Applying Real-Time Scheduling Theory to the Synchronous Data Flow Model of Computation*

Abhishek Singh[1], Pontus Ekberg[2], and Sanjoy K. Baruah[3]

1 **The University of North Carolina, Chapel Hill, NC, USA**
abh@cs.unc.edu
2 **Uppsala University, Uppsala, Sweden**
pontus.ekberg@it.uu.se
3 **The University of North Carolina, Chapel Hill, NC, USA**
baruah@cs.unc.edu

Abstract

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 modeled using the synchronous data-flow graph (SDFG) model. An enhancement to the standard SDFG model is proposed, that permits the specification of a real-time latency constraint between a specified input and a specified output of an SDFG. A technique is derived for transforming such an enhanced SDFG to a collection of traditional 3-parameter sporadic tasks, thereby allowing for the analysis of systems of SDFG tasks using the methods and algorithms that have previously been developed within the real-time scheduling community for the analysis of systems of such sporadic tasks. 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 recurrent processes that are each represented using the enhanced SDFG model.

1998 ACM Subject Classification C.3 Real-Time and Embedded Systems, C.3 Signal Processing Systems

Keywords and phrases real-time systems, synchronous dataflow (SDF), hard real-time streaming dataflow applications, algorithms

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2017.8

1 Introduction

The research discussed in this document is inspired in part by problems that arise in designing base stations for wireless cellular communication systems. A base-station can handle a certain number of connections; for each such handled connection, streams of data packets arrive at the base-station and go through a number of stages of data-flow processing. (The precise nature of such processing depends upon the kind of connection, and may, therefore, be different for different connections.) A minimum duration is assumed between the arrival of successive data packets of the same connection at the base station, and the processing of the packet is required to complete within a specified duration of its arrival.

It is quite natural to model such processing requirements using *sporadic task models* of the kind that have been very widely studied in the real-time scheduling literature (see [26, 24] for

* This research is supported by NSF grants CNS 1115284, CNS 1218693, CNS 1409175, and CPS 1446631, AFOSR grant FA9550-14-1-0161, and ARO grant W911NF-14-1-0499. It was conducted while the second author was visiting the University of North Carolina.

a survey), in which the minimum inter-arrival duration of successive data packets is modeled by the 'period' parameter, and the duration allowed for the processing of each packet by the 'relative deadline' parameter, of sporadic tasks. However, the modeling of the actual processing of the packets, which, for telecom applications, are typically represented using the Synchronous Data Flow Graph (SDFG) [15] modeling abstraction[1], proves more challenging: there does not appear to be a straight-forward means of directly modeling such processing requirements using the concepts and terminology of real-time scheduling theory. Although the SDFG abstraction, which is widely used in the modeling, design, and analysis of real-time streaming applications in telecommunications and other domains, has been studied for decades, the run-time scheduling of computational workloads that are represented in the SDFG model has traditionally been done via static scheduling methods (e.g., [13, 14]; see [23] for a text-book description), in which scheduling tables are determined prior to run-time and these pre-computed tables are used for making run-time scheduling decisions. Techniques for constructing such tables have been developed from first principles, and these techniques display little commonality with the ones that are used in the real-time scheduling theory community. As telecommunications and other streaming applications become increasingly more computation-intensive and efficiency of implementation becomes an increasing concern, however, efforts are being made to apply the more efficient dynamic scheduling approaches, of the kind that is very familiar to the real-time scheduling community, to the scheduling of such systems represented using the SDFG model. Examples of such efforts include the following (this is not an exhaustive list):

1. Bouakaz et al. [5] apply EDF scheduling in order to implement multiple independent applications each specified in the SDFG model upon a shared (uniprocessor or partitioned multiprocessor) computing platform.
2. Bamakhrama and Stefanov [2, 3] 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.
3. Ali et al. [1] have developed techniques for transforming SDFG specifications of a different kind, called *cyclic homogeneous* SDFG, to collections of periodic tasks.
4. Mohaqeqi et al. [18] describe how to represent SDFG specifications in the *digraph* real-time (DRT) task model [25].

To our knowledge, these approaches are all proved correct but none claim optimality; indeed, it is fairly easy to construct example instances in which each such approach will result in implementations that make very inefficient use of platform computing resources. Other data-flow approaches that have been explored in the real-time systems community, such as the Processing Graph Method (PGM) [20] similarly suffer from an absence of optimality results. Some other recent research to have explored the relationship between SDF and real-time scheduling include [11, 12], which 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).

Motivation

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-

[1] The SDFG model is described in Section 2.

time workloads that are represented using the SDFG model. We believe that there are plenty of opportunities here: while real-time scheduling theory has made tremendous progress in recent years, this progress has, by and large, remained focused upon the workload models popular within the community. Meanwhile, data-flow models such as SDFG are finding increasing use in many embedded application domains, but research on these models have thus far primarily concentrated on ensuring correctness of design rather than enhancing efficiency of implementation. It is only now, 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 offers us in the real-time scheduling theory community an opportunity to demonstrate the usefulness and applicability of our research endeavors, simultaneously providing us with a rich source of interesting new problems that are of immediate interest outside the real-time scheduling theory community.

This research

In this paper, we report on our efforts at developing algorithms that allow us to transform, for the purposes of analysis, recurrent tasks that are specified using the SDFG model into a task model that is widely studied in the real-time scheduling theory literature: the 3-parameter sporadic task model [19, 4]. We do so by

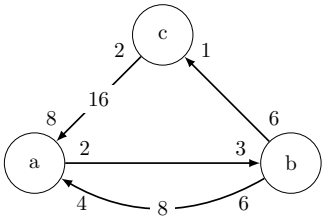
1. detailing extensions to the basic SDFG model that render it suitable for representing real-time requirements; and then
2. deriving an algorithm that allows us to transform any task that is represented using this SDFG model into a collection of 3-parameter sporadic tasks that have worst-case computational requirement (as represented using the *demand bound function abstraction* [4]) exactly equal to the worst-case computational requirement of the SDFG task.

In this manner, determining schedulability for a collection of independent SDFG tasks implemented upon a shared platform is reduced to the problem of determining schedulability of a collection of 3-parameter sporadic tasks implemented upon a shared platform – this is a problem that is well understood in real-time scheduling theory.[2] In particular, this means that we can schedule a collection of independent SDFG tasks *optimally* upon a preemptive uniprocessor platform, by exploiting the well-known result concerning the optimality of the earliest deadline first scheduling algorithm (EDF) upon preemptive uniprocessors [17, 7].

Organization

The remainder of this paper is organized as follows. In Section 2 we describe the SDFG model of computation and detail some enhancements to the basic model that make it better suited for representing real-time constraints. In Section 3 we briefly discuss the 3-parameter sporadic task model, and present without proof some results concerning analysis of systems represented using this model. In Section 4 we present, prove correct, and characterize the effectiveness of, our algorithm for transforming any SDFG task to a collection of equivalent 3-parameter sporadic tasks.

[2] As an added benefit, this transformation allows us to schedule collections of tasks, some of which are represented using the SDFG model and others, using traditional real-time task models, upon a shared platform and analyzed within a common framework.



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

Figure 1 An example SDFG. From left to right: the SDFG, its topology matrix Γ, and its repetition vector q. (Rows and columns of the topology matrix are labeled by channel and actor respectively, and rows of the repetition vector are labeled by actor.)

A note on the presentation. This paper is aimed at a readership that is familiar with the real-time scheduling literature, but not necessarily the literature on synchronous and other data-flow models. We have therefore chosen to provide a rather detailed explanation of the SDFG model, and have made certain simplifications by essentially ignoring aspects of the model that are orthogonal to our perspective of scheduling to provide real-time guarantees. In contrast, we have been terse with our discussion of the 3-parameter sporadic task model, assuming that the reader is already very familiar with this model.

2 A real-time SDF model

In this section we describe both the basic SDF model [13, 15], and several extensions that have been proposed to the model in order to enhance its capabilities to more accurately depict real-time considerations.

2.1 Synchronous Data Flow Graphs

We now provide a brief introduction to the synchronous data-flow graph model of computation; we refer the interested reader to [16, Ch 6.3.2] for a text-book description and additional references.

A dataflow graph is a directed graph[3] 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. In a *synchronous dataflow graph (SDFG)*, 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 $G = (V, E, \mathsf{prod}, \mathsf{cons}, \mathsf{delay})$ where

[3] 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 are 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.

- V denotes the set of actors.
- $E \subseteq (V \times V)$ is the set of channels. For each channel $e = (u, v)$, we denote u as $\mathsf{tail}(e)$ and v as $\mathsf{head}(e)$. We assume that $u \neq v$; i.e., there are no channels leading from an actor back to itself.
- $\mathsf{prod} : E \to \mathbb{N}_{>0}$. For each $e \in E$, $\mathsf{prod}(e)$ tokens are added to channel e each time the actor $\mathsf{tail}(e)$ fires.
- $\mathsf{cons} : E \to \mathbb{N}_{>0}$. For each $e \in E$, $\mathsf{cons}(e)$ tokens are removed from channel e each time the actor $\mathsf{head}(e)$ fires.
- $\mathsf{delay} : E \to \mathbb{N}_{\geq 0}$. For each $e \in E$, $\mathsf{delay}(e)$ 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 the number is > 0.

Some additional terminology:

- The channels leading into an actor v from other actors are called *input channels* of v and are collectively denoted as $\mathrm{In}(v)$. *Output channels* of v are defined similarly and denoted as $\mathrm{Out}(v)$.
- If each channel $e \in \mathrm{In}(v)$ contains at least $\mathsf{cons}(e)$ tokens, then actor v is said to be *enabled*.
- An enabled actor may *fire*, i.e., execute. The firing of actor v removes $\mathsf{cons}(e)$ tokens from each $e \in \mathrm{In}(v)$, and adds $\mathsf{prod}(e)$ tokens to each $e \in \mathrm{Out}(v)$.

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 [15] 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.

SDFG analysis techniques and algorithms have been developed [13, 15] for determining, for a given SDFG, whether sequences of firings of actors 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. We briefly summarize some of these results below.

▶ **Definition 2** (Topology Matrix)**.** For an SDFG $G = (V, E, \mathsf{prod}, \mathsf{cons}, \mathsf{delay})$, 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:

$$\Gamma(G)[i,j] \stackrel{\text{def}}{=} \begin{cases} \mathsf{prod}(e_i), & \text{if } \mathsf{tail}(e_i) = v_j \\ -\mathsf{cons}(e_i), & \text{if } \mathsf{head}(e_i) = v_j \\ 0, & \text{otherwise} \end{cases}$$

The topology matrix for the SDFG depicted in Figure 1 is shown in the figure. 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 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 [13, 15]:

1. There are methods to determine whether a given SDFG G is deadlock-free.
2. A connected[4] An SDFG G, with n actors, that is deadlock-free will not suffer from buffer overflow if and only if the rank[5] of its topology matrix $\Gamma(G)$ equals $(n-1)$.
   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 $\vec{v}$ such that

$$\Gamma(G) \cdot \vec{v} = \mathbf{0}. \tag{1}$$

   (Since the topology matrix $\Gamma(G)$ has n columns it is evident that for Equation 1 to be well-formed, any such $\vec{v}$ must comprise n components – recall that n denotes the number of actors.)

If we interpret the n components of $\vec{v}$ as number of firings of the n 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 positive integer vector $\vec{v}$ for which Equation 1 holds, and is denoted by $\mathrm{q}(G)$. An *iteration* is a set of actor firings with as many firings as the repetitions vector entry for each actor.

Hence the number of tokens in each channel remains unchanged upon completion of an iteration, during which each actor would fire as many times as indicated by its corresponding entry in the repetitions vector. This is formally stated in the following *balance equation* for each channel $e \in E$:

$$\mathsf{prod}(e) \cdot \mathrm{q}(G)[\mathsf{tail}(e)] = \mathsf{cons}(e) \cdot \mathrm{q}(G)[\mathsf{head}(e)] \tag{2}$$

In the remainder of the text when the SDFG G under consideration is evident, we will often ***simplify our notation and write*** Γ ***(***q***, respectively) for*** $\Gamma(G)$ ***(***$\mathrm{q}(G)$***, resp.).***

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:

$$\Gamma \cdot \mathrm{q} = \begin{pmatrix} 2 & -3 & 0 \\ -4 & 6 & 0 \\ 0 & 6 & -1 \\ -8 & 0 & 2 \end{pmatrix} \cdot \begin{pmatrix} 3 \\ 2 \\ 12 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

[4] An SDFG $G = (V, E, \mathsf{prod}, \mathsf{cons}, \mathsf{delay})$ is said to be *connected* if its set of actors and set of channels, when interpreted as the vertices and edges of a directed graph, represents a weakly connected digraph.

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

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.

We close this section out defining a restricted kind of SDFG.

▶ **Definition 4** (Homogeneous SDFG (HSDFG))**.** An *homogeneous* SDFG is one in which all the prod and cons rates are equal to one.

For such homogeneous SDFG's, 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: $\mathrm{q}(a) = 1$ for all $a \in V$.

Bounding buffer sizes

A significant amount of the SDFG research literature deals with the problem of minimizing the size of the channel buffers; i.e., minimizing the maximum number of tokens that need to be stored in individual channels. While this is an important dimension to SDFGs, we have chosen to ignore this aspect of the problem in the current paper, leaving its consideration as future work. (We have considered several ad hoc approaches for dealing with buffer capacity constraints on channels by, e.g., associating a deadline (temporal) with the actor at the head of the channel, but these ad hoc solutions have not yet been proved optimal.)

2.2 Incorporating real time

We now discuss extensions that have been made to the basic SDFG model described above over the years in order to incorporate real-time considerations.

Actor execution times

The SDFG model, as described in Section 2.1 above, does not incorporate consideration of real time; it was subsequently extended [10] to additionally associate an execution time with each actor. That is, along with the parameters V, E, prod, cons, and delay as defined in Definition 1, we specify an additional parameter, a *worst-case execution time* function

$$\mathsf{wcet} : V \to \mathbb{N}_{\geq 0}$$

with the interpretation that for each $v \in V$, $\mathsf{wcet}(v)$ is the worst-case execution time of a (single) firing of actor v.

Input-output Latency

As we had stated earlier, the SDFG model as originally proposed did not explicitly model the interaction of the SDFG with the external world: 'Connections to the outside world are not considered' [15]. But our interest is expressly in this interaction: as stated in the introduction, the kinds of applications we are analyzing typically require that the processing of a packet complete within a specified (real-time) duration of its arrival. The SDFG model may be extended as follows in order to incorporate such real-time considerations. For each SDFG, we additionally specify

- a single *input actor* v_{in} and a single *output actor* v_{out}.[6] External tokens are assumed to arrive at the input actor v_{in}. That is, we can imagine an additional channel e_{in}

[6] The rationale behind the decision to restrict the number of input and output actors to being one is explained in Section 2.3.

with $\mathsf{head}(e_{\text{in}}) = v_{\text{in}}$ and $\mathsf{tail}(e_{\text{in}})$ not specified, but rather representing the external environment within which the SDFG is operating. The consume rate $\mathsf{cons}(e_{\text{in}})$ is one, while no produce rate $\mathsf{prod}(e_{\text{in}})$ is specified; instead, tokens 'appear' on channel e_{in} according to the period parameter (discussed next).

- a *period* parameter, denoting the minimum duration between successive arrivals of external tokens on the input channel e_{in}, and
- 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 5).

In the example SDFG of Figure 1, we could, for instance, specify that actor a is the input actor, actor b the output actor, the period is 5 time units and the relative deadline is 2 time units. This would mean that

- Actor a may only fire when there are at least 8 tokens on its input channel (c, a) and at least 4 tokens on its input channel (b, a), *and* at least one external input token; its firing consumes 8 tokens from (c, a), 4 tokens from (b, a), and one external input token.
- The duration between the arrival of successive external input tokens at a is no smaller than 5 time units.
- The maximum duration that may elapse between the arrival of the external input token at a and the completion of the firing of a corresponding execution of b is 2 time units.

Let us now formalize this notion of correspondence.

Suppose that the first external input token arrives at actor a at some time instant, thereby causing actor a to fire. Observe that since the production rate $\mathsf{prod}(a, b)$ of the channel leading from a to b is two while the consumption rate $\mathsf{cons}(a, b)$ is three, at least two firings of actor a must occur before actor b may fire for the first time. But since the period of the SDFG denotes only a *lower* bound on duration between the arrival of successive external input tokens, we cannot provide an upper bound upon the time instant at which actor b is enabled – this depends upon when the second external input token arrives at actor a. 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.

This conundrum was resolved by Ghamarian et al. [10] based on the reasoning that an entire iteration (see Definition 3) of an SDFG should be thought of as representing a single logical chunk of computation. Therefore 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 an external input token that initiates the first firing of the input actor within an iteration, and the completion of the execution of the last firing of the output actor during that iteration. We do so by preprocessing an SDFG to make the following changes (Figure 2 illustrates the result of applying these changes to the example of Figure 1):

- Add two new actors src and dst, with $\mathsf{wcet}(\mathsf{src}) = \mathsf{wcet}(\mathsf{dst}) = 0$, and ensure (see below) that each will execute exactly once per iteration. These now become the designated input and output actors, while 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
  - $\mathsf{delay}(e_1) = \mathsf{delay}(e_2) = 0$,
  - $\mathsf{prod}(e_1) = \mathrm{q}(v_{\text{in}})$, $\mathsf{cons}(e_1) = 1$, (recall that each actor a executes $\mathrm{q}(a)$ times per iteration), and
  - $\mathsf{prod}(e_2) = 1$, $\mathsf{cons}(e_2) = \mathrm{q}(v_{\text{out}})$.

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

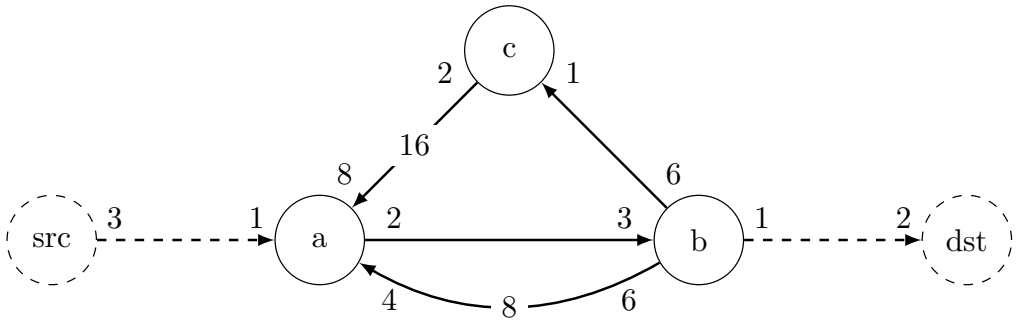


Figure 2 Applying the transformations of Section 2.2 to the example SDFG shown in Figure 1. Suppose that actors a and b are identified as the input and output actors of the example SDFG of Figure 1. The (dashed) actors and channels designated src and dst are added, and become the designated input and output actors. Recall from Figure 1 that $\mathrm{q}(a) = 3$ and $\mathrm{q}(b) = 2$; 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 characterized with a relative deadline and a period parameter, both being positive integers.

After the transformation, the arrival of $\mathrm{q}(v_{\mathrm{in}})$ external input tokens at actor v_{in} in the original SDFG is modeled as the arrival of one external input token at src.[7] 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 $\mathrm{q}(v_{\mathrm{in}})$; the relative deadline parameter may also need to be modified suitably. (Indeed, the interpretation of the relative deadline parameter is ambiguous when 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. We will also assume that each actor is 'reachable' via channels from src, and that dst 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 removed from consideration during pre-processing to be executed in the background during run-time.)

An additional point of interest arises from the tokens that populate each channel initially, before the first arrival of an external input token at src. There are $\mathsf{delay}(e)$ such tokens on each $e \in E$; since each $\mathsf{delay}(e)$ 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 src firing for the first time. The *dependency distance* denotes the maximum number of times dst can be fired before exhausting the initially-supplied tokens:

[7] 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 $\mathrm{q}(v_{\mathrm{in}})$ 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.) An analogous interpretation may be made for dst.

▶ **Definition 5** (Dependency Distance δ [22]; Correspondence)**.** Due to the initial distribution of tokens on the channels specified by delay, dst can fire some δ times before src fires for the first time. The number δ is called the dependency distance.

For any $k \in \mathbb{N}$, the k-th firing of src is said to *correspond to* the $(k+\delta)$-th firing of dst, 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 2, $\mathsf{delay}(a,b)$ were equal to 10 rather than zero (i.e, 10 tokens were initially provided in this channel). Since $\mathsf{cons}(a,b) = 3$, 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 $k \in \mathbb{N}$ 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

$$\mathsf{delay}(a,b) = 1; \quad \mathsf{delay}(b,\mathsf{dst}) = 1; \quad \mathsf{delay}(b,a) = 26;$$
$$\mathsf{delay}(b,c) = 0; \quad \mathsf{delay}(c,a) = 52; \quad \mathsf{delay}(\mathsf{src},a) = 0.$$

In order to keep things simple, in 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 $k \in \mathbb{N}$.

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:

$$G \stackrel{\text{def}}{=} \Big\langle (V,\, E,\, \mathsf{prod},\, \mathsf{cons},\, \mathsf{delay}), \mathsf{wcet}, \mathsf{src}, \mathsf{dst}, D, T \Big\rangle \tag{3}$$

with

- V, E, prod, cons, and delay as specified for traditional SDFGs;
- $\mathsf{wcet} : V \to \mathbb{N}_{\geq 0}$ specifying the worst-case execution times of the actors;
- Actors $\mathsf{src} \in V$ and $\mathsf{dst} \in V$ being specified as the unique input and output actor, respectively; and
- $D \in \mathbb{N}$ and $T \in \mathbb{N}$ 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: $\mathrm{q}(\mathsf{src}) = \mathrm{q}(\mathsf{dst}) = 1$.

We now briefly discuss the rationale behind some of the design decisions 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 minimum inter-arrival duration, but no maximum inter-arrival duration, specified. 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 $\mathsf{wcet} = 0$) 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** ($\mathrm{q}(\mathsf{src}) = \mathrm{q}(\mathsf{dst}) = 1$). This was discussed above, when introducing the transformation of adding the single source actor src: 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 src that is added, and guaranteed to have $\mathrm{q}(\mathsf{src}) = 1$.

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: $\mathrm{q}(v) = 1$ 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** src**, and** dst **is reachable from each actor.** It is easily seen that any actor that is either not reachable from src, or from which dst is not reachable, need not fire at all in order to ensure that dst fires in response to a firing of src. Hence, we need not execute such actors during run-time to ensure real-time correctness, and 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 run-time computational workload. We, therefore, assume this in the remainder of this paper. However, we point out that this is not necessary – our algorithms are easily extended to deal with the case where such pre-processing is not done for whatever reason, and $\delta > 0$.

3 The three-parameter sporadic task model

We now provide a very brief introduction to the 3-parameter sporadic task model [19], which is widely used in real-time scheduling theory. (This introduction is primarily intended to introduce terminology and notation; we assume that the reader is already very familiar with this model.)

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* [4]. For any sporadic task τ_i and any real number $t > 0$, the demand bound function $\mathsf{dbf}(\tau_i, t)$ 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 $t_o + T_i$, $t_o + 2T_i$, $t_o + 3T_i, \ldots$ (this fact is formally proved in [4]). We, therefore, have [4]:

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

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

$$\mathsf{load}(\tau) \stackrel{\text{def}}{=} \max_{t>0}\left(\frac{\sum_{\tau_i \in \tau} \mathsf{dbf}(\tau_i, t)}{t}\right) .$$

It has been shown [4] 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 $\mathsf{load}(\tau) \leq 1$. Pseudo-polynomial algorithms are known [4, 21, 28] for computing $\mathsf{load}(\tau)$, 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 $\mathsf{load}(\tau)$ in polynomial time, to any desired degree of accuracy [9].

4 Optimal uniprocessor scheduling of sporadic real-time SDFGs

In this section, we will develop an optimal algorithm, and an associated exact schedulability test, for scheduling a collection of independent sporadic real-time SDFGs upon a preemptive uniprocessor. The algorithm is *optimal* in the following sense: if any run-time scheduling algorithm can guarantee to schedule the collection to always meet all deadlines for all permissible arrival sequences of external input tokens, then our algorithm also guarantees to always meet all deadlines for all permissible arrival sequences of external input tokens.

Our run-time scheduling algorithm is EDF-based: individual firings of actors are assigned deadlines, and at each instant in time, the earliest-deadline enabled actor firing that has not

yet completed execution is executed. The manner in which deadlines are assigned to firings of actors is described later.

We start with a high-level overview of our schedulability test. As with 3-parameter sporadic tasks (Section 3), we will characterize the execution requirement of a sporadic real-time SDFG by a demand bound function (dbf): for any sporadic real-time SDFG G (characterized as in Expression 3) and any positive real number t, let $\mathsf{dbf}(G,t)$ 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:

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

(In the remainder of the text when the SDFG G under consideration is evident, we will ***simplify our notation and write*** $k(t)$ ***for*** $k(G,t)$.)

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 src for which the corresponding firings of 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 src triggers one iteration (see Definition 3) of G, an upper bound for $\mathsf{dbf}(G,t)$ may be obtained by simply assuming that each actor a fires a total of $\mathrm{q}[a]$ times during each such iteration, thereby obtaining a bound of

$$k(t) \times \sum_{a \in V} \big(\mathrm{q}[a]\mathsf{wcet}(a)\big) . \tag{5}$$

However this bound, while safe, is not necessarily tight – the presence of initial tokens on some of the channels (as represented by the $\mathsf{delay}(c)$ values) means that not all firings of all actors need take place. Consider, for instance, the example of Figure 2 and consider a value of t_o satisfying $D < t_o < T + D$, so that $k(t_o)$ evaluates to 1 by Equation 4. Even though we had previously computed (see Figure 1) that $\mathrm{q}[c] = 12$, the reader may verify that firing actor c just four times suffices to ensure that dst is able to fire. Hence over such a t_o, $\mathsf{dbf}(G,t_o)$ equals

$$\Big(3\,\mathsf{wcet}(a) + 2\,\mathsf{wcet}(b) + \mathbf{4}\,\mathsf{wcet}(c)\Big), \text{ rather than } \Big(3\,\mathsf{wcet}(a) + 2\,\mathsf{wcet}(b) + \mathbf{12}\,\mathsf{wcet}(c)\Big)$$

as suggested by the upper bound in Expression 5 above.

In the remainder of this section, we will describe the computation of a *skip vector* $\mathrm{s}(G)$ 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.[8] That is, we will show that for each actor a the computed skip-vector value $\mathrm{s}(G)[a]$ is the largest integer possessing the property that actor a will need to complete no more than

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

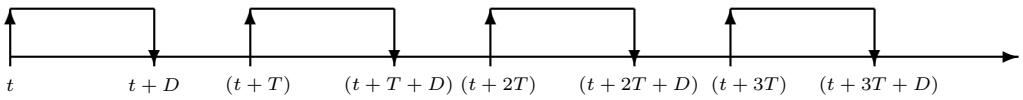
firings over any contiguous interval of duration t.

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

(As we did with the topology matrix Γ and the repetitions vector q, when the SDFG G under consideration is evident we will often ***simplify our notation and write*** s ***for*** s(G).)

Before deriving formulae for computing the skip vector, let us briefly illustrate how the skip-vector values, once computed, will be used during run-time for assigning deadlines to the firings of actors. Suppose that we have determined, for a particular actor a in given sporadic real-time SDFG G, that s[a] = 10 and q[a] = 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 src at some time instant t, we would schedule two firings of actor a to complete by a deadline $t + 3T + D$, and a further firing of actor a to complete by a deadline $t + 4T + D$. (This is equivalent, for the purposes of developing a uniprocessor EDF schedulability test, to representing actor a's computational requirements by two 3-parameter sporadic tasks: one with parameters $(2\,\mathsf{wcet}(a), 3T + D, T)$, and one with parameters $(\mathsf{wcet}(a), 4T + D, T)$.)

Let us now seek to understand the rationale behind this strategy. If we were to not schedule any firings of actor a in response to the arrival of the external input token at time instant t, we would have 'used up' three of the ten skips that are permitted; over four iterations, all the skips would thus be completely used up, and future iterations would need to complete three firings by their deadlines. Recall that our objective is to minimize the run-time 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 a associated with the arrival of the external input token at 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. As shown in the following figure, we know that the next three external input tokens cannot arrive before time-instants $t + T$, $t + 2T$, and $t + 3T$:



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 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 the figure above shows, 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 a to complete by a deadline $t + 3T + D$, and a further firing of actor a to complete by a deadline $t + 4T + D$.

Generalizing the example above from q[a] ← 3 and s[a] ← 10 to arbitrary values for q[a] and s[a], it is straightforward to show that in response to an external input token's arrival at time-instant t, we would schedule each actor a to have

$$\big(\mathrm{q}[a] - (\mathrm{s}[a] \bmod \mathrm{q}[a])\big) \text{ firings with a deadline at } \Big(\big\lfloor \mathrm{s}[a]/\mathrm{q}[a] \big\rfloor \cdot T + D\Big)$$

and the remaining

$$\big(\mathrm{s}[a] \bmod \mathrm{q}[a]\big) \text{ firings with a deadline at } \Big(\big(\big\lfloor \mathrm{s}[a]/\mathrm{q}[a] \big\rfloor + 1\big) \cdot T + D\Big).$$

4.1 Computing the skip vector

Our intent is that the skip vector value $\mathrm{s}[a]$ denote the maximum number of times the execution of actor a may be skipped, due to the presence of initial tokens on the edges. Let us now consider any channel $e \in E$ of the sporadic real-time SDFG under consideration, and let $u = \mathsf{tail}(e)$, $v = \mathsf{head}(e)$. Let n_u and n_v denote the number of times that actors u and v have fired by some point in time; it must be the case that

$$n_u \cdot \mathsf{prod}(e) + \mathsf{delay}(e) \geq n_v \cdot \mathsf{cons}(e). \tag{6}$$

Let us instantiate Equation 6 above to the end of the k'th iteration of the sporadic real-time SDFG under consideration. At that point in time, $n_u \leftarrow (k \cdot \mathrm{q}[u] - \mathrm{s}[u])$ and $n_v \leftarrow (k \cdot \mathrm{q}[v] - \mathrm{s}[v])$; hence we have

$$\begin{aligned}
& \Big(k \cdot \mathrm{q}[u] - \mathrm{s}[u]\Big) \cdot \mathsf{prod}(e) + \mathsf{delay}(e) && \geq && \Big(k \cdot \mathrm{q}[v] - \mathrm{s}[v]\Big) \cdot \mathsf{cons}(e) \\
\Leftrightarrow\ & k \cdot \mathrm{q}[u] \cdot \mathsf{prod}(e) - \mathrm{s}[u] \cdot \mathsf{prod}(e) + \mathsf{delay}(e) && \geq && k \cdot \mathrm{q}[v] \cdot \mathsf{cons}(e) - \mathrm{s}[v] \cdot \mathsf{cons}(e) \\
\Leftrightarrow\ & k \cdot \underbrace{\Big(\mathrm{q}[u] \cdot \mathsf{prod}(e) - \mathrm{q}[v] \cdot \mathsf{cons}(e)\Big)}_{=0 \text{ by the balance equation (Eqn. 2)}} + \mathsf{delay}(e) && \geq && \mathrm{s}[u] \cdot \mathsf{prod}(e) - \mathrm{s}[v] \cdot \mathsf{cons}(e) \\
\Leftrightarrow\ & \mathsf{delay}(e) && \geq && \mathrm{s}[u] \cdot \mathsf{prod}(e) - \mathrm{s}[v] \cdot \mathsf{cons}(e)
\end{aligned}$$

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

$$\mathrm{s}[\mathsf{tail}(e)] \cdot \mathsf{prod}(e) - \mathrm{s}[\mathsf{head}(e)] \cdot \mathsf{cons}(e) \leq \mathsf{delay}(e)$$

$$\Leftrightarrow \quad \mathrm{s}[\mathsf{tail}(e)] \leq \left\lfloor \frac{\mathsf{delay}(e) + \mathrm{s}[\mathsf{head}(e)] \cdot \mathsf{cons}(e)}{\mathsf{prod}(e)} \right\rfloor \tag{7}$$

to help us compute the skip vector: our objective is to determine the largest values for $\mathrm{s}[a]$ for all actors a, such that Equation 7 is satisfied across all channels of the SDFG. Before doing so, we prove in Lemma 6 below, that there cannot be multiple incomparable skip vectors for the same SDFG.

▶ **Lemma 6.** *The skip vector* s *is unique.*

Proof. We prove the lemma by contradiction. Assume for this purpose that there exists two different skip vectors s and s′ for which Equation 7 holds for all channels $e \in E$. Assume also that both s and s′ are maximal, so that Equation 7 would not hold for either of them if we increased some values of s or s′.

Now let s″ be defined so that $\mathrm{s}''[v] = \max(\mathrm{s}[v], \mathrm{s}'[v])$ for all $v \in V$. For any edge $e \in E$ we have

$$\begin{aligned}
\mathrm{s}''[\mathsf{tail}(e)] &= \max(\mathrm{s}[\mathsf{tail}(e)], \mathrm{s}'[\mathsf{tail}(e)]) \\
&\leq \max\left(\left\lfloor \frac{\mathsf{delay}(e) + \mathrm{s}[\mathsf{head}(e)] \cdot \mathsf{cons}(e)}{\mathsf{prod}(e)} \right\rfloor, \left\lfloor \frac{\mathsf{delay}(e) + \mathrm{s}'[\mathsf{head}(e)] \cdot \mathsf{cons}(e)}{\mathsf{prod}(e)} \right\rfloor\right) \\
&= \left\lfloor \frac{\mathsf{delay}(e) + \mathrm{s}''[\mathsf{head}(e)] \cdot \mathsf{cons}(e)}{\mathsf{prod}(e)} \right\rfloor,
\end{aligned}$$

but then Equation 7 holds for all edges $e \in E$ also when using skip vector s″. It follows that s and s′ can not both be maximal. ◀

We now derive our algorithm for determining this skip vector. We start defining some additional terminology and notation. For each actor a, let $\check{s}[a]$ denote an upper bound on the value of $s[a]$; 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:

$$\check{s}[a] \leftarrow \begin{cases} 0, & \text{if } a = \mathsf{dst} \\ \infty, & \text{otherwise} \end{cases} \tag{8}$$

It is evident that these initial values on $\check{s}$ 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, and recall from Section 2.3 that our model assumes that that the dependency distance between the input and output actors equal zero ($\delta = 0$).[9]

Relaxing (along) a channel

For a given assignment of $\check{s}$ values to all the actors, the process of *relaxing* a channel consists of identifying a channel e for which the current skip estimates violate Condition 7:

$$\check{s}[\mathsf{tail}(e)] > \left\lfloor \frac{\mathsf{delay}(e) + \check{s}[\mathsf{head}(e)] \cdot \mathsf{cons}(e)}{\mathsf{prod}(e)} \right\rfloor$$

and updating (by decreasing) the skip estimate of $\mathsf{tail}(e)$ in order to cause it to satisfy Condition 7:

$$\check{s}[\mathsf{tail}(e)] \leftarrow \left\lfloor \frac{\mathsf{delay}(e) + \check{s}[\mathsf{head}(e)] \cdot \mathsf{cons}(e)}{\mathsf{prod}(e)} \right\rfloor \tag{9}$$

If no channel can be relaxed, then the current assignment of values to $\check{s}$ is the desired skip vector. Our algorithm for computing the skip vector can thus be stated as follows:

> ***Procedure*** Compute Skip-Vectors. *Repeatedly relax channels until no further channel relaxations are possible.*

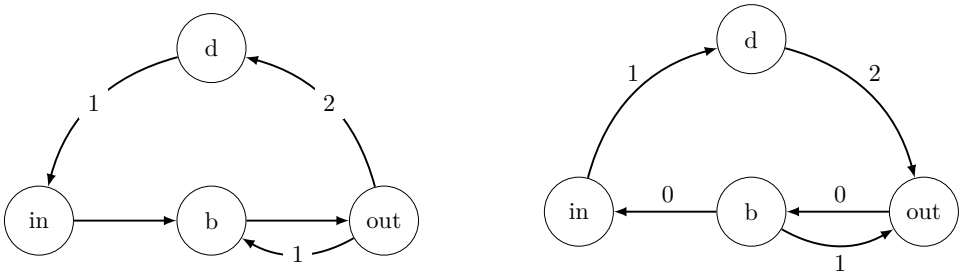
We elaborate upon the details of this algorithm first for the case of homogeneous SDFGs in Section 4.1.1 below; the case of general (i.e., not necessarily homogeneous) SDFGs is then considered in Section 4.1.2.

4.1.1 Homogeneous SDFGs

As stated in Definition 4, all produce and consume rates for homogeneous SDFGs are equal to one; for homogeneous SDFGs, Expression 9 above can therefore be simplified in the following manner:

$$\begin{aligned} &\check{s}[\mathsf{tail}(e)] \leftarrow \left\lfloor \frac{\mathsf{delay}(e) + \check{s}[\mathsf{head}(e)] \cdot \mathsf{cons}(e)}{\mathsf{prod}(e)} \right\rfloor \\ &\Leftrightarrow \check{s}[\mathsf{tail}(e)] \leftarrow \left\lfloor \frac{\mathsf{delay}(e) + \check{s}[\mathsf{head}(e)] \cdot 1}{1} \right\rfloor \\ &\Leftrightarrow \check{s}[\mathsf{tail}(e)] \leftarrow \Big(\mathsf{delay}(e) + \check{s}[\mathsf{head}(e)]\Big) \end{aligned} \tag{10}$$

[9] 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 is easily extended to handle non-zero dependency distances: we omit details here.



(a) Example homogeneous SDFG

(b) Shortest-path graph

Figure 3 Illustrating the computation of skip vectors for homogeneous SDFGs.

Observe that across any channel e, Condition 10 specifies that $\check{s}[\mathsf{tail}(e)]$ be no larger than $\check{s}[\mathsf{head}(e)]$ plus $\mathsf{delay}(e)$. This is similar to the constraints on single-source shortest-path problems on directed graphs (see, e.g., [6, Chapter 24] for a textbook discussion): if (u, v) is an edge of cost w in a directed graph, the shortest path from some designated vertex to v is of length no greater than the shortest path to u plus the cost w of the edge (u, v). This observation motivates our transformation of the skip vector computation problem to a single-source shortest path problem.

We will describe an algorithm for transforming the problem of assigning $\check{s}[v]$ values for all actors $v \in V$ in such a manner that no further channels relaxations are possible, to a single-source shortest paths problem in graphs. Let us attempt to obtain some intuition by working through the simple example HSDFG depicted in Figure 3 a. (In an HSDFG, since all repetition vector entries are always equal to one, the correspondence between the firing of input and output actors is well-defined and hence it is not necessary to add additional src and dst vertices.)

Let us consider the computation of the skip vector on the HSDFG of Figure 3 (a).

1. Upon initialization, the $\check{s}$ values are as follows:

$$\check{s}[\text{in}] = \check{s}[\text{b}] = \check{s}[\text{d}] = \infty;\ \check{s}[\text{out}] = 0$$

2. It is evident from visual inspection of Figure 3 (a) that the only channels that can be relaxed immediately after initialization are those for which the actor out is the head, i.e., the channel (b, out). Since $\mathsf{delay}(\text{b}, \text{out}) = 0$, such relaxation, which consists of applying Expression 10 to the channel $e = (\text{b}, \text{out})$, results in $\check{s}[\text{b}] \leftarrow 0$.
3. As a consequence, channels for which the actor b is the head potentially become relaxable. There are two such channels: (in, b) and (out, b). Relaxing the channel (in, b) updates $\check{s}[\text{in}]$ to 0, but it may be verified that channel (out, b) already satisfies Condition 7 and is hence not relaxable.
4. The update to $\check{s}[\text{in}]$ (in the step above) renders channels for which the actor in is the head potentially relaxable. This is the channel (d, in); since $\mathsf{delay}(\text{d}, \text{in}) = 1$, relaxing this channel updates $\check{s}[\text{d}]$ according to Expression 10 to a value of 1.
5. The update to $\check{s}[\text{d}]$ renders channels for which the actor d is the head potentially relaxable. This is the channel (out, d); it may be verified that this channel already satisfies Condition 7 and is hence not relaxable.

No further channels are relaxable, and hence the algorithm terminates with the following $\check{s}$ values:

$$\check{s}[\text{in}] = \check{s}[\text{out}] = \check{s}[\text{b}] = 0;\ \check{s}[\text{d}] = 1$$

(The reader may verify that these values are indeed correct, by observing that the presence of one initial token on the channel from actor d to actor in permits us to skip one firing of actor d, and that no further skips are possible.)[10]

We now describe the algorithm for transforming a homogeneous SDFG into a directed graph, such that solving a single-source shortest paths problem upon this graph will compute the skip vector for all the actors in that HSDFG.

1. The graph is constructed to have one vertex corresponding to each actor in the homogeneous SDFG.
   The graph for our example is depicted in Figure 3 (b); since the homogeneous SDFG of Figure 3 (a) has four actors this graph has four vertices, each labeled with the name of its corresponding actor.
2. For each channel c with $\mathsf{tail}(c) = u$ and $\mathsf{head}(c) = v$; we add an edge from the vertex corresponding to actor v to the vertex corresponding to actor u, and assign this edge a cost equal to $\mathsf{delay}(c)$.
   Observe that the edges of the graph depicted in Figure 3 (b) are *reversed* from the channel they correspond to, and that each is assigned a cost equal to the number of initial tokens (delays) on the channel.
3. As a consequence of the structural similarity of Condition 10 to shortest-path constraints, it follows by a direct application of shortest-paths arguments (see, e.g., [6, Sec 24.4]), that the skip vector value for each actor is equal to the shortest path in the graph to the vertex corresponding to it, from the vertex corresponding to the output vertex (out).
   In Figure 3 (b), it is evident that the shortest paths from the vertex labeled out to the vertices labelled in, out, b, and d are 0, 0, 0, and 1 respectively. Therefore, we conclude that the skip vector values are as follows: s[in] = 0; s[out] = 0; s[b] = 0; and s[d] = 1.

Observe, additionally, that since none of the $\mathsf{delay}(e)$ values are negative, the shortest-paths problem is easily solved using Dijkstra's shortest-path algorithm [8], for which implementations are known that have $O(|V| \log |V| + |E|)$ running time.

4.1.2 General SDFGs

We now turn our attention to general SDFGs; for such SDFGs, produce and consume rates are allowed to be arbitrary non-negative integers, and we cannot, therefore, simplify Expression 9 to a more tractable form (as we did for HSDFGs, in Expression 10). Hence, procedure Compute Skip-Vectors, which had been defined earlier as

> ***Procedure*** Compute Skip-Vectors. *Repeatedly relax channels until no further channel relaxations are possible.*

is run through in its entirety. In this section, we informally argue that this procedure concludes upon performing no more than exponentially many relaxations; since each relaxation takes constant time, this immediately yields an exponential-time upper bound on the running time of procedure Compute Skip-Vectors.

[10] This example also illustrates the advantage of the *pre-processing* we advocate, of maximally firing all actors prior to the first arrival of an external input token. In the example of Figure 3 (a), such pre-processing would fire actor d twice, yielding three tokens on channel (d, in) (and removing the two tokens from channel (out, d). Repeated relaxations on the resulting configuration would increase š[d] to 3, while keeping the other values unchanged. I.e., a further decrease in the run-time computational requirement is possible.

As we saw in our HSDFG example, initially the only channels that can be relaxed are those for which the dst actor is the head: the š values assigned to these actors are updated from ∞ to a value that is polynomially bounded in the values of the prod, cons, and delay parameters of the SDFG. Each such actor with a non-∞ š value, in turn, causes the š values of other actors to be reduced from ∞ to a value that is polynomially bounded in its value and the values of the prod, cons, and delay parameters. We state without proof the following facts:

1. For each actor u that is not dst, the first relaxation that changes $\check{s}[u]$ from ∞ assigns it a value that is polynomially bounded in the values of the prod, cons, and delay parameters of the SDFG, and the current š values of actors that were previously assigned values other than ∞; and
2. Each relaxation decreases $\check{s}[u]$ for some actor u by at least one.

Since the composition of polynomially many polynomial functions is an exponential, the first fact above implies that the maximum value that each $\check{s}[u]$ may have, other than ∞, is exponentially bounded. Hence a total of $(|V| - 1)$ relaxations – one per actor except for dst – reduces all the š values to be exponentially bounded. Henceforth, each relaxation reduces one of the š values by at least one. Since the sum of $(|V| - 1)$ values each of which is exponentially bounded is also exponentially bounded, it follows that the total number of relaxations is exponentially bounded in the values of the parameters characterizing the SDFG.

Experimental evaluation

Above, we are only able to provide an exponential-time upper bound on the running time of the algorithm for determining the skip vector for a general (i.e., not necessarily homogeneous) SDFG. We have implemented and tested our algorithm on randomly-generated SDFGs; these experiments indicate that the convergence occurs rather more rapidly than is implied by the exponential bound. Specifically, we used the SDFG random graph generator that is provided[11] as part of the SDF3 [27] tool-suite to generate 1000 deadlock-free and consistent, weakly-connected, cyclic SDFGs. This tool allows for rates and degrees of actors to be specified using minimum and maximum bounds, average value, and variance. The probability that initial tokens are added to a channel and the sum of the repetition vector can also be specified. We generated all these specifications randomly from uniform distributions; in all 1000 cases, convergence occurred upon performing no more than $|V| \times |E|$ relaxations.

We close this section with an example illustrating, at a high level, the execution of procedure Compute Skip-Vectors, upon the example SDFG of Figure 2.

1. Upon initialization, the skip estimate values are

$$\Big\langle \check{s}[\mathsf{src}] = \infty,\ \check{s}[a] = \infty,\ \check{s}[b] = \infty,\ \check{s}[c] = \infty,\ \check{s}[\mathsf{dst}] = 0 \Big\rangle.$$

2. The only channel that can be relaxed now is $(\mathrm{b}, \mathsf{dst})$; the skip estimate values are updated to

$$\Big\langle \check{s}[\mathsf{src}] = \infty,\ \check{s}[a] = \infty,\ \check{s}[b] = 0,\ \check{s}[c] = \infty,\ \check{s}[\mathsf{dst}] = 0 \Big\rangle.$$

[11] Available for download at `www.es.ele.tue.nl/sdf3/` (accessed January 2017)

3. Let us suppose that the channel (a, b) is relaxed next. The skip estimate values are updated to
$$\Big\langle \check{\mathrm{s}}[\mathsf{src}] = \infty,\ \check{\mathrm{s}}[a] = 0,\ \check{\mathrm{s}}[b] = 0,\ \check{\mathrm{s}}[c] = \infty,\ \check{\mathrm{s}}[\mathsf{dst}] = 0 \Big\rangle.$$
4. Let us suppose that the channel $(\mathsf{src}, \mathrm{a})$ is relaxed next. The skip estimate values are updated to
$$\Big\langle \check{\mathrm{s}}[\mathsf{src}] = 0,\ \check{\mathrm{s}}[a] = 0,\ \check{\mathrm{s}}[b] = 0,\ \check{\mathrm{s}}[c] = \infty,\ \check{\mathrm{s}}[\mathsf{dst}] = 0 \Big\rangle.$$
5. Let us suppose that the channel (c, a) is relaxed next. From Expression 9 we have
$$\check{\mathrm{s}}[c] \leftarrow \left\lfloor \frac{\mathsf{delay}(\mathrm{c}, \mathrm{a}) + \check{\mathrm{s}}[a] \cdot \mathsf{cons}(\mathrm{c}, \mathrm{a})}{\mathsf{prod}(\mathrm{c}, \mathrm{a})} \right\rfloor = \left\lfloor \frac{16 + 0 \cdot 8}{2} \right\rfloor = 8.$$
Hence the skip estimate values are updated to
$$\Big\langle \check{\mathrm{s}}[\mathsf{src}] = 0,\ \check{\mathrm{s}}[a] = 0,\ \check{\mathrm{s}}[b] = 0,\ \check{\mathrm{s}}[c] = 8,\ \check{\mathrm{s}}[\mathsf{dst}] = 0 \Big\rangle.$$

It may be verified that no further relaxations are possible: procedure Compute Skip-Vectors has converged after just four relaxations. The final values for the skip vector that are computed by procedure Compute Skip-Vectors, are therefore the estimates given above. (Observe that this matches with what we had informally argued at the beginning of Section 4, when we had reasoned that although $\mathrm{q}[c] = 12$, it suffices to fire actor c four times (i.e., skip $(12 - 4) = 8$ firings.)

5 Conclusions

The Synchronous Data Flow Graph (SDFG) model is widely used in the modeling of embedded real-time systems. In this research, we have attempted to apply ideas, techniques, and results from real-time scheduling theory to the analysis of systems represented using this model. We have developed what is, to our knowledge, the first optimal algorithm for dynamically scheduling a collection of such tasks upon a preemptive uniprocessor platform. Our algorithm achieves optimality by exploiting the presence of initial tokens to 'skip' (actually, delay) the executions of some actors. Significant improvement in performance over prior approaches depends upon the presence of a relatively large number of initial tokens in the SDFG under consideration; while this may be the case for only a limited class of systems, we hope that the theoretical insights provided by our optimal algorithm will lead to additional results that may be applicable to a wider variety of systems.

In addition to this particular result, we believe that a major contribution of this paper lies in its opening up a plethora of problems concerning real-time data-flow models to scrutiny by the real-time scheduling theory community. There are many aspects of the SDF model that are of interest to the SDF community that we have chosen to ignore in this paper, that merit further attention – of particular note are consideration of bounds on channel buffer sizes, and extension to multiprocessor platforms. We are optimistic that some of these aspects will prove amenable to analysis using recently-developed techniques of real-time scheduling theory.

References

1 H. I. Ali, B. Akesson, and L. M. Pinho. Generalized extraction of real-time parameters for homogeneous synchronous dataflow graphs. In *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 701–710, March 2015. doi:10.1109/PDP.2015.57.

2 M. Bamakhrama and T. Stefanov. Hard-real-time scheduling of data-dependent tasks in embedded streaming applications. In *Proceedings of the Ninth ACM International Conference on Embedded Software*, EMSOFT'11, pages 195–204, New York, NY, USA, 2011. ACM. doi:10.1145/2038642.2038672.

3 M. Bamakhrama and T. Stefanov. 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*, CODES+ISSS'12, pages 83–92, New York, NY, USA, 2012. ACM. doi:10.1145/2380445.2380464.

4 S. Baruah, A. Mok, and L. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the 11th Real-Time Systems Symposium*, pages 182–190, Orlando, Florida, 1990. IEEE Computer Society Press. doi:10.1109/REAL.1990.128746.

5 A. Bouakaz, T. Gautier, and J. P. Talpin. Earliest-deadline first scheduling of multiple independent dataflow graphs. In *2014 IEEE Workshop on Signal Processing Systems (SiPS)*, pages 1–6, Oct 2014. doi:10.1109/SiPS.2014.6986102.

6 T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, third edition, 2009.

7 M. Dertouzos. Control robotics : the procedural control of physical processors. In *Proceedings of the IFIP Congress*, pages 807–813, 1974.

8 E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, December 1959. doi:10.1007/BF01386390.

9 N. Fisher, T. Baker, and S. Baruah. 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*, Sydney, Australia, August 2006. IEEE Computer Society Press. doi:10.1109/RTCSA.2006.12.

10 A. H. Ghamarian, S. Stuijk, T. Basten, M. C. W. Geilen, and B. D. Theelen. Latency minimization for synchronous data flow graphs. In *Digital System Design Architectures, Methods and Tools, 2007. DSD 2007. 10th Euromicro Conference on*, pages 189–196, Aug 2007. doi:10.1109/DSD.2007.4341468.

11 Jad Khatib, Alix Munier-Kordon, Enagnon Cédric Klikpo, and Kods Trabelsi-Colibet. 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'16, pages 87–96, New York, NY, USA, 2016. ACM. doi:10.1145/2997465.2997479.

12 Enagnon Cédric Klikpo and Alix Munier-Kordon. 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'16, pages 77–86, New York, NY, USA, 2016. ACM. doi:10.1145/2997465.2997474.

13 E. A. Lee. *A Coupled Hardware and Software Architecture for Programmable Digital Signal Processors*. PhD thesis, EECS Department, University of California, Berkeley, 1986. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/1986/715.html.

14 E. A. Lee and D. G. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Trans. Comput.*, 36(1):24–35, January 1987. doi:10.1109/TC.1987.5009446.

15 E. A. Lee and D. G. Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235–1245, Sept 1987. doi:10.1109/PROC.1987.13876.

16 E. A. Lee and S. A. Seshia. *Introduction to Embedded Systems, A Cyber-Physical Systems Approach*. MIT Press, 2011. URL: http://LeeSeshia.org.

17 C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, 1973. doi:10.1145/321738.321743.

18 M. Mohaqeqi, J. Abdullah, and W. Yi. 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*, pages 15–29, New York, NY, USA, 2016. Springer-Verlag New York, Inc. doi:10.1007/978-3-319-39083-3_2.

19 A. Mok. *Fundamental Design Problems of Distributed Systems for The Hard-Real-Time Environment*. PhD thesis, Laboratory for Computer Science, Massachusetts Institute of Technology, 1983. Available as Technical Report No. MIT/LCS/TR-297. URL: http://hdl.handle.net/1721.1/15670.

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

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

22 F. Siyoum. *Worst-case temporal analysis of real-time dynamic streaming applications*. PhD thesis, PhD thesis, Eindhoven University of Technology, 2014. doi:10.6100/IR780952.

23 S. Sriram and S. S. Bhattacharya. *Embedded Multiprocessors: Scheduling and Synchronization*. Marcel Dekker, Inc., 2000.

24 M. Stigge. *Real-Time Workload Models: Expressiveness vs. Analysis Efficiency*. PhD thesis, Ph.D. thesis, Uppsala University, 2014.

25 M. Stigge, P. Ekberg, N. Guan, and W. Yi. The digraph real-time task model. In *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 71–80, April 2011. doi:10.1109/RTAS.2011.15.

26 M. Stigge and W. Yi. Graph-based models for real-time workload: A survey. *Real-Time Syst.*, 51(5):602–636, September 2015. doi:10.1007/s11241-015-9234-z.

27 S. Stuijk, M. C. W. Geilen, and T. Basten. SDF^3: SDF For Free. In *Application of Concurrency to System Design, 6th International Conference, ACSD 2006, Proceedings*, pages 276–278. IEEE Computer Society Press, Los Alamitos, CA, USA, June 2006. URL: http://www.es.ele.tue.nl/sdf3, doi:10.1109/ACSD.2006.23.

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