>0}$ specifying the relative deadline and period parameters of this sporadic real-time SDFG task.

Additionally, we assume that the SDFG has been validated to be deadlock-free and free from buffer overflow, and to have the repetition rates for the input and output actors equal to one: $\mathbf{q}(v_{\text{in}}) = \mathbf{q}(v_{\text{out}}) = 1$.

We now briefly discuss the rationale behind some of the design decisions that we have made in the specification of the sporadic real-time SDFG task model.

1. **A single input actor.** Tokens are assumed to arrive at an input actor in a sporadic manner, with a specified minimum inter-arrival duration, but no maximum inter-arrival duration. Latency or response time is measured from the instant that such an input token arrives, to the instant that the corresponding firing of the output actor completes. The following simple example illustrates the problem with allowing multiple independent input actors.
   Suppose that there are two input actors a and b; external tokens arrive sporadically at each. Suppose that there are channels (a, c) and (b, c) leading from a and b to a third actor c, and both a and b must complete firing in order for c to fire. After an external token arrives at a, there is no upper bound on the duration of time before an external token arrives at b; hence, we cannot bound the duration of time between the arrival of the input token at a and the firing of c.
   It is, of course, possible to have the *same* sporadic input stream of tokens arrive at multiple actors, but this is effectively modeled by having a single dummy input actor (with $\text{WCET} = 0$) from which channels lead out to all the original input actors receiving this stream of tokens.
2. **The input and output actors execute once per repetition** ($\mathbf{q}(v_{\text{in}}) = \mathbf{q}(v_{\text{out}}) = 1$). This was discussed above, when introducing the transformation of adding the single source actor: since we cannot bound the duration between the arrival of successive external tokens from above, the concept of latency is not meaningful except in considering arrivals of a group of external input tokens for an entire iteration of the SDFG. This concept is abstracted into the new input actor v_{in} that is added, and guaranteed to have $\mathbf{q}(v_{\text{in}}) = 1$.
3. **A single output actor.** This is not a necessary restriction – it is quite possible to specify multiple output actors, with different latencies ("relative deadlines") specified for each. (Of course, each output actor so specified must satisfy the property that it executes exactly once per iteration: $\mathbf{q}(v) = 1$ for each such output actor v.) In this paper we restrict consideration to a single output actor per task in order to keep things simple; our results are easily extended to deal with multiple output actors.
4. **Each actor is reachable from v_{in}, and v_{out} is reachable from each actor.** Any actor from which v_{out} is not reachable need not fire at all in order to ensure that v_{out} fires in response to a firing of v_{in}.
   An actor v that is not reachable from v_{in}, but from which v_{out} can be reached, does not need external input tokens and may be fired repeatedly prior to run time to create a buildup of supply tokens for v_{out}, subject to memory constraints. The $\text{DELAY}(e)$ values of the outgoing channels of v can

be modified to reflect the accumulated tokens. Our scheduling algorithm may be able to assign deadlines to firings of v without any modification. However, we choose to simplify our analysis by assuming that actors like v are not present in the SDFG.
Actors that cannot be reached from v_{in} *and* from which v_{out} is unreachable need not be scheduled during run time to ensure real-time correctness: their presence has no impact on schedulability. In practice, such actors may be executed in the background when there are no real-time actors awaiting execution.

5. **No actors are enabled before the first external input arrives.** (And as a result, $\delta = 0$.) As we had argued above, performing pre-processing prior to run time by maximally firing all enabled actors is a reasonable strategy from the perspective of minimizing the computational workload. We, therefore, assume this in the remainder of this paper. However, we point out that this is not necessary – we describe in Section 4.3 how our algorithms may be extended to deal with the case where such pre-processing is not done for whatever reason, and δ is potentially a positive integer.

3 Three-parameter sporadic tasks

We now provide a very brief introduction to terminology, notation, and some basic concepts associated with the 3-parameter sporadic task model (Mok (1983)), which is widely used in real-time scheduling theory. A 3-parameter sporadic task $\tau_i = (C_i, D_i, T_i)$ is characterized by a WCET C_i, a relative deadline parameter D_i, and a period T_i. Such a task generates an unbounded sequence of jobs, with each job having an execution requirement $\leq C_i$ and successive arrivals at least T_i time units apart. Each job is required to complete by a deadline that is D_i time units after its arrival time.

The scheduling of systems of 3-parameter sporadic tasks upon preemptive uniprocessors by the earliest deadline first scheduling algorithm (EDF) has been extensively studied, and algorithms derived for determining whether a given task system is EDF-schedulable or not. These algorithms make use of the concept of the *demand bound function* (Baruah et al (1990)). For any sporadic task τ_i and any real number $t > 0$, the demand bound function $\text{DBF}(\tau_i, t)$ is the largest cumulative execution requirement of all jobs that can be generated by τ_i to have both their arrival times and their deadlines within a contiguous interval of length t. It is evident that the cumulative execution requirement of jobs of τ_i over an interval $[t_o, t_o + t)$ is maximized if one job arrives at the start of the interval – i.e., at time-instant t_o – and subsequent jobs arrive as rapidly as permitted — i.e., at instants $t_o + T_i$, $t_o + 2T_i$, $t_o + 3T_i$, ... (this fact is formally proved in Baruah et al (1990)). We therefore have (Baruah et al (1990)):

$$\text{DBF}(\tau_i, t) \stackrel{\text{def}}{=} \max\left(0, \left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1\right) \times C_i\right)$$

A *load* parameter, based upon the DBF function, may be defined for any sporadic task system τ as follows:

$$\text{LOAD}(\tau) \stackrel{\text{def}}{=} \max_{t>0} \left(\frac{\sum_{\tau_i \in \tau} \text{DBF}(\tau_i, t)}{t} \right)$$

It has been shown (Baruah et al (1990)) that a necessary and sufficient condition for 3-parameter sporadic task system τ to be EDF-schedulable on a unit-speed preemptive uniprocessor is that $\text{LOAD}(\tau) \leq 1$. Pseudo-polynomial algorithms are known (Baruah et al (1990); Ripoll et al (1996); Zhang and Burns (2009)) for computing $\text{LOAD}(\tau)$, for task systems τ possessing the additional property that the quantity $(\sum_{\tau_i \in \tau} C_i/T_i)$ is bounded from above by a constant < 1. Polynomial-time approximation schemes (PTAS's) have also been derived that are able to compute an approximation to $\text{LOAD}(\tau)$ in polynomial time, to any desired degree of accuracy (Fisher et al (2006)).

4 DBF representation of computational demand

In this section we will develop a polynomial-time algorithm that accepts as input any sporadic real-time SDFG G and determines a collection of at most $2 \times |V|$ 3-parameter sporadic tasks (here, $|V|$ denotes the number of actors in the sporadic real-time SDFG), that is equivalent to it from the perspective of the demand bound function: for any $t \in \mathbb{R}_{\geq 0}$, the maximum cumulative execution requirement by jobs of these 3-parameter sporadic tasks with both arrival times and deadlines within any contiguous interval of duration t is exactly equal to the maximum cumulative execution requirement by the sporadic real-time SDFG over any contiguous interval of duration t. Such a transformation allows us to use the vast DBF-based machinery for schedulability analysis that has been developed in the real-time scheduling theory community; for instance, it allows for the optimal EDF-based scheduling of collections of independent sporadic real-time SDFGs upon a preemptive uniprocessor (or indeed of collections of independent tasks represented using a mixture of different models –SDFG's, 3-parameter sporadic tasks, etc.– for all of which techniques have been developed for representing their computational demand using the DBF abstraction).

As with 3-parameter sporadic tasks (Section 3), let us characterize the execution requirement of a sporadic real-time SDFG by a demand bound function (DBF): for any sporadic real-time SDFG G and any positive real number t, let $\text{DBF}(G, t)$ denote the maximum cumulative execution requirement that could be generated by SDFG G over a contiguous interval of duration t. Let $k(G, t)$ denote the following function:

$$k(G,t) \stackrel{\text{def}}{=} \max \left(0, \left(\left\lfloor \frac{t-D}{T} \right\rfloor + 1 \right) \right) \tag{4}$$

(As we did in writing Γ for $\Gamma(G)$ and $\mathbf{q}$ for $\mathbf{q}(G)$, henceforth we will often simplify our notation and write $k(t)$ rather than $k(G,t)$ when the SDFG G under consideration is evident.)

It is evident, using an argument analogous to those used in computing DBF for 3-parameter sporadic tasks, that over any contiguous time-interval of duration t there may be at most $k(t)$ external input tokens arriving at v_{src} for which the corresponding firings of v_{dst} must occur within the interval (this happens when the first external input token arrives at the start of the interval, and successive external input tokens arrive exactly T time units apart). Since each arrival of an external input token at v_{src} triggers one iteration (see Definition 3) of G, an upper bound for $\text{DBF}(G,t)$ may be obtained by simply assuming that each actor $v \in V$ fires a total of $\mathbf{q}[v]$ times during each such iteration, thereby obtaining the bound

$$\text{DBF}(G,t) \leq k(t) \times \sum_{v \in V} \big(\mathbf{q}[v]\text{WCET}(v)\big) \tag{5}$$

However this bound, while safe, is not necessarily tight – the presence of initial tokens on some of the channels (as represented by the $\text{DELAY}(e)$ values) means that not all firings of all actors need to take place during the current iteration – some firings may be postponed to later (thereby "spreading out" the computational requirement of this SDFG). Our approach to determining which firings may be postponed in this manner is to compute a *skip vector* $\mathbf{s}(G)$ of non-negative integers, with $|V|$ components, which will represent the maximum number of firings of each actor that we may "skip" as a consequence of the presence of initial tokens on the channels[6]. That is, we will show that for each actor $v \in V$ the computed skip-vector value $\mathbf{s}(G)[v]$ is the largest integer possessing the property that actor a will need to complete no more than

$$\max\Big(0, \big(k(t) \times \mathbf{q}[v]\big) - \mathbf{s}(G)[v]\Big)$$

firings over any contiguous interval of duration t.

(As we have done at previous points in this paper, we will often simplify our notation and write $\mathbf{s}$ for $\mathbf{s}(G)$ when the SDFG G under consideration is evident.)

In Section 4.1, we illustrate how these skip-vector values, once computed, will be used during run time for assigning deadlines to the firings of actors. In Section 4.2, we will derive an algorithm that computes the skip vector for any sporadic real-time SDFG with a run time that is polynomial in the representation of the sporadic real-time SDFG. In Section 4.3, we will extend this algorithm to deal with the case where all actors have not been fired maximally before run time and the dependency distance δ is not necessarily zero.

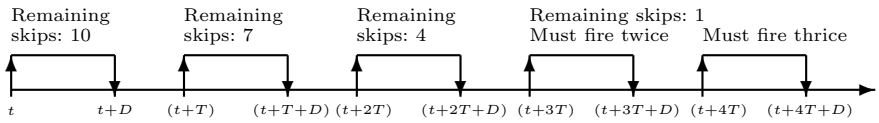


Fig. 5: Illustrating the use of skip vectors, for an actor v with $\mathbf{q}[v] = 3$ and $\mathbf{s}[v] = 10$.

4.1 Run-time scheduling

Suppose that we have determined, for a particular actor $v \in V$ in given sporadic real-time SDFG G, that $\mathbf{s}[v] = 10$ and $\mathbf{q}[v] = 3$ (i.e., the actor fires three times per iteration, but a maximum of ten firings may be skipped). Suppose an external input token arrives at v_{in} at some time-instant t, we would schedule two firings of actor v to complete by a deadline $t + 3T + D$, and a third firing of actor v to complete by a deadline $t + 4T + D$. This is equivalent, from the perspective of having an equivalent demand bound function for all values of t, to representing actor v's computational requirements by two 3-parameter sporadic tasks: one with parameters $(2\,\text{WCET}(v), 3T + D, T)$, and one with parameters $(\text{WCET}(v), 4T + D, T)$.

We now explain the rationale behind this strategy. If we were to not schedule any firings of actor v in response to the arrival of the first external input token at v_{in}, we would have "used up" three of the ten skips that are permitted. Similarly not scheduling any firings of v in response to the arrivals of the second and third input tokens at v_{in} would use up an additional six skips; the arrival of the fourth input token at v_{in} would require us to schedule two firings of v to complete within an interval of duration D from the token's arrival, and subsequent arrivals of input tokens would require us to schedule three firings of v within an interval of duration D from each such token's arrival.

Recall that our objective is to minimize the computational demand of the task, which, as quantified by DBF, is a *worst case* measure; under such a strategy, the DBF for the task is defined by these future iterations during which no firings may be skipped — applied to all the actors, this would be exactly equal to the upper bound of Expression 5. So instead we do schedule the three firings of v associated with the arrival of each external input token at v_{in} (at, say, time-instant t), but rather than assigning them a deadline at $t + D$, we assign them later deadlines, thereby "spreading out" their contribution to the DBF. We know that the next three external input tokens cannot arrive before time-instants $t + T$, $t + 2T$, and $t + 3T$ – see Figure 5. Since we are allowed to skip 10 firings of the actor, we may skip all three firings for the first three iterations of the SDFG, and one of the firings for the next (i.e., fourth) iteration of the SDFG; however, we cannot skip the other two firings for the fourth iteration, nor any for the fifth (and future) iterations. We, therefore, schedule

[6] We will show, in Lemma 2, that this skip vector is uniquely defined for a given G.

two of the firings of a associated with the current iteration to complete by the deadline of the fourth iteration, and the third to complete by the deadline of the fifth iteration. As shown in Figure 5, the deadline of the fourth iteration is $\geq (t + 3T + D)$ while the deadline of the fifth iteration is $\geq (t + 4T + D)$; hence the decision to schedule two firings of actor v to complete by a deadline $t+3T+D$, and a further firing of actor v to complete by a deadline $t+4T+D$.

We now generalize the example above from $\mathbf{q}[v] \leftarrow 3$ and $\mathbf{s}[v] \leftarrow 10$ to arbitrary values for $\mathbf{q}[v]$ and $\mathbf{s}[v]$. Letting

$$r[v] \stackrel{\text{def}}{=} \mathbf{s}[v] \bmod \mathbf{q}[v]$$
$$\text{and } f[v] \stackrel{\text{def}}{=} \left\lfloor \mathbf{s}[v]/\mathbf{q}[v] \right\rfloor,$$

it is straightforward to show that in response to an external input token's arrival at time-instant t, we would schedule each actor v to have $\big(\mathbf{q}[v] - r[v]\big)$ firings with a deadline at $\big(f[v] \cdot T + D\big)$, and the remaining $r[v]$ firings with a deadline at $\big((f[v] + 1) \cdot T + D\big)$. Equivalently, for all $t \in \mathbb{R}_{\geq 0}$ actor v's contribution to the SDFG task's demand bound function is equal to the demand bound function of the two 3-parameter sporadic tasks:

WCET	RELATIVE DEADLINE	PERIOD
$\big(\mathbf{q}[v] - r[v]\big) \cdot \text{WCET}(v)$	$f[v] \cdot T + D$	T
$r[v] \cdot \text{WCET}(v)$	$\big(f[v] + 1\big) \cdot T + D$	T

Optimality of run-time scheduler If a sporadic real-time SDFG task set is EDF-schedulable for some deadline and period assignments, it is also EDF-schedulable using the above assignments since the above assignments minimize the DBF. Thus, the above assignments are optimal for EDF-schedulability.

EDF itself is optimal in the sense of feasibility, as shown by Dertouzos (1974). EDF can find a feasible schedule for a set of independent jobs with arbitrary release times and deadlines on a preemptive uniprocessor if a feasible schedule exists. Although the firings of the actors have precedence constraints between them, the time-slice swapping argument that supports the optimality of EDF still holds for the firings since the above assignments ensure that the deadline of a preceding firing does not exceed the deadline of the following firing.

4.2 Computing the skip vector

Our intent is that the skip vector value $\mathbf{s}[v]$ denote the maximum number of times the execution of actor v may be skipped, due to the presence of initial tokens on the edges. The larger the components of the skip vector, the better in the sense that deadlines assigned to more firings of actors may be postponed. (We will show below, in Lemma 2, that such a quest is well-defined in that there cannot be multiple incomparable skip vectors for the same SDFG.)

Consider any channel $e \in E$ of the sporadic real-time SDFG under consideration, and let $u = \text{TAIL}(e)$, $v = \text{HEAD}(e)$. Let n_u and n_v denote the number

of times that actors u and v have fired by some point in time; since the number of tokens consumed during the firing of this channel cannot exceed the number of tokens produced in it plus the number of tokens initially placed upon it, it must be the case that

$$n_u \cdot \text{PROD}(e) + \text{DELAY}(e) \geq n_v \cdot \text{CONS}(e). \tag{6}$$

Let us instantiate Equation 6 above to the end of the k'th iteration of the sporadic real-time SDFG under consideration, when $n_u \leftarrow (k \cdot \mathbf{q}[u] - \mathbf{s}[u])$ and $n_v \leftarrow (k \cdot \mathbf{q}[v] - \mathbf{s}[v])$. Therefore

$$\begin{aligned}
&\Big(k \cdot \mathbf{q}[u] - \mathbf{s}[u]\Big) \cdot \text{PROD}(e) + \text{DELAY}(e) \\
&\qquad \geq \Big(k \cdot \mathbf{q}[v] - \mathbf{s}[v]\Big) \cdot \text{CONS}(e) \\
\Leftrightarrow &k \cdot \mathbf{q}[u] \cdot \text{PROD}(e) - \mathbf{s}[u] \cdot \text{PROD}(e) + \text{DELAY}(e) \\
&\qquad \geq k \cdot \mathbf{q}[v] \cdot \text{CONS}(e) - \mathbf{s}[v] \cdot \text{CONS}(e) \\
\Leftrightarrow &k \cdot \overbrace{\Big(\mathbf{q}[u] \cdot \text{PROD}(e) - \mathbf{q}[v] \cdot \text{CONS}(e)\Big)}^{=0 \text{ by the balance equation (Eqn. 2)}} + \text{DELAY}(e) \\
&\qquad \geq \mathbf{s}[u] \cdot \text{PROD}(e) - \mathbf{s}[v] \cdot \text{CONS}(e) \\
\Leftrightarrow \quad &\text{DELAY}(e) \geq \mathbf{s}[u] \cdot \text{PROD}(e) - \mathbf{s}[v] \cdot \text{CONS}(e)
\end{aligned}$$

We will use this relationship that we have just derived above (replacing u and v with $\text{TAIL}(e)$ and $\text{HEAD}(e)$):

$$\begin{aligned}
&\mathbf{s}[\text{TAIL}(e)] \cdot \text{PROD}(e) - \mathbf{s}[\text{HEAD}(e)] \cdot \text{CONS}(e) \leq \text{DELAY}(e) \\
\Leftrightarrow \quad &\mathbf{s}[\text{TAIL}(e)] \leq \left\lfloor \frac{\text{DELAY}(e) + \mathbf{s}[\text{HEAD}(e)] \cdot \text{CONS}(e)}{\text{PROD}(e)} \right\rfloor
\end{aligned} \tag{7}$$

to help us compute the skip vector: our objective is to determine the largest values for $\mathbf{s}[a]$ for all actors a, such that Equation 7 is satisfied across all channels of the SDFG. Before doing so, we prove in Lemma 2 below, that there cannot be multiple incomparable skip vectors for the same SDFG.

Lemma 2 *The skip vector* $\mathbf{s}$ *is unique.*

Proof By contradiction. Assume that there exist two distinct skip vectors $\mathbf{s}$ and $\mathbf{s}'$ for which Equation 7 holds for all channels $e \in E$. Assume also that both $\mathbf{s}$ and $\mathbf{s}'$ are maximal, so that Equation 7 would not hold for either of them if we increased some values of $\mathbf{s}$ or $\mathbf{s}'$.

Now let $\mathbf{s}''$ be defined as follows: $\mathbf{s}''[v] \leftarrow \max(\mathbf{s}[v], \mathbf{s}'[v])$ for all $v \in V$. For each channel $e \in E$ of the SDFG we have

$$\begin{aligned}\mathbf{s}''[\text{TAIL}(e)] &= \max(\mathbf{s}[\text{TAIL}(e)], \mathbf{s}'[\text{TAIL}(e)]) \\ &\leq \max\left(\begin{array}{l}\left\lfloor \frac{\text{DELAY}(e)+\mathbf{s}[\text{HEAD}(e)]\cdot\text{CONS}(e)}{\text{PROD}(e)} \right\rfloor \\ \\ \left\lfloor \frac{\text{DELAY}(e)+\mathbf{s}'[\text{HEAD}(e)]\cdot\text{CONS}(e)}{\text{PROD}(e)} \right\rfloor\end{array}\right) \\ &= \left\lfloor \frac{\text{DELAY}(e) + \mathbf{s}''[\text{HEAD}(e)] \cdot \text{CONS}(e)}{\text{PROD}(e)} \right\rfloor,\end{aligned}$$

from which it follows that Equation 7 holds for all channels $e \in E$ of the SDFG also when using skip vector $\mathbf{s}''$. It follows that $\mathbf{s}$ and $\mathbf{s}'$ cannot both be maximal. □

We now derive our algorithm for determining this (unique maximal) skip vector. We start out defining some additional terminology and notation. For each actor v, let $\mathbf{s}^{\text{ub}}[v]$ denote an upper bound on the value of $\mathbf{s}[v]$; we will refer to these upper bounds as *skip estimates*. Our algorithm for computing the skip vector values will initialize these skip estimates as follows:

$$\mathbf{s}^{\text{ub}}[v] \leftarrow \begin{cases} 0, & \text{if } v = v_{\text{out}} \\ \infty, & \text{otherwise} \end{cases} \tag{8}$$

It is evident that these initial values on $\mathbf{s}^{\text{ub}}$ are indeed upper bounds on the skip vector values: since all skip vector values are necessarily finite, ∞ is an upper bound on the actual skip-vector values. Recall from Definition 6 that our model assumes that the dependency distance between the input and output actors equal zero ($\delta = 0$)[7]. Thus, v_{out} cannot be skipped at all, i.e. $\mathbf{s}[v_{\text{out}}] = 0$, and $\mathbf{s}^{\text{ub}}[v_{\text{out}}] = 0$ is a valid upper bound.

We will additionally maintain, as auxiliary variables[8], a *constraining actor* $\boldsymbol{\pi}[v] \in V \bigcup \{\text{NIL}\}$ for each actor $v \in V$, that are initialized as follows

$$\boldsymbol{\pi}[v] \leftarrow \text{NIL} \quad \text{for all } v \tag{9}$$

The intended semantics of these constraining actors will be explained a bit later.

Relaxing (along) a channel. For a given assignment of $\mathbf{s}^{\text{ub}}$ values to all the actors, the process of *relaxing* a channel e consists of the following steps (also depicted in Algorithm 1):

[7] As we have stated in Section 2.3, in this paper we have assumed $\delta = 0$ in order to keep things simple. However, our algorithm may be extended to handle non-zero dependency distances: we briefly describe the extension in Section 4.3.

[8] These are *auxiliary* variables in the sense that they will not appear in the actual code that implements the algorithm. They are defined solely to record information that is useful in proving properties (in our case, computational complexity) of the algorithm.

Algorithm 1 Relaxing channel e

procedure RELAX(e)
  if $\left(\mathbf{s}^{\text{ub}}[\text{TAIL}(e)] > \left\lfloor \frac{\text{DELAY}(e)+\mathbf{s}^{\text{ub}}[\text{HEAD}(e)]\cdot\text{CONS}(e)}{\text{PROD}(e)} \right\rfloor\right)$ **then**
    $\mathbf{s}^{\text{ub}}[\text{TAIL}(e)] \leftarrow \left\lfloor \frac{\text{DELAY}(e)+\mathbf{s}^{\text{ub}}[\text{HEAD}(e)]\cdot\text{CONS}(e)}{\text{PROD}(e)} \right\rfloor$;
    $\boldsymbol{\pi}[\text{TAIL}(e)] \leftarrow \text{HEAD}(e)$;
  end if
end procedure

§1. Determine whether the current skip estimates violate Condition 7 across channel e:

$$\mathbf{s}^{\text{ub}}[\text{TAIL}(e)] > \left\lfloor \frac{\text{DELAY}(e) + \mathbf{s}^{\text{ub}}[\text{HEAD}(e)] \cdot \text{CONS}(e)}{\text{PROD}(e)} \right\rfloor \tag{10}$$

If HEAD(e) is skipped $\mathbf{s}^{\text{ub}}[\text{HEAD}(e)]$ times, then the maximum number of times TAIL(e) can be skipped is specified by the right-hand side of the expression above. Hence if the current skip estimate for HEAD(e) is equal to $\mathbf{s}^{\text{ub}}[\text{HEAD}(e)]$ and the expression above holds, then the current skip estimate for TAIL(e) is too high — we must *reduce* it to equal the right-hand side of the expression above. (If Condition 7 is not violated – i.e., the expression above does not hold – then there is nothing more to be done, and the relaxation is complete.)

§2. If Condition 10 is true, we decrease the skip estimate of TAIL(e) in order to have it satisfy Condition 7:

$$\mathbf{s}^{\text{ub}}[\text{TAIL}(e)] \leftarrow \left\lfloor \frac{\text{DELAY}(e) + \mathbf{s}^{\text{ub}}[\text{HEAD}(e)] \cdot \text{CONS}(e)}{\text{PROD}(e)} \right\rfloor \tag{11}$$

§3. If such an update occurs, the fact is "recorded" by setting

$$\boldsymbol{\pi}[\text{TAIL}(e)] \leftarrow \text{HEAD}(e) \tag{12}$$

Thus the value of $\boldsymbol{\pi}[a]$, if not NIL, denotes the HEAD of the channel whose relaxation has resulted in the value currently assigned to $\mathbf{s}^{\text{ub}}[a]$.

We now prove some properties that are maintained by the $\mathbf{s}^{\text{ub}}$ and $\boldsymbol{\pi}$ values as we perform repeated relaxations.

Lemma 3 *Suppose that the* $\mathbf{s}^{\text{ub}}$ *and* $\boldsymbol{\pi}$ *values are initialized as specified in Expressions 8-9, and a series of relaxations performed. The following holds after each relaxation:*

$$\forall\ v \in V\ \ : \mathbf{s}^{\text{ub}}[v] \geq \mathbf{s}[v] \tag{13}$$

Proof This lemma is proved by induction on the number of relaxations that are performed. It is obviously correct at initialization (as pointed out in the paragraph following Expression 8). For the inductive step, suppose for an inductive hypothesis (IH) that it is true prior to the instant that a relaxation

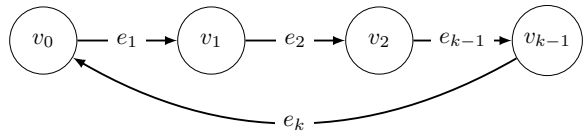


Fig. 6: Illustrating the proof of Lemma 4. (The labels on the edges denote the channel names, *not* the delays on the channel.)

operation is performed across the channel e). The only skip estimate that is changed is $\mathbf{s}^{\text{ub}}[\text{TAIL}(e)]$; it changes as follows:

$$\begin{aligned}\mathbf{s}^{\text{ub}}[\text{TAIL}(e)] &\leftarrow \left\lfloor \frac{\text{DELAY}(e) + \mathbf{s}^{\text{ub}}[\text{HEAD}(e)] \cdot \text{CONS}(e)}{\text{PROD}(e)} \right\rfloor \\ &\geq \left\lfloor \frac{\text{DELAY}(e) + \mathbf{s}[\text{HEAD}(e)] \cdot \text{CONS}(e)}{\text{PROD}(e)} \right\rfloor \text{ (By the IH)} \\ &\geq \mathbf{s}[\text{TAIL}(e)] \text{ (By Expression 7)}\end{aligned}$$

and the invariant is thereby maintained. □

We now use the constraining actor auxiliary variables (the $\boldsymbol{\pi}[\cdot]$ values) to define a *constraint graph* $G_{\boldsymbol{\pi}}$ as follows[9]. The vertices in $G_{\boldsymbol{\pi}}$ are exactly the actors in G. The edges in $G_{\boldsymbol{\pi}}$ are the pairs $(v, \boldsymbol{\pi}(v))$, where v is an actor in G and $\boldsymbol{\pi}(v) \neq \text{NIL}$.

Lemma 4 *Suppose that the* $\mathbf{s}^{\text{ub}}$ *and* $\boldsymbol{\pi}$ *values are initialized as specified in Expressions 8-9, and a series of relaxations performed. The following invariant is maintained: the constraint graph* $G_{\boldsymbol{\pi}}$ *contains no cycle.*

Proof This is obviously correct at initialization: since $\boldsymbol{\pi}(v) = \text{NIL}$ for all v, $G_{\boldsymbol{\pi}}$ contains no edges. Now consider $G_{\boldsymbol{\pi}}$ after a series of relaxation steps have been carried out. Suppose for the sake of contradiction that some relaxation step creates a cycle in $G_{\boldsymbol{\pi}}$. Let this cycle comprise the actors $\langle v_o, v_1, \ldots, v_{k-1}, v_k = v_o \rangle$, such that $\boldsymbol{\pi}(v_{i-1}) = v_i$ for each i, $1 \leq i \leq k$. Without loss of generality, assume that relaxing the channel (v_{k-1}, v_o) caused the cycle. Let e_i denote the channel (v_{i-1}, v_i) for each i, $1 \leq i \leq k$. (Figure 6 illustrates such a hypothetical cycle for the case when the cycle comprises four actors.)

We now examine the skip estimates just prior to the relaxation of e_k (i.e., of channel (v_{k-1}, v_o)). It must be the case that for all i, $1 \leq i \leq (k-1)$,

$$\boldsymbol{\pi}[v_{i-1}] = v_i \tag{14}$$

$$\mathbf{s}^{\text{ub}}[v_{i-1}] \geq \left\lfloor \frac{\mathbf{s}^{\text{ub}}[v_i] \cdot \text{CONS}(e_i) + \text{DELAY}(e_i)}{\text{PROD}(e_i)} \right\rfloor \tag{15}$$

[9] As with the $\boldsymbol{\pi}[\cdot]$ auxiliary variables, the constraint graph is a concept used only in our proofs — no such graph is explicitly constructed by our algorithm.

Since we are assuming that the relaxation of channel e_k would lead to the cycle in the constraint graph (by setting $\boldsymbol{\pi}(v_{k-1}) \leftarrow v_0$), it must be the case that

$$\mathbf{s}^{\text{ub}}[v_{k-1}] > \left\lfloor \frac{\mathbf{s}^{\text{ub}}[v_o] \cdot \text{CONS}(e_k) + \text{DELAY}(e_k)}{\text{PROD}(e_k)} \right\rfloor \tag{16}$$

Now let us relax along the channels $e_{k-1}, e_{k-2}, \ldots, e_1$ in order, so that all of Eqn 15 becomes a strict equality: for all i, $1 \leq i \leq (k-1)$,

$$\mathbf{s}^{\text{ub}}[v_{i-1}] = \left\lfloor \frac{\mathbf{s}^{\text{ub}}[v_i] \cdot \text{CONS}(e_i) + \text{DELAY}(e_i)}{\text{PROD}(e_i)} \right\rfloor \tag{17}$$

Note that Eqn 16 continues to hold since $\mathbf{s}^{\text{ub}}[v_{k-1}]$ has not changed and $\mathbf{s}^{\text{ub}}[v_o]$ has not *in*creased.

Let us now consider the reverse SDFG G^R of G (see Definition 5 for a quick refresher). Recall that all the channels in G^R are reversed versions of the channels in G; for $1 \leq i \leq k$, let e'_i denote the channel (v_i, v_{i-1}) in G^R. We will consider execution traces of G^R in which the number of firings of the individual actors are specified according to the $\mathbf{s}^{\text{ub}}$ values shown above to satisfy Eqns 16 and 17. More specifically, consider execution traces of G^R in which the actor v_{k-1} fires exactly

$$f[v_{k-1}] \stackrel{\text{def}}{=} \mathbf{s}^{\text{ub}}[v_{k-1}] \tag{18}$$

times. Since G is assumed to be deadlock-free, it follows from Lemma 1 that G^R is deadlock-free as well. Such traces therefore exist.

Now, let $f[v_i]$ denote the maximum number of firings of actor v_i that are possible in any such trace of G^R. We claim the following:

Fact 1 *For each i, $1 \leq i \leq (k-1)$,*

$$f[v_{i-1}] \leq \mathbf{s}^{\text{ub}}[v_{i-1}]$$

Proof (Fact 1) We prove Fact 1 by induction on i, starting with $i \leftarrow (k-1)$ and decreasing it in steps of size 1 until $i = 1$.

The base case, $i = (k-1)$, is true by the definition of $f[v_{k-1}]$.

For the inductive step, assume, as an inductive hypothesis (IH), that $f[v_i] \leq \mathbf{s}^{\text{ub}}[v_i]$. Now observe that the total number of tokens available on the channel $e'_i = (v'_i, v'_{i-1})$ is at most

$$\big(f[v_i] \cdot \text{PROD}'(e'_i) + \text{DELAY}'(e'_i)\big).$$

Since $\text{CONS}'(e'_i)$ tokens are consumed each time actor v'_{i-1} fires, it must be the case that

$$\begin{aligned} f[v_{i-1}] &\leq \left\lfloor \frac{f[v_i] \cdot \text{PROD}'(e'_i) + \text{DELAY}'(e'_i)}{\text{CONS}'(e'_i)} \right\rfloor \\ &= \left\lfloor \frac{f[v_i] \cdot \text{CONS}(e_i) + \text{DELAY}(e_i)}{\text{PROD}(e_i)} \right\rfloor \quad (\text{Since } \text{PROD}'(e'_i) = \\ &\quad \text{CONS}(e_i); \text{CONS}'(e'_i) = \text{PROD}(e_i), \text{ and } \text{DELAY}'(e'_i) \\ &\quad = \text{DELAY}(e_i) - \text{see Definition 5}) \\ &\leq \left\lfloor \frac{\mathbf{s}^{\text{ub}}[v_i] \cdot \text{CONS}(e_i) + \text{DELAY}(e_i)}{\text{PROD}(e_i)} \right\rfloor \quad (\text{From the IH}) \\ &= \mathbf{s}^{\text{ub}}[v_{i-1}] \quad (\text{By Eqn 17}) \end{aligned}$$

□

Consider now the channel (v_o, v_{k-1}) in G^R. The total number of tokens that are made available on this channel is at most

$$\begin{aligned} & f[v_o] \times \text{PROD}'(e'_k) + \text{DELAY}'(e'_k) \\ \leq \quad & (\text{By Fact 1}) \quad \mathbf{s}^{\text{ub}}[v_o] \times \text{PROD}'(e'_k) + \text{DELAY}'(e'_k) \\ = \quad & (\text{Since } \text{PROD}'(e'_k) = \text{CONS}(e_k) \text{ and } \text{DELAY}'(e'_k) = \text{DELAY}(e_k)) \\ & \mathbf{s}^{\text{ub}}[v_o] \times \text{CONS}(e_k) + \text{DELAY}(e_k) \end{aligned}$$

Hence the maximum number of times the actor v_{k-1} may fire in these traces of G^R is

$$\begin{aligned} \leq \quad & \left\lfloor \frac{\mathbf{s}^{\text{ub}}[v_o] \times \text{CONS}(e_k) + \text{DELAY}(e_k)}{\text{CONS}'(e'_k)} \right\rfloor \\ = \quad & \left\lfloor \frac{\mathbf{s}^{\text{ub}}[v_o] \times \text{CONS}(e_k) + \text{DELAY}(e_k)}{\text{PROD}(e_k)} \right\rfloor \\ = \quad & \left\lfloor \frac{\mathbf{s}^{\text{ub}}[v_k] \times \text{CONS}(e_k) + \text{DELAY}(e_k)}{\text{PROD}(e_k)} \right\rfloor \end{aligned}$$

(since $\text{CONS}(e_k) = \text{PROD}'(e'_k)$, from the definition of reverse – see Definition 5 — and $v_o = v_k$.)

Recall that we had chosen $f[v_{k-1}] \leftarrow \mathbf{s}^{\text{ub}}[v_{k-1}]$ (Equation 18). We have thus shown that

$$\mathbf{s}^{\text{ub}}[v_{k-1}] \leq \left\lfloor \frac{\mathbf{s}^{\text{ub}}[v_k] \times \text{CONS}(e_k) + \text{DELAY}(e_k)}{\text{PROD}(e_k)} \right\rfloor .$$

But this contradicts Inequality 16; we therefore have a contradiction to our assumption that there is a cycle in the constraint graph $G_{\boldsymbol{\pi}}$ □

To understand the significance of Lemma 4, let us consider the constraint graph when all the skip bounds (the $\mathbf{s}^{\text{ub}}[\cdot]$ values) have converged to a fixed point – Expression 7 holds for all channels in the SDFG. Since none of these fixed-point values of $\mathbf{s}^{\text{ub}}[\cdot]$ can be reduced any further, they are, by Lemmas 2 and 3, equal to the $\mathbf{s}[\cdot]$ values that we seek to determine. At this stage, the following Lemma establishes that the constraint graph comprises a tree that is rooted at v_{out}.

Lemma 5 *The constraint graph $G_{\boldsymbol{\pi}}$ is a directed in-tree rooted at v_{out}.*

Proof After convergence, only $\boldsymbol{\pi}[v_{\text{out}}] = \text{NIL}$ as by assumption v_{out} is reachable from all actors and therefore no actor v can have $s[v] = \infty$. Consider any directed path in $G_{\boldsymbol{\pi}}$. By Lemma 4 this path has no cycles, and therefore can contain no actor twice. As each actor except v_{out} has an outgoing edge, it follows that all paths, if extended, must eventually end in v_{out}. In addition, the undirected graph given by ignoring the directions on the edges of $G_{\boldsymbol{\pi}}$ has $|V|$ vertices, $|V|-1$ edges and is connected, hence it must be a tree (by basic graph theory). The lemma follows. □

Let us define the *constraint distance* of any actor v to denote the length of the (unique) path in this tree from v to v_{out}. These constraint distances are not known beforehand; however, we point out the obvious fact that the maximum constraint distance for any actor is $(|V|-1)$ where $|V|$ denotes the number of actors.

1. Recall that upon initialization, $\mathbf{s}^{\text{ub}}[v_{\text{out}}] = \mathbf{s}[v_{\text{out}}] = 0$ (assuming that the dependency distance $\delta = 0$).
2. Consider any actor v that is at constraint distance 1, and let c denote the channel for which $\text{TAIL}(c) = v$ and $\text{HEAD}(c) = v_{\text{out}}$. Upon relaxing the channel c, we will have $\mathbf{s}^{\text{ub}}[v] = \mathbf{s}[v]$.
   We do not a priori know which actors are at constraint distance 1. However, if we were to relax *every* channel once, then upon having done so we would know that each actor at constraint distance 1 has its $\mathbf{s}^{\text{ub}}$ value equal to its $\mathbf{s}$ value.
3. Once this is done and all actors at constraint distance 1 have their $\mathbf{s}^{\text{ub}}$ value equal to their $\mathbf{s}$ value, consider some actor v that is at constraint distance 2, and let c denote the channel for which $\text{TAIL}(c) = v$ and $\text{HEAD}(c) = v'$, where v' is the (unknown) actor at constraint distance 1 that lies on the path in the eventual (i.e., upon convergence to a fixed point) constraint graph from v to v_{out}. Upon relaxing the channel c, we will have $\mathbf{s}^{\text{ub}}[v] = \mathbf{s}[v]$.
   As was the case for constraint distance 1 (argued above), we do not know beforehand which actors are at constraint distance 2. However, upon relaxing every channel once again we would know that all the actors at constraint distance 2 have their $\mathbf{s}^{\text{ub}}$ value equal to their $\mathbf{s}$ value.
4. Repeating the above argument $(|V|-1)$ times, we would know that all the actors at constraint distance $\leq (|V|-1)$ have their $\mathbf{s}^{\text{ub}}$ value equal to their $\mathbf{s}$ value. And since all actors have their constraint distances in the range

$[0, 1, \ldots, |V| - 1]$, this implies that we will have successfully computed the $\mathbf{s}$ values for all actors in the SDFG.

Algorithm 2 Computing the skip vector when all enabled actors are fired maximally before run time

```
1: procedure ComputeSkipVector(G)
2:     Initialize s^ub and π according to Expressions 8–9
3:     for i ← 1 to |V| − 1 do
4:         for all channels e do
5:             Relax(e)
6:         end for
7:     end for
8:     return s^ub
9: end procedure
```

The argument above is presented in pseudo-code form in Algorithm 2. A formal proof of correctness is easily established by induction, with the loop invariant for the outer for loop that at the start of the i'th iteration, all actors at constraint distance at most $(i-1)$ have their $\mathbf{s}^{\text{ub}}$ value equal to their $\mathbf{s}$ value.

Run-time complexity. Since we perform $(|V| - 1) \times |E|$ relaxations and each relaxation takes $\Theta(1)$ time, the computational complexity of determining the skip vector is $\Theta(|V| \times |E|)$. This is clearly polynomial (quadratic) in the representation of the sporadic real-time SDFG.

4.3 Extended algorithm for the case where all enabled actors are not fired maximally before run time

As we had stated in Section 2.3, it makes sense from a real-time perspective to pre-process as much as possible, thereby minimizing the computational demand that must be accommodated during run time; hence, we believe that preprocessing the SDFG to have a dependency distance equal to zero makes sense. However, our algorithm is easily generalized to deal with SDFGs that have not been preprocessed in this manner (so that the dependency distance is some value $\delta \geq 0$).

A key idea behind the generalization is that skip values can be negative. While a positive skip value for an actor indicates the maximum number of firings of the actor that may be skipped, a negative skip value indicates the additional number of firings of the actor that need to be completed before the first deadline. A positive skip value allows the computational demand to be diffused over a number of iterations; a negative skip value forces the concentration of higher computational demand in the first iteration.

Now that negative skip values are allowed, we can describe the skip value of v_{out} when δ is potentially greater than zero. Recall from Definition 6 that

the $(k+\delta)$'th firing of v_{out} corresponds to the k'th firing of v_{in}; hence, v_{out} must fire $(\delta+1)$ times before the first deadline. It follows that v_{out} needs to be fired δ times in addition to its repetitions vector entry, $\mathbf{q}[v_{\text{out}}]=1$, in the first iteration. Thus, $\mathbf{s}[v_{\text{out}}]=-\delta$. The skip value of v_{in} is even easier to characterize. In any iteration, a single input token arrives at v_{in} and v_{in} fires exactly once, conforming to its repetitions vector entry, $\mathbf{q}[v_{\text{in}}]=1$. Thus, $\mathbf{s}[v_{\text{in}}]$ must be 0.

Knowing that $\mathbf{s}[v_{\text{out}}]$ must be $-\delta$, we could simply initialize $\mathbf{s}^{\text{ub}}[v_{\text{out}}]$ as $-\delta$ in Algorithm 2, if δ is known a priori. However, we do not need to know δ in advance because the fact that $\mathbf{s}[v_{\text{in}}]$ must be zero can be used to modify Algorithm 2 to produce the correct skip values and δ gets computed as a side effect. Let us examine the skip estimate update step of relaxation in Equation 11 when we have a different skip estimate $\mathbf{s}^{\text{ub}'}[\text{HEAD}(e)]$ for $\text{HEAD}(e)$ such that $\mathbf{s}^{\text{ub}'}[\text{HEAD}(e)]=\mathbf{s}^{\text{ub}}[\text{HEAD}(e)]+k\cdot\mathbf{q}[\text{HEAD}(e)]$ and $k\in\mathbb{Z}$:

$$
\begin{aligned}
\mathbf{s}^{\text{ub}'}[\text{TAIL}(e)] &\leftarrow \left\lfloor \frac{\text{DELAY}(e)+\mathbf{s}^{\text{ub}'}[\text{HEAD}(e)]\cdot\text{CONS}(e)}{\text{PROD}(e)} \right\rfloor \\
&= \left\lfloor \frac{\text{DELAY}(e)+(\mathbf{s}^{\text{ub}}[\text{HEAD}(e)]+k\cdot\mathbf{q}[\text{HEAD}(e)])\cdot\text{CONS}(e)}{\text{PROD}(e)} \right\rfloor \\
&= \left\lfloor \frac{\text{DELAY}(e)+\mathbf{s}^{\text{ub}}[\text{HEAD}(e)]\cdot\text{CONS}(e)}{\text{PROD}(e)} + \frac{k\cdot\mathbf{q}[\text{HEAD}(e)]\cdot\text{CONS}(e)}{\text{PROD}(e)} \right\rfloor \\
&= \left\lfloor \frac{\text{DELAY}(e)+\mathbf{s}^{\text{ub}}[\text{HEAD}(e)]\cdot\text{CONS}(e)}{\text{PROD}(e)} \right\rfloor + k\cdot\mathbf{q}[\text{TAIL}(e)] \\
&\qquad\qquad \text{(using balance equation (Eqn. 2).)} \\
&= \mathbf{s}^{\text{ub}}[\text{TAIL}(e)]+k\cdot\mathbf{q}[\text{TAIL}(e)]
\end{aligned}
$$

Thus, adding $k\cdot\mathbf{q}[v]$ to the skip estimate of an actor v results in addition of $k\cdot\mathbf{q}[u]$ to all actors u, where $\boldsymbol{\pi}[u]=v$. Recall from Lemma 5 that the constraint graph is a directed in-tree rooted at v_{out}. This, combined with the fact that $\mathbf{q}[v_{\text{out}}]=1$, implies that we can compute the skip values for a different initial value $\alpha\in\mathbb{Z}$ of $\mathbf{s}^{\text{ub}}[v_{\text{out}}]$ by adding α times the repetitions vector to the skip vector originally computed by Algorithm 2. Since $\mathbf{s}[v_{\text{in}}]$ must be zero, we must choose α to be $-\mathbf{s}^{\text{ub}}[v_{\text{in}}]$. This modification to Algorithm 2 is shown in Algorithm 3. To the best of our knowledge, Algorithm 3 is also the first polynomial-time algorithm proposed for computing the dependency distance δ.

The reduction to sporadic tasks described in Section 4.1 should be modified as follows only for actors with negative skip values:

WCET	RELATIVE DEADLINE	PERIOD
$\mathbf{q}[v]\cdot\text{WCET}(v)$	D	T
$-\mathbf{s}[v]\cdot\text{WCET}(v)$	D	∞

The second row is a job that represents the additional computational demand that needs to be met before the first deadline. We stress that it is impossible

Algorithm 3 Computing the skip vector when all enabled actors are *not* fired maximally before run time

```
1: procedure ComputeSkipVector(G)
2:     Initialize s^ub and π according to Expressions 8–9
3:     for i ← 1 to |V| − 1 do
4:         for all channels e do
5:             Relax(e)
6:         end for
7:     end for
8:     δ ← s^ub[v_in]
9:     s ← s^ub − (δ × q)
10:    return s
11: end procedure
```

for the DBF to be lower for a different set of deadline and period assignments. Thus, we have extended our EDF-schedulability test for the case where all enabled actors are not fired maximally before run time.

5 Conclusions

The Synchronous Data Flow Graph (SDFG) model is widely used in the modeling of embedded real-time systems. In this paper, we have made an effort to combine the core competencies of two communities – those studying dataflow methodologies, and researchers in real-time scheduling theory – to obtain a better understanding of the problem of achieving highly resource-efficient implementations of SDFG-modeled real-time systems.

Acknowledgements This research has been supported by NSF grants CNS 1115284, CNS 1218693, CNS 1409175, and CPS 1446631, AFOSR grant FA9550-14-1-0161, and ARO grant W911NF14-1-0499.

References

(1987) PGM – Processing Graph Method Specification. Naval Research Laboratory, prepared by the Naval Research Laboratory for use by the Navy Standard Signal Processing Program Office (PMS-412). Version 1.0

Ali HI, Akesson B, Pinho LM (2015) Generalized extraction of real-time parameters for homogeneous synchronous dataflow graphs. In: 2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, pp 701–710, DOI 10.1109/PDP.2015.57

Bamakhrama M, Stefanov T (2011) Hard-real-time scheduling of data-dependent tasks in embedded streaming applications. In: Proceedings of the Ninth ACM International Conference on Embedded Software, ACM, New York, NY, USA, EMSOFT '11, pp 195–204, DOI 10.1145/2038642.2038672, URL http://doi.acm.org/10.1145/2038642.2038672

Bamakhrama MA, Stefanov T (2012) Managing latency in embedded streaming applications under hard-real-time scheduling. In: Proceedings of the Eighth IEEE/ACM/IFIP

International Conference on Hardware/Software Codesign and System Synthesis, ACM, New York, NY, USA, CODES+ISSS '12, pp 83–92, DOI 10.1145/2380445.2380464, URL http://doi.acm.org/10.1145/2380445.2380464

Baruah S, Mok A, Rosier L (1990) Preemptively scheduling hard-real-time sporadic tasks on one processor. In: Proceedings of the 11th Real-Time Systems Symposium, IEEE Computer Society Press, Orlando, Florida, pp 182–190

Bouakaz A, Gautier T, Talpin JP (2014) Earliest-deadline first scheduling of multiple independent dataflow graphs. In: 2014 IEEE Workshop on Signal Processing Systems (SiPS), pp 1–6, DOI 10.1109/SiPS.2014.6986102

Dertouzos M (1974) Control robotics : the procedural control of physical processors. In: Proceedings of the IFIP Congress, pp 807–813

Fisher N, Baker T, Baruah S (2006) Algorithms for determining the demand-based load of a sporadic task system. In: Proceedings of the International Conference on Real-time Computing Systems and Applications, IEEE Computer Society Press, Sydney, Australia

Ghamarian AH, Stuijk S, Basten T, Geilen MCW, Theelen BD (2007) Latency minimization for synchronous data flow graphs. In: Digital System Design Architectures, Methods and Tools, 2007. DSD 2007. 10th Euromicro Conference on, pp 189–196, DOI 10.1109/DSD.2007.4341468

Khatib J, Kordon AM, Klikpo EC, Trabelsi-Colibet K (2016) Computing latency of a real-time system modeled by synchronous dataflow graph. In: Proceedings of the 24th International Conference on Real-Time Networks and Systems, RTNS 2016, Brest, France, October 19-21, 2016, pp 87–96, DOI 10.1145/2997465.2997479, URL http://doi.acm.org/10.1145/2997465.2997479

Klikpo EC, Kordon AM (2016) Preemptive scheduling of dependent periodic tasks modeled by synchronous dataflow graphs. In: Proceedings of the 24th International Conference on Real-Time Networks and Systems, RTNS 2016, Brest, France, October 19-21, 2016, pp 77–86, DOI 10.1145/2997465.2997474, URL http://doi.acm.org/10.1145/2997465.2997474

Lee EA (1986) A coupled hardware and software architecture for programmable digital signal processors. PhD thesis, EECS Department, University of California, Berkeley, URL http://www2.eecs.berkeley.edu/Pubs/TechRpts/1986/715.html

Lee EA, Messerschmitt DG (1987a) Static scheduling of synchronous data flow programs for digital signal processing. IEEE Transactions on Computers C-36(1):24–35

Lee EA, Messerschmitt DG (1987b) Synchronous data flow. Proceedings of the IEEE 75(9):1235–1245, DOI 10.1109/PROC.1987.13876

Lee EA, Seshia SA (2011) Introduction to Embedded Systems, A Cyber-Physical Systems Approach. MIT Press, URL http://LeeSeshia.org

Liu C, Layland J (1973) Scheduling algorithms for multiprogramming in a hard real-time environment. Journal of the ACM 20(1):46–61

Mohaqeqi M, Abdullah J, Yi W (2016) Modeling and analysis of data flow graphs using the digraph real-time task model. In: Proceedings of the 21st Ada-Europe International Conference on Reliable Software Technologies — Ada-Europe 2016 - Volume 9695, Springer-Verlag New York, Inc., New York, NY, USA, pp 15–29

Mok A (1983) Fundamental design problems of distributed systems for the hard-real-time environment. PhD thesis, Laboratory for Computer Science, Massachusetts Institute of Technology, available as Technical Report No. MIT/LCS/TR-297

Neuendorffer S (2005) The SDF Domain. In: Brooks C, Lee EA, Liu X, Neuendorffer S, Zhao Y, Zheng H (eds) Heterogeneous Concurrent Modeling and Design in Java, vol 3 (Ptolemy II Domains), EECS, University of California, Berkeley, chap 3, pp 49–60, memorandum UCB/ERL M05/23

Ripoll I, Crespo A, Mok AK (1996) Improvement in feasibility testing for real-time tasks. Real-Time Systems: The International Journal of Time-Critical Computing 11:19–39

Singh A, Ekberg P, Baruah S (2017) Applying real-time scheduling theory to the synchronous data flow model of computation. In: 2017 29th Euromicro Conference on Real-Time Systems, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik

Siyoum F (2014) Worst-case temporal analysis of real-time dynamic streaming applications. PhD thesis, PhD thesis, Eindhoven University of Technology

Stigge M, Ekberg P, Guan N, Yi W (2011) The digraph real-time task model. In: Proceedings of the IEEE Real-Time Technology and Applications Symposium (RTAS), IEEE Computer Society Press, Chicago, pp 71–80

Zhang F, Burns A (2009) Schedulability analysis for real-time systems with EDF scheduling. IEEE Transactions on Computers 58(9):1250–1258, DOI 10.1109/TC.2009.58