Efficient Computation of Buffer Capacities for Cyclo-Static Real-Time Systems with Back-Pressure

Maarten H. Wiggers [1], Marco J.G. Bekooij [2] , Pierre G. Jansen [1], Gerard J.M. Smit [1]
[1] University of Twente, Enschede, The Netherlands
[2] NXP Semiconductors, Eindhoven, The Netherlands
m.h.wiggers@utwente.nl

Abstract. *This paper describes a conservative approximation algorithm that derives close to minimal buffer capacities for an application described as a cyclo-static dataflow graph. The resulting buffer capacities satisfy constraints on the maximum buffer capacities and end-to-end throughput and latency constraints. Furthermore we show that the effects of run-time arbitration can be included in the response times of dataflow actors. We show that modelling an MP3 playback application as a cyclo-static dataflow graph instead of a multi-rate dataflow graph results in buffer capacities that are reduced up to 39%. Furthermore, the algorithm is applied to a real-life car-radio application, in which two independent streams are processed.*

1. Introduction

In the multi-media domain, applications are often implemented as task graphs and executed on an Multi-Processor System-on-Chip (MPSoC), e.g. in smart-phones and car-radio's. This is because an MPSoC provides high data processing capabilities in a power efficient manner, which results in a cost-effective solution.

Applications consist of multiple jobs, where each job is a task graph that can be started or stopped by the end-user. A job has real-time requirements, such as end-to-end throughput and latency, that can be firm or soft. In our system [1], we allow jobs with firm and soft real-time requirements to share resources of the MPSoC, such as for instance processors.

The tasks that constitute a job communicate containers over fixed capacity First-In First-Out (FIFO) buffers. A task only starts its execution when sufficient containers filled with data are available on its input buffers and sufficient empty containers are available on its output buffers such that the task will not need to block during its execution. The fact that a task only starts when sufficient empty containers are present leads to back-pressure and is an effective mechanism to prevent buffer overflow. In contrast with traffic shapers, back-pressure enables a task to start as soon as there are sufficient containers.

Since jobs are started and stopped by the user, run-time arbitration is required on resources that are shared by multiple jobs. If jobs with soft and firm real-time requirements share a resource, then we only allow pre-emptive arbitration mechanisms that provide resource budget guarantees [13], and thereby enable a separate analysis of these jobs. Resource budget guarantees are not required if upper bounds are known on the execution times of all tasks that share the resource.

In [20], we presented a conservative approximation algorithm that derived close to minimal buffer capacities, which satisfy constraints on the maximum buffer capacities and a throughput constraint. This algorithm is applicable to derive buffer capacities of jobs, if each task can be modelled as a Multi-Rate Dataflow (MRDF) actor [11]. Alternative approaches are exact, which can lead to excessive run-times and memory requirements since they operate on the corresponding Single-Rate Dataflow (SRDF) graph [17], which can be exponentially larger [16].

In this work, we extend [20] and show that the algorithm can be applied on a Cyclo-Static Dataflow (CSDF) graph [2, 15] that conservatively models the temporal behaviour of a task graph that is mapped on an MPSoC. We furthermore show that the effects of run-time arbitration can be included in the response times of the actors and that the algorithm can satisfy latency constraints. We also show that there is an interesting class of SRDF graphs for which this work provides a low complexity algorithm to verify temporal constraints.

By modelling a task with a CSDF actor, we remove the restriction that a task needs to produce and consume a constant number of containers in every execution and we no longer need to determine a single bound on the response times of all executions. Instead we allow variation in the production and consumption rates and in the bound on the response time, as long as we can identify a period with which this behaviour repeats. We will show an MP3 playback example for which this leads to a more detailed model and significantly reduced buffer capacities. Other examples that show the difference between MRDF and CSDF are discussed in [2, 15]. The consequence of the variance for the algorithm is that no longer a strictly periodic schedule can be constructed, as was done in [20] for MRDF graphs.

Our approach relies on the fact that container arrival times can be bounded from above, because the corresponding CSDF graph has a *monotonic* temporal behaviour. Monotonicity implies that a shorter response time or an earlier start time does not lead to a later container arrival time. This means that we can construct a conservative schedule that satisfies the temporal constraints to derive buffer capacities. The production times of this schedule are conservative compared to the container production times when executing the task graph. This means that buffer capacities derived with the conservative schedule are sufficient buffer capacities in the implementation.

Scheduling approaches that do not include run-time arbitration, like the time-triggered [10] and static-order [9] approaches, are difficult to apply for a system that includes a mix of streams that have firm or soft real-time constraints. This is because soft real-time streams often have execution times that are impractical to bound, and can have a data dependent number of task executions. Through application of arbiters that provide resource budget guarantees, we can analyse firm real-time streams separately from soft real-time streams.

Scheduling approaches that include run-time arbitration, as for instance presented by Jersak [8, 7], Goddard [5], or Maxiaguine [12], do not allow cyclic dependencies that influence the temporal behaviour of the system. Not only do these cycles occur, because of functional constraints, this also means that back-pressure through bounded FIFO buffers cannot be applied in these approaches. In the single-processor context, the multiframe task model [14] is related in the sense that in this model tasks also go through a sequence of execution times in a cyclic fashion. However, the multiframe task model does not consider dependencies between tasks.

Two alternative approaches that determine buffer capacities for CSDF graphs are known to us. The back-tracking approach to derive a schedule as applied by [2] can be extended to deal with temporal constraints, constraints on buffer capacities, and buffer capacity minimisation. Another approach is to back-track over the possible buffer capacities of the CSDF graph, and check whether the temporal constraints are satisfied by applying MCM analysis on the corresponding SRDF graph [18]. Both approaches suffer from an exponential space and time complexity, because they schedule SRDF actors, but are able to satisfy the throughput requirement and buffer capacity constraints for a larger set of problem instances, because they are exact.

The organization of this paper is as follows. Relevant properties of CSDF graphs are summarised in Section 2, while in Section 3 the relation between an implementation and its CSDF graph is discussed. In Section 4 we present our algorithm. By applying the algorithm on an MP3 playback application, we investigate the run-time and accuracy of the algorithm in Section 5. In this section we furthermore provide a case study that shows the applicability of our approach to a real-life car-radio application. We conclude in Section 6.

2. Dataflow graph definition

The input to our algorithm is a CSDF [2] graph that models the application. A CSDF graph is a directed graph $G = (V, E, \delta, \rho, \pi, \gamma, \theta)$ that consists of a finite set of actors V, and a set of directed edges, $E = \{(v_i, v_j) | v_i, v_j \in V\}$. Actors synchronise by communicating tokens over edges that represent queues. The graph G has an initial token placement $\delta : E \to \mathbb{N}$. An actor v_i has $\theta(v_i)$ distinct phases of execution, with $\theta : V \to \mathbb{N}$, and transitions from phase to phase in a cyclic fashion. An actor is enabled to fire when the number of tokens that will be consumed is available on all its input edges. The number of tokens consumed in firing k by actor v_i is determined by the edge and the current phase of the token consuming actor, $\gamma : E \times \mathbb{N} \to \mathbb{N}$, and therefore equals $\gamma(e, ((k-1) \bmod \theta(v_i)) + 1)$ tokens. The specified number of tokens is consumed in an atomic action from all input edges when the actor is started. The response time $\rho(v_i, f)$, $\rho : V \times \mathbb{N} \to \mathbb{R}^+$, is the difference between the finish and the start time of phase f of actor v_i. The response time of actor v_i in firing k is therefore $\rho(v_i, ((k-1) \bmod \theta(v_i))+1)$. When actor v_i finishes, then it produces the specified number of tokens on each output edge $e_{ij} = (v_i, v_j)$ in an atomic action. The number of tokens produced in a phase will be denoted by $\pi : E \times \mathbb{N} \to \mathbb{N}$.

For edge $e_{ij} = (v_i, v_j)$, we define $\Pi(e_{ij})$ as the number of tokens produced in one cyclo-static period, with $\Pi(e_{ij}) = \sum_{f=1}^{\theta(v_i)} \pi(e_{ij}, f)$, while $\Gamma(e_{ij})$ provides the number of tokens consumed in one cyclo-static period, with $\Gamma(e_{ij}) = \sum_{f=1}^{\theta(v_j)} \gamma(e_{ij}, f)$. We further define the topology matrix Ψ as a $|E| \times |V|$ matrix, where

$$\Psi_{ij} = \begin{cases} \Pi(e_i) & \text{if } e_i = (v_j, v_k) \\ -\Gamma(e_i) & \text{if } e_i = (v_k, v_j) \\ \Pi(e_i) - \Gamma(e_i) & \text{if } e_i = (v_j, v_j) \\ 0 & \text{otherwise} \end{cases}$$

If the rank of Ψ is $|V| - 1$, then a connected CSDF graph is said to be consistent [2]. A consistent CSDF graph requires queues with finite capacity, while an inconsistent CSDF graph requires infinite queue capacity.

We define the vector $\mathbf{s}$ of length $|V|$, as a positive integer solution to $\Psi\mathbf{s} = \mathbf{0}$. This vector determines the relative firing frequencies of the cyclo-static periods. The repetition vector $\mathbf{q}$ of the CSDF graph determines the relative firing frequencies of the actors and is given by

$$\mathbf{q} = \Theta\mathbf{s} \quad \text{with} \quad \Theta_{jk} = \begin{cases} \theta(v_j) & \text{if } j = k \\ 0 & \text{otherwise} \end{cases}$$

The repetition rate $\mathbf{q}_x$ of actor v_x is therefore the number of phases of v_x within one cyclo-static period times the relative firing frequency of this cyclo-static period.

The throughput constraint is specified by a period μ in which each actor v_i has to fire $\mathbf{q}_i$ times. In the remainder of this paper, we assume that the required period μ is given. The case study in Section 5.2 provides an example that shows how a period μ can be chosen. The specification of an end-to-end latency constraint is discussed in Section 4.3.

2.1. Temporal monotonic execution

If a CSDF graph is executed in a self-timed manner, then actors start execution as soon as they are enabled. Further we say that a CSDF graph maintains a FIFO ordering of tokens, if each actor either has a constant response time, or has a self-cycle with one initial token. This is because queues by definition maintain FIFO ordering of tokens, which means that tokens cannot overtake each other in such a CSDF graph.

An important property is that self-timed execution of a strongly connected CSDF graph that maintains a FIFO ordering of tokens is monotonic in time, which is defined as follows.

Definition 1 *A CSDF graph executes monotonically in time if no decrease in response time or start time of any firing k of any actor v_i can lead to a later enabling of any firing l of any actor v_j.*

If a CSDF graph G maintains FIFO ordering of tokens, then the self-timed execution of G is monotonic. This is because a decrease in response time or start time can only lead to earlier token production times, and therefore only to an earlier actor enabling.

In [2], every CSDF actor implicitly has a self-cycle with a single token. In contrast with [2] we explicitly model the self-cycle with a single token. This is because we would like to check whether the constraint imposed by a self-cycle with a single token does not lead to a violation of the throughput constraint. And furthermore, we allow CSDF actors with a constant response time without a single token self-cycle, because actors with a constant response time also maintain FIFO ordering of tokens.

2.2. Example

An example CSDF graph is shown in Figure 1. In this graph, v_p has the response time sequence $r_p = \langle 2, 1\rangle$ and v_c has the response time sequence $r_c = \langle 1, 1\rangle$. The vector $\mathbf{s}$ is $[2\,3]^{\mathrm{T}}$, and the repetition vector is $\mathbf{q} = [4\,6]^{\mathrm{T}}$. One instance of the problem discussed in this paper is to find a token placement δ such that the period μ of the graph in Figure 1 is at least 12, and the constraints on d_1 and d_3 are satisfied. The presented algorithm will tell us that buffer capacities $d_1 = d_3 = 1$, and $d_2 = 3$ are sufficient to satisfy the constraints.

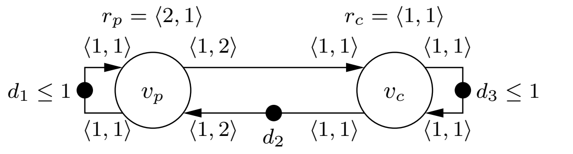


Figure 1. Example buffer capacity problem.

3. Dataflow graph construction

The CSDF graph described in the previous section is a model of the temporal behaviour of an implementation. In this section we define a task graph and the construction rules that given a task graph result in a CSDF graph. We will show that, also if tasks are run-time scheduled, worst case temporal behaviour of the implementation can be derived from the constructed CSDF graph. This is a contribution of this paper. We conclude this section with a discussion on the expressivity of CSDF compared to MRDF.

3.1. Task graph definition

The implementation is a task graph, which is a weakly connected directed graph $T = (W, B, \zeta, \eta, \lambda, \xi, \sigma, \psi, \vartheta)$. A weakly connected directed graph is a graph in which for every pair of vertices x and y, a path exists from x to y or from y to x. The finite set of vertices W represents the set of tasks, while the set of edges $B = \{(w_i, w_j)|w_i, w_j \in W\}$ represents the set of FIFO buffers. Tasks synchronise on containers and communicate over FIFO buffers. The capacity of a FIFO buffer b is the number of containers that can be stored in this buffer, and is denoted $\zeta(b)$, with $\zeta : B \rightarrow \mathbb{N}$. The amount of data that can be stored in a container is defined per FIFO buffer. The number of containers stored in a buffer b at the start of the application equals $\eta(b)$, with $\eta : B \rightarrow \mathbb{N}$. Each task w_i executes $\vartheta(w_i)$ code segments in a cyclic fashion, with $\vartheta : W \rightarrow \mathbb{N}$. A code segment is enabled when the number of containers that will be consumed during its execution is available in its input buffers and space for the number of containers that will be produced is available in its output buffers. A code segment checks if it is enabled before it starts. This check prevents that the code segment needs to wait for additional data or space before it can finish its execution. Code segment k of task w_j consumes $\lambda(b, k)$ containers per execution from buffer $b = (w_i, w_j)$, with $\lambda : B \times \mathbb{N} \rightarrow \mathbb{N}$. These containers are consumed before the code segment finishes its execution. A code segment k of task w_j produces $\xi(b, k)$ containers per execution in the buffer $b = (w_j, w_i)$, with $\xi : B \times \mathbb{N} \rightarrow \mathbb{N}$. The difference between the enabling and finish time of a code segment is the response time of this code segment. The response time of code segment k of task w_i is smaller than or equal to its worst-case response time (WCRT) $\sigma(w_i, k)$, with $\sigma : W \times \mathbb{N} \rightarrow \mathbb{R}^+$. In Section 3.4, we will show that upper bounds on the response times can be determined given the task to processor assignments, arbiter settings, and the worst case execution times. The specified number of containers are produced *before* the code segment finishes its execution. If a code segment is not pre-empted during its execution, then this code segment will finish its execution within its worst-case execution time (WCET) after it has started. The WCET of code segment k of task w_i is denoted by $\psi(w_i, k)$, with $\psi : W \times \mathbb{N} \rightarrow \mathbb{R}^+$.

Figure 2 shows a task graph with buffer $b = (w_p, w_c)$, container productions $\xi(b, 1) = 1$, $\xi(b, 2) = 2$, container consumptions $\lambda(b, 1) = 1$, $\lambda(b, 1) = 1$, and one initial container. Further we have for task w_p the response time sequence $u_p = \langle\sigma(w_p, 1), \sigma(w_p, 2)\rangle = \langle 2, 1\rangle$ and for task w_c the response time sequence $u_c = \langle\sigma(w_c, 1), \sigma(w_c, 2)\rangle = \langle 1, 1\rangle$.

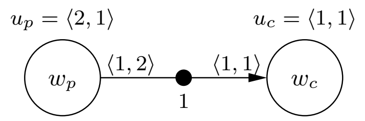


Figure 2. An example task graph.

3.2. Dataflow model of the application

In this section a CSDF graph is constructed that has a one-to-one correspondence with the task graph.

Each task w_i in the implementation is modeled by an actor v_i in the dataflow model. The response time of phase k of actor v_i is equal to the upper bound on the response time of the corresponding code segment, i.e. $\rho(v_i, k) = \sigma(w_i, k)$. For each actor v_i, a self-edge (v_i, v_i) with one initial token is added in the dataflow model on which each phase produces a token and from which each phase consumes a token. This captures the constraint that a code segment cannot start before the previous code segment has finished. Each FIFO buffer $b_i = (w_i, w_j)$ is modeled by an edge $e_{ij} = (v_i, v_j)$ and an edge $e_{ji} = (v_j, v_i)$, and each container is modeled by a token. The number of initial tokens on edge e_{ij} equals

$\eta(b_i)$, while the number of initial tokens on edge e_{ji} equals $\zeta(b_i) - \eta(b_i)$. The number of tokens that are consumed in the execution of code segment k from edge e_{ij} is $\lambda(b_i, k)$, which equals the number of tokens produced by this code segment on edge e_{ji}. Furthermore, the number of tokens that are produced in the execution of code segment k on edge e_{ij} equals $\xi(b_i, k)$, which equals the number of tokens consumed from edge e_{ji} by this code segment.

If we apply this procedure on the task graph shown in Figure 2, then we obtain the CSDF graph in Figure 1, i.e. if $d_1 = d_3 = 1$, and with no initial containers.

Since the task graph is weakly connected and each buffer in the task graph results in two edges in opposite direction in the CSDF graph, a task graph is modelled by a strongly connected CSDF graph. FIFO ordering of tokens is maintained because each actor has a self-edge with one initial token. In Section 2.1 we have shown that the self-timed execution of a strongly connected CSDF graph that maintains FIFO ordering is monotonic in time.

The CSDF graph is constructed such that there is a one-to-one correspondence with the task graph. This is because for each task there exists a corresponding actor with the same enabling condition. Further we have that each buffer in the task graph corresponds with two edges in the CSDF model. The availability of data or space in the buffer corresponds with the presence of tokens on one of these edges, and further if a code segment consumes a container, then it creates space in a FIFO buffer. This action can be seen as the production of an empty container. The production of an empty container corresponds with the production of a token in the CSDF graph.

3.3. Worst-case production times

In this section we show that container production times are not later than the corresponding token production times, i.e. if code segments are not enabled later than their corresponding phases, and if both the task graph and the CSDF graph execute in a self-timed manner. This is an important contribution of this paper. We arrive at this conclusion for the following reasons. First of all, there is a one-to-one correspondence between the task graph and the CSDF graph. Relevant differences are that (1) code segments produce their containers before their execution finishes, while a phase produces tokens at the end of its firing, and (2) a code segment can have a response time that is smaller than its worst-case response time, while the response time of a phase equals the worst-case response time. In both cases, when code segment and phase are enabled at the same time, this only leads to earlier container productions than token productions. Since we know that the self-timed execution of the CSDF graph has a monotonic temporal behaviour and we have a one-to-one relation between CSDF graph and task graph, we arrive at the following conclusion. If during self-timed execution, the first execution of the first code segment of each task is not enabled later than the corresponding phase, then a shorter response time of a code segment or producing containers before the code segment finishes cannot result in a later production of containers than the production of the corresponding tokens.

Sriram [18] bases his work on a similar observation about the arrival time of containers compared to tokens. A subtle but important difference is that we allow a code segment to start later than the corresponding phase. This is possible, because we use worst-case response times of code segments instead of their worst-case execution times. Given the worst-case response time of the code segment we have shown that even if the execution of a code segment starts later, e.g. because a different task is currently executing on the same resource, the code segment will still produce its containers earlier than the corresponding phase produces its tokens. The reason is that we know that while a code segment may start later than the corresponding phase, it is not enabled later than this phase. And while the execution time is defined relative to the start of a code segment, the response time is defined relative to the enabling of a code segment.

In Section 4, we will describe an algorithm, with a low run-time, that creates a conservative schedule of actor phase firings from which conservative container arrival times can be derived.

3.4. Response time calculation

In our multi-processor system, we only apply arbiters that provide resource budgets. These arbiters allow the derivation of a WCRT of a code segment based on only the WCET of the code segment and the arbiter settings, such as e.g. Time Division Multiplex (TDM). However, even though an increasing number of processors support pre-emption, classical digital signal processors, as for instance applied in the MPSoC of our case study in Section 5.2, do not support pre-emption. Therefore, we also apply arbiters that allow the derivation of the WCRT based on only the arbiter settings and the WCET of all code segments that execute on the same processor, such as e.g. Round-Robin (RR). Application of these arbiters requires that the execution times of all code segments that execute on this processor can be bounded. Arbiters for which the derivation of the WCRT of one code segment is dependent on the inter arrival times of containers consumed by a code segment of a higher priority task, such as e.g. static priority based arbiters, are not applied in our multi-processor system. This is because the WCRT is an input to our analysis, while inter arrival times depend, among others, on the buffer capacity, which is the result of our analysis.

If TDM arbitration is applied, then the WCRT of code segment k of task w_i can be calculated with Equation (1), in which the length of the TDM period is m and the length of the time slice of task w_i is n_i. The WCRT can be computed with this equation because code segment k of task w_i is at most $\lceil \psi(w_i,k)/n_i \rceil$ times pre-empted during its execution. The total time that the task cannot make progress because it is pre-empted, is thus at most $(m - n_i)\lceil \psi(w_i,k)/n_i \rceil$.

$$\sigma(w_i, k) = \psi(w_i, k) + (m - n_i) \left\lceil \frac{\psi(w_i, k)}{n_i} \right\rceil \qquad (1)$$

If n_i is small compared to $\psi(w_i, k)$, then the effect of rounding to the next larger integer value is small and a tight bound on the WCRT is computed with Equation (1).

A code segment can do busy waiting if it is not enabled. The time, during which a code segment is busy waiting, is not included in the response time, because the response time is defined as the interval of time between enabling and finish.

An earlier arrival of a container allows for a longer stay in a FIFO buffer. Therefore, a potentially too early arrival of containers should not increase the worst-case response time estimate. According to Equation (1) this is the case for dataflow models. This is, however, not the case for event-models [8]. As a consequence, a more accurate estimate of the minimal throughput and maximum end-to-end latency can be obtained with dataflow models than with event-models.

If round-robin arbitration is applied, then the maximum time between enabling of a code segment of task w_i and the start of the execution of this code segment is given by $K(w_i)$ as determined by Equation 2, where P is the set of tasks in the round-robin list.

$$K(w_i) = h + \sum_{w_l \in P \setminus \{w_i\}} \max(\{\psi(w_l, j) | 1 \leq j \leq \vartheta(w_l)\}) \tag{2}$$

This is because it can occur that the code segment of task w_i needs to wait on a code segment of each of the other tasks that share this processor, and by taking the code segments with the maximum execution time into account we obtain a conservative bound on this waiting time. The time h is added because the container can arrive just after it was checked whether the task is enabled, and this check still requires time to return before other tasks can start their execution. After it is detected that the actor is enabled it takes at most $\psi(w_i, k)$ before the task finishes its execution. Equation 3 therefore provides the worst-case response time of code segment k of task w_i.

$$\sigma(w_i, k) = \psi(w_i, k) + K(w_i) \tag{3}$$

3.5. Expressivity of CSDF

A task executes a sequence of code segments. We will show that CSDF actors can model a larger class of tasks than MRDF actors. Figure 3 shows an example CSDF graph taken from [15] in which merging the actors v_1 and v_3 into an MRDF actor v_m will lead to deadlock, i.e. this creates a cycle with no initial tokens where v_2 requires a token on an edge from the actor v_m and the actor v_m requires a token from actor v_2 in order to be enabled. A CSDF actor can merge the actors v_1 and v_3 without introducing deadlock, since a CSDF actor can still allow the firing sequence $\langle v_1, v_2, v_3 \rangle$. Therefore, the task consisting of the code segments as modelled by actors v_1 and v_3 can be modelled with a CSDF actor, but should not be modelled with an MRDF actor. This shows that CSDF is more expressive than MRDF, if a one-to-one correspondence between tasks and actors is maintained.

4. Buffer capacity algorithm

In this section we will present an algorithm that determines close to minimal buffer capacities, which satisfy constraints on the maximal buffer capacities and on the end-to-end temporal behaviour.

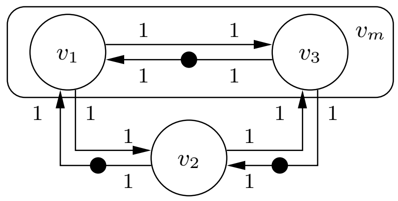


Figure 3. Deadlock if v_m is an MRDF actor.

The basic idea is to first construct for each actor v_i a schedule of phase finish times such that a schedule is created that is periodic with $\theta(v_i)$ finishes every μ/s_i time. From these finish times we can derive the times at which tokens are consumed and produced.

For example, for the graph in Figure 1, we first separately construct a token production schedule $\mathrm{p}(t)$ for actor v_p and a token consumption schedule $\mathrm{c}(t)$ for actor v_c that satisfy the throughput constraint. We will discuss the construction of these schedules in Section 4.1.

The following argument is illustrated by Figure 4. After construction of the two schedules $\mathrm{p}(t)$ and $\mathrm{c}(t)$, they are linearly bounded by $\mathrm{p}_l(t)$ and $\mathrm{c}_u(t)$. The bound $\mathrm{p}_l(t)$ gives a lower bound on the number of tokens that are produced on edge e_{pc} while the bound $\mathrm{c}_u(t)$ gives an upper bound on the number of tokens that are consumed from edge e_{pc}.

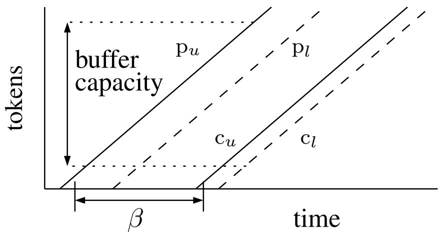


Figure 4. In order to minimise the buffer capacity, the distance $\mathrm{p}_u - \mathrm{c}_l$ needs to be minimised.

In Section 4.1, we will show that schedules can be constructed that have linear bounds $\mathrm{p}_l(t)$ and $\mathrm{c}_u(t)$ with the same slope. We further derive a scheduling distance β_{ij}, which is a sufficient – but close to minimal – distance between the start times of the first firings of two actors v_i and v_j that are connected by an edge (v_i, v_j). Since minimisation of the difference between these linear bounds leads to smaller buffer capacities, we determine a scheduling distance β_{ij} such that the linear bounds are equal.

The set of scheduling distances forms a set of constraints. In Section 4.3, we apply the algorithm from [20] that determines start times of all actors such that all scheduling constraints are satisfied. The topology of the graph may lead to scheduling distances ϵ_{ij} that are larger than the minimal scheduling distances β_{ij}. This occors if multiple paths exist between a pair of actors, and the sums of scheduling distances of these paths are different.

Through application of the scheduling distances ϵ_{pc} and the linear upper bound $\mathrm{p}_u(t)$ on tokens consumed from edge e_{cp} and the linear lower bound $\mathrm{c}_l(t)$ on tokens produced on edge e_{cp}, we can derive a sufficient buffer capacity for the example shown in Figure 1. This is discussed in detail in Section 4.2.

Note that while buffer capacities are derived such that the constructed schedule always has sufficient tokens available, the self-timed schedule in the implementation does experience back-pressure. And furthermore, since (1) start times in the constructed schedule are only delayed compared to the self-timed schedule and (2) the implementation has monotonic temporal behaviour, the self-timed schedule will also meet the throughput constraint.

In contrast with [20] the presented algorithm is applicable to CSDF, which is more expressive than MRDF. The fact that each firing can have a different response time and token transfer behaviour prevented the derivation of closed form expressions for both the minimum scheduling distance β and the buffer capacity. Furthermore, Section 4.3 shows that the algorithm can also include latency constraints and is applicable to an interesting class of SRDF graphs. For this class of SRDF graphs, our algorithm has a lower complexity than alternative algorithms.

4.1. Schedule construction

In this section periodic schedules for two communicating actors are constructed such that the firing rules are not violated, the required temporal behaviour is satisfied, and a minimal buffer capacity is required. The concepts are illustrated using the example CSDF graph, as shown in Figure 1.

We start by defining a number of variables that denote properties of an edge and the actors connected to it. In this section we look at a single edge $e = (v_p, v_c)$ of a strongly connected and consistent CSDF graph. Edge e connects a token producing actor $v_p \in V$ to a token consuming actor $v_c \in V$, and has $d = \delta(e)$ initial tokens.

In this section we define schedules in which the finish time of phase f of actor v_k is $z_{k,f}$ later than the finish time of phase $f - 1$, with

$$z_{k,f} = \frac{\mu}{\mathrm{s}_k \sum_{g=1}^{\theta(v_k)} \rho(v_k, g)} \rho(v_k, f) \qquad (4)$$

By application of Equation 4, the time between subsequent finishes is increased with the same relative amount, such that the q_k firings finish just within the period μ, leading to a schedule that satisfies the throughput constraint.

Token production schedule When considering an edge $e = (v_p, v_c)$, we construct a token production schedule in which phase 1 of actor v_p finishes at $z_{p,1}$ and phase $f > 1$ finishes $z_{p,f}$ time after the finish time of phase $f - 1$. We define $\mathrm{p}(t)$ with $\mathrm{p} : \mathbb{R} \to \mathbb{N}$ as the function that returns the number of tokens produced in the interval $[0, t]$ on edge e. And we define $\mathrm{p}^{-1}(i)$ with $\mathrm{p}^{-1} : \mathbb{N} \to \mathbb{R}$ as the function that returns the production time of token i on edge e. The function $\mathrm{p}(t)$ for edge (v_p, v_c) from the example CSDF graph is shown in Figure 5 where the upward arrows denote the start times of v_p. The full and empty circles show the discontinuities that are caused by the instantaneous token productions.

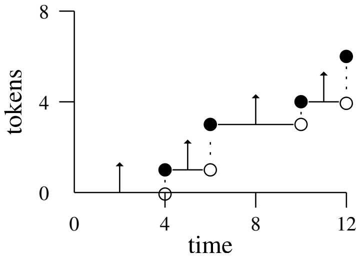


Figure 5. Producer $\mathrm{p}(t)$ schedule.

Token consumption schedule Similarly to the token production schedule, the token consumption schedule is constructed such that phase 1 of actor v_c finishes $\beta + \rho(v_c, 1)$ later than the start time of the first phase of actor v_p, where β is such that the token consumption schedule does not violate the firing rules. Phase $f > 1$ of actor v_c finishes $z_{c,f}$ time after the finish time of phase $f - 1$. We define $\mathrm{c}(t)$ with $\mathrm{c} : \mathbb{R} \to \mathbb{N}$ as the function that returns the number of tokens consumed in the interval $[0, t]$ on edge e, as shown in Figure 6. And we define $\mathrm{c}^{-1}(i)$ with $\mathrm{c}^{-1} : \mathbb{N} \to \mathbb{R}$ as the function that returns the consumption time of token i on edge e.

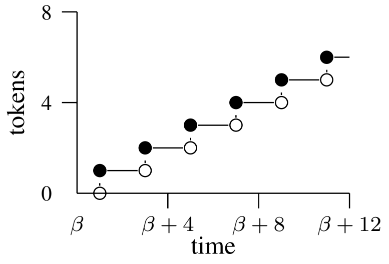


Figure 6. Consumer $\mathrm{c}(t)$ schedule.

We will now determine a minimal value for the schedule distance β. The approach is to derive $\mathrm{p}_u^{-1}(i) = \alpha_p i + \beta_p$, a linear upper bound on the production time of token i, and $\mathrm{c}_l^{-1} = \alpha_c i - \beta_c + \beta$, a linear lower bound on the consumption time of token i.

On edge $e = (v_p, v_c)$, every $\mathrm{s}_p\theta(v_p)$ firings of v_p, $\mathrm{s}_p\Pi(e)$ tokens are produced, and every $\mathrm{s}_c\theta(v_c)$ firings of v_c, $\mathrm{s}_c\Gamma(e)$ tokens are consumed. In the constructed schedule both $\mathrm{s}_p\theta(v_p)$ firings of v_p and $\mathrm{s}_c\theta(v_c)$ firings of v_c require μ time. We therefore take $\alpha_p = {}^{\mu}/_{(\mathrm{s}_p \cdot \Pi(e))}$, and $\alpha_c = {}^{\mu}/_{(\mathrm{s}_c \cdot \Gamma(e))}$. Due to consistency, we have that $\mathrm{s}_p\Pi(e) = \mathrm{s}_c\Gamma(e)$ and thus that $\alpha_p = \alpha_c$.

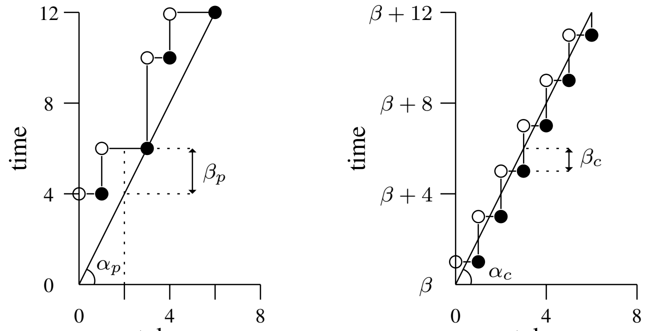
In order to determine an upper bound on the token production times, we will inspect the token production schedule and determine the maximum difference between the token production times as given by the bound with $\beta_p = 0$ and the token production times as given by the actual schedule. We will be able to show that a suitable upper bound is obtained by taking β_p equal to this maximum difference.

The difference between the bound and the token production schedule only needs to be determined for every

$\gamma(e,f), \gamma(e,f) \neq 0$ tokens that are produced. This is because the token consuming actor is only enabled after $\gamma(e,f)$ tokens are produced, if $\gamma(e,f) \neq 0$, and it is therefore sufficient that the upper bound on the token production times is only valid every $\gamma(e,f), \gamma(e,f) \neq 0$, tokens. The difference between the bound and the actual schedule needs to be determined until a periodic pattern arises in the observed differences, which occurs after a number of tokens have been produced that equals the least common multiple of $\Pi(e)$ and $\Gamma(e)$). This can result in problematic run-times since the result of a least common multiple can be exponentially larger than its operands. We therefore apply the following approach, for which we first define λ as the greatest common divisor (gcd) of the production and consumption rates on edge e: $\lambda = \gcd(\{\Pi(e)\} \cup \{\gamma(e,g) | g \in [1, \theta(v_c)] \wedge \gamma(e,g) \neq 0\})$. By observing the difference for every λ tokens that are produced, a repetitive pattern of differences will already arise after $\Pi(e)$ tokens have been produced. This is because λ divides $\Pi(e)$. Furthermore, combining the fact that the difference will repeat with period $\Pi(e)$ with the fact that every cumulative number of tokens produced can be described as $a\Pi(e) + b\lambda$, with $a, b \in \mathbb{N}$, leads to the conclusion that the maximum difference will be found if the difference is observed every λ tokens until $\Pi(e)$ tokens are produced.

Lemma 1 *The linear upper bound on the production time of token i on edge e, $\mathrm{p}_u^{-1} = \alpha_p i + \beta_p$, is found by taking $\beta_p = \max\{\mathrm{p}^{-1}(\lambda i) - \alpha_p \lambda i | i \in [1, {}^{\Pi(e)}/_{\lambda}]\}$.*

Proof. The proof follows by construction, an example is shown in Figure 7(a). We have already argued that the bound only needs to be valid every λ tokens. In order to create an exact linear upper bound, a linear function l is constructed with the same slope α_p and the maximum difference between the function that we bound, p^{-1}, and the linear function l leads to the distance over which l needs to be vertically shifted in order to be an exact upper bound for p^{-1}. $\square$



(a) Producer p^{-1} schedule with $\beta_p = 2$

(b) Consumer c^{-1} schedule with $\beta_c = 1$

Figure 7. Example derivation of β_p and β_c.

In order to determine a lower bound on the token consumption times, we only need to inspect the consumption times of every $\gamma(e,f)$ tokens consumed. This is because $\gamma(e,f)$ tokens are consumed simultaneously.

Lemma 2 *The linear lower bound on the consumption time of token i on edge e when assuming $\beta = 0$, and thus $\mathrm{c}_l^{-1} = \alpha_c i - \beta_c$, is found by taking $\beta_c = \max\{\alpha_c j - \mathrm{c}^{-1}(j) | j \in \{\gamma(e,1), \ldots, \Gamma(e) - \gamma(e, \theta(v_c) - 1), \Gamma(e)\}\}$.*

Proof. The proof follows by construction, an example is shown in Figure 7(b). The maximum difference between $\alpha_c i$ and $\mathrm{c}^{-1}(i)$ occurs when $i \in \{\gamma(e,1), \ldots, \Gamma(e) - \gamma(e, \theta(v_c) - 1), \Gamma(e)\}$, because $\gamma(e,f)$ tokens are consumed simultaneously. Symmetrically to the proof of Lemma 1, an exact lower bound is found by vertically shifting $\alpha_c i$ over the maximum difference between $\alpha_c i$ and $\mathrm{c}^{-1}(i)$. $\square$

Theorem 1 *If we let the first firing of actor v_c start after $\beta = \beta_p + \beta_c + z_{c,1} - r_{c,1} - z_{p,1} + r_{p,1} - \lambda \lfloor {}^{\delta(e)}/_{\lambda} \rfloor \alpha_c$ time then no token will be consumed before it is available, with $d = \delta(e)$.*

Proof. Without initial tokens, we have that $\mathrm{p}^{-1}(i) \leq \alpha_p i + \beta_p$ and $\mathrm{c}^{-1}(i) \geq \alpha_c i - \beta_c + \beta \leftrightarrow \alpha_c i - \beta_c + \beta \leq \mathrm{c}^{-1}(i)$. In order not to consume a token that is not available, it should hold that $\forall i \in [0, \Pi(e)], \mathrm{p}^{-1}(i) \leq \mathrm{c}^{-1}(i)$. This constraint is satisfied when $\alpha_p i + \beta_p \leq \alpha_c i - \beta_c + \beta$, since $\alpha_p = \alpha_c$ this means that $\beta_p + \beta_c$ is a sufficient difference between the start of the schedule of v_c and the start of the schedule of v_p. The first firing of v_p starts $z_{p,1} - r_{p,1}$ later than the start of its schedule and the first firing of v_c starts $z_{c,1} - r_{c,1}$ later than the start of its schedule. This results in $(\beta_p + \beta_c) - (z_{p,1} - r_{p,1}) + (z_{c,1} - r_{c,1})$.

With d initial tokens only $\lambda \lfloor {}^{d}/_{\lambda} \rfloor$ tokens can contribute to an earlier enabling of v_c. According to the lower bound on the consumption times, actor v_c consumes a token every α_c time. Which means that, with d initial tokens, the schedule of actor v_c can start $\lambda \lfloor {}^{d}/_{\lambda} \rfloor \alpha_c$ earlier. $\square$

For the example CSDF graph, as shown in Figure 1, β equals 2. Which means that if the first firing of actor v_c starts 2 time units later than the first firing of actor v_p, no tokens are consumed before they are produced. And further, for this example, β is minimal, since tokens are consumed at $t = 4$ that are also produced at $t = 4$.

4.2. Buffer capacity

In this section we will determine the required buffer capacity of a FIFO buffer modelled with a cycle C through the two actors v_p and v_c. The cycle C is formed by the edges $e_{pc} = (v_p, v_c)$ and $e_{cp} = (v_c, v_p)$. The edge e_{pc} determines a difference $\epsilon \in \mathbb{R}$ between the start times of the first firings of v_p and v_c, while tokens can be placed on edge e_{cp}.

While $\mathrm{p}(t)$ and $\mathrm{c}(t)$ are defined on the edge e_{pc}, the number of tokens consumed by v_p at time t from edge $e_{cp} = (v_c, v_p)$ is given by $\mathrm{b}(t)$ with $b : \mathbb{R} \to \mathbb{N}$. As required by the model, consumption of tokens by v_p on e_{cp} in phase f occurs $\rho(v_p, f)$ earlier than production of tokens on e_{pc}. Production of tokens on e_{cp} by v_c occurs when v_c finishes, and the number of tokens produced by v_c at time t is given by the function $\mathrm{o}(t)$ with $\mathrm{o} : \mathbb{R} \to \mathbb{N}$. Further the consumption time of token i is given by $\mathrm{b}^{-1}(i)$ with $\mathrm{b}^{-1} : \mathbb{N} \to \mathbb{R}$. And the production time of token i by v_c on edge e_{cp} is given by $\mathrm{o}^{-1}(i)$ with $\mathrm{o}^{-1} : \mathbb{N} \to \mathbb{R}$.

The required number of tokens on the cycle C can be found by determining the maximum difference between the number of tokens consumed by v_p, $\mathrm{b}(t)$, and the number of tokens produced by v_c, $\mathrm{o}(t)$, that will ever occur when v_p and v_c execute according to the constructed schedules.

We will determine a linear upper bound on the number of tokens consumed at time t: $b_u(t) = \alpha_b t + \kappa_b$. And a linear lower bound on the number of tokens produced at time t: $o_l(t) = \alpha_o t - \kappa_o$. Similarly to α_p and α_c, due to consistency, we have that $\alpha_b = \alpha_o = {}^{(\mathbf{s}_c \cdot \Pi(e_{cp}))}/_{\mu} = {}^{(\mathbf{s}_p \cdot \Gamma(e_{cp}))}/_{\mu}$.

Lemma 3 *An upper bound on the number of tokens consumed on edge e_{cp} is found by taking $\kappa_b = \max\{\mathrm{b}(t) - \alpha_b t | t \in [0, \sum_{f=1}^{\theta(v_p)} \rho(v_p, f)]\}$.*

Proof. Since $\mathrm{b}(t) \leq \mathrm{b}_u(t) = \alpha_b t + \kappa_b$, which can be rewritten to $\kappa_b \geq \mathrm{b}(t) - \alpha_b t$ taking the maximum difference between $\kappa_b \geq \mathrm{b}(t)$ and $\alpha_b t$ results in the minimum value for κ_b. □

Lemma 4 *A lower bound on the number of tokens produced on edge e_{cp} is found by taking $\kappa_o = \max\{\alpha_o t - \mathrm{o}(t) | t \in [0, \sum_{f=1}^{\theta(v_c)} \rho(v_c, f)]\}$.*

Proof. Since $\mathrm{o}(t) \geq \mathrm{o}_l(t) = \alpha_o t - \kappa_o$, which can be rewritten to $\kappa_o \geq \alpha_o t - \mathrm{o}(t)$, taking the maximum difference between $\alpha_o t$ and $\mathrm{o}(t)$ results in the minimum value for κ_o. □

Theorem 2 *A sufficient number of tokens to be placed on edge e_{cp} equals $\lambda \lfloor {}^{(\kappa_b + \kappa_o)}/_{\lambda} \rfloor$.*

Proof. An upper bound on the maximum difference between the number of tokens consumed from e_{cp} and the number of tokens produced on e_{cp} is $\mathrm{b}_u(t) - \mathrm{o}_l(t) = (\alpha_b t + \kappa_b) - (\alpha_o t - \kappa_o) = \kappa_b + \kappa_o$, because $\alpha_b = \alpha_o$.

Further because the different consumption and production rates are multiples of λ also the difference needs to be a multiple of λ. Since we have obtained an upper bound on the maximum difference, we can round to the next smaller value that is a multiple of λ. □

4.3. Algorithm

In this section we present the algorithm to compute sufficient buffer capacities of a strongly connected and consistent CSDF graph $G = (V, E, \delta, \rho, \pi, \gamma, \theta)$, given a repetition vector $\mathbf{q}$, and a required period μ.

Applicability While the results from the previous two sections are applicable to strongly connected and consistent CSDF graphs that execute monotonically, the presented algorithm is applicable to a subset of CSDF graphs. In terms of the implementation, the algorithm can be applied to applications in which each individual bounded buffer is either full or empty. And in terms of the analysis model, we consider the class of strongly connected and consistent CSDF graphs where, on each simple cycle that models a bounded buffer, initially all tokens are placed on one edge, and each actor has a self-cycle with one token. Each simple cycle has one edge on which tokens can be placed, and the initial tokens $\delta(e)$ on that edge are considered as the constraint on the buffer capacity.

Algorithm In our algorithm, the first firing of each actor v_i is assigned an as-soon-as-possible $\mathrm{asap}(v_i)$ and an as-late-as-possible $\mathrm{alap}(v_i)$ start time, such that the constraints are satisfied and the required buffer capacity is close to minimal. See [20] for a discussion about the various steps of this algorithm.

The set of edges E is partitioned in a set B that includes all edges on which initially tokens can be placed, and a set $\overline{B} = E \setminus B$ on which no initial tokens can be placed. The subgraph $\overline{D} = (V, \overline{B}, \delta, \rho, \pi, \gamma, \theta)$ is a connected acyclic graph [20].

1. Set $\mathrm{asap}(v_i) = 0$, and $\mathrm{alap}(v_i) = \infty$ for each actor v_i
2. Determine β_{ij} for each edge $(v_i, v_j) \in E$ as presented in Section 4.1
3. Create a graph $\overline{D} = (V, \overline{B}, \delta, \rho, \pi, \gamma, \theta)$
4. Create a list L of length $|V|$, where the actors of $\overline{D}$ are topologically sorted. If the topological sort fails, then report deadlock
5. Visit the actors in L from front to back, and for each actor v_i : $\mathrm{asap}(v_i) = \max(\{\mathrm{asap}(v_i)\} \cup \{\mathrm{asap}(v_x) + \beta_{xi} | (v_x, v_i) \in \overline{B}\})$
6. For all actors v_l that do not have successors in $\overline{D}$: $\mathrm{alap}(v_l) = \max(\{\mathrm{asap}(v_x) | v_x \in V\})$
7. Visit the actors in L from back to front, and for each actor v_i : $\mathrm{alap}(v_i) = \min(\{\mathrm{alap}(v_i)\} \cup \{\mathrm{alap}(v_x) - \beta_{ix} | (v_i, v_x) \in \overline{B}\})$
8. Visit the actors in L from front to back, and for each actor v_i : $\mathrm{alap}(v_i) = \min(\{\mathrm{alap}(v_i)\} \cup \{\mathrm{alap}(v_x) - \beta_{ix} | (v_i, v_x) \in B \wedge v_i \neq v_x\})$, and report a violation of the constraints if $\mathrm{asap}(v_i) > \mathrm{alap}(v_i) \vee \mu < s_k \sum_{g=1}^{\theta(v_k)} \rho(v_k, g)$
9. For each edge $(v_j, v_i) \in B$, a sufficient number of tokens can be determined using Theorem 2, if we take $\epsilon = \mathrm{alap}(v_j) - \mathrm{alap}(v_i)$.

Latency constraints We assume that a latency L is defined as a property of a single concept in the implementation, e.g. an audio sample or video frame, relative to a strictly periodically executing task w_i. In the CSDF graph this translates to two actors v_i and v_j that have the same repetition rate, $\mathbf{q}_i = \mathbf{q}_j$. Furthermore, all phases f of actor v_i need to have the same response time, $\rho_{i,f} = \rho(i, 1)$, and a violation of the throughput constraint should result if v_i does not execute strictly periodically, i.e. $\mu = \mathbf{q}_i \cdot \rho(i, 1)$. Further also all phases g of actor v_j need to have the same response time, $\rho_{j,g} = \rho(j, 1)$. With these constraints, we have that in the constructed schedule, the difference between the start times of firings of v_i and v_j is constant, and therefore equals the difference between the start times of the first firings of both actors. Through addition of an edge (v_j, v_i) to D with $\beta_{ji} = -L$ we add the constraint $s(v_i, k) \geq s(v_j, k) + \beta_{ji}$, which equals $s(v_j, k) \leq s(v_i, k) + L$, where $s(v_i, k)$ is the start time of firing k of actor v_i. A maximum latency constraint, with $L \geq 0$, only needs only to be verified at design time. This is because if tokens arrive in time to enable v_j, then also containers will arrive in time to enable task w_j, because we have shown that token arrival times are conservative container arrival times.

Complexity analysis Steps 1, 5, 6, and 7 each have complexity $O(|V|)$, steps 3 and 4 have complexity $O(|V| + |E|)$ [3], and step 8 has a complexity $O(|E|)$. In step 2, we have that, for each edge, the determination of the minimum scheduling distance is linear in the number of phases. This also holds for the determination of the buffer capacities in step 9. Steps 2 and 9 therefore have a complexity $O(|E|F)$, with $F = \max_i(\theta(v_i))$. The complexity of this algorithm is therefore $O(|V| + |E|F)$.

Steps 1, 5, 6, 7, and 8 determine the asap and alap times and have a complexity $O(|V| + |E|)$. This low complexity is due to the specific class of CSDF graphs that this algorithm can deal with. For CSDF graphs that include buffers that are initially partially filled, a single source longest path algorithm, as for instance the Bellman-Ford algorithm with complexity $O(|V||E|)$ [3], is required to determine asap and alap times [18]. Applying Bellman-Ford will however only check whether the constraints can be satisfied; a subsequent optimisation step is required to obtain buffer capacities that are close to minimal.

Application to SRDF graphs If a CSDF graph from the considered class is converted to an SRDF graph, then again the SRDF graph is a member of the considered class of CSDF graphs. This is because each CSDF actor with a self-cycle is converted into a connected graph of SRDF actor copies. And because between each CSDF actor there is at least one edge without initial tokens, also between each cluster of SRDF actors there is at least one edge without initial tokens. Consequently this algorithm can be applied on the corresponding SRDF by partitioning the set of edges of the SRDF graph in a set of edges with initial tokens and a set of edges without initial tokens.

In a SRDF graph each edge $e = (v_p, v_c)$ forms the constraint, $s(v_c, i) \geq s(v_p, i) + r_p - \mu\delta(e)$ [17], where $s(v_x, j)$ is the start time of firing j of actor v_x and r_x is the response time of the single phase that the SRDF actor v_x has.

Since the presented algorithm considers for each edge e the constraint $s(v_c, i) \geq s(v_p, i) + \beta_{pc}$, we need to show that $\beta_{pc} = r_p - \mu\delta(e)$. Theorem 1 states that $\beta_{pc} = \beta_p + \beta_c + z_c - r_c - z_p + r_p - \lambda \lfloor \delta(e)/\lambda \rfloor \alpha_c$. We have that $z_p = z_c = \mu$. According to the constructed token production schedule a token is produced every μ time, which equals α_p, resulting in $\beta_p = 0$. According to the constructed token consumption schedule a token is consumed at $\mu - r_c$ and subsequently every μ time. This means that $\beta_c = r_c$. With the additional knowledge that $\lambda = 1$, we derive that $\beta_{pc} = \beta_p + \beta_c + z_c - r_c - z_p + r_p - \delta(e)\alpha_c = \beta_p + \beta_c - r_c + r_p - \delta(e)\alpha_c = r_c - r_c + r_p - \delta(e)\alpha_c = r_p - \delta(e)\alpha_c = r_p - \delta(e)\mu$.

Reiter [17] defines the MCM as the minimal period for which a periodic schedule exists, and [18] shows that the MCM equals the minimal period of the self-timed schedule. Therefore if our algorithm cannot find a periodic schedule for the SRDF graph, then also no self-timed schedule exists.

This insight implies that exact throughput constraint satisfaction analysis is possible for strongly connected and consistent CSDF graphs, if on each simple cycle that models a bounded buffer the maximum number of tokens is constrained and if each CSDF actor has a self-cycle with one token, with a lower complexity than the traditionally applied Bellman-Ford algorithm [18]. Note that the resulting SRDF graph depends on the topology of the CSDF graph and on the number of tokens in the CSDF graph, therefore constraint satisfaction verification is possible on the SRDF graph but derivation of buffer capacities is not.

5. Experimental validation

In this section we will first show the behaviour of the presented algorithm by comparing the buffer capacities for an MP3 playback application that can be modelled as an MRDF graph with the buffer capacities for this application when the application has CSDF behaviour. Subsequently we will describe a real-life case study in which the presented algorithm is applied to determine buffer capacities that satisfy the throughput and latency constraints.

5.1. Experimental results

We will now apply our algorithm to determine buffer capacities for an MP3 playback application. The implementation of this application is first modelled as an MRDF graph, and subsequently as a CSDF graph. For both models the runtime and accuracy of the presented algorithm is compared with alternative approaches. The algorithm can be applied on the MRDF model, because MRDF is a subclass of CSDF, where MRDF restricts each actor to only have a single phase.

In the MP3 playback application, of which the MRDF graph is shown in Figure 8, with $R = 1152$, and $S = 1$, the compressed audio is decoded by the MP3 task into a 48 kHz audio sample stream. These samples are converted by the Sample Rate Converter (SRC) task into a 44.1 kHz stream, after which the Audio Post-Processing (APP) task enhances the perceived quality of the audio stream and sends the samples to a Digital to Analogue Converter (DAC).

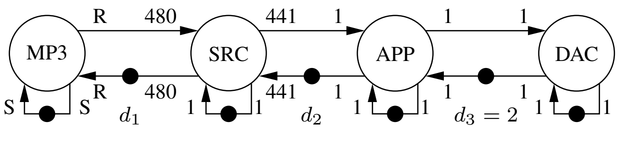


Figure 8. Dataflow model of the MP3 playback application

We performed an experiment with the MP3 playback application in which the topology of the MRDF graph is fixed to the topology shown in Figure 8. The repetition vector is $[5\ 12\ 5292\ 5292]^T$. The frequency of the DAC is 44.1 kHz, leading to $r_{DAC} = 1/44100$ s. And further $r_{MP3} = 7.51$ ms, $r_{SRC} = z_{SRC}$, and $r_{APP} = z_{APP} = 1/44100$ s. The required period μ is fixed to $\mathbf{q}_{DAC} * r_{DAC} = 120$ ms. This means that $r_{SRC} = {}^{120}/_{12} = 10\,\text{ms}$. In this experiment, the response time of the sample rate converter is varied in order to show the behaviour of the presented algorithm.

Table 1 lists the resulting buffer capacities for this experiment, both as determined by the algorithm presented in Section 4.3 and as determined through back-tracking where the throughput of the MRDF graph is determined with Maximum Cycle Mean (MCM) analysis [18], which is applied on the SRDF graph.

	Buffer Capacity						rel.
	d_1		d_2		$d_1 + d_2$		inc.
	alg.	opt.	alg.	opt.	alg.	opt.	(%)
r_{SRC}	2304	2016	882	882	3186	2898	10
$^3/_4 \cdot r_{SRC}$	2208	1824	772	988	2980	2812	6
$^1/_2 \cdot r_{SRC}$	2112	1536	662	772	2774	2308	20
$^1/_4 \cdot r_{SRC}$	2016	1536	552	551	2568	2087	23

Table 1. Our results (alg.), the optimal results (opt.) for the MRDF model of the MP3 playback application, and the relative increase.

The behaviour of the MP3 task can however be modelled in more detail, which, as the following experiment shows, results in smaller buffer capacities. The decoding of every MP3 frame involves the decoding of the header, decoding of 2 granules, and decoding of 18 sub-bands that each consist of 32 samples per granule. Only after the sub-bands of a granule are decoded is the decoded data sent to the SRC. This behaviour can be concisely modelled with a CSDF actor. In this case R equals $\langle 0, 0, 18\text{x}32, 0, 18\text{x}32 \rangle$ and S equals $\langle 1, 1, 1, 1, 1 \rangle$ in Figure 8.

The CSDF model has a different R, which results in a repetition vector $[195 \quad 12 \quad 5292 \quad 5292]^{\mathrm{T}}$, and further $r_{MP3} = \langle 670, 2700, 18\text{x}40, 2700, 18\text{x}40 \rangle$ µs. Again the response time of the sample rate converter is varied to show the behaviour of the presented algorithm.

Table 2 lists the resulting buffer sizes for this experiment, both as determined by the algorithm presented in Section 4.3 and as determined through back-tracking where the throughput of the CSDF graph is determined with Maximum Cycle Mean (MCM) analysis [18], which is applied on the SRDF graph. In this experiment, we see that the application of CSDF instead of MRDF leads to a reduction of up to 39% in the total buffer capacities, as determined by our algorithm, for the case in which the response time of the sample rate converter is divided by four.

The presented algorithm confirms that one token on each self-cycle and $d_3 = 2$ is sufficient. Further, as shown in Tables 1 and 2, the accuracy of the algorithm decreases as the number of possible schedules that satisfy the throughput constraint increases. The run-time of the algorithm for both the MRDF and the CSDF graph is in the order of 10^{-2} s. In our multi-processor system, often more than one FIFO buffer will be mapped on a particular memory. This results in a constraint on a sum of buffer capacities. Since the algorithm does not allow for constraints that include multiple buffer capacities, many iterations using this algorithm are required. The low run-time of a single iteration enables this approach.

	Buffer Capacity						rel.
	d_1		d_2		$d_1 + d_2$		inc.
	alg.	opt.	alg.	opt.	alg.	opt.	(%)
r_{SRC}	1376	960	882	882	2258	1842	23
$^3/_4 \cdot r_{SRC}$	1280	576	772	910	2052	1486	38
$^1/_2 \cdot r_{SRC}$	1152	480	662	661	1814	1141	59
$^1/_4 \cdot r_{SRC}$	1024	480	552	551	1576	1031	49

Table 2. Our results (alg.), the optimal results (opt.) for the CSDF model of the MP3 playback application, and the relative increase.

Even though we have been able to apply back-tracking in combination with MCM analysis for this example, we feel that, in general, this is not a feasible approach since one iteration of the Howard algorithm [4], which we used to derive the MCM, requires a minute. Application of the algorithm presented in Section 4.3 on the SRDF graph instead of the Howard or Bellman-Ford algorithm reduces the run-time by two orders of magnitude. Even though this is a significant improvement, we feel that the exponential memory and run-time, as caused by the conversion to a single-rate dataflow graph and by the back-tracking approach, remains problematic.

Govindarajan's [6] linear programming formulation can be used to determine minimal buffer capacities for the MRDF model of the MP3 playback implementation. However, application of this approach on the MP3 playback application was infeasible, because the solver from the GNU Linear Programming Kit (GLPK) runs out of memory. Removal of the APP task from the graph enables the application of Govindarajan's approach, but still has a run-time of half an hour. Since CSDF is more expressive than MRDF, we expect that extensions of Govindarajan's approach to include CSDF graphs will also have problematic run-times and memory requirements.

It seems possible to extend the work presented in [19] to make it applicable to CSDF graphs. This approach does not require the conversion to the corresponding SRDF graph in order to determine the MCM, and leverages the fact that a strongly connected and consistent MRDF graph enters a periodic regime after a transitional phase. Even though good experimental results are provided, no bound on the complexity is provided, and we know of no bound on the length of the transitional phase.

In this section, we have shown that a CSDF model can lead to reduced resource requirements compared to an MRDF model and that the presented algorithm can leverage the more detailed information.

5.2. Case-study

In this section we use our algorithm to derive buffer capacities that satisfy throughput and latency constraints of a car-radio application. In this car-radio application, see its block diagram in Figure 9, a phone call from a cell phone with Bluetooth (BT) is handled while playing music – at a lower volume – from an MP3 encoded song. Audio echo cancellation is used to prevent the howling effect and to cancel the sound from the loudspeakers, such that a signal for the Bluetooth device is obtained that only contains the speech of the user. The latency between the microphone and the BT device is required to be maximally 30 ms.

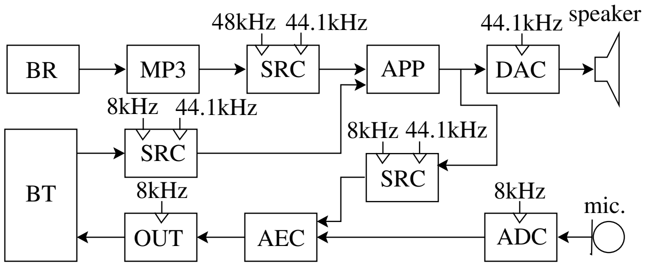


Figure 9. Audio echo cancellation in a car-radio application.

In this application, three streams are processed simultaneously. These streams are (1) a stream that decodes MP3 encoded songs as provided by a block reader (BR) task from a compact disk, (2) a stream from the Bluetooth device is converted into a 44.1 kHz signal and mixed with the stream from the MP3 decoder and sent to the loudspeakers, and (3) a stream originating from a microphone from which the sound from the loudspeakers is removed with an Audio Echo Cancellation (AEC) task. The stream from the MP3 decoder should be independent from the other streams, because it must be possible to start or stop one stream while continuing the other without interruption of the sound. For example, the speech signal should not be interrupted if the MP3 decoder task stops at the end of a song.

In our system [1], we execute the BR task on an ARM7 processor. Pre-Emptive arbitration is applied because non real-time system configuration software is executed on the same processor. The computational load generated by the APP task plus the OUT and SRC tasks requires one DSP, while the MP3 decoder and the AEC task execute on another DSP. Each sample rate converter is implemented with 2 tasks, and each SRC task runs in its own interrupt service routine that is activated with a fixed frequency. The MP3 decoder and the AEC task are round-robin scheduled because the applied DSPs do not support pre-emptive task switching. We apply round-robin scheduling instead of static order scheduling, because the MP3 decoder can stop at any moment and this should not prevent execution of the AEC task.

MP3 decoding and audio echo cancellation can be studied in isolation because round-robin arbitration is applied and the SRC tasks and the OUT task are executed strictly periodically. We will now derive buffer capacities, of the most interesting parts of this application, such that the throughput constraint for MP3 decoding and the throughput and latency for the audio echo cancellation are satisfied.

MP3 decoding The task graph with the MP3 decoding task is shown in Figure 10. The SRC task is executed every 1⁄48 ms. The worst-case response time of the BR task depends on the behavior of the compact disk player. We will assume a worst-case response time of 11 ms. The FIFO buffer $b_1 = (\mathrm{MP3}, \mathrm{SRC})$ has a capacity of m_1 containers and is initially empty. The FIFO buffer $b_2 = (\mathrm{MP3}, \mathrm{BR})$ has a capacity of two containers and is initially full.

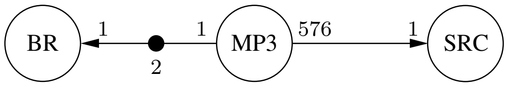


Figure 10. Task graph with the MP3 task.

The BR task stores in FIFO buffer $b_3 = (\mathrm{BR}, \mathrm{MP3})$ encoded MP3 data. The MP3 decoder task consumes each execution a variable number of bytes from this buffer. This buffer is not shown in Figure 10 because we ensure that it can be omitted during analysis, thereby allowing the MP3 decoder to be modelled in an MRDF graph. FIFO buffer b_3 can be omitted during analysis if this buffer does not influence the temporal behaviour of the BR and MP3 tasks, which requires that sufficient space and data is always available in buffer b_3. To achieve this, the MP3 task informs the BR how many bytes with encoded data it has consumed during its previous execution. This is achieved by sending a container to the BR task via buffer b_2. The value stored in this container is equal to the number of bytes consumed during the previous execution of the MP3 task. The BR task will write this number of bytes in buffer b_3. Initially two containers are stored in buffer b_2. The initial value in each container is equal to the maximum number of bytes that the MP3 task can consume per execution.

The corresponding MRDF graph with an MP3 actor is shown in Figure 11. The response time of the SRC actor and BR actor is 1⁄48 ms and 11 ms, respectively. Because round-robin arbitration is applied, the response time of the MP3 actor is equal to the sum of the WCET of the AEC and MP3 task plus $h = 1\,\mu\mathrm{s}$. With a WCET of the MP3 task that equals 4.090 ms and a WCET of the AEC task that equals 5.000 ms this results in a WCRT of the MP3 task of $5.000 + 4.090 + 0.001 = 9.091$ ms.

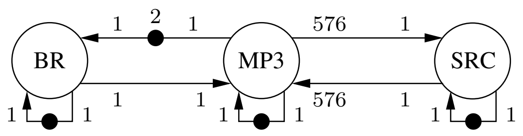


Figure 11. MRDF graph with the MP3 actor.

The repetition vector of the MRDF graph in Figure 11 is $[1\;1\;576]^{\mathrm{T}}$. The SRC task should be able to execute strictly periodically, which means that 576 firings of the SRC actor should complete in a period of $576 \cdot 1/48 = 12$ ms. Our algorithm determined that 1013 tokens are sufficient to satisfy this throughput requirement.

The MP3 task has a single code segment and not multiple code segments, because the increase in the number of code segments leads to an increase in the schedule uncertainty. In combination with RR arbitration this leads to response times that do not allow satisfaction of the throughput requirement. Since we have seen that reducing the synchronisation granularity leads to smaller buffer capacities, application of a different arbiter with a smaller schedule uncertainty is attractive and seen as future work. The MP3 decoder in this case study can be set to operate on frames or granules. In order to meet the throughput constraints we set the MP3 decoder to operate on granules instead of on frames, as done in Section 5.1.

Audio echo cancellation The task graph with the AEC task is shown in Figure 12. The SRC, the ADC and the OUT task are executed every 1/8 ms. The FIFOs have a capacity of n_1, n_2 and n_3 containers and are initially empty.

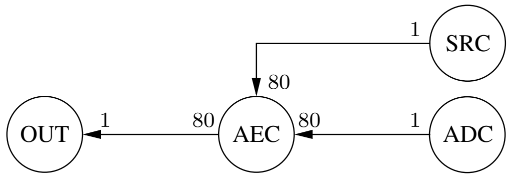


Figure 12. Task graph with the AEC task

The corresponding MRDF graph is shown in Figure 13. The response time of the AEC actor is equal to the sum of the WCET of the AEC and MP3 task plus $h = 1$ µs, i.e. $5.000 + 4.090 + 0.001 = 9.091$ ms. The response time of the SRC, the ADC and the OUT actor is 1/8 ms.

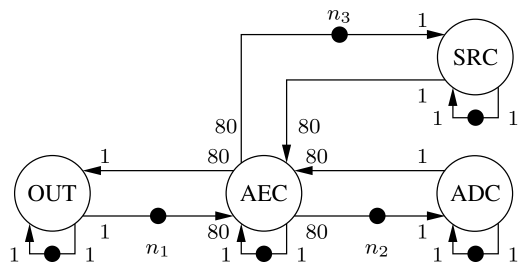


Figure 13. MRDF graph with the AEC actor

It is specified that the maximum latency between the ADC task and the OUT task is 30 ms. This latency includes the 6 ms algorithmic delay of the AEC task as caused by its 48 tap filter. The latency constraint between the OUT and ADC actor is $30 - 6 = 24$ ms, because the algorithmic delay is not modeled in the MRDF graph. The repetition vector of the MRDF graph in Figure 11 is $[80\ 1\ 80\ 80]^{\mathrm{T}}$. The OUT, SRC, and ADC tasks are required to execute strictly periodically, this means that 80 executions of these tasks are required within a period of $80 \cdot 1/8 = 10$ ms. If we assume that the SRC and ADC task start at the same time, then after adding the three constraints between the start times of the OUT, SRC, and ADC actors, our algorithm determines that each FIFO requires a capacity of 158 containers to satisfy these constraints, and that the OUT task should start 19.091 ms later than the SRC and ADC tasks.

6. Conclusion

In this work we have presented an algorithm that determines close to minimal buffer capacities for cyclo-static dataflow graphs such that throughput and latency requirements and constraints on maximum buffer capacities are satisfied. Because our algorithm does not require a conversion from a cyclo-static dataflow graph to a single-rate dataflow graph, we do not suffer from the exponential complexity associated with this conversion and obtain an algorithm with a low order polynomial complexity. Related approaches do make this conversion and have excessive run-times and memory requirements for realistic cyclo-static dataflow graphs.

We have further shown that the effects of run-time arbitration can be conservatively modelled in the response times of the dataflow actors. A real-life car-radio application involving time-division multiplex and round-robin arbitration, multiple independent streams and a combination of data driven and time driven tasks showed the applicability of the presented approach.

To complement the buffer capacity derivation, as e.g. done in the case study, we are currently setting up a mapping flow that determines both scheduler settings and the task to processor assignment. In order to derive a configuration that meets all constraints, we expect that this mapping flow will need to evaluate many different scheduler settings and task to processor assignments, for which the presented algorithm is an important contribution.

References

[1] M. J. G. Bekooij *et al.* *Dataflow Analysis for Real-Time Embedded Multiprocessor System Design*, chapter 15. Dynamic and Robust Streaming Between Connected CE Devices. Kluwer Academic Publishers, 2005.

[2] G. Bilsen *et al.* Cyclo-Static Dataflow. *IEEE Transactions on Signal Processing*, 44(2):397–408, February 1996.

[3] T. H. Cormen *et al.* *Introduction to Algorithms*. MIT press, 2001.

[4] A. Dasdan. Experimental Analysis of the Fastest Optimum Cycle Ratio and Mean Algorithms. *ACM Transactions on Design Automation of Embedded Systems*, 9(4):385–418, October 2004.

[5] S. Goddard and K. Jeffay. Managing Latency and Buffer Requirements in Processing Graph Chains. *The Computer Journal*, 44(6), 2001.

[6] R. Govindarajan *et al.* Minimizing Buffer Requirement under Rate-Optimal Schedules in Regular Dataflow Networks. *Journal of VLSI Signal Processing*, 31(3), 2002.

[7] M. Jersak *et al.* Context-Aware Performance Analysis for Efficient Embedded System Design. In *Proc. Design, Automation and Test in Europe (DATE)*, March 2004.

[8] M. Jersak *et al.* Performance Analysis of Complex Embedded Systems. *International Journal of Embedded Systems*, 1(1-2):33–49, 2005.

[9] V. Kianzad *et al.* An Architectural Level Design Methodology for Embedded Face Detection. In *Proc. Int'l Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, p. 136–141, September 2005.

[10] H. Kopetz. *Real-Time Systems: Design Principles for Distibuted Embedded Applications*. Kluwer Academic Publishers, 1997.

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

[12] A. Maxiaguine *et al.* Tuning SoC Platforms for Multimedia Processing: Identifying Limits and Tradeoffs. In *Proc. Int'l Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, September 2004.

[13] C. W. Mercer *et al.* Processor Capacity Reserves: Operating System Support for Multimedia Applications. In *Proc. IEEE Int'l Conference on Multimedia Computing and Systems*, p. 90–99, 1994.

[14] A. K. Mok and D. Chen. A Multiframe Model for Real-Time Tasks. *IEEE Transactions on Software Engineering*, 23(10):635–645, October 1997.

[15] T. M. Parks *et al.* A Comparison of Synchronous and Cyclo-Static Dataflow. In *Proc. IEEE Asilomar Conference on Signals, Systems and Computers*, p. 204–210, October 1995.

[16] J. L. Pino and E. A. Lee. Hierarchical Static Scheduling of Dataflow Graphs onto Multiple Processors. In *Proc. IEEE Int'l Conference on Acoustics, Speech, and Signal Processing*, May 1995.

[17] R. Reiter. Scheduling Parallel Computations. *Journal of the ACM*, 15(4):590–599, October 1968.

[18] S. Sriram and S.S. Bhattacharyya. *Embedded Multiprocessors: Scheduling and Synchronization*. Marcel Dekker Inc., 2000.

[19] S. Stuijk *et al.* Exploring Trade-Offs in Buffer Requirements and Throughput Constraints for Synchronous Dataflow Graphs. In *Proc. Design Automation Conference (DAC)*, p. 889–904, July 2006.

[20] M. H. Wiggers *et al.* Efficient Computation of Buffer Capacities for Multi-Rate Real-Time Systems with Back-Pressure. In *Proc. Int'l Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, October 2006.