Modelling Run-Time Arbitration by Latency-Rate Servers in Dataflow Graphs

Maarten Wiggers
University of Twente
Enschede, The Netherlands
m.h.wiggers@utwente.nl

Marco Bekooij
NXP Semiconductors
Eindhoven, The Netherlands
marco.bekooij@nxp.com

Gerard Smit
University of Twente
Enschede, The Netherlands
g.j.m.smit@utwente.nl

ABSTRACT

In order to obtain a cost-efficient solution, tasks share resources in a Multi-Processor System-on-Chip. In our architecture, shared resources are run-time scheduled. We show how the effects of Latency-Rate servers, which is a class of run-time schedulers, can be included in a dataflow model. The resulting dataflow model, which can have an arbitrary topology, enables us to provide guarantees on the temporal behaviour of the implementation.

Traditionally, the end-to-end behaviour of multiple Latency-Rate servers has been analysed with Latency-Rate analysis, which is a Network Calculus. This paper bridges a gap between Network Calculi and dataflow analysis techniques, since we show that a class of run-time schedulers can now be included in dataflow models, or, from a Network Calculus perspective, that restrictions on the topology of graphs that include run-time scheduling can be removed.

1. INTRODUCTION

Decreasing feature sizes have made it possible to implement multiple processing cores on a single chip, resulting in so-called Multi-Processor System-on-Chip (MPSoC) designs. These MPSoCs provide a high data processing throughput in a cost and energy-efficient way, making them an ideal match with multi-media applications found in TV-sets, set-top boxes, and smart-phones.

MPSoCs simultaneous processing of multiple streams of data. Each stream is processed by a job, where a job is started or stopped by an external event, e.g. by the end-user. Jobs often have temporal constraints, such as throughput and latency, and these temporal constraints can be firm or soft real-time constraints. In order to reduce costs, jobs share resources on the MPSoC, i.e. processors, interconnects and memories.

The functional behaviour of a job can often be intuitively described as a YAPI [8] task graph. YAPI task graphs are realisations of Kahn process networks [15]. It is shown in [2] that a YAPI task graph can be modelled as a deterministic Dynamic Dataflow (DDF) graph of which performance guarantees can be provided through simulation. Cyclo-Static Dataflow (CSDF) [5] is an important subclass of DDF of which performance guarantees can be provided, which are independent of the input data, by application of dataflow analysis techniques [21].

The tasks of a job are assigned to processors of the MPSoC, as e.g. shown in Figure 1. In this architecture, the inter-task communication FIFO buffers are placed in the local memory of the tile onto which the data consuming task is assigned. In this way, latency-sensitive read operations are kept local to the tile.

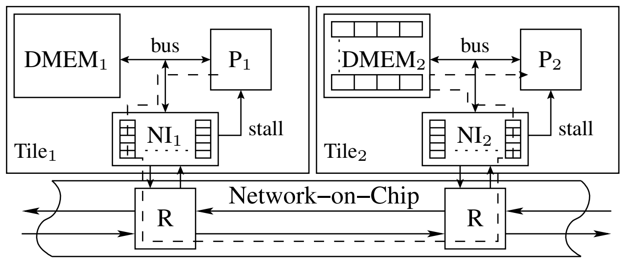


Figure 1: Example MPSoC architecture.

In this MPSoC there are many shared resources that each require a scheduler, e.g. the processors, memory ports, and links in the Network-on-Chip. In such an MPSoC, run-time scheduling is attractive for e.g. the following reasons, (1) a high resource utilisation can be obtained even in cases where there are tasks with a significant variation in their execution time and/or execution rate, (2) we do not need to compute and store a schedule for each combination of jobs that is simultaneously active, and (3) the transition from one combination of jobs to another is simplified.

Traditionally, dataflow analysis is applied when tasks are scheduled in a fully static or static order fashion [16, 21]. However, for the mentioned reasons, we apply run-time scheduling even in cases where the tasks all have the same fixed communication behaviour, i.e. they can be modelled with the most restrictive dataflow – Single-Rate Dataflow (SRDF) [20] – model.

The contribution of this paper is that we show that worst-case effects of a class of run-time schedulers can be modelled in a dataflow graph of arbitrary topology, and that through analysis of the resulting dataflow graph we can guarantee that the implementation satisfies its temporal constraints.

This is achieved by restricting the class of run-time schedulers to the class of Latency-Rate ($\mathcal{LR}$) servers [22]. $\mathcal{LR}$ servers were originally proposed as a modeling paradigm to model the effect of scheduling on traffic passing through a chain of heterogeneous routers in a packet-switched network in which connections provide quality of service guarantees. Many schedulers have been shown to be $\mathcal{LR}$ servers, e.g. round-robin and variants of round-robin [22],

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...$5.00.

and priority based schedulers that include a rate-controller [24].

We will show that the effects of scheduling by an $\mathcal{LR}$ server can be conservatively modelled with two vertices, which are called actors, in a dataflow graph. However, traditionally [2, 23] the proof that worst case temporal behaviour of the implementation can be determined by analysis of the dataflow graph is based on two conditions, (1) a one-to-one relation between the implementation and the dataflow graph, and (2) monotonic execution in time of the dataflow graph. In this paper we will relax the first condition, and show that the latter condition holds for a larger class of dataflow graphs than previously shown by Poplavko [19].

This paper is structured as follows. First in Section 2 we briefly present related work. Then in Section 3 we present the SRDF model that we use in subsequent sections to derive our initial results. We proceed in Section 4 by showing that single-rate dataflow graphs are temporally monotonic. In Section 5 we describe the relation between implementations and their dataflow models. After which, in Section 6 $\mathcal{LR}$ servers are introduced. This enables us to present an SRDF component, in Section 7, of which we show that this component has an input-output behaviour that is equivalent to the $\mathcal{LR}$ model of a task that is scheduled on an $\mathcal{LR}$ server. In Section 8 we show how the application of $\mathcal{LR}$ servers improves upon previous work. And in Section 9 we show that dataflow analysis leads to smaller buffer capacities than $\mathcal{LR}$ analysis. In Section 10 we generalise the results obtained using SRDF and show that these results remain valid for DDF. Proofs of the various results are placed in the appendix in order to focus on the main concepts.

2. RELATED WORK

Resource sharing has been included in dataflow models in [4, 2], however only Time Division Multiplex and Round Robin scheduling are considered, while the class of $\mathcal{LR}$ servers is broader. Furthermore, the models as proposed in [4, 2] are less accurate than the dataflow component that is presented in this paper. This is because the model in [4] models both the time between the enabling and finish of a task and the throughput by the response time of a single actor. In order to conservatively model the time between enabling and finish, the response time needs to be rounded up to an integer number of pre-emptions. We will show that this results in an overly pessimistic model of the throughput. In [2] a more accurate model is presented that includes a notion of execution sequences which is similar to a so-called busy period, which is one of the core concepts of $\mathcal{LR}$ servers. We will, however, show that application of the concept of busy periods leads to a more accurate model. Furthermore, the approach from [2] relies on simulation of the dataflow graph, while the approach presented in this paper does not require this.

In contrast with the presented approach, other performance analysis approaches that include run-time scheduling, as for instance presented by Jersak [13], Goddard [10], or Maxiaguine [17], do not allow feedback cycles that influence the temporal behaviour of the system. Not only do these cycles exist, because of functional constraints, they can also model that tasks only start their execution when sufficient space is available in the output buffers, i.e. back-pressure. Applying back-pressure has the advantage that the system does not require means to control the jitter, such as e.g. traffic shapers, in order to prevent buffer overflow.

The goal of the $\mathcal{LR}$ analysis in [22] is to be able to determine whether traffic that is injected into the network according to a specific traffic model arrives at its destination in time, and to determine sufficient buffer capacities such that no buffer overflow occurs. A limitation of $\mathcal{LR}$ analysis is that it cannot deal with cycles that influence the temporal behaviour, while it is well known that flow control, and in particular local flow control, can significantly reduce the required buffer capacities [1].

Goyal [11] and Hung [12] have presented approaches that are very similar to $\mathcal{LR}$ servers. These approaches are special cases of the service curve framework presented by Agrawal [1], which is a network calculus based on the work by Cruz [6, 7]. In [1] buffer capacities are derived such that cycles, which result from flow-control, in a chain of routers do not influence the temporal behaviour. In contrast with [1], we will derive the throughput of a graph, with an arbitrary topology, when buffer capacities are given, this includes the case that cycles in the graph influence the temporal behaviour.

3. SINGLE-RATE DATAFLOW

In this section we introduce Single-Rate Dataflow (SRDF) graphs [20]. An SRDF graph is a directed graph $G = (V, E, d, r)$ 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 $d : E \rightarrow \mathbb{N}$. An actor is enabled to fire when a token is available on each input edge. The response time $r(v_i)$, $r : V \rightarrow \mathbb{R}$, is the difference between the finish and start time of actor v_i. When actor v_i finishes, then it produces a token on each output edge in one atomic action.

For a strongly connected SRDF graph, we can derive the period μ of the graph through Maximum Cycle Mean (MCM) analysis [20, 21]. To determine the MCM, the maximum of the cycle means of all simple cycles in the SRDF graph needs to be determined, where the cycle mean $\mu(c)$ of a cycle c is

$$\mu(c) = \frac{\sum_{v_i \in V(c)} r(v_i)}{\sum_{e \in E(c)} d(e)} \tag{1}$$

where $V(c)$ is the set of actors traversed by cycle c and $E(c)$ is the set of edges traversed by cycle c. A simple cycle is a cycle that traverses actors maximally once. The MCM of an SRDF graph G is therefore

$$\mu(G) = \max_{c \in C(G)} \mu(c) \tag{2}$$

where $C(G)$ is the set of simple cycles of SRDF graph G. The maximal attainable throughput of the graph relates to μ^{-1}. In case the run-time of the MCM analysis is problematic, a conservative approximation technique [23] can be applied.

4. MONOTONIC EXECUTION

In this section we will first define temporal monotonicity of an SRDF graph, and then extend [19] by proving that every SRDF graph is temporally monotonic. Since existing analysis techniques [3, 2, 23] rely on temporal monotonicity, in order to guarantee the temporal behaviour of the implementation, this increases the applicability of these techniques.

This is an extension of [19], which states that an SRDF graph is monotonic in the response times, if the SRDF graphs maintains a first-in first-out (FIFO) ordering of tokens. An SRDF graph maintains a FIFO ordering of tokens if each actor either has a constant response time or a self-edge with one initial token [9]. This is because the queues by definition maintain a FIFO ordering of tokens. Temporal monotonicity is defined as follows.

DEFINITION 1. *An SRDF graph is temporally monotonic if (1) no decrease in response time, (2) increase in number of initial tokens, or (3) decrease in the difference between actor enabling time and actor start time leads to a later actor start time.*

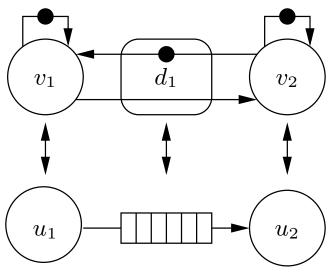


Figure 2: Example of the traditional one-to-one relation between dataflow graph and task graph.

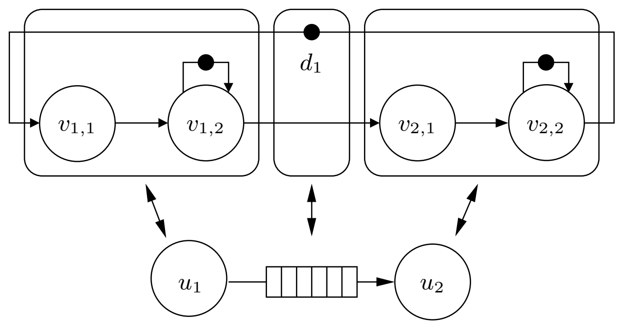


Figure 3: Example one-to-one relation between component graph and task graph.

No proof was provided in [19] to show that an SRDF graph is monotonic in the response times, only the observation that the evolution equations as given in Chapter 2 of [9] are monotonic in the response times. These evolution equations only hold for SRDF graphs that maintain a FIFO ordering of tokens. This is because in the evolution equations the start time of the k'th firing of actor v_j depends on firings $k - m, 0 \leq m \leq M$ of actors v_i, where M is the maximum number of initial tokens on any edge. If FIFO ordering is not maintained, then firing k of actor v_j is enabled by firing q of actor v_k, where q can be larger than, smaller than, or equal to k.

We, however, present the following more general theorem. This theorem does not require FIFO ordering of tokens to be maintained in the graph, since the response times do not depend on the tokens that lead to enabling.

THEOREM 1. *An SRDF graph is temporally monotonic.*

In Appendix A we have included the proof of Theorem 1, which shows that an SRDF graph is also temporally monotonic without the constraint that FIFO ordering is maintained. This is one of the key contributions of this paper.

5. ANALYSIS MODEL AND IMPLEMENTATION

While traditionally [3, 2, 23] every task is modelled by one actor, as e.g. in Figure 2, we will in later sections like to model a task that is scheduled on an $\mathcal{LR}$ server with two actors. However, since in that case there is no longer a one-to-one relation between the dataflow graph and the implementation, we can no longer use the traditional argumentation to obtain the result that worst-case temporal behaviour of the implementation can be derived from the dataflow model. In this section, we will show that a correspondence between model and implementation and a sufficient condition exists such that worst-case temporal behaviour of the implementation can still be computed with dataflow analysis techniques.

We assume that the application is implemented as a weakly connected directed task graph G_A, of which the vertices represent tasks and the edges represent FIFO buffers with a fixed capacity. A weakly connected graph is a graph in which for every pair of vertices a and b a path exists from a to b and/or from b to a. Tasks only communicate fixed sized containers over FIFO buffers, where a container can be full or empty. A task produces one container on each output FIFO and consumes one container from each input FIFO in every execution. Furthermore, the execution of a task only starts when a full container is present on every input FIFO buffer and an empty container is present on every output FIFO buffer. The finish time of each task is at most the worst case response time later than the enabling time. And further at most one instance of a task can execute at any time.

We define I_x to be the set of input FIFO buffers and O_x to be the set of output FIFO buffers both of a task u_x. We further define $a_A(m, j)$ to be the arrival time of the j-th container in the input FIFO $m \in I_x$ and $a_A(n, j), n \in O_x$ to be the arrival time of the j-th container in the output FIFO n both of a task u_x in the implementation.

We will now describe a sufficient condition for the SRDF graph such that this SRDF graph can be used to derive worst-case container arrival times in the implementation. Let us assume an SRDF graph G_M. We define C as a partitioning of the set of actors V, i.e. $\forall C_x, C_y \in C, C_x \neq C_y \Rightarrow C_x \cap C_y = \emptyset, V = \{C_i | C_i \in C\}$. Each element in C is called a component. Components consume tokens from component input queues and produce tokens on component output queues. A component input queue of component C_x is an edge (v_i, v_j) in the SRDF graph that has an actor $v_j \in C_x$ as destination and an actor $v_i \notin C_x$ as its source. A component output queue of component C_x is an edge (v_i, v_j) in the SRDF graph that has an actor $v_i \in C_x$ as its source and an actor $v_j \notin C_x$ as its destination.

The component graph needs to match with the task graph, i.e. there should be a one-to-one correspondence between tasks and components, and between bounded FIFO buffers in the task graph and pairs of edges in the component graph, as e.g. in Figure 3. With a slight abuse of notation, we define I_x as the set of input queues of component C_x, and O_x as the set of output queues of C_x.

We define $a_M(m, j)$ to be the arrival time of the j-th token in the component input queue $m \in I_x$ and $a_M(n, j), n \in O_x$ to be the arrival time of the j-th token in the component output queue n both of a component C_x in the SRDF graph.

DEFINITION 2. *The token production and consumption behaviour of component C_x is conservative with respect to the container production and consumption behaviour of task u_x, if the following holds*

$$\begin{aligned}\forall m \in I_x, a_A(m, j) \leq a_M(m, j) \Rightarrow \\ \forall n \in O_x, a_A(n, j) \leq a_M(n, j)\end{aligned} \tag{3}$$

The condition that needs to hold for the SRDF graph such that worst-case container arrival times can be derived with the SRDF graph is the following.

THEOREM 2. *If every component C_x is conservative with respect to task u_x, i.e. Equation 3 holds, then the worst-case arrival times of containers in the output FIFO buffers of every task u_x can be computed with dataflow analysis techniques.*

A proof of this theorem is provided in Appendix B. An implication of Theorem 2 is that the execution of one task in the system can be represented by a component which can be a collection of dataflow actors as long as the component is conservative with respect to the task. In the next sections, we will use this property to model tasks that are scheduled using an $\mathcal{LR}$ server with a component that consists of two dataflow actors, similar to Figure 3, of which we will show that this component conservatively models the execution of a task on an $\mathcal{LR}$ server. In the traditional approach as for example shown in Figure 2 this relationship between model and implementation was not possible.

6. LATENCY-RATE SERVERS

In this section we introduce $\mathcal{LR}$ servers from a dataflow perspective. Let $b(u_x, i), b : U \times \mathbb{N} \to \mathbb{R}$, with U the set of tasks, be the time at which sufficient containers are available on all input FIFOs, such that execution i of task u_x can start, and say that execution i of task u_x is externally enabled at $b(u_x, i)$. A task is only enabled if also its previous execution has finished. We define $A_x(s,t), A : \mathbb{R} \times \mathbb{R} \to \mathbb{N}$, as the number of external enablings of task u_x within the interval $(s,t]$. Further, let $f(u_x, i), f : U \times \mathbb{N} \to \mathbb{R}$, be the finish time of execution i of task u_x. And let $W_x(s,t), W : \mathbb{R} \times \mathbb{R} \to \mathbb{N}$ be the number of finishes of task u_x in the interval $(s,t]$.

A busy period is defined in [22] as a maximum interval of time $(s,t]$ for which Equation (4) holds. This means that at any time τ in a busy period the number of external enablings since the start of the busy period, $A(s,\tau)$, is at least equal to the number of finishes at the allocated rate $\rho_x \in \mathbb{R}$, as provided by the bound $H(s,\tau)$.

$$\forall \tau \in (s,t] : A_x(s,\tau) \geq H(s,\tau) = \rho_x \cdot (\tau - s) \tag{4}$$

According to [22], if and only if we can show for a scheduler that a nonnegative constant $L_x \in \mathbb{R}$ can be found such that Equation (5) holds, then this scheduler is an $\mathcal{LR}$ server for task u_x with rate ρ_x. Equation (5) requires that the number of finishes from s to τ, in a busy period $(s,t]$, is bounded by $Q(s,\tau)$, which is an expression in terms of a latency L_x and a rate ρ_x. The value Θ_x is the minimum L_x such that Equation (5) holds, and is called the latency of the scheduler.

$$\forall \tau \in (s,t] : W_x(s,\tau) \geq Q(s,\tau) = \max(0, \rho_x \cdot (\tau - s - L_x)) \tag{5}$$

Since the number of finishes is an integer, $W_x(s,t) \in \mathbb{N}$, we obtain a tighter lower bound $\breve{Q}$ on $W_x(s,t)$, by enforcing that $\breve{Q} \in \mathbb{N}$. This is done by taking the floor of Q and we therefore require that a nonnegative constant L_x can be found such that Equation (6) holds. And again, the minimum L_x, for which Equation (5) holds, is defined to be the latency Θ_x of the scheduler for task u_x.

$$\forall \tau \in (s,t] : W_x(s,\tau) \geq \breve{Q}(s,\tau) = \max(0, \lfloor \rho_x \cdot (\tau - s - L_x) \rfloor) \tag{6}$$

Figure 4 illustrates the relation between the number of external enablings and the number of finishes. A discussion on the differences between these definitions and the definitions in [22] is included in Appendix C.

7. $\mathcal{LR}$ SERVERS IN SRDF

In this section we present an SRDF component that conservatively models a task that executes on an $\mathcal{LR}$ server by showing that the minimum number of token releases by this dataflow construct in an interval $(s,t]$ equals the minimum number of finishes, and therefore the minimum number of token releases, in the interval $(s,t]$ as defined in Equation (6).

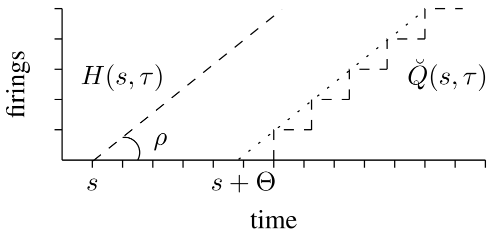


Figure 4: The lines starting at s and $s + \Theta$ both have a slope ρ. The number of external enablings should be at least $H(s,\tau)$ and the number of finishes should be at least $\breve{Q}(s,\tau)$.

We will first show that bounding the finish time of an execution in a busy period with respect to the start of the busy period, as expressed by Equation (7), is equivalent to bounding the number of finishes over a time interval that starts from the start of the busy period, as expressed by Equation (6). Subsequently we show that bounding the finish time of an execution, as expressed by Equation (7), is equivalent to bounding the finish time of an execution in a busy period with respect to either its external enabling time or the finish time of the previous execution, as expressed by Equation (9). Finally we arrive at the main result of this paper, which is that Equation (9) is equivalent to an SRDF component that models a task with one input and one output FIFO that is scheduled on an $\mathcal{LR}$ server.

In the following we will only consider a task u_x and we therefore use the shorthand notation $b(j)$ and $f(j)$ to mean the external enabling time of execution j of task u_x and the finish time of task u_x.

LEMMA 1. *Let execution k be the first execution in a busy period $(s,t]$ and let execution j occur in the same busy period, that is $b(k) = s$ and $j \geq k$ and $b(j) \leq t$. If the scheduler is an $\mathcal{LR}$ server with latency Θ and rate ρ, then the bound τ_j on the finish time of execution j as provided by Equation* (7) *is equivalent to the bound as provided by Equation* (6).

$$f(j) \leq \tau_j = b(k) + \Theta + \frac{j-k+1}{\rho} \tag{7}$$

See Appendix D for a proof of Lemma 1.

LEMMA 2. *Let execution k be the first execution in a busy period $(s,t]$ and let execution j occur in the same busy period, that is $b(k) = s$ and $j \geq k$ and $b(j) \leq t$. If the scheduler is an $\mathcal{LR}$ server with rate ρ, then the bound ϕ_j on the external enabling time of execution j as in Equation* (8) *is equivalent to bounding the number of externally enabled executions as in Equation* (4) *in every busy period.*

$$b(j) \leq \phi_j = b(k) + \frac{j-k}{\rho} \tag{8}$$

See Appendix E for a proof of Lemma 2.

LEMMA 3. *The bound on the finish time of execution j, τ_j, as defined by Equation* (7) *is equivalent to the bound $g(j)$ as defined by Equation* (9).

$$g(j) = \begin{cases} \max(b(j) + \Theta, g(j-1)) + \frac{1}{\rho} & \text{if } j > 0 \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

See Appendix F for a proof of Lemma 3. A transformation from Equation (5), which assumes continuous service, to an Equation similar to Equation (9), has been provided in [14]. Since we have discretised the service, we obtain a different expression for the latency of the server. We will now present the SRDF component that is equivalent to a $\mathcal{LR}$ model of a task that is scheduled on an $\mathcal{LR}$ server. This is the main contribution of this paper.

THEOREM 3. *The dataflow construct shown in Figure 5 models a task u_x with one input FIFO and one output FIFO that executes on an $\mathcal{LR}$ server with latency Θ_x and allocated rate ρ_x.*

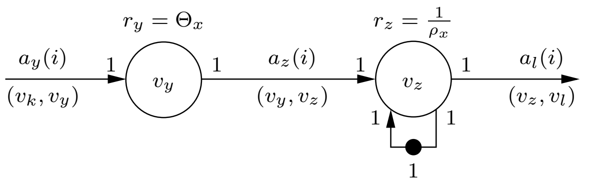


Figure 5: A dataflow component that models a $\mathcal{LR}$ server.

See Appendix G for a proof of this theorem.

8. EXAMPLE

In this section we will show that TDM scheduling is an $\mathcal{LR}$ server, and that this insight leads to a more accurate analysis than previously applied in [4]. This is because the model in [4] models both the time between enabling and finish of the task and the throughput by the response time of a single actor. In order to conservatively model the time between enabling and finish of the task, the response time needs to be rounded up to an integer number of pre-emptions. We will show that this leads to an overly pessimistic model of the throughput.

8.1 TDM is an $\mathcal{LR}$ server

Let P be the TDM period, S_x be the time slice allocated to task u_x, $D_{x,i}$ be the execution time of execution i of task u_x, and $\overline{D}_x$ be the worst-case execution time of task u_x. We will derive in this section an expression for the latency and rate of a TDM scheduler in terms of the period, time-slice, and worst-case execution time, such that Equation (5) holds.

The difference between subsequent task finishes is $f(u_x, i) - f(u_x, i-1) = D_{x,i}\frac{P}{S_x}$ in a busy period. This is because, in a busy period, without resource sharing execution i of task u_x will finish $D_{x,i}$ time later than the finish of firing $i-1$. However, with TDM scheduling we have that in every period P, there is only a time interval of length S_x time allocated to task u_x.

The guaranteed rate ρ_x at which task u_x finishes in a busy period with TDM scheduling is given by Equation (10). This is because $\forall i \in \mathbb{N} : f(u_x, i) - f(u_x, i-1) \leq \overline{D}_x \frac{P}{S_x}$, with $\overline{D}_x$ the worst case execution time of u_x. The guaranteed rate ρ_x is therefore $\frac{1}{\overline{D}_x \frac{P}{S_x}}$ which can be rewritten to obtain Equation (10).

$$\rho_x = \frac{1}{\overline{D}_x}\frac{S_x}{P} \tag{10}$$

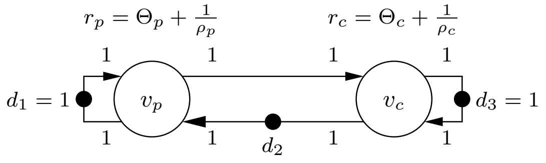

$r_p = \Theta_p + \frac{1}{\rho_p}$
$r_c = \Theta_c + \frac{1}{\rho_c}$
$d_1 = 1$
v_p
v_c
$d_3 = 1$
d_2
1
1
1
1
1
1
1
1

Figure 6: Previous inclusion of TDM arbitration.

To derive the latency Θ_x we use a result from [4] to obtain that an exact upper bound on the response time of the first execution in a busy period is given by Equation (11).

$$\overline{r}_x = \overline{D}_x + (P - S_x)\left\lceil \frac{\overline{D}_x}{S_x} \right\rceil \tag{11}$$

We know that the response time of the first execution in a busy period is smaller than or equal to $\Theta_x + \frac{1}{\rho_x}$, and that this is the tightest bound. Therefore it should hold that

$$\Theta_x + \frac{1}{\rho_x} = \overline{D}_x + (P - S_x)\left\lceil \frac{\overline{D}_x}{S_x} \right\rceil \tag{12}$$

We will now rewrite $\frac{1}{\rho_x}$ to obtain an expression for the latency in terms of the period, time-slice and worst-case execution time. First we rewrite Equation (10) to obtain

$$\frac{1}{\rho_x} = P\frac{\overline{D}_x}{S_x} \tag{13}$$

which we can rewrite into

$$\frac{1}{\rho_x} = (P - S_x)\frac{\overline{D}_x}{S_x} + \overline{D}_x \tag{14}$$

After combining Equation (14) with Equation (12), we obtain

$$\Theta_x = (P - S_x)(\left\lceil \frac{\overline{D}_x}{S_x} \right\rceil - \frac{\overline{D}_x}{S_x}) \tag{15}$$

Equation (15) shows that an L_x exists such that Equation (6) holds, which means that TDM is an $\mathcal{LR}$ server.

8.2 Improved dataflow analysis results

Figure 6 shows how the effects of TDM arbitration are included with the approach from [4], while Figure 7 shows how the effects of TDM arbitration are included with the approach presented in this paper.

The MCM of the SRDF graph in Figure 6 is

$$\mu_a = \max(\{\Theta_p + \frac{1}{\rho_p}, \frac{\Theta_p + \frac{1}{\rho_p} + \Theta_c + \frac{1}{\rho_c}}{d_2}, \Theta_c + \frac{1}{\rho_c}\}) \tag{16}$$

And the MCM of the SRDF graph in Figure 7 is

$$\mu_b = \max(\{\frac{1}{\rho_p}, \frac{\Theta_p + \frac{1}{\rho_p} + \Theta_c + \frac{1}{\rho_c}}{d_2}, \frac{1}{\rho_c}\}) \tag{17}$$

Since by definition Θ_p and Θ_c are non-negative, we have that $\mu_a \geq \mu_b$. The more accurate model, as shown in Figure 7, can thus lead to a lower MCM, which corresponds to a higher throughput, and can therefore guarantee the satisfaction of more stringent throughput constraints for the same resource requirements in comparison with the model proposed in [4].

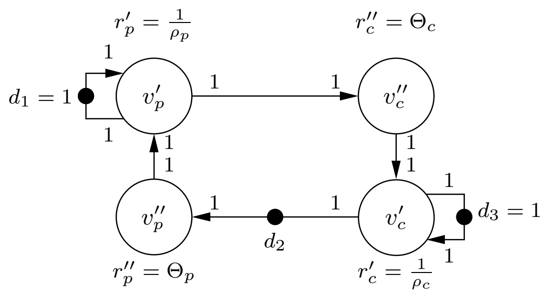


Figure 7: Inclusion of TDM arbitration using an $\mathcal{LR}$ model.

9. DATAFLOW ANALYSIS COMPARED WITH $\mathcal{LR}$ ANALYSIS

In this section we will show an example in which an implementation where tasks only execute when there is output space available, as our applications are implemented, leads to smaller buffer capacities than an implementation in which tasks execute as soon as input data is available as the $\mathcal{LR}$ analysis model assumes.

Figure 8(a) shows the $\mathcal{LR}$ analysis model and an SRDF model of a chain of three tasks that operate on a single stream. Let us assume that inputs arrive strictly periodic, e.g. from an Analog-to-Digital Converter, which is, for the sake of simplicity, left out of this example. In the $\mathcal{LR}$ analysis model such a stream is modelled with a (σ, ρ) model, where σ equals the maximum burst size and ρ equals the rate. Let us assume that $\sigma = 1$ and $\rho = 1$ accurately model the input stream. Let us further assume that the worst case execution time $\overline{D}_x$ of each task u_x is 1.

In the dataflow model as shown in Figure 8(b), we need to guarantee that the throughput of the graph equals the throughput of the input stream. Since $\rho = 1$, which means that one token arrives per time unit, it is required that for the MCM of the graph $\mu(G)$ it needs to hold that $\mu(G) \leq 1$. If we further assume that $\Theta_i^{DF} = \rho_i^{DF} = 1$, for $i = 1, 2, 3$, then it can be verified that $d_1 = 4$ and $d_2 = 4$ are sufficient buffer capacities to let the SRDF graph have an MCM of 1.

In the $\mathcal{LR}$ analysis model, the maximum number of tokens in the input queue of the k'th server equals $\sigma + \rho^{LR} \sum_{j=1}^{k} \Theta_j^{LR}$. This expression is derived using the bound $Q(s, t)$, which therefore means that $\Theta^{LR} = \Theta^{DF} + \frac{1}{\rho^{LR}}$, with $\rho_i^{LR} = \rho_i^{DF} \overline{D}_i$ this means that $\Theta_i^{LR} = 2$ for $i = 1, 2, 3$. The maximum number of tokens in the input queue of the server s_2 is therefore $d_1 = 1 + 1 * 4 = 5$ and the maximum number of tokens in the input queue of the server s_3 is therefore $d_2 = 1 + 1 * 6 = 7$. However, the $\mathcal{LR}$ analysis assumes that no space in the output buffer is required until the task has produced a token. Therefore, we need to increase each buffer capacity with one token. This results in $d_1 = 6$ and $d_2 = 8$.

From the expression for the maximum backlog it becomes clear that, in the $\mathcal{LR}$ analysis, the required buffer capacity scales with the length of the chain. This does not occur in the dataflow analysis. In the example the buffer capacities derived using $\mathcal{LR}$ analysis are larger than the buffer capacities as derived using dataflow analysis.

An important reason for the difference in buffer capacities is that our implementations include a local flow-control mechanism, because tasks only start when sufficient space is available in their output FIFOs, i.e. if a task rapidly produces a burst of containers it will eventually be slowed down because there will be no more empty containers left. $\mathcal{LR}$ analysis cannot model this behaviour, because this is a cyclic dependency between two tasks that can influence the temporal behaviour. We note that in this case we can apply the analysis based on Network Calculus from [1] to come to the same results as the dataflow analysis. However, this is only because this is a chain of servers, and the cycles due to local flow-control do not determine the throughput.

$\mathcal{LR}$ analysis cannot leverage the fact that traffic shapers might be part of the implementation, as e.g. a (σ, ρ)-regulator [6] or a rate- [25] or credit- [18] based scheduler. This is in contrast with [1] where application of traffic shapers can lead to reduced buffer capacities, since traffic shapers limit the burstiness of traffic. In dataflow analysis, traffic shapers are not required to reduce buffer capacities, since the local flow control mechanism as mentioned in the previous paragraph already controls the burstiness. However, traffic shapers are required to enable the local analysis of priority based schedulers.

10. GENERALISATION

In this section we will generalise the results obtained until now to include tasks with multiple input and output FIFO buffers and with less constrained container consumption and production behaviour than the behaviour as defined in Section 5. We will generalise two results, (1) we will show that not only an SRDF graph is temporally monotonic but also a deterministic Dynamic Dataflow graph (DDF), and (2) we will show how to model a task with multiple input and output FIFO buffers that consumes and produces multiple containers per execution and is executed on an $\mathcal{LR}$ server.

This means that jobs, which can be modelled as CSDF graphs and execute on resources that are scheduled with $\mathcal{LR}$ servers, can be analysed using traditional dataflow analysis techniques [5, 21]. Jobs that cannot be modelled as CSDF graphs and need to be modelled as DDF graphs, because of data dependent consumption and production of containers, can be analysed through simulation of the DDF graph [2].

10.1 Monotonicity

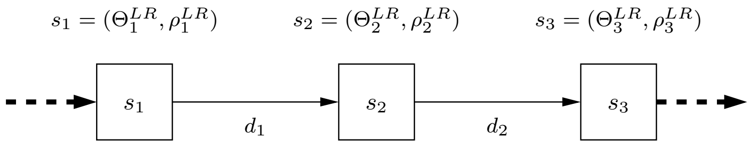
We can generalise the result from Theorem 1 to conclude that deterministic DDF graphs as defined in [2] are temporally monotonic. The start time of a firing of a DDF actor depends on the token production times on specific edges, and therefore on finishes of specific firings of specific DDF actors. For every firing of a DDF actor v_x this can be formulated like Equations (18) and (19), where however the set of actors that determines the enabling time of v_x can change from firing to firing. The proof of Theorem 1 does not depend on a fixed set of actors that determines the enabling time and is therefore also applicable to DDF graphs. This means that deterministic DDF graphs are temporally monotonic.

Also for a DDF graph we can define a component graph that partitions the DDF graph. If there is a one-to-one relation between these components and the task graph, and each component is conservative with respect to the corresponding task, then since DDF graphs are temporally monotonic we have that token arrival times in the DDF graph are worst-case container arrival times in the implementation.

10.2 Execution on $\mathcal{LR}$ servers

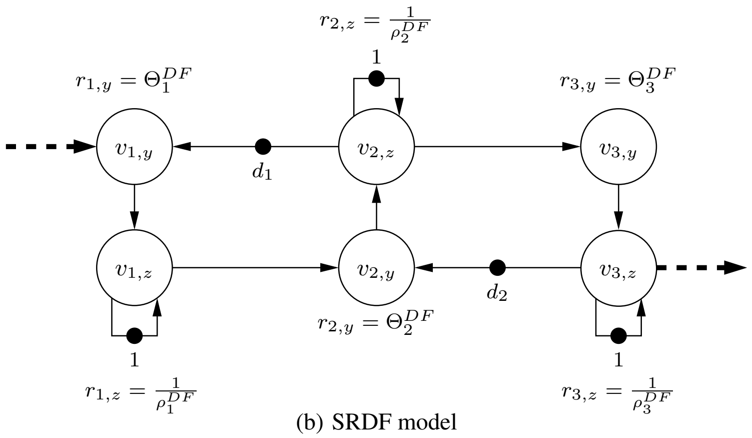
The result of Theorem 3 can be generalised to arrive at the conclusion that any task, which can be modelled with a DDF actor and executes on an $\mathcal{LR}$ server, can be modelled as a DDF component.

This is because the result of Theorem 3 deals with the relation between the external enabling time and the finish time, and not with the enabling rule and finish rule. The condition that determines the external enabling of the task therefore becomes the condition that determines the enabling of the latency actor, i.e. actor v_y in Figure 5. Further when the task finishes it produces containers on



$s_1 = (\Theta_1^{LR}, \rho_1^{LR})$
$s_2 = (\Theta_2^{LR}, \rho_2^{LR})$
$s_3 = (\Theta_3^{LR}, \rho_3^{LR})$
s_1
s_2
s_3
d_1
d_2

(a) $\mathcal{LR}$ analysis model



$r_{2,z} = \frac{1}{\rho_2^{DF}}$
1
$r_{1,y} = \Theta_1^{DF}$
$r_{3,y} = \Theta_3^{DF}$
$v_{1,y}$
$v_{2,z}$
$v_{3,y}$
d_1
$v_{1,z}$
$v_{2,y}$
$v_{3,z}$
d_2
$r_{2,y} = \Theta_2^{DF}$
1
1
$r_{1,z} = \frac{1}{\rho_1^{DF}}$
$r_{3,z} = \frac{1}{\rho_3^{DF}}$

(b) SRDF model

Figure 8: A chain of tasks that operates on a stream, in which all tasks are scheduled on $\mathcal{LR}$ servers, the dashed arrows do not belong to the models, but show the input and output streams.

various output FIFO buffers. Since the finish time is modelled by the rate actor, i.e. actor v_z in Figure 5, this actor will now produce tokens on the corresponding queues.

Also the worst case execution time of different firings of a dataflow actor can be different. Since with our definitions Θ_x and ρ_x are dependent on the worst case execution time, this can be modelled by parameterising Θ_x and ρ_x, resulting in $\Theta_x(i)$ and $\rho_x(i)$ in order to make them dependent on the worst case execution time of firing i.

11. CONCLUSION

This paper describe a relation between concepts from the Network Calculus and dataflow domains. We have shown that Latency-Rate servers, which is a class of run-time schedulers, can be included in a dataflow model by an actor that models the rate and an actor that models the latency. The resulting dataflow model is shown to be monotonic in time, which enabled us to prove that worst case temporal behaviour of the implementation can be observed in the dataflow model.

Currently, we are setting up a design flow that determines a task to processor binding, scheduler settings, and buffer capacities such that the application's temporal constraints are satisfied. We feel that this work is an important contribution to such a design flow that likely needs to deal with a heterogeneous set of run-time schedulers.

12. REFERENCES

[1] R. Agrawal. Performance Bounds for Flow Control Protocols. *IEEE/ACM Transactions on Networking*, 7(3):310–323, June 1999.

[2] M. J. G. Bekooij, S. Parmar, and J. van Meerbergen. Performance Guarantees by Simulation of Process Networks. In *Proc. Int'l Workshop on Software and Compilers for Embedded Systems (SCOPES)*, 2005.

[3] M. J. G. Bekooij *et al.* Predictable Embedded Multi-Processor System Design. In *Proc. Int'l Workshop on Software and Compilers for Embedded Systems (SCOPES)*, 2004.

[4] 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.

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

[6] R. L. Cruz. A Calculus for Network Delay, Part I: Network Elements in Isolation. *IEEE Transactions on Information Theory*, 37(1):114–131, January 1991.

[7] R. L. Cruz. A Calculus for Network Delay, Part II: Network Analysis. *IEEE Transactions on Information Theory*, 37(1):132–141, January 1991.

[8] E. A. de Kock. YAPI: Application Modeling for Signal Processing Systems. In *Proc. Design Automation Conference (DAC)*, 2000.

[9] F. Bacelli, G. Cohen, G.J. Olsder, and J-P. Quadrat. *Synchronization and Linearity: An Algebra for Discrete Event Systems*. Wiley, 1992.

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

[11] P. Goyal *et al.* Determining End-to-End Delay Bounds in Heterogeneous Networks. *Multimedia Systems*, 5:157–163, 1997.

[12] A. Hung and G. Kesidis. Bandwidth Scheduling for Wide-Area ATM Networks Using Virtual Finishing Times. *IEEE/ACM Transactions on Networking*, 4(1):49–54, February 1996.

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

[14] Y. Jiang. Relationship between Guaranteed Rate Server and Latency Rate Server. *Computer Networks*, 43(3):307–315, October 2003.

[15] G. Kahn. The Semantics of a Simple Language for Parallel Programming. In *Information Processing*, Stockholm, August 1974.

[16] E. A. Lee and S. Ha. Scheduling Strategies for Multi-Processor Real-Time DSP. In *Proc. IEEE Global Telecommunications Conference and Exhibition (GLOBECOM)*, November 1989.

[17] 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.

[18] C. Otero Pérez *et al. Dynamic and Robust Streaming between Connected CE Devices*, chapter Resource Reservations in Shared Memory Multiprocessor SoCs. Kluwer Academic Publishers, 2005.

[19] P. Poplavko *et al.* Task-Level Timing Models for Guaranteed Performance in Multiprocessor Networks-on-Chip. In *Proc. Int'l Conference on Compilers, Architectures, and Synthesis of Embedded Systems (CASES)*, November 2003.

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

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

[22] D. Stiliadis and A. Varma. Latency-Rate Servers: A General Model for Analysis of Traffic Scheduling Algorithms. *IEEE/ACM Transactions on Networking*, 6(5):611–624, October 1998.

[23] 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.

[24] H. Zhang. Service Disciplines for Guaranteed Performance Service in Packet-Switching Networks. *Proceedings of the IEEE*, 83(10):1374–1396, October 1995.

[25] H. Zhang and D. Ferrari. Rate-Controlled Service Disciplines. *Journal of High-Speed Networks*, 3(4), 1994.

APPENDIX

A.

PROOF OF THEOREM 1. The enabling time of firing k of actor v_i, $l(v_i, k)$, is determined by

$$l(v_i, k) = \max(\{s(v_x, k') + r(v_x, k')|(x, i) \in E\}) \quad (18)$$

where $s(v_x, k')$ is the start time of actor v_x and $r(v_x, k')$ is the response time of actor v_x of some firing k'. The start time of firing k of actor v_i is therefore given by

$$s(v_i, k) = l(v_i, k) + \epsilon(v_i, k) \quad (19)$$

where $\epsilon(v_i, k) \geq 0$ is the difference between the enabling time and the start time of firing k of actor v_i.

Since $\forall b : a' \leq a \Rightarrow \max(a', b) \leq \max(a, b)$, a decrease of the start times or response times of firing k' of actor v_x, which is a predecessor of v_i, cannot lead to an increase of the enabling time of firing k of an actor v_i. Since the SRDF graph does not maintain FIFO ordering of tokens, it can occur that a decrease of the start time or response times of firing k'' of an actor v_x, which is a predecessor of v_i, determines the enabling time of v_x in iteration k. This can only occur if firing k'' of v_x produces earlier than firing k' and thus $s(v_x, k'') + r(v_x, k'') < s(v_x, k') + r(v_x, k')$. This situation can therefore not lead to a later enabling time of firing k of actor v_i, and therefore also not to a later start time of firing k of actor v_i.

If the number of initial tokens $d((v_x, v_i))$ is increased of some edge (v_x, v_i) to become $d'((v_x, v_i))$, with $d'((v_x, v_i)) > d((v_x, v_i))$, then the enabling time of firing k of actor v_i becomes dependent on the start time and response time of firing $\widehat{k}, \widehat{k} \neq k'$ of actor v_x, with $s(v_x, \widehat{k}) + r(v_x, \widehat{k}) \leq s(v_x, k) + r(v_x, k)$. This is because firing k is no longer dependent on the $k - d((v_x, v_i))$'th token to arrive on edge (v_x, v_i) but on the $k - d'((v_x, v_i))$'th token to arrive. Since $d'((v_x, v_i)) > d((v_x, v_i))$, the $k - d'((v_x, v_i))$'th token arrives no later than the $k - d((v_x, v_i))$'th token.

Further if the difference $\epsilon(v_i, k)$ between the start time and enabling time of firing k of actor v_i is decreased to become $\epsilon'(v_i, k), 0 \leq \epsilon'(v_i, k) < \epsilon(v_i, k)$, then this does not lead to an increase of the start time of firing k of actor v_i.

Since for any actor v_i and any firing k no decrease of the response time, increase in the number of initial tokens, or decrease of the difference between start time and enabling time leads to a later start time, we conclude that a SRDF graph is temporally monotonic. □

B.

PROOF OF THEOREM 2. We know that (1) Equation 3 holds, (2) every SRDF graph is temporally monotonic, and (3) there is a one-to-one correspondence between the component graph and the task graph. The one-to-one correspondence combined with the fact that Equation 3 holds, means that for constant response times token arrival times form a conservative bound on container arrival times. This is because tokens are produced when the actor finishes, while containers are produced before a task finishes. Furthermore we know that an SRDF graph is temporally monotonic, which means that no reduction of the response time of an actor in the component graph leads to a later start time of an actor in the component graph. Combining this fact with the facts that the components are conservative with respect to the tasks and that there is a one-to-one correspondence between the component graph and the task graph, leads to the conclusion that token arrival times in the component graph are conservative container arrival times in the task graph, and are therefore worst-case container arrival times. □

C.

Because $W_x(s, t) \in \mathbb{N}$, using the bound $\breve{Q}(s, t)$ leads to a latency Θ'_x that is $\frac{1}{\rho}$ lower than the latency Θ''_x derived using the bound $Q(s, t)$. This is because when using $\breve{Q}(s, t)$, we have that $\breve{Q}_x(s, \tau) \leq n$ for $\tau < s + \Theta'_x + \frac{n+1}{\rho_x}$, while when using $Q(s, t)$, we have that $Q_x(s, \tau) \leq n$ for $\tau < s + \Theta''_x + \frac{n}{\rho_x}$. Since at the finish time of some execution m both bounds are exact, we have that $\Theta'_x = \Theta''_x - \frac{1}{\rho_x}$. In [22] latencies of a number of $\mathcal{LR}$ servers are derived using the bound $Q(s, t)$.

Traditionally, the arrival function A and service function W are expressed in the amount of time requested from and provided by the $\mathcal{LR}$ server, while we have defined them in terms of number of enablings and finishes of task executions. However, in worst-case analysis these two notions are equivalent. This is because, in worst-case analysis we can associate the worst case execution time, $\overline{ET}_x$, which is the maximum time required by the execution of task u_x when executed in isolation, with every request or provision of service. If with every arrival and provision of service the $\overline{ET}_x$ is associated, then ρ_x would be multiplied by $\overline{ET}_x$ to become $\rho'_x = \rho_x \cdot \overline{ET}_x$, while Θ_x remains the same. In this case, ρ'_x would mean the fraction of time that a requester is allowed to execute on the $\mathcal{LR}$ server, i.e. after Θ_x time, which corresponds with the traditional notion of rate. Therefore $\frac{1}{\rho'_x}$ would mean the fraction with which the execution time of the requested is stretched due to execution on the $\mathcal{LR}$ server, excluding Θ_x. And therefore $\frac{\overline{ET}_x}{\rho'_x}$ would be the worst case time required to finish a request for service, excluding Θ_x. However, we have that $\frac{\overline{ET}_x}{\rho'_x} = \frac{\overline{ET}_x}{\rho_x \cdot \overline{ET}_x} = \frac{1}{\rho_x}$, which means that the definitions as used in the present discussion are equivalent to the traditionally used definitions. Furthermore the definitions as used in the present discussion ease the notation in the next section where we show an SRDF component that models the same behaviour as an $\mathcal{LR}$ server.

D.

PROOF OF LEMMA 1. According to Equation 7, execution j of actor v_x is guaranteed to have finished at $\tau_j = b(k) + \Theta + \frac{j-k+1}{\rho}$. For Equation 7 to equal Equation 6, it should hold that starting from τ_j the guaranteed number of finishes is at least $j - k + 1$. More formally it should hold that

$$\forall \tau' \geq \tau_j : \breve{Q}(b(k), \tau') \geq j - k + 1, \text{and} \quad (20)$$

$$\forall \tau'', b(k) \leq \tau'' < \tau_j : \breve{Q}(b(k), \tau'') < j - k + 1 \quad (21)$$

Equation (20) holds, because, according to Equation 6,

$$\breve{Q}(s, \tau_j) = \max(0, \lfloor \rho \cdot (\tau_j - b(k) - \Theta) \rfloor) \quad (22)$$

$$\breve{Q}(s, \tau_j) = \max(0, \lfloor \rho \cdot (b(k) + \Theta + \frac{j-k+1}{\rho} - b(k) - \Theta) \rfloor) \quad (23)$$

$$\breve{Q}(s,\tau_j) = \max(0, \lfloor \rho \cdot (\frac{j-k+1}{\rho}) \rfloor) \tag{24}$$

$$\breve{Q}(s,\tau_j) = \max(0, \lfloor j-k+1 \rfloor) = j-k+1 \tag{25}$$

And Equation (21) holds, because for $\tau'' = \tau_j - \epsilon, 0 < \epsilon \leq \tau_j - b(k)$, we have that

$$\breve{Q}(s,\tau'') = \max(0, \lfloor \rho \cdot (\tau_j - \epsilon - b(k) - \Theta) \rfloor) \tag{26}$$

$$\breve{Q}(s,\tau'') = \max(0, \lfloor \rho \cdot (b(k) + \Theta + \frac{j-k+1}{\rho} - \epsilon - b(k) - \Theta) \rfloor) \tag{27}$$

$$\breve{Q}(s,\tau'') = \max(0, \lfloor \rho \cdot (\frac{j-k+1}{\rho} - \epsilon) \rfloor) \tag{28}$$

$$\breve{Q}(s,\tau'') = \max(0, \lfloor j-k+1-\epsilon \rfloor) < j-k+1 \tag{29}$$

Furthermore execution k is not guaranteed to have finished at any $\dot{\tau}$ for which holds

$$b(k) < \dot{\tau} < \tau_k = b(k) + \Theta + \frac{1}{\rho} \tag{30}$$

Equation (30) is true because for $\dot{\tau} = \tau_k - \epsilon, 0 < \epsilon < b(k)$ we have that

$$\breve{Q}(b(k), \tau_k - \epsilon) \geq \max(0, \lfloor \rho \cdot (b(k) + \Theta + \frac{1}{\rho} - \epsilon - b(k) - \Theta) \rfloor) \tag{31}$$

$$\breve{Q}(b(k), \tau_k - \epsilon) \geq \max(0, \lfloor \rho \cdot (\frac{1}{\rho} - \epsilon) \rfloor) \tag{32}$$

$$\breve{Q}(b(k), \tau_k - \epsilon) \geq \max(0, \lfloor 1 - \rho \cdot \epsilon \rfloor) = 0 \tag{33}$$

□

E.

PROOF OF LEMMA 2. According to Equation (8), execution j of actor v_x is guaranteed to be externally enabled at $\phi_j = b(k) + \frac{j-k}{\rho}$. For the bounds, as provided by Equation (8) and Equation (4), to be equal, it should hold that starting from ϕ_j the number external enablings is at least $j - k$. More formally, it should hold that

$$\forall \phi' \geq \phi_j : H(b(k), \phi') \geq j-k, \text{and} \tag{34}$$

$$\forall \phi'' < \phi_j : H(b(k), \phi'') < j-k \tag{35}$$

Equation (34) holds, because, according to Equation (4)

$$H(b(k), \phi_j) = \rho \cdot (\phi_j - b(k)) \tag{36}$$

$$H(b(k), \phi_j) = \rho \cdot (b(k) + \frac{j-k}{\rho} - b(k) = j-k \tag{37}$$

And further for $\phi'' = \phi_j - \epsilon, 0 < \epsilon \leq \phi_j - b(k)$, we have that

$$H(b(k), \phi'') = \rho \cdot (\phi_j - \epsilon - b(k)) \tag{38}$$

$$H(b(k), \phi'') = \rho \cdot (b(k) + \frac{j-k}{\rho} - \epsilon - b(k) = j-k-\rho\epsilon < j-k \tag{39}$$

□

F.

PROOF OF LEMMA 3. We will first show the correctness for the first busy period by induction on the number of executions, and subsequently use this result as the base step to prove this lemma by induction on the number of busy periods.

Base step – induction on the number of executions: For the first execution we have that

$$g(1) = \max(b(1) + \Theta, g(1-1)) + \frac{1}{\rho} \tag{40}$$

and because $g(1-1) = g(0) = 0$.

$$g(1) = b(1) + \Theta + \frac{1}{\rho} = \tau_1 \tag{41}$$

Inductive step – induction on the number of executions: We assume that for execution $j, j \geq 1$ holds that $g(j) = \tau_j$, and thus that

$$b(k) + \Theta + \frac{j-k+1}{\rho} = \max(b(j) + \Theta, g(j-1)) + \frac{1}{\rho} \tag{42}$$

We will now show that given this assumption also for execution $j+1$ we have $g(j+1) = \tau_{j+1}$, and thus that

$$b(k) + \Theta + \frac{(j+1)-k+1}{\rho} = \max(b(j+1) + \Theta, g(j)) + \frac{1}{\rho} \tag{43}$$

We know from Equation (9) and Equation (8) that

$$g(j+1) = \max(b(j+1) + \Theta, g(j)) + \frac{1}{\rho} \tag{44}$$

$$b(j+1) \leq \phi_{j+1} = b(k) + \frac{(j+1)-k}{\rho} \tag{45}$$

Given the induction hypothesis from Equation (42), we know that

$$g(j) = \tau_j = b(k) + \Theta + \frac{j-k+1}{\rho} \tag{46}$$

We therefore conclude that

$$b(j+1) + \Theta \leq \phi_{j+1} + \Theta = b(k) + \Theta + \frac{j-k+1}{\rho} = \tau_j = g(j) \tag{47}$$

This results in $\max(b(j+1) + \Theta, g(j)) = g(j)$, and therefore Equation (44) results in

$$g(j+1) = g(j) + \frac{1}{\rho} = \tau_{j+1} \tag{48}$$

We have now shown the equivalence of $g(j)$ and τ_j for the first busy period. This forms the base step for the proof by induction that this equivalence holds for all busy periods.

Inductive step – induction on the number of busy periods: We assume that $g(j) = \tau_j$ for all executions up to and including busy period β_l starting at $b(h)$, we now show that given this assumption $g(j) = \tau_j$ is also true for busy period β_{l+1} starting at $b(k)$.

Let k be the first execution in the busy period β_{l+1}, then execution $k-1$ belonged to the previous busy period β_l starting at $b(h)$. From Equation (8), we have that

$$b(k-1) \leq \phi_{k-1} = b(h) + \frac{(k-1)-h}{\rho} \tag{49}$$

And since execution k does not belong to busy period β_h, we have that

$$b(k) > b(h) + \frac{k-h}{\rho} \tag{50}$$

Furthermore we know from Equation (7) that

$$\tau_{k-1} = b(h) + \Theta + \frac{(k-1) - h + 1}{\rho} \tag{51}$$

$$\tau_{k-1} = b(h) + \Theta + \frac{k-h}{\rho} \tag{52}$$

Using Equation (50)

$$\tau_{k-1} < b(k) + \Theta \tag{53}$$

Using the induction hypothesis we obtain $\tau_{k-1} = g(k-1)$, which leads to

$$g(k) = \max(b(k) + \Theta, g(k-1)) + \frac{1}{\rho} \tag{54}$$

$$g(k) = b(k) + \Theta + \frac{1}{\rho} = \tau_k \tag{55}$$

The earlier part of this proof, *Inductive step – induction on the number of executions*, that showed that $g(j+1) = \tau_{j+1}$, if $g(j) = \tau_j$, for the first busy period, did not assume that the executions j and $j+1$ were in the first busy period and therefore holds for any busy period. □

G.

PROOF OF THEOREM 3. Let $a_y(i)$ be the arrival time of token i on edge (v_k, v_y), let $a_z(i)$ be the arrival time of token i on the edge (v_y, v_z), and let $a_l(i)$ be the arrival time of token i on the edge (v_z, v_l), see Figure 5. From Figure 5 we can derive that

$$a_z(i) = a_y(i) + \Theta_x \tag{56}$$

$$a_l(i) = \max(a_z(i), a_l(i-1)) + \frac{1}{\rho_x} \tag{57}$$

Substitution of Equation (56) in Equation (57) results in

$$a_l(i) = \max(a(i) + \Theta_x, a_l(i-1)) + \frac{1}{\rho_x} \tag{58}$$

Since every arrival of a token on the edge (v_k, v_y) leads to an external enabling of v_y, we have that $b(v_y, i) = a_y(i)$. And since every firing of v_z produces one token on the edge (v_z, v_l), we have that a finish of firing i of actor v_z corresponds with the release of a token on the edge (v_z, v_l): $f(v_z, i) = a_l(i)$. This means that Equation (58) becomes

$$f(v_z, i) = \max(b(v_y, i) + \Theta_x, f(v_z, i-1)) + \frac{1}{\rho_x} \tag{59}$$

By defining $f(v_z, i) = 0$ for $i \leq 0$, we have that substitution of $g(j) = f(v_z, i)$ and $b(j) = b(v_y, i)$ results in an equality of Equations (59) and (9). Lemma 3 and Lemma 1 tells us that the bound on the finish time of actor v_x as obtained by Equation (9) is equivalent to the bound on the number of finishes of actor v_x as obtained by Equation (6).

This means that in any busy period $(s, t]$ the number of releases by actor v_z in the interval $(s, \tau], s < \tau \leq t$ equals $\check{Q}$ as defined by Equation (6). □