Technical Note NXP-TN-2007-00119

Issued: 9/2007

Real-Time Scheduling of Hybrid Systems using Credit-Controlled Static-Priority Arbitration

Benny Åkesson, Liesbeth Steffens,
Eelke Strooisma and Kees Goossens

Authors' address Benny Akesson benny.akesson@nxp.com
Liesbeth Steffens liesbeth.steffens@nxp.com
Eelke Strooisma eelke.strooisma@nxp.com
Kees Goossens kees.goossens@nxp.com

Title: Real-Time Scheduling of Hybrid Systems using Credit-Controlled Static-Priority Arbitration

Author(s): Benny Åkesson, Liesbeth Steffens,
Eelke Strooisma and Kees Goossens

Reviewer(s): Martijn Coenen, Maarten Wiggers

Technical Note: NXP-TN-2007-00119

Additional Numbers:

Subcategory:

Project: Hijdra (2002-213), Variomatic (2006-188)

Customer: NXP Semiconductors

Keywords: CCSP, Arbitration, Scheduling, QoS, Predictability

Abstract: The convergence of application domains in new systems-on-chip results in hybrid systems with many intellectual property components with a mix of soft and hard real-time requirements. Resources, such as memories and interconnect, are shared between requestors to reduce cost. However, resource sharing introduces interference between the sharing components, referred to as requestors, making it difficult to satisfy their real-time requirements. Existing arbiters do not efficiently satisfy the requirements of the requestors in these systems, as they couple rate or allocation granularity to latency, or cannot run at sufficiently high speeds in hardware with a low-cost implementation.

The contribution of this document is an arbiter called Credit-Controlled Static-Priority (CCSP) consisting of a rate regulator and a static-priority scheduler. The rate regulator isolates requestors by regulating the amount of provided service. Regulating provided service, as opposed to regulating requested service has two benefits: 1) the implementation of the regulator is less complex, and 2) the amount of work associated with a particular request does not have to be known. We show that CCSP belongs to the class of latency-rate servers and guarantees the allocated service rate within a maximum latency, required by hard real-time requestors. We present an implementation of the arbiter in the context of a DDR2 SDRAM controller that has been efficiently integrated into the network interface of a network-on-chip. The implementation supports service allocation with negligible over-allocation, and its area is small and scales linearly with the number of requestors. An instance with six ports runs at 250 MHz and requires 0.0175 mm^2 in CMOS090LP.

Conclusions: In this document, we present an arbiter called Credit-Controlled Static-Priority (CCSP) for scheduling access to resources, such as interconnect and memories in systems-on-chip. CCSP resembles an arbiter with a rate regulator that enforces a (σ, ρ) constraint on requested service together with a static-priority scheduler. However, instead of enforcing an upper bound on requested service, CCSP enforces an upper bound on provided service. Regulating provided service, as opposed to regulating requested service has two benefits: 1) the implementation of the regulator is less complex, and 2) the amount of work associated with a particular request does not have to be known. We show that CCSP enjoys these benefits, without increasing worst-case latency or buffering, compared to an arbiter regulating requested service. We furthermore show that CCSP belongs to the class of latency-rate ($\mathcal{LR}$) servers and guarantees the allocated service rate within a maximum latency, required by hard real-time requestors.

We present an implementation of the arbiter in the context of a DDR2 SDRAM controller that has been efficiently integrated into the network interface of a network-on-chip. The area of implementation is small and scales linearly with the number of requestors. An instance with six ports runs at 250 MHz and requires 0.0175 mm^2 in CMOS090LP. The efficiency of the service allocation is demonstrated in a use-case involving an H.264 decoder, where only 0.0294% of the resource capacity is lost due to over-allocation.

Future work involves creating a methodology to automatically derive service allocations and find a suitable priority assignment, such that the latency and service requirements of all requestors are met. This is a challenging problem that may require a heuristic approach, as the latency of the CCSP arbiter is computed by a non-linear equation that does not lend itself to integer-linear programming, and the configuration space is furthermore likely to be too large to allow exhaustive searches.

Contents

Section 1

Introduction

Chip design is getting increasingly complex, as technological advances allow highly integrated systems on a single piece of silicon. A contemporary multi-processor system-on-chip (SoC) features a large number of intellectual property components (IP), such as streaming hardware accelerators and processors with caches. Resources, such as memories and interconnects, are shared between IPs in order to reduce system cost. Access to these resources is provided by resource arbiters with a low-cost hardware implementation that runs at high speeds. However, resource sharing introduces interference between IPs making it difficult to satisfy their real-time requirements. We refer to users of the resources as *requestors*, which correspond, for example, to processes or threads in the context of CPUs, or to communication channels in case of a memory or an interconnect.

We consider resource scheduling in *hybrid systems* [1] that contain requestors with both soft and hard real-time requirements. Hard real-time requestors, such as a digital-to-analog converter, typically have predictable and regular request patterns. Their deadlines are not very tight, but must always be met in order to guarantee the functional correctness of the SoC [1, 2, 3]. These requestors require a *guaranteed minimum service rate and a bounded maximum latency* that can be analytically verified at design-time. In contrast, a soft real-time requestor, such as a software video decoder, is typically very bursty and has tight deadlines on a much coarser grain than their hard real-time counterparts. These deadlines may span thousands of requests, making the worst-case latency of a single request less interesting. Missing a soft deadline reduces the quality of the application output, such as causing a frame skip in video playback, which may be acceptable as long is it does not occur too frequently [1,3]. These requestors require a *guaranteed minimum service rate and a low average latency* to minimize the number of missed deadlines.

Existing resource arbiters are unable to cater to the above-mentioned requirements for at least one of the following three reasons: 1) allocation granularity is coupled to latency resulting in long latencies or over-allocation due to discretization errors, 2) latency is coupled to rate making the arbiter unable to provide low latency to requestors with low rate requirements without over-allocating, or 3) they cannot run at sufficiently high speeds in hardware with a low-cost implementation.

The contribution of this document is a novel arbiter called Credit-Controlled Static-Priority (CCSP) consisting of a rate regulator and a static-priority scheduler. The rate regulator isolates requestors by regulating the amount of provided service in a way that decouples allocation granularity and latency. The static-priority scheduler decouples latency and rate, such that low latency can be provided to any requestor, regardless of the allocated rate. We show that CCSP belongs to the class of latency-rate ($\mathcal{LR}$) servers and provides a minimum amount of service within a maxi-

mum latency, required by hard real-time requestors. We present an implementation of the arbiter in the context of a DDR2 SDRAM controller that has been efficiently integrated into the network interface of a network-on-chip. The implementation supports service allocation with negligible over-allocation, and its area is small and scales linearly with the number of requestors.

This document is organized as follows. In Section 2, we review related work and discuss why existing arbiters do not satisfy the requirements of hybrid systems in SoCs. We introduce a formal model in Section 3 and show how service curves are used to describe the interaction between requestors and the arbiter. We introduce the CCSP arbiter in Section 4 and have explain the operation of the rate regulator and static-priority arbiter. We analyze the arbiter in Section 5, to derive a service guarantee and to prove that CCSP belongs to the class of $\mathcal{LR}$ servers. In Section 6, we show how to extend the arbiter and analysis to cover preemption and work-conservation. An efficient hardware implementation is presented in Section 7 before studying experimental results for a hybrid system running an H.264 decoder in Section 8. Lastly, conclusions and future work are presented in Section 9.

Section 2

Related work

Many arbiters have been proposed in the context of communication networks. Several of these are based on the Round-Robin algorithm because it is simple and starvation-free. Weighted Round-Robin [4] and Deficit Round-Robin [5] (DRR) are extensions that guarantee each requestor a minimum service, proportional to an allocated rate, in a frame of fixed size. This type of *frame-based* arbitration suffers from a coupling between allocation granularity and latency, where the allocation granularity is inversely proportional to the frame size [6]. Larger frame sizes result in finer allocation granularities, reducing over-allocation, at the cost of increased latencies for all requestors. This granularity issue is addressed in [7,8,9] with hierarchical framing strategies and in [10], where tracking debits and credits accomplishes exact allocation over multiple frames. It is shown in [10] that fine-grained allocation in certain cases allows systems with a large number of requestors to achieve 100% increase in utilization of the resource compared to regular frame-based approaches. The above-mentioned algorithms, as well as the family of Fair Queuing algorithms [6], are unable to distinguish different latency requirements, as the rate is the only parameter affecting scheduling. This results in an unwanted coupling between latency and rate where latency, in the best case, is inversely proportional to the allocated rate. Requestors with low rate requirements hence suffer from high latency unless their rates are increased, reducing resource utilization.

Scheduling of hybrid systems is previously considered in [1, 11]. Hard real-time requestors are scheduled in [1] by an earliest-deadline-first (EDF) scheduler, while a constant-bandwidth server is used for soft real-time requestors. This approach is not immediately applicable to resource arbiters in SoCs, as it is difficult to provide a low-cost hardware implementation of an EDF scheduler that runs at sufficiently high speeds. The implementation of an EDF scheduler in [12] uses a tree of comparators to compare deadlines in the priority queue, which is too slow for our considered resources. A more scalable implementation is provided in [13]. However, this implementation requires double-ported SRAM memory, which is too expensive for many resource arbiters in a SoC. The scheduling approach in [11] employs a static-priority arbiter with a simple implementation. High priority is assigned to soft real-time requestors to achieve low average latency. Rate regulation is used to protect the lower priority hard real-time requestors, such that a bounded worst-case latency can be provided. The approach of scheduling hybrid systems using a static-priority arbiter has the benefit of being cheap to implement in hardware. However, the proposed arbiter has significant short-comings, as the rate regulator is frame-based and couples allocation granularity, latency and rate, despite the use of priorities.

To efficiently employ the ideas of [11], a static-priority scheduler that decouples latency, rate and allocation granularity, isolates requestors, and has a low-cost hardware implementation is

required. Two implementations of static-priority arbiters with rate regulation are provided in [14, 15], none of which meets these requirements. Rate-Controlled Static-Priority [14] controls rate by holding requests until certain constraints on minimum and average inter-arrival times between requests from a requestor are satisfied. This requires a potentially large number of time-stamps to be stored in the arbiter, which is not feasible for a resource arbiter in a SoC. ALG [15], is an asynchronous scheduling discipline that uses priorities and has an efficient implementation. Service is allocated in discrete chunks, the size of which depends on the priority of the requestor and the total number of requestors sharing the resource. This couples allocation granularity and rate. The authors claim that this is solved by virtually increasing the number of requestors sharing the resource, and assigning multiple chunks per requestor. However, the efficiency of this extension and its implications on latency and ordering of requests are not straight-forward and are not discussed in the paper. Moreover, at most 84% of the resource capacity can be used for guaranteed service.

We propose Credit-Controlled Static-Priority arbitration for scheduling access to resources, such as interconnect and memories in SoCs. CCSP resembles an arbiter with a rate regulator that enforces a (σ, ρ) constraint [16][1] on requested service together with a static-priority scheduler, a combination we refer to as SRSP in this document. Similarly to SRSP, the CCSP rate regulator replenishes the service available to a requestor incrementally, instead of basing it on frames, decoupling allocation granularity and latency. Both arbiters furthermore use priorities to decouple latency and rate. However, instead of enforcing an upper bound on requested service, CCSP enforces an upper bound on provided service. Regulating provided service reduces the complexity of the hardware implementation, and allows the arbiter to efficiently handle requests with unknown sizes. We furthermore show that CCSP has a low-cost implementation that runs in high speeds.

[1] Different from a (σ, ρ)-regulator presented in the same document, since that regulator does not actually enforce the constraint.

Section 3

Formal model

In this section, we introduce the formal model used in this document. We explain how service curves are used to model the interaction between the requestors and the resource in Section 3.1. We proceed by discussing the models used to bound *requested service* and *provided service* in Section 3.2 and Section 3.3, respectively. Table 3.1 lists the notation conventions that are used throughout this document. We use subscripts to disambiguate between variables belonging to different requestors, although for clarity these subscripts are omitted when they are not required. We use $\mathbb{N}$ to denote the set of non-negative integers, and $\mathbb{N}^+$ to denote the set of positive integers.

To emphasize the generality of our approach, and its applicability to a wide range of resources, we abstract from a particular target resource, such as memories or (multi-hop) interconnects. We adopt an abstract resource view, where a service unit corresponds to the access granularity of the resource. A time unit, referred to as a *cycle*, is defined as the time required to serve such a service unit. Time is discrete and counting from zero.

3.1 Service curves

We use service curves [17] to model the interaction between the resource and the requestors. These service curves are typically cumulative and monotonically non-decreasing in time. We start by defining a an operator for retrieving the value of a service curve in Definition 1. We use closed discrete time intervals throughout this document. The interval $[\tau, t]$ hence includes all cycles in the set $\{\tau, \tau + 1, ..., t - 1, t\}$. Definition 2 defines notation for expressing the difference in values between the endpoints of such an interval.

Definition 1. *$\xi(t)$ denotes the value of a service curve ξ at the beginning of a cycle t.*

Definition 2. *$\xi(\tau, t)$ denotes the difference in values between the endpoints of the closed interval $[\tau, t]$, where $t \geq \tau$, and is defined as $\xi(\tau, t) = \xi(t + 1) - \xi(\tau)$.*

Table 3.1: The notation used throughout this document.

Description	*Example*
Capital letter denotes a set	A
Hat denotes an upper bound	$\hat{a}$
Check denotes a lower bound	$\check{a}$
Bar denotes an average value	$\bar{a}$

The resource is shared between a set of requestors, as stated in Definition 3. A requestor generates service requests of variable but bounded size, as defined in Definition 4, Definition 5, and Definition 6.

Definition 3 (Set of requestors). *The set of requestors sharing the resource is denoted R.*

Definition 4 (Set of requests). *The set of requests from a requestor $r \in R$ is denoted Ω_r.*

Definition 5 (Request). *The k:th request ($k \in \mathbb{N}$) from a requestor $r \in R$ is denoted $\omega_r^k \in \Omega_r$.*

Definition 6 (Request size). *The size of a request ω_r^k is denoted $s(\omega_r^k) : \Omega_r \rightarrow \mathbb{N}^+$.*

Requests arrive at the resource according to Definition 7. For clarity, it is assumed that only a single request arrives per requestor in a particular cycle, as stated in Assumption 1, although this is easy to generalize. A request is considered to arrive as an impulse when it is ready to be served by the resource, which for instance in the case of a memory controller is upon arrival of the last bit of the request. This is captured by the requested service curve, w, defined in Definition 8. Note that Definition 7 and Definition 8 state that a requested service curve at time $t + 1$ accounts for a request with arrival time $t + 1$.

Definition 7 (Arrival time). *The arrival time of a request ω_r^k from a requestor $r \in R$ is denoted $t_a(\omega_r^k) : \Omega_r \rightarrow \mathbb{N}^+$, and corresponds to the cycle in which the last bit of ω_r^k arrives.*

Assumption 1. *It holds that $\forall k \in \mathbb{N}^+, \forall r \in R.\ t_a(\omega_r^k) > t_a(\omega_r^{k-1}) > 0$.*

Definition 8 (Requested service curve). *The requested service curve of a requestor $r \in R$ is denoted $w_r(t) : \mathbb{N} \rightarrow \mathbb{N}$, and is defined as*

$$w_r(t+1) = \begin{cases} w_r(t) + s(\omega_r^k) & \exists \omega_r^k.\ t_a(\omega_r^k) = t+1 \\ w_r(t) & \nexists \omega_r^k.\ t_a(\omega_r^k) = t+1 \end{cases}$$

where $w_r(0) = 0$.

The scheduler in the resource arbiter attempts to schedule a requestor every cycle according to its particular scheduling policy. The scheduled requestor is denoted according to Definition 9. The first cycle in which a request ω^k is scheduled is referred to as its starting time, $t_s(\omega^k)$, defined in Definition 10. A request cannot be scheduled until is has arrived, as stated in Assumption 2.

Definition 9 (Scheduled requestor). *The scheduled requestor at a time t is denoted $\gamma(t) : \mathbb{N} \rightarrow R \cup \{\varnothing\}$.*

Definition 10 (Starting time of a request). *The starting time of a request ω_r^k is denoted $t_s(\omega_r^k) : \Omega_r \rightarrow \mathbb{N}$, and is defined as the smallest t at which ω_r^k is scheduled.*

Assumption 2. *It holds that $\forall r \in R, \omega_r^k \in \Omega_r.\ t_s(\omega_r^k) \geq t_a(\omega_r^k)$*

The provided service curve, w', defined in Definition 11, reflects the amount of service units provided by the resource to a requestor. A service unit takes one cycle to serve. This is reflected in that provided service is increased at $t+1$, if a requestor is scheduled at t. A request leaves the resource when the last service unit of the request has been served, corresponding to when the last bit is read or written in case of a memory controller. An illustration of a requested service curve and a provided service curve is provided in Figure 3.1. For reasons of clarity, the curves in the figure are drawn as continuous functions, although their values are only defined at discrete points in time.

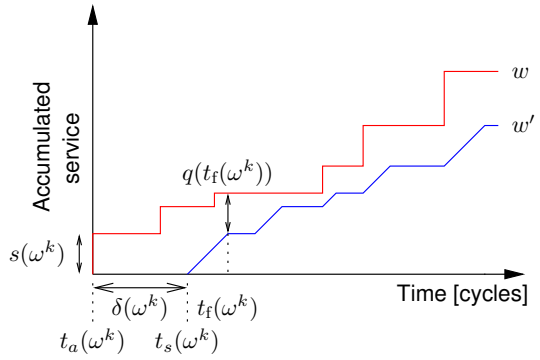

Figure 3.1: A requested service curve, a provided service curve and representations of the surrounding concepts.

Definition 11 (Provided service curve). *The provided service curve of a requestor $r \in R$ is denoted $w'_r(t) : \mathbb{N} \rightarrow \mathbb{N}$, and is defined as*

$$w'_r(t+1) = \begin{cases} w'_r(t) + 1 & \gamma(t) = r \\ w'_r(t) & \gamma(t) \neq r \end{cases}$$

where $w'_r(0) = 0$.

Corollary 1. *It follows from Assumption 2 that $w(t) \geq w'(t)$.*

The finishing time of a request corresponds to the first cycle in which a request is completely served, as defined in Definition 12. We initially consider a non-preemptive scheduler, which means that a requestor is scheduled during $s(\omega^k)$ consecutive cycles after service of ω^k is started at $t_s(\omega^k)$. This is formally expressed in Definition 13. This allows us to express the finishing time of a request in a non-preemptive scheduler, as shown in Corollary 2. We extend the model to cover preemptive scheduling in Section 6.1.

Definition 12 (Finishing time of a request). *The finishing time of a request ω_r^k is denoted $t_f(\omega_r^k) : \Omega_r \rightarrow \mathbb{N}$, and is defined as $t_f(\omega_r^k) = \min(\{t \mid t \in \mathbb{N} \wedge w'_r(t) = w'_r(t_s(\omega_r^k)) + s(\omega_r^k)\})$.*

Definition 13 (Non-preemption). *A non-preemptive scheduler is defined such that $\forall r \in R, k \in \mathbb{N}$. $w'_r(t_s(\omega_r^k) + s(\omega_r^k)) = w'_r(t_s(\omega_r^k)) + s(\omega_r^k)$.*

Corollary 2. *The finishing time of a request ω_r^k in a non-preemptive arbiter is denoted $t_f^{np}(\omega_r^k) : \Omega_r \rightarrow \mathbb{N}$. It follows from Definition 12 and Definition 13 that $t_f^{np}(\omega_r^k) = t_s(\omega_r^k) + s(\omega_r^k)$.*

We are now ready to define the concepts of backlog and delay. The backlog of a requestor, defined in Definition 14, corresponds to the amount of requested service that has not yet been served at a particular time. The set of requestors that are backlogged at a particular time is defined in Definition 15. The delay of a request is the time from the arrival of the request until it begins receiving service, as stated in Definition 16. The graphical interpretations of backlog and delay are shown in Figure 3.1.

Definition 14 (Backlog). *The backlog of a requestor $r \in R$ at a time t is denoted $q_r(t) : \mathbb{N} \rightarrow \mathbb{N}$, and is defined as $q_r(t) = w_r(t) - w'_r(t)$.*

Definition 15 (Set of backlogged requestors). *The set of requestors that are backlogged at t is defined as* $R_t^q = \{r \mid \forall r \in R \wedge q_r(t) > 0\}$.

Definition 16 (Delay). *The delay of a request* ω_r^k *is denoted* $\delta(\omega_r^k) : \Omega_r \rightarrow \mathbb{N}$*, and is defined as* $\delta(\omega_r^k) = t_s(\omega_r^k) - t_a(\omega_r^k)$

In order to work with service curves analytically, abstraction is provided through a variety of traffic models that characterize the behavior of the curves. This abstraction has the benefit that analytical results can be derived without exact knowledge of a service curve, as long as it can be accurately characterized [6]. Characterizations that bound the requested and provided service curves are required to provide upper bounds on backlog and delay, which are needed to satisfy the requirements of hard real-time requestors.

3.2 Requested service model

We use the (σ, ρ) model [16] to characterize the requested service curve. The model uses a linear function to express a burstiness constraint, and is frequently used in literature to upper bound the requested service curve in an interval. The bounding function is determined by two parameters, σ and ρ, corresponding to burstiness and average request rate, respectively. The definition of a (σ, ρ)-constrained service curve is found in Definition 17, and its graphical interpretation is shown in Figure 3.2.

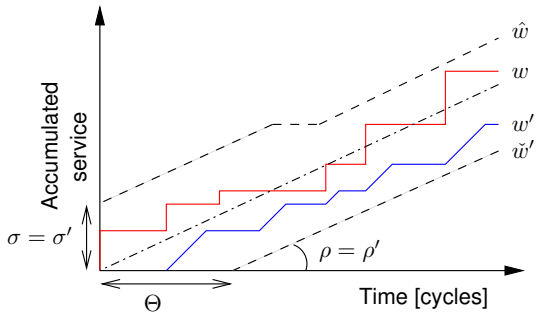

Figure 3.2: A requested service curve and a provided service curve along with their corresponding bounds.

Definition 17 ((σ, ρ) constraint). *A service curve, ξ, is defined to be (σ, ρ) constrained in an interval $[\tau, t]$ if* $\hat{\xi}(\tau, t) = \sigma + \rho \cdot (t - \tau + 1)$.

Hard real-time requestors typically correspond to hardware IPs with regular and predictable access patterns that lend themselves to characterization. Soft real-time requestors, however, are typically burstier than their hard real-time counterparts, and may hence have a σ that is very large. Soft real-time requestors may additionally be very difficult to characterize, as applications become more dynamic and input dependent. However, in this document, we assume that all requestors have been accurately characterized, according to Definition 18.

Definition 18 (Requestor). *A requestor $r \in R$ is characterized by a tuple $((\sigma_r, \rho_r), \hat{s}_r)$, where (σ_r, ρ_r) is a (σ, ρ) constraint on w_r, and* $\hat{s}_r = \max_{\forall k \in \mathbb{N}} s(\omega_r^k)$*. It holds that* $\sigma_r, \rho_r \in \mathbb{R}^+$ *and* $\rho < 1$.

3.3 Provided service model

The purpose of the provided service model is to provide a lower bound on the provided service curve based on the service allocation of a requestor. The service allocated to a requestor in our model depends on two parameters, as defined in Definition 19. These are the allocated service rate, ρ', and allocated burstiness, σ', respectively. The definition states three constraints that must be satisfied in order for a configuration to be valid: 1) the allocated service rate must be at least equal to the average request rate, ρ, to satisfy the service requirement of the requestor, and to get finite buffers, 2) it is not possible to allocate more service to the requestors than what is offered by the resource, and 3) the allocated burstiness must be sufficiently large to accommodate a maximally sized request. The last constraint will be further discussed in Section 5.2, along with additional guidelines on how to allocate burstiness.

Definition 19 (Allocated service). *The service allocation of a requestor $r \in R$ is defined as (σ'_r, ρ'_r), where $\sigma', \rho' \in \mathbb{R}^+$. For a valid allocation it holds that $\forall r \in R.\ \rho'_r \geq \rho_r$, $\sum_{\forall r \in R} \rho'_r \leq 1$, and $\forall r \in R.\ \sigma'_r \geq \hat{s}_r$.*

Our provided service model is based on the notion of *active periods*, defined in Definition 20. The definition states that a requestor is active at t if it is either live at t (Definition 21), backlogged at t, or both. Active periods hence depend on both w and w'. Definition 21 states that a requestor must have requested service according to its allocated rate, on average, since the start of the latest active period to be considered live at a time t. The requested service in this interval is expressed as $w_r(\tau_1 - 1, t - 1)$, where τ_1 is the start of the last active period. Note that $\tau_1 - 1$ is used in the expression since $w(\tau_1)$ accounts for requests that arrived at τ_1, according to Definition 8. The sets of requestors that are active and live at a particular time are defined in Definition 22 and Definition 23, respectively.

Definition 20 (Active period). *An active period of a requestor $r \in R$ is defined as a maximum interval $[\tau_1, \tau_2]$, such that $\forall t \in [\tau_1, \tau_2].\ w_r(\tau_1 - 1, t - 1) \geq \rho'_r \cdot (t - \tau_1 + 1) \vee q_r(t) > 0$.*

Definition 21 (Live requestor). *A requestor $r \in R$ is defined as live at a time t during an active period $[\tau_1, \tau_2]$ if $w_r(\tau_1 - 1, t - 1) \geq \rho'_r \cdot (t - \tau_1 + 1)$.*

Definition 22 (Set of active requestors). *The set of requestors that are active at t is defined as $R^a_t = \{r \mid \forall r \in R \wedge r \text{ active at } t\}$.*

Definition 23 (Set of live requestors). *The set of requestors that are live at t is defined as $R^l_t = \{r \mid \forall r \in R \wedge r \text{ live at } t\}$.*

Corollary 3. *It follows from Definitions 20-23 that $R^a_t = R^q_t \cup R^l_t$.*

Figure 3.3 illustrates the relation between being live, backlogged and active. Three requests arrive starting from τ_1. The relation between size and inter-arrival time of these request is sufficient to keep the requestor live until τ_3, considering its allocated rate. The requestor is initially both live and backlogged, but the provided service curve catches up with the requested service curve at τ_2, as the third request finishes receiving service. This requestor is hence in a live and not backlogged state until τ_3. The requestor is not live nor backlogged between τ_3 and τ_4, as no additional requests arrive at the resource. The requestor becomes live and backlogged again at τ_4, since two additional requests arrive within a small period of time. The requestor stays in this state until τ_5, since the requests keep the requestor live until this point, and not enough service

is provided to remove the backlog. The requestor is hence backlogged but not live at τ_5, and remains such until τ_6. The requestor in Figure 3.3 is active between τ_1 and τ_3 and between τ_4 and τ_6, according to Definition 20. Note from this example that a live requestor is not necessarily backlogged, nor vice versa.

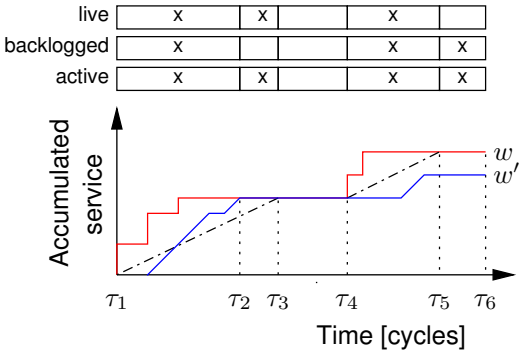

Figure 3.3: Example requested service curve and provided service curve that illustrate the relation between being live, backlogged and active.

The service provided to a requestor is defined by two parameters Θ and ρ', being latency and allocated rate, respectively. To disambiguate, we refer to Θ, defined in Definition 24, as *service latency* throughout this document. The definition states that service is provided to an active requestor according to the allocated rate, ρ', after an initial latency, Θ. This means that ρ' and Θ defines a lower bound, $\check{w}'$, on the provided service curve, as shown in Figure 3.2.

Definition 24 (Service latency). *The service latency of a requestor $r \in R$ is defined as the minimum $\Theta_r \in \mathbb{N}$, such that during any active period $[\tau_1, \tau_2]$ it holds that $\forall t \in [\tau_1, \tau_2]$. $\check{w}'_r(\tau_1, t) = \max(0, \rho'_r \cdot (t - \tau_1 + 1 - \Theta_r))$.*

We show in Section 5 that CCSP belongs to the class of $\mathcal{LR}$ servers [18], which are a general frame-work for analyzing scheduling algorithms. The lower bound on provided service in Definition 24 is a key characteristic of $\mathcal{LR}$ servers. The authors of [18] use this bound to derive general bounds on buffering and delay that are valid for any combination of $\mathcal{LR}$ servers in sequence. They furthermore show that many well-known schedulers, such as Generalized Processor Sharing [19], DRR [5], and several varieties of Fair Queuing [6] belong to the class. The $\mathcal{LR}$ server model was conceived in the context of packet switched networks, and is intended for analysis using Network Calculus [16]. Recent work has shown a link between the $\mathcal{LR}$ server model and data-flow analysis. In [20], it is shown that that a $\mathcal{LR}$ server can be modeled as a cyclo-static data-flow graph. This allows the arbiter to be used also in data-flow analysis, which has the added benefit that the presence of flow control can be accurately modeled, and that application-level throughput constraints can be satisfied. This is, however, outside the scope of this document.

Section 4

Credit-Controlled Static-Priority Arbitration

In this section, we present the CCSP arbiter. A CCSP arbiter consists of a rate regulator and a scheduler, following the decomposition from [14]. This partitioning provides a separation of concerns, but also emphasizes the modularity and re-usability of the components. We start in Section 4.1 by providing an overview of the main idea, before discussing the rate regulator and scheduler separately in Sections 4.2 and 4.3, respectively.

4.1 Overview

A rate regulator provides *accounting* and *enforcement* and thus determines which requests that are *eligible* for scheduling at a particular time, considering their allocated service. There are two types of enforcement, as an arbiter is either *work-conserving*, or *non-work-conserving*. A work-conserving arbiter uses weak enforcement and is never idle when there is a backlogged requestor. A rate regulator in a non-work-conserving arbiter, however, does not schedule a request until it becomes eligible, even though the resource may be idle. This is referred to as strict enforcement. A non-work-conserving arbiter clearly leads to a lower resource utilization, but is beneficial for networks of servers, as it limits the increase in burstiness of the provided service at their respective outputs [16, 14, 18]. It hence allows trading lower resource utilization for smaller buffers in a network. We consider both types of enforcement throughout this document, although for clarity we initially assume a non-work-conserving arbiter. We extend the model in Section 6.2 to also cover the work-conserving case.

The purpose of a rate regulator is to isolate requestors from each other and to protect requestors that do not ask for more service than they are allocated from the ones that do. This form of protection is a key property in providing guaranteed service to requestors with timing constraints [6]. A rate regulator protects requestors by enforcing burstiness constraints on either requested service or provided service. When enforcing an upper bound on requested service, requests are delayed until a particular burstiness constraint, such as a minimum inter-arrival time, is met before releasing them into the request buffers. All requests in the request buffers are considered eligible, and are scheduled according to the policy of the particular scheduler. Examples of this kind of regulators are found in [16,14]. Figure 4.1 shows an arbiter that enforces an upper bound on requested service. The rate regulator is positioned before the request buffers, allowing it to regulate the arriving requests by buffering them until the burstiness constraint is satisfied. Note that there is no communication between the scheduler and the rate regulator. A rate regu-

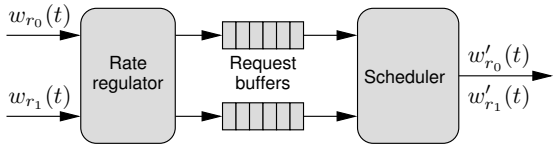

Figure 4.1: An arbiter with a rate regulator that enforces an upper bound on requested service.

lator that enforces an upper bound on provided service, such as those in [4, 5, 11] and the CCSP rate regulator, is shown in Figure 4.2. As seen in the figure, the rate regulator is positioned after the request buffers. It is hence only aware of requests at the heads of the buffers, and cannot constrain arrivals of requests in any way. The scheduler communicates the id of the scheduled requestor, $\gamma(t)$, back to the rate regulator every cycle. The regulator uses this information to update the accounting mechanism. This type of rate regulator operates by simply determining if the request at the head of each request buffer is eligible for scheduling.

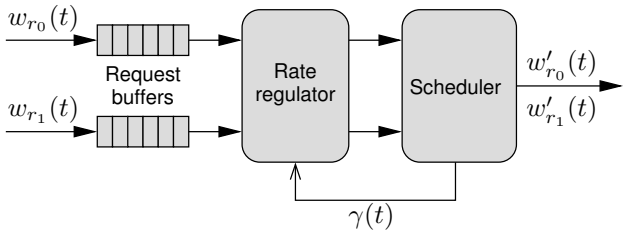

Figure 4.2: An arbiter with a rate regulator that enforces an upper bound on provided service.

Enforcing an upper bound on provided service has two benefits: 1) the implementation of the regulator is less complex, and 2) the amount of work associated with a particular request does not have to be known. We proceed by discussing these benefits in more detail.

A regulator that enforces an upper bound on provided service only requires knowledge about the size of the request at the head of each request queue. A regulator that enforces an upper bound on requested service, on the other hand, needs to know the sizes of all requests that arrive during a cycle. This incurs additional complexity in a hardware implementation, especially if requests can arrive with higher frequency than they are parsed.

A difficulty in resource arbitration is that the amount of work associated with a particular request is not always known before it has been served. For instance, the amount of time required to decode a video frame on a processor is not known when the work is scheduled. Similarly, the time required by most SDRAM controllers to serve a request of a particular size depends on the state of the memory. This in turn is predominantly determined by the previous requests, such as which SDRAM banks that were accessed, the request size, and if it was a read or a write access. These situations cannot be handled if requested service is regulated, unless worst-case assumptions are used to estimate the amount of work. This approach, however, is very inefficient if the variance in the amount of work is large, as in the examples mentioned above. This is efficiently handled when regulating provided service by charging for a single service unit at a time. This allows a preemptive scheduler to interrupt a requestor that runs out of budget and schedule another one.

We show in Section 5.3 that CCSP enjoys both these benefits, without increasing worst-case latency or buffering, compared to SRSP. CCSP furthermore uses incremental replenishment of service, in contrast to the frame-based provided service regulators in [4, 5, 11]. We show in

Section 7 that this decouples allocation granularity and latency.

4.2 Rate regulator

The CCSP rate regulator enforces an upper bound on provided service, as explained in Section 4.1. We regulate provided service based on active periods, and define the upper bound on provided service according to Definition 25. The intuition behind the definition is that the upper bound on provided service of an active requestor increases according to the allocated rate every cycle. Conversely for an inactive requestor, the bound is limited to $w'(t) + \sigma'$, a value that depends on the allocated burstiness. This prevents a requestor from becoming inactive for an extended period of time and increase the bound, possibly resulting in starvation of other requestors once it becomes active again. Note that this implies that the upper bound on provided service is not necessarily monotonically non-decreasing in time, as shown in Figure 4.3. The requestor in the figure is live until τ_1, but remains active until τ_2 where w' catches up to w. According to Definition 25, this results in $\hat{w}'(\tau_2 + 1) < \hat{w}'(\tau_2)$, since $\hat{w}'(\tau_2) > w'(\tau_2) + \sigma'$. The requestor starts a new active period at τ_3, causing $\hat{w}'$ to increase again.

Definition 25 (Provided service bound). *The enforced upper bound on provided service of a requestor $r \in R$ is denoted $\hat{w}'_r(t) : \mathbb{N} \rightarrow \mathbb{R}^+$, and is defined according to*

$$\hat{w}'_r(t+1) = \begin{cases} \hat{w}'_r(t) + \rho'_r & r \in R^a_t \\ w'_r(t) + \sigma'_r & r \notin R^a_t \end{cases} \tag{4.1}$$

where $\hat{w}'_r(0) = \sigma'_r$.

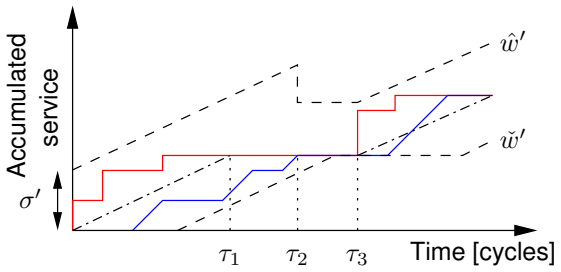

Figure 4.3: The upper bound on provided service is not necessarily monotonically non-decreasing.

It is not possible to perform accounting and enforcement in hardware based on $\hat{w}'$, since $\lim_{t\to\infty} \hat{w}'(t) = \infty$. Instead, the accounting mechanism in the rate regulator is based on the potential of a requestor, as defined in Definition 26. We show in Section 5.3 that the potential of a requestor is bounded since the arbiter guarantees a lower bound on provided service. The accounting is updated every cycle, according to Definition 27. Lemma 1 shows that the accounting mechanism in Definition 27 corresponds to a recursive definition of potential.

Definition 26 (Potential). *The potential of a requestor $r \in R$ is denoted $\pi_r(t) : \mathbb{N} \rightarrow \mathbb{R}$, and is defined as $\pi_r(t) = \hat{w}'_r(t) - w'_r(t)$.*

Definition 27 (Accounting). *The accounted potential of a requestor $r \in R$ is denoted $\pi^*_r(t) : \mathbb{N} \rightarrow \mathbb{R}$, and is defined according to*

$$\pi_r^*(t+1) = \begin{cases} \pi_r^*(t) + \rho_r' - 1 & r \in R_t^a \wedge \gamma(t) = r \\ \pi_r^*(t) + \rho_r' & r \in R_t^a \wedge \gamma(t) \neq r \\ \sigma_r' & r \notin R_t^a \wedge \gamma(t) \neq r \end{cases}$$

where $\pi_r^*(0) = \sigma_r'$.

Lemma 1. $\forall t \in \mathbb{N}.\ \pi(t) = \pi^*(t)$.

Proof. We prove the lemma by induction.
Base case: The lemma holds when $t = 0$, since $\pi(0) = \hat{w}'(0) - w'(0) = \pi^*(0)$, according to Definition 11, Definition 25 and Definition 27.
Inductive step: For the inductive step, we prove that if the lemma holds at a time t then it also holds for $t+1$. According to Definition 26, potential at $t+1$ is defined as $\pi_r(t+1) = \hat{w}_r'(t+1) - w_r'(t+1)$. We substitute $\hat{w}_r'(t+1)$ and $w_r'(t+1)$, according to the recursive definitions in Definition 25 and Definition 11, respectively. Definition 11 has two cases and depends on whether the requestor is scheduled or not. Similarly, Definition 25 has two cases depending on if the requestor is active or not. The resulting equation, shown in Equation (4.2), is hence supposed to have four cases. However, one case is eliminated since $r \notin R_t^a$ implies $\gamma(t) \neq r$.

$$\hat{w}'(t+1) - w'(t+1) = \begin{cases} (\hat{w}'(t) + \rho') - (w'(t) + 1) & r \in R_t^a \wedge \gamma(t) = r \\ (\hat{w}'(t) + \rho') - w'(t) & r \in R_t^a \wedge \gamma(t) \neq r \\ (w'(t) + \sigma') - w'(t) & r \notin R_t^a \wedge \gamma(t) \neq r \end{cases} \tag{4.2}$$

Finally, we substitute $\pi(t) = \hat{w}'(t) - w'(t)$, in accordance with Definition 26, after which we arrive at the the accounting mechanism in Definition 27. □

Enforcement in the rate regulator is performed by determining if a request from a requestor is eligible for scheduling. A request becomes eligible at its eligibility time, defined in Definition 28. The definition states that three conditions must be satisfied for a request at the eligibility time in a non-preemptive arbiter: 1) all previous requests from the requestor must have been served, as they are served in FIFO order, 2) the considered request must have arrived, and 3) the requestor must have enough potential to serve the complete request. The last condition is required to satisfy the definition of a non-preemptive system in Definition 13. The eligibility criterion for a requestor in a non-preemptive arbiter is formally defined in Definition 29. We proceed by defining the set of requestors that is eligible at a particular time in Definition 30.

Definition 28 (Eligibility time). *The eligibility time of a request* ω_r^k *from a requestor* $r \in R$ *in a non-preemptive arbiter is denoted* $t_e^{np}(\omega_r^k)$*, and is defined as the smallest* t *for which the following conditions apply:*

1. $\forall i < k.\ t \geq t_f(\omega_r^i)$*, and*
2. $w_r(t) \geq w_r'(t) + s(\omega_r^k)$*, and*
3. $\pi_r(t) \geq s(\omega_r^k) - \rho_r'(t)$

Definition 29 (Eligible requestor). r *is defined as eligible in a non-preemptive arbiter* $\forall k \in \mathbb{N}, t \in [t_e^{np}(\omega_r^k), t_f(\omega_r^k) - 1]$.

Definition 30 (Set of eligible requestors). *The set of requestors that are eligible for scheduling at t is defined as $R_t^e = \{r \mid \forall r \in R \wedge r \text{ eligible at } t\}$.*

The rest of this section contain lemmas that prove general properties of the rate regulator. These properties are used in the arbiter analysis in Section 5 and in the hardware implementation, presented in Section 7. We start in Lemma 2 by showing that an eligible requestor is backlogged and hence active. Note, however, that the reverse relation does not hold since an active requestor does not necessarily have enough potential to be eligible.

Lemma 2. $r \in R_t^e \Rightarrow q_r(t) > 0 \Rightarrow r \in R_t^a$

Proof. According to the eligibility criterion in Definition 29, a requestor is eligible for scheduling of ω_r^k in the interval $[t_e^{np}(\omega_r^k), t_\mathrm{f}(\omega_r^k) - 1]$. The second rule of Definition 28 states that $q_r(t_e^{np}(\omega_r^k)) \geq s(\omega_r^k)$. From this, Definition 11 and Definitions 13-14 give us that $\forall t \in [t_e^{np}(\omega_r^k), t_\mathrm{f}(\omega_r^k) - 1]$. $q_r(t) > 0$. This concludes the proof, since Definition 20 states that $q_r(t) > 0$ is a sufficient condition for a requestor to be active. □

We show in Lemma 3 that $\hat{w}'$ is truly an upper bound for w', given the accounting mechanism in Definition 27 and enforcement strategy in Definition 29. The intuition behind the lemma is that the eligibility criterion asserts that the requestor has enough potential already at the eligibility time for the bound never to be exceeded.

Lemma 3. *It holds that* $\forall t.\ \hat{w}'(t) \geq w'(t)$.

Proof. According to Lemma 1, proving this lemma is equivalent to showing that $\forall t.\ \pi(t) \geq 0$. This holds at $t = 0$, since Definition 27 defines $\pi(0) = \sigma'$. The accounting mechanism in Definition 27 additionally states that if $\pi(t) \leq \sigma'$ then $\pi(t+1) < \pi(t)$ only if the requestor is scheduled at t. It is hence sufficient to show that $\forall t \in [t_s(\omega^k), t_\mathrm{f}(\omega^k)].\ \pi(t) \geq 0$.

In a non-preemptive arbiter, ω^k can only be scheduled after becoming eligible at $t_e^{np}(\omega^k) \leq t_s(\omega^k)$. We know from Definition 29 that $\pi(t_e^{np}(\omega^k)) \geq s(\omega^k) - \rho'$. According to Definition 27, it holds that $\forall t \in [t_e^{np}(\omega^k), t_s(\omega^k) - 1].\ \pi(t+1) \geq \pi(t)$, since the requestor is eligible, and hence active (Lemma 2, and not scheduled in the interval. We hence know that

$$\pi(t_s(\omega^k)) \geq \pi(t_e^{np}(\omega^k)) \geq s(\omega^k) - \rho' \geq 0$$

According to Definition 13 and Definition 12, we get that $w'(t_\mathrm{f}(\omega^k)) = w'(t_s(\omega^k)) + s(\omega^k)$. From Definition 13 and Definition 27, we additionally get that $\hat{w}'(t_\mathrm{f}(\omega^k)) = \hat{w}'(t_s(\omega^k)) + s(\omega^k) \cdot \rho'$. Combining these results with the definition of potential in Definition 26 results in a potential at $t_\mathrm{f}(\omega^k)$ according to

$$\pi(t_\mathrm{f}(\omega^k)) = \pi(t_e^{np}(\omega^k)) + \rho' \cdot s(\omega^k) - s(\omega^k) \geq (s(\omega^k) - 1) \cdot \rho' \geq 0$$

□

Lemma 4 shows some important relations between the requested service curve and the provided service curve at the start of an active period. These are used for substitutions in other lemmas.

Lemma 4. *If τ_1 is the start of an active period then* $w(\tau_1) > w(\tau_1 - 1) = w'(\tau_1) = w'(\tau_1 - 1)$.

Proof. According to Definition 20, if τ_1 starts an active period then the requestor was inactive at $\tau_1 - 1$ and hence $q(\tau_1 - 1) = 0$. We know from Definition 14 that if $q(\tau_1 - 1) = 0$ then $w(\tau_1 - 1) = w'(\tau_1 - 1)$. This implies that the requestor cannot be scheduled at $\tau_1 - 1$, which according to Definition 11 results in that $w'(\tau_1) = w'(\tau_1 - 1)$. Definition 20 states that if an active period starts at τ_1 then $q(\tau_1) > 0$ or $w(\tau_1 - 1, \tau_1 - 1) \geq \rho'$. These cases all imply $w(\tau_1) > w(\tau_1 - 1)$. □

Lemma 5 shows how to express the potential of a requestor at any time during an active period. This is used in Lemma 6 to establish a relation between potential and provided service.

Lemma 5. *During an active period* $[\tau_1, \tau_2]$*, it holds that*

$$\forall t \in [\tau_1, \tau_2].\ \pi(t) = \hat{w}'(\tau_1) - w'(\tau_1) + \hat{w}'(\tau_1, t-1) - w'(\tau_1, t-1) \tag{4.3}$$

Proof. Rewriting the right hand side according to Definition 2 yields $\hat{w}'(\tau_1) - w'(\tau_1) + \hat{w}'(t-1+1) - \hat{w}'(\tau_1) - (w'(t-1+1) - w'(\tau_1))$. According to the definition of potential in Definition 26, this is equivalent to $\pi(\tau_1) + \pi(t) - \pi(\tau_1) = \pi(t)$. □

Lemma 6. *During an active period* $[\tau_1, \tau_2]$*, it holds that* $\forall t \in [\tau_1, \tau_2].\ \pi(t) \leq \sigma' - \rho' \iff w'(\tau_1, t-1) \geq \rho' \cdot (t - \tau_1 + 1)$.

Proof. We know from Lemma 5 that Equation (4.3) holds during an active period $[\tau_1, \tau_2]$. Definition 25 and the fact that the requestor is inactive at $\tau_1 - 1$ results in $\hat{w}'(\tau_1) - w'(\tau_1) = \sigma'$, and $\hat{w}'(\tau_1, t-1) = (t - \tau_1) \cdot \rho'$. Substituting these results into Equation (4.3) yields

$$\pi(t) = \sigma' + (t - \tau_1) \cdot \rho' - w'(\tau_1, t-1) \leq \sigma' - \rho'$$

The proof is concluded by solving for $w'(\tau_1, t-1)$. □

We proceed by showing that there is a relation between the potential and being live for non-backlogged requestors in Lemma 7. Lastly, we bound the increase in the upper bound on provided service during an interval in Lemma 8.

Lemma 7. *For a requestor* $r \in R$ *during an active period* $[\tau_1, \tau_2]$*, it holds that* $\forall t \in [\tau_1, \tau_2],\ q_r(t) = 0.\ \pi_r(t) \leq \sigma'_r - \rho'_r \iff w(\tau_1 - 1, t-1) \geq \rho' \cdot (t - \tau_1 + 1)$.

Proof. According to Lemma 6, we know $w'_r(t) - w'_r(\tau_1) \geq \rho'_r \cdot (t - \tau_1 + 1) \iff \pi_r(t) \leq \sigma'_r - \rho'_r$. Definition 14 states that $w'_r(t) = w_r(t)$ since $q_r(t) = 0$. From Lemma 4, we additionally know that $w'_r(\tau_1) = w_r(\tau_1 - 1)$. We conclude the proof by substituting these results into the expression from Lemma 6. □

Lemma 8. *It holds that* $\hat{w}'_r(\tau, t) \leq \rho' \cdot (t - \tau + 1)$.

Proof. We prove the lemma by showing that $\hat{w}'_r(\tau, t)$ is maximized if $\tau, t \in [\tau_1, \tau_2]$, where $[\tau_1, \tau_2]$ is an active period. This in turn is proven by showing that the first rule of Equation (4.1) implies $\hat{w}'_r(t+1) > \hat{w}'_r(t)$, while the second rule implies $\hat{w}'_r(t+1) \leq \hat{w}'_r(t)$.

The first rule in Equation (4.1) implies that $\hat{w}'_r(t+1) > \hat{w}'_r(t)$, since it follows from Definition 17 and Definition 19 that $\rho'_r \geq 0$.

We split the analysis of the second rule in Equation (4.1) into two cases. In the first case, the requestor is inactive at both $t - 1$ and t, corresponding to multiple cycles of inactivity. In the second case, the requestor is active at $t-1$ and inactive at t, meaning it is ending its active period.

Case 1: $r \notin R^a_{t-1} \wedge r \notin R^a_t$
From the second rule in Equation (4.1), we get that $\hat{w}'_r(t+1) = w'_r(t) + \sigma'_r$. Since an inactive requestor cannot be scheduled, it must hold that $w'_r(t) = w'_r(t-1)$. It hence follows that $\hat{w}'_r(t+1) = \hat{w}'_r(t)$ if $r \notin R^a_{t-1} \wedge r \notin R^a_t$

Case 2: $r \in R^a_{t-1} \wedge r \notin R^a_t$
We proceed by showing that $\hat{w}'_r(t+1) < \hat{w}'_r(t)$ if $r \in R^a_{t-1} \wedge r \notin R^a_t$. Let $t = \tau_2 + 1$, where $[\tau_1, \tau_2]$ defines an active period. We must hence show that

$$\hat{w}'_r(\tau_2 + 2) < \hat{w}'_r(\tau_2 + 1) \tag{4.4}$$

According to Definition 2, $\hat{w}'_r(\tau_2+1) = \hat{w}'_r(\tau_1)+\hat{w}'_r(\tau_1, \tau_2)$. From Lemma 4 and the second rule in Equation (4.1), we get that $\hat{w}'_r(\tau_1) = w'_r(\tau_1 - 1) + \sigma'_r = w'_r(\tau_1) + \sigma'_r$, since $r \notin R^a_{\tau_1 - 1}$. We furthermore know from the first rule in Equation (4.1) that $\hat{w}'_r(\tau_1, \tau_2) = \rho'_r \cdot (\tau_2 - \tau_1 + 1)$, since $\forall t \in [\tau_1, \tau_2]$. $r \in R^a_t$. This results in

$$\hat{w}'_r(\tau_2 + 1) = w'_r(\tau_1) + \sigma'_r + \rho'_r \cdot (\tau_2 - \tau_1 + 1) \tag{4.5}$$

The second rule in Equation (4.1) states that $\hat{w}'_r(\tau_2+2) = w'_r(\tau_2+1)+\sigma'$ since $r \notin R^a_{\tau_2+1}$. Rewriting this using Definition 2 results in $\hat{w}'_r(\tau_2 + 2) = w'_r(\tau_1) + w'_r(\tau_1, \tau_2) + \sigma'_r$. From Definition 20 and Lemma 4, we know that $r \notin R^a_{\tau_2+1} \Rightarrow w'_r(\tau_1 - 1, \tau_2) = w_r(\tau_1 - 1, \tau_2) < \rho'_r \cdot (\tau_2 - \tau_1 + 1)$, as the requestor is neither live nor backlogged at $\tau_2 + 1$. Putting these results together gives us

$$\hat{w}'_r(\tau_2 + 2) < w'_r(\tau_1) + \sigma'_r + \rho'_r \cdot (\tau_2 - \tau_1 + 1) \tag{4.6}$$

By substituting Equation (4.5) and Equation (4.6) into Equation (4.4), we see that $\hat{w}'_r(\tau_2 + 2) < \hat{w}'_r(\tau_2 + 1)$.

We hence conclude that $\hat{w}'_r(\tau, t)$ is maximized if $\tau, t \in [\tau_1, \tau_2]$, where $[\tau_1, \tau_2]$ is an active period. According to the first rule of Definition 27, this implies that $\hat{w}'_r(\tau, t) \leq \rho' \cdot (t - \tau + 1)$. □

4.3 Scheduler

A scheduler controls the latency of requestors by controlling the order in which they access the resource. A scheduler is either *preemptive* or *non-preemptive* depending on the nature of the resource. A preemptive scheduler can interrupt a request in progress to make a new scheduling decision. A non-preemptive scheduler, however, must finish serving a scheduled request before a new decision is made. In this document, we consider both preemptive and non-preemptive scheduling. We initially assume a non-preemptive scheduler, but return to discuss preemptive scheduling in Section 6.1.

The CCSP arbiter uses a static-priority scheduler. A static-priority scheduler has three important features: 1) it decouples latency and rate, 2) it is cheap to implement in hardware, and 3) it is convenient to analyze. Each requestor is assigned a priority level, p, as stated in Definition 31, where a lower level indicates higher priority. We do not allow requestors to share priority levels. Sharing priorities, as done in [14], results in a situation where equal priority requestors must assume that they all have to wait for each other in the worst-case, resulting in less tight bounds. Alternatively, a second-level arbiter, such as round-robin, is required to break ties, at the cost of increased area and reduced clock speed of the implementation. We use the priority

level to define the set of requestors that have higher priority and lower priority than a particular requestor in Definition 32 and Definition 33, respectively.

Definition 31 (Priority level). *A requestor $r \in R$ has a priority level p_r, such that $\forall r_i, r_j \in R.\ p_{r_i} \neq p_{r_j}$.*

Definition 32 (Set of higher priority requestors). *The set of requestors with higher priority than $r_i \in R$ is denoted $R^+_{r_i}$, and is defined as $R^+_{r_i} = \{r_j \mid \forall r_j \in R \wedge p_{r_i} > p_{r_j}\}$.*

Definition 33 (Set of lower priority requestors). *The set of requestors with lower priority than $r_i \in R$ is denoted $R^-_{r_i}$, and is defined as $R^-_{r_i} = \{r_j \mid \forall r_j \in R \wedge p_{r_i} < p_{r_j}\}$.*

An idle non-preemptive non-work-conserving static-priority scheduler operates by picking the highest priority requestor that is eligible for scheduling. Once a request is chosen by the scheduler, it is scheduled every cycle until its finishing time, as stated in Definition 13. No requestor is scheduled if there are no eligible requestors, causing the resource to idle, even though there may be backlogged requestors. The operation of a non-preemptive (np) non-work-conserving (nwc) static-priority scheduler is formally defined in Definition 34. We examine the case of work conservation in Section 6.2. The scheduled requestor is uniquely defined, even though a set notation is used in the first two cases in Equation (4.7). In the first case, the uniqueness stems from that only a single request can be in progress of receiving service in a non-preemptive arbiter, according to Definition 13, and in the second case because Definition 31 states that there can be only one eligible requestor with the lowest priority.

Definition 34 (Non-preemptive non-work-conserving static-priority scheduler). *The scheduled requestor at a time t in a non-preemptive non-work-conserving static-priority scheduler is denoted $\gamma^{nwc}_{np}(t) : \mathbb{N} \rightarrow R^e_t \cup \{\varnothing\}$, and is defined as*

$$\gamma^{nwc}_{np}(t) = \begin{cases} \{r_i \mid \exists k \in \mathbb{N}.\ t_s(\omega^k_{r_i}) < t < t_f(\omega^k_{r_i})\} & \exists \omega^k_{r_i}.\ t_s(\omega^k_{r_i}) < t < t_f(\omega^k_{r_i}) \wedge R^e_t \neq \varnothing \\ \{r_i \mid r_i \in R^e_t \wedge \nexists r_j \in R^e_t.\ p_{r_j} < p_{r_i}\} & \nexists \omega^k_{r_i}.\ t_s(\omega^k_{r_i}) < t < t_f(\omega^k_{r}) \wedge R^e_t \neq \varnothing \\ \varnothing & R^e_t = \varnothing \end{cases} \tag{4.7}$$

Section 5

Arbiter Analysis

In this section, we derive analytical properties of the CCSP arbiter. We begin in Section 5.1 by upper bounding interference in an interval. We use this result to derive the service guarantee of CCSP, and to prove that CCSP belongs to the class of $\mathcal{LR}$ servers. The interference bound, and hence the service guarantee depends on the service allocation of the requestors, which is discussed in Section 5.2. Lastly, we show in Section 5.3 that active period rate regulation does not result in a burstier characterization of w', nor requires additional request buffering or increase maximum delay, compared to enforcing a (σ, ρ) constraint on the requested service.

5.1 Latency Analysis

In order to perform a latency analysis, we require interference to be defined and bounded. Interference in a CCSP arbiter consists of two parts. The first part is interference due to blocking that accounts for interference from lower priority requestors that occurs if the arbiter is non-preemptive. The second part is interference due to preemption, accounting for interference from higher priority requestors.

We start by defining blocking in Definition 35. We then proceed by defining the aggregate potential of a set of requestors in Definition 36, as this is required for our definition of preemption in Definition 37. After defining blocking and preemption, we define interference in Definition 38. Our definitions of blocking and interference make an assumption about the eligibility of requestors. We discuss how to allocate service to deliver on this assumption in Section 5.2.

Definition 35 (Blocking). *Blocking of a requestor $r_i \in R$ in a non-preemptive non-work-conserving arbiter during any interval $[\tau_1, \tau_2]$, for which it holds that $\forall t \in [\tau_1, \tau_2].\ r_i \in R^e_t$, is denoted $b^{nwc}_{r_i} : R \rightarrow \mathbb{N}$, and is defined as the number of interfering service units provided to a requestor $r_j \in R^-_{r_i}$.*

Definition 36 (Aggregate potential). *The aggregate potential of a set of requestors $R' \subseteq R$ is defined according to*

$$\sum_{\forall r \in R'} \pi_r(t) = \sum_{\forall r \in R'} \hat{w}'_r(t) - \sum_{\forall r \in R'} w'_r(t)$$

Definition 37 (Preemption). *Preemption of a requestor $r_i \in R$ in an interval $[\tau_1, \tau_2]$ is denoted $\phi_{r_i}(\tau_1, \tau_2) : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{R}$, and is defined as*

$$\phi_{r_i}(\tau_1, \tau_2) = \sum_{\forall r_j \in R_{r_i}^+} \pi_{r_j}(\tau_1) + \hat{w}'_{r_j}(\tau_1, \tau_2) \tag{5.1}$$

Definition 38 (Interference). *The interference of a requestor $r \in R$ during an interval $[\tau_1, \tau_2]$, for which it holds that $\forall t \in [\tau_1, \tau_2].\ r \in R_t^e$, is denoted $i_r(\tau_1, \tau_2) : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{R}$, and is defined as $i_r(\tau_1, \tau_2) = b_r + \phi_r(\tau_1, \tau_2)$.*

Having defined interference, we proceed with the latency analysis by upper bounding it. Bounding interference is accomplished by deriving upper bounds on each of its two parts. We start by deriving an upper bound on blocking in Lemma 9.

Lemma 9. *A requestor $r_i \in R$ in a non-preemptive non-work-conserving arbiter can maximally be blocked during any interval $[\tau_1, \tau_2]$, for which it holds that $\forall t \in [\tau_1, \tau_2].\ r_i \in R_t^e$, according to*

$$\hat{b}_{r_i}^{nwc} = \max_{\forall r_j \in R_{r_i}^-} \hat{s}_{r_j} - 1 \tag{5.2}$$

Proof. The maximum blocking of a requestor $r_i \in R$ in a non-preemptive non-work-conserving arbiter occurs when a request, $\omega_{r_i}^k$, becomes eligible at $t_e^{np}(\omega_{r_i}^k) = \tau_1$, and a maximally sized request from the lower priority requestor with the largest request size was scheduled at $t_e^{np}(\omega_{r_i}^k) - 1$. In this case, r_i may be blocked by up to $\max_{\forall r_j \in R_{r_i}^-} \hat{s}_{r_j} - 1$ service units by a lower priority requestor. □

Before deriving an upper bound on preemption, we require some additional lemmas that relate the active, backlogged and eligible states using potential.

Lemma 10. *For a requestor $r \in R$, it holds that $\forall t \in \mathbb{N},\ q_r(t) = 0.\ \pi_r(t) > \sigma'_r - \rho'_r \Rightarrow r \notin R_t^a$.*

Proof. We get from Lemma 7 that $q_r(t) = 0$ and $\pi_r(t) > \sigma'_r - \rho'_r$ implies that $w_r(\tau_1 - 1, t-1) < \rho'_r \cdot (t - \tau_1 + 1)$, where τ_1 is the start of the last active period. By negating Definition 20, we know that $r \notin R_t^a$ iff $q_r(t) = 0$ and $w_r(\tau_1 - 1, t - 1) < \rho'_r \cdot (t + 1 - \tau_1)$. □

Lemma 11. *For a requestor $r \in R_t^a.\ \pi_r(t) > \sigma'_r - \rho'_r \Rightarrow q_r(t) > 0$.*

Proof. We prove the lemma by contradiction. We know that $r \in R_t^a$ and $\pi_r(t) > \sigma'_r - \rho'_r$. Lemma 10 states that if $q_r(t) = 0$ then $r \notin R_t^a$, which is a contradiction. Hence, it must hold that $q_r(t) > 0$ □

Lemma 12. *For a requestor $r \in R_t^a.\ \pi_r(t) > \sigma'_r - \rho'_r \Rightarrow r \in R_t^e$.*

Proof. We must show that for a requestor $r \in R_t^a.\ \pi_r(t) > \sigma'_r - \rho'_r$ implies that the eligibility criterion in Definition 29 is satisfied. This results in two cases in a non-preemptive non-work-conserving arbiter depending on whether a request is in progress of receiving service or not.

Case 1: $\exists \omega_r^k.\ t_s(\omega_r^k) < t < t_{\mathrm{f}}(\omega_r^k)$.
This case is in itself a sufficient for r to be eligible at t since $t_s(\omega_r^k) \geq t_e^{np}(\omega_r^k)$ in a non-work-conserving arbiter and Definition 29 states that $\forall t \in [t_e^{np}(\omega_r^k), t_{\mathrm{f}}(\omega_r^k) - 1].\ r \in R_t^e$.

Case 2: $\nexists \omega_r^k.\ t_s(\omega_r^k) < t < t_{\mathrm{f}}(\omega_r^k)$.
In this case the requestor is not eligible unless $t_e^{np}(\omega_r^k) \leq t < t_s(\omega_r^k)$, which requires the three conditions in Definition 28 to be satisfied. The first condition is satisfied since the definition of the second case states that $\nexists \omega_r^k.\ t_s(\omega_r^k) < t < t_{\mathrm{f}}(\omega_r^k)$. We know from Lemma 11 that $r \in R_t^a$ and $\pi_r(t) > \sigma'_r - \rho'_r$ implies $q_r(t) > 0$. This satisfies the second condition since Definition 8 states that a request is not accounted for until it is completely arrived. The third condition is satisfied since $\pi_r(t) > \sigma'_r - \rho'_r \geq \hat{s}_r - \rho'_r$, according to Definition 19. □

Lemma 13 is a key element in our analysis, as it shows that the aggregate potential of a set of requestors cannot increase, as long as a requestor in the set is scheduled every cycle. This result allows us to derive an upper bound on aggregate potential of higher priority requestors in Lemma 14, which is essential when bounding preemption in Lemma 15. Lemma 14 and Lemma 15 both consider a preemptive arbiter and does not take blocking into account. We account for the extra aggregate potential that is built up during blocking when bounding maximum interference in Lemma 16.

Lemma 13. *For a set of requestors $R' \subseteq R$, it holds that*

$$\exists r_k \in R'.\ \gamma(t) = r_k \Rightarrow \sum_{\forall r \in R'} \pi_r(t+1) \leq \sum_{\forall r \in R'} \pi_r(t)$$

Proof. According to Definition 2 and the definition of aggregate potential in Definition 36

$$\sum_{\forall r \in R'} \pi_r(t+1) = \sum_{\forall r \in R'} \pi_r(t) + \sum_{\forall r \in R'} \hat{w}'_r(t,t) - \sum_{\forall r \in R'} w'_r(t,t)$$

According to Lemma 8, $\sum_{\forall r \in R'} \hat{w}'(t,t) \leq \sum_{\forall r \in R'} \rho'_r$, where equality is reached if all requestors are active at t. From Definition 11, we also get that $\sum_{\forall r \in R'} w'_r(t,t) = 1$ if a requestor in R' is scheduled at t. Hence, if $\forall r \in R'.\ r \in R_t^a$ and $\exists r_k \in R'.\ \gamma(t) = r_k$, then

$$\sum_{\forall r \in R'} \pi_r(t+1) \leq \sum_{\forall r \in R'} \pi_r(t) + \sum_{\forall r \in R'} \rho'_r - 1$$

The proof is concluded by observing that $\sum_{\forall r \in R'} \rho'_r \leq 1$, according to Definition 19. □

Lemma 14. *For a requestor $r_i \in R$ in a preemptive arbiter, it holds that $\forall t \in \mathbb{N}.\ \sum_{\forall r_j \in R_{r_i}^+} \pi_{r_j}(t) \leq \sum_{\forall r_j \in R_{r_i}^+} \sigma'_{r_j}$. The maximum value occurs at any time t for which $\forall r_j \in R_{r_i}^+.\ r_j \notin R_{t-1}^a$.*

Proof. We prove the lemma by induction on t.
Base case: The lemma holds at $t = 0$, since Definition 27 states that $\forall r \in R.\ \pi_r(0) = \sigma'_r$.

Inductive step: For the inductive step, we prove that if the lemma holds at a time t then it also holds for $t+1$. We examine two different cases for cycle t. In the first case there exists a higher priority eligible requestor, and in the second case there does not.
Case 1: $(R_{r_i}^+ \cap R_t^e) \neq \varnothing$
This case implies that $\exists r_k \in R_t^e.\ \gamma(t) = r_k$. Since a static-priority scheduler schedules the highest priority eligible requestor, according to Definition 34, it follows that $r_k \in R_{r_i}^+$. Applying Lemma 13 hence gives us

$$\sum_{\forall r_j \in R_{r_i}^+} \pi_{r_j}(t+1) \leq \sum_{\forall r_j \in R_{r_i}^+} \pi_{r_j}(t) \leq \sum_{\forall r_j \in R_{r_i}^+} \sigma'_{r_j}$$

The equation shows that if the lemma holds at t, then it also holds at $t+1$, proving the first case.
Case 2: $(R^+_{r_i} \cap R^e_t) = \varnothing$
No higher priority requestor is eligible in this case. We will show that this implies that $\pi(t+1) \leq \sigma'$ both for requestors with $\pi(t) > \sigma' - \rho'$ and $\pi(t) \leq \sigma' - \rho'$.

According to Lemma 12, it must hold that $\forall r_j \in R^+_{r_i}, \pi_{r_j}(t) > \sigma'_{r_j} - \rho'_{r_j}.\ r_j \notin R^a_t$. The third rule of Definition 27 hence states that $\forall r_j \in R^+_{r_i}, \pi_{r_j}(t) > \sigma'_{r_j} - \rho'_{r_j}.\ \pi_{r_j}(t+1) = \sigma'_{r_j}$. It furthermore follows from Definition 27 that $\forall r_j \in R^+_{r_i}, \pi_{r_j}(t) \leq \sigma'_{r_j} - \rho'_{r_j}.\ \pi_{r_j}(t+1) \leq \sigma'_{r_j}$. It hence holds that $\forall r_j \in R^+_{r_i}.\ \pi_{r_j}(t+1) \leq \sigma'_{r_j}$. This means that $\sum_{\forall r_j \in R^+_{r_i}} \pi_{r_j}(t+1) \leq \sum_{\forall r_j \in R^+_{r_i}} \sigma'_{r_j}$, which proves the second case.

We conclude that the aggregate potential of higher priority requestors is maximized if $\forall r_j \in R^+_{r_i}.\ \pi_{r_j}(t) = \sigma'_{r_j}$, which occurs at any time t for which $\forall r_j \in R^+_{r_i}.\ r_j \notin R^a_{t-1}$. □

Lemma 15. *A requestor $r_i \in R$ in a preemptive arbiter can maximally be preempted for a time $\hat{\phi}_{r_i}(\tau_1, \tau_2)$ during an interval $[\tau_1, \tau_2]$, for which it holds that $\forall t \in [\tau_1, \tau_2].\ r_i \in R^e_t$, according to*

$$\hat{\phi}_{r_i}(\tau_1, \tau_2) = \sum_{\forall r_j \in R^+_{r_i}} \sigma'_{r_j} + \rho'_{r_j} \cdot (\tau_2 - \tau_1 + 1) \tag{5.3}$$

The maximum preemption occurs when all higher priority requestors start an active period at τ_1, and remain active $\forall t \in [\tau_1, \tau_2]$.

Proof. We know from Equation (5.1) that preemption is defined as $\phi_{r_i}(\tau_1, \tau_2) = \sum_{\forall r_j \in R^+_{r_i}} \pi_{r_j}(\tau_1) + \hat{w}'_{r_j}(\tau_1, \tau_2)$. From Lemma 14, we know that that $\sum_{\forall r_j \in R^+_{r_i}} \pi_{r_j}(\tau_1) \leq \sum_{\forall r_j \in R^+_{r_i}} \sigma'_{r_j}$. This expression is maximized if all higher priority requestors are inactive at $\tau_1 - 1$. We furthermore know from Lemma 8 that $\sum_{\forall r_j \in R^+_{r_i}} \hat{w}'_{r_j}(\tau_1, \tau_2) \leq \sum_{\forall r_j \in R^+_{r_i}} \rho'_{r_j} \cdot (\tau_2 - \tau_1 + 1)$, which is maximized if $\forall t \in [\tau_1, \tau_2].\ r_j \in R^a_t$. Hence, $\hat{\phi}_{r_i}(\tau_1, \tau_2) = \sum_{\forall r_j \in R^+_{r_i}} \sigma'_{r_j} + \rho'_{r_j} \cdot (\tau_2 - \tau_1 + 1)$, and the maximum occurs when all higher priority requestors start an active period at τ_1, and remain active $\forall t \in [\tau_1, \tau_2]$. □

After bounding blocking and preemption, we are ready to derive an upper bound on maximum interference in Lemma 16. Note that maximum preemption accounts for the increase in aggregate potential while the higher priority requestors are blocked.

Lemma 16 (Maximum interference). *The maximum interference experienced by a requestor $r \in R$ during an interval $[\tau_1, \tau_2]$, for which it holds that $\forall t \in [\tau_1, \tau_2].\ r_i \in R^e_t$, equals*

$$\hat{i}_{r_i}(\tau_1, \tau_2) = \hat{b}_{r_i} + \sum_{\forall r_j \in R^+_{r_i}} \sigma'_{r_j} + \rho'_{r_j} \cdot (\tau_2 - \tau_1 + 1) \tag{5.4}$$

The maximum interference occurs when all higher priority requestors start an active period at τ_1 and remain active $\forall t \in [\tau_1, \tau_2]$.

Proof. The interference in an interval $[\tau_1, \tau_2]$ is defined as $i_r(\tau_1, \tau_2) = b_{r_i} + \phi_{r_i}(\tau_1, \tau_2)$, according to Definition 38. We conclude the proof by using the bounds on blocking and preemption from Equation (5.2) and Equation (5.3), respectively. It follows from Lemma 15 that the worst-case occurs when all higher priority requestors start an active period at τ_1, and remain active $\forall t \in [\tau_1, \tau_2]$. □

We are now ready to derive the service guarantee of a CCSP arbiter, and to compute its service latency. This is done in Theorem 1.

Theorem 1 (Service guarantee). *An active requestor $r_i \in R$ is guaranteed a minimum service during an active period $[\tau_1, \tau_2]$ according to $\forall t \in [\tau_1, \tau_2]$. $\check{w}'_{r_i}(\tau_1, t) = \max(0, \rho'_{r_i} \cdot (t - \tau_1 + 1 - \Theta_{r_i}))$, where*

$$\Theta_{r_i} = \frac{\hat{b}_{r_i} + \sum_{\forall r_j \in R^+_{r_i}} \sigma'_{r_j}}{1 - \sum_{\forall r_j \in R^+_{r_i}} \rho'_{r_j}} \tag{5.5}$$

Proof. It suffices to show that the theorem holds for intervals where $\tau_2 - \tau_1 + 1 > \Theta_{r_i}$, as these are the only intervals for which $\check{w}'_{r_i}(\tau_1, \tau_2) > 0$. For these intervals, we must show that

$$\forall t \in [\tau_1, \tau_2].\ \check{w}'_{r_i}(\tau_1, t) = \rho'_{r_i} \cdot (t - \tau_1 + 1 - \Theta_{r_i}) \tag{5.6}$$

We prove the theorem by splitting the active period in two cases. In the first case, we look at the behavior of r_i during backlogged periods within the active period, where the k:th backlogged period is denoted $[\alpha_k, \beta_k]$. Just like in Lemma 16, it is assumed that $\forall t \in [\alpha_k, \beta_k].\ r_i \in R^e_t$. In the second case, the requestor is in a live and not backlogged state.

Case 1: $\forall t \in [\alpha_k, \beta_k].\ r_i \in R^e_t$

There are $(\beta_k - \alpha_k + 1)$ units of service available in the backlogged interval. An eligible requestor in a static-priority scheduler cannot access the resource whenever it is used by higher priority requestors, or when it is blocked. The minimum service available to r_i, denoted $\check{w}^a_{r_i}$, can hence be expressed according to

$$\check{w}^a_{r_i}(\alpha_k, \beta_k) = \beta_k - \alpha_k + 1 - \hat{i}_{r_i}(\alpha_k, \beta_k)$$

Since r_i is continuously backlogged and eligible in the interval, it follows that $\check{w}'_{r_i}(\alpha_k, \beta_k) = \check{w}^a_{r_i}(\alpha_k, \beta_k)$. We proceed by using the result from Lemma 16 to bound the maximum possible interference.

$$\check{w}'_{r_i}(\alpha_k, \beta_k) = \check{w}^a_{r_i}(\alpha_k, \beta_k) = \beta_k - \alpha_k + 1 - \sum_{\forall r_j \in R^+_{r_i}} \sigma'_{r_j} - \sum_{\forall r_j \in R^+_{r_i}} \rho'_{r_j} \cdot (\beta_k - \alpha_k + 1) - \hat{b}_{r_i} \tag{5.7}$$

We combine the results from Equation (5.6) and Equation (5.7), resulting in

$$\beta_k - \alpha_k + 1 - \sum_{\forall r_j \in R^+_{r_i}} \sigma'_{r_j} - \sum_{\forall r_j \in R^+_{r_i}} \rho'_{r_j} \cdot (\beta_k - \alpha_k + 1) - \hat{b}_{r_i} = \rho'_{r_i} \cdot (\beta_k - \alpha_k + 1 - \Theta_{r_i})$$

We replace ρ'_{r_i} by $1 - \sum_{\forall r_j \in R^+_{r_i}} \rho'_{r_j}$, which is valid since $1 - \sum_{\forall r_j \in R^+_{r_i}} \rho'_{r_j} \geq \rho'_{r_i}$, according to Definition 19. Solving for Θ_{r_i} results in Equation (5.5), proving the first case. *Case 2:* $r_i \in R^l_t \wedge r_i \notin R^q_t$ According to Definition 21, $r_i \in R^l_t$ implies that $\check{w}_{r_i}(\tau_1 - 1, t-1) = \rho'_{r_i} \cdot (t - \tau_1 + 1)$. On the other hand, Definition 14 states that $r_i \notin R^q_t$ means that $w_{r_i}(t) = w'_{r_i}(t)$. By combining these results we get that

$$\check{w}'_{r_i}(\tau_1 - 1, t - 1) = \rho'_{r_i} \cdot (t - \tau_1 + 1) \tag{5.8}$$

We know from Lemma 4 that $w'_{r_i}(\tau_1 - 1) = w'_{r_i}(\tau_1)$. We also know from Definition 11 that $w'_{r_i}(t, t) \geq 0$. Substituting these results into Equation (5.8) gives us $\check{w}_{r_i}(\tau_1, t) = \rho'_{r_i} \cdot (t - \tau_1 + 1)$, which proves the second case. □

Next, we show in Theorem 2 that CCSP belongs to the class of $\mathcal{LR}$ servers.

Theorem 2 ($\mathcal{LR}$ server). *A CCSP arbiter belongs to the class of $\mathcal{LR}$ servers, and the service latency of a requestor $r_i \in R$ equals*

$$\Theta_{r_i} = \frac{\hat{b}_{r_i} + \sum_{\forall r_j \in R_{r_i}^+} \sigma'_{r_j}}{1 - \sum_{\forall r_j \in R_{r_i}^+} \rho'_{r_j}}$$

Proof. According to [18], it is sufficient to show that $\forall t \in [\tau_1, \tau_2].\ r \in R_t^e \Rightarrow \breve{w}'_{r_i}(\tau_1, \tau_2) = \max(0, \rho'_{r_i} \cdot (\tau_2 - \tau_1 + 1 - \Theta_{r_i}))$. This is shown in the first case of the proof of Theorem 1. □

Theorem 2 proves that CCSP belongs to the class of $\mathcal{LR}$ servers. Our derived service latency is furthermore the same as that of SRSP, derived in [21]. This is the case even though the CCSP regulator admits burstier traffic, as we will show in Section 5.3.

The service latencies of several schedulers are presented in [18]. We compare the service latency of our arbiter to that of Packet-level GPS [19] (PGPS) and Deficit Round-Robin [5] (DRR). These schedulers are chosen as PGPS has properties that are very typical for the Fair-Queuing family [6] of schedulers, and DRR represents a typical frame-based scheduler. We have translated their respective service latencies to fit with our notation. The service latency of PGPS, shown in Equation (5.9), depends on the maximum packet size of the requestors in the system. This reflects that a PGPS server is never more than one packet behind an ideal server where service is infinitely divisible, such as GPS [19]. It is apparent from Equation (5.9) that given a set of requestors, we can only reduce service latency by increasing the allocated rate.

$$\Theta_r^{PGPS} = \frac{\hat{s}_r}{\rho'_r} + \max_{\forall r \in R} \hat{s}_r \tag{5.9}$$

The service latency of DRR is shown in Equation (5.10), where F is the frame size of the scheduler and ϕ the allocation of a requestor within a frame. From this it follows that $\rho' = \phi/F$. We see in the equation that the service latency is coupled to the allocated rate, which implies that a high allocated rate is required to provide low latency. We furthermore note that the service latency is proportional to the frame size. A small frame size is hence required to provide low latency, although this increases over-allocation, as we will see in Section 7.

$$\Theta_r^{DRR} = 3 \cdot F - 2 \cdot \phi_r = F \cdot (3 - 2 \cdot \rho'_r) \tag{5.10}$$

We see in Equation (5.5) that service latency and allocated rate is decoupled in CCSP by means of priorities. This allows us to provide low service latency to requestors with low service requirements without over-allocating resources. This is a benefit of priority schedulers.

5.2 Configuration

In this section, we discuss how to allocate service to requestors in order to deliver on the assumption on eligibility from Section 5.1.

The maximum interference bound in Lemma 16, and hence the results of Theorem 1 and Theorem 2, are not valid for requestors that are not eligible during backlogged periods. The reason for this is that a non-eligible requestor may encounter additional interference from lower priority requestors. By negating the eligibility criterion in Definition 29, we learn that non-eligible requestors are requestors that: 1) do not have requests pending, nor have requests in progress of being served, or 2) do not have sufficient potential for their pending requests to be served. We

are not interested in latency bounds for requests that have not arrived, and hence disregard of the first case. The second case, however, implies that the arbiter must be configured such that the requestor always has enough potential to serve a pending request, and is not slowed down by the rate regulator. We get from Definition 19 that $\forall r \in R.\ \rho'_r \geq \rho_r$. The allocated burstiness is configured slightly differently depending on the real-time requirements of a requestor. Hard real-time requestors require $\sigma' \geq \sigma$, to guarantee enough potential, assuming that the requestor sticks to its characterization. We configure these requestors according to $\sigma' = \sigma$, since there is no benefit in allocating higher burstiness than requested. Configuring $\sigma' < \sigma$ causes the regulator to limit the burstiness of a requestor. This is useful to protect hard real-time requestors from bursty soft real-time requestors that are not interested in service latency bounds. Note, however, that these requestors are still guaranteed their allocated service, although with a service latency that is maximally $s(\omega^k)/\rho'$ longer than derived in Equation (5.5). This extra time accounts for the maximum time required to build up sufficient potential to become eligible, required by Definition 28. We return to the topic of service allocation in Section 8, as we experimentally study its impact.

As shown in Theorem 1, the CCSP arbiter guarantees the allocated service rate within a maximum latency. However, a structured methodology is required to derive this configuration given a set of application requirements. This is a challenging problem that may require a heuristic approach, as the latency of the CCSP arbiter, expressed by Equation (5.5), is a non-linear equation that does not lend itself to integer-linear programming, and the configuration space is furthermore likely to be too large to allow exhaustive searches. This topic is out of the scope of this document and is left as future work.

5.3 Arbiter characterization

We derived the service latency of the CCSP arbiter in Section 5.1. We proceed in this section by using this result to show that CCSP admits burstier traffic than an SRSP arbiter without resulting in a burstier characterization of w'. We furthermore derive upper bounds on backlog and delay. To facilitate this, we require some additional lemmas. We start by deriving the maximum potential of a requestor in Lemma 17.

Lemma 17. *The maximum potential of a requestor $r \in R$ during an active period $[\tau_1, \tau_2]$ equals*

$$\hat{\pi}_r(t) = \sigma'_r + \rho'_r \cdot \Theta_r$$

Proof. From Lemma 5, we know that Equation (4.3) holds during an active period $[\tau_1, \tau_2]$ and hence that $\forall t \in [\tau_1, \tau_2]$. $\pi(t) = \hat{w}'(\tau_1) - w'(\tau_1) + \hat{w}'(\tau_1, t-1) - w'(\tau_1, t-1)$. We know from Definition 27 and the fact that the requestor is inactive at $\tau_1 - 1$ that $\hat{w}'_r(\tau_1) - w'_r(\tau_1) = \sigma'_r$. We also know from Lemma 8 that $\hat{w}'_r(\tau_1, t-1) \leq \rho'_r \cdot (t - \tau_1)$. To get the maximum potential, we use $\check{w}'_r$ instead of w'_r in Equation (4.3), which according to Theorem 1 equals $\check{w}'_r(\tau_1, t-1) = \max(0, \rho'_r \cdot (t - \tau_1 - \Theta_r))$. Substituting these results into Equation (4.3) gives us

$$\hat{\pi}_r(t) \leq \sigma'_r + \rho'_r \cdot (t - \tau_1) - \rho'_r \cdot \max(0, t - \tau_1 - \Theta_r)$$

The equation is maximized for intervals satisfying $t - \tau_1 + 1 \geq \Theta_r$, for which it equals $\hat{\pi}_r(t) = \sigma'_r + \rho'_r \cdot \Theta_r$. □

Next, we derive a measure on how much service that can be admitted by the rate regulator. For this purpose, we use the the eligible service curve, w^*, defined in Definition 39. The eligible

service curve represents the amount of service that has become eligible at a particular time. The definition is consistent with the eligibility criterion in Definition 29, since we know that $\hat{w}'(t) \geq w'(t)$ and $w(t) \geq w'(t)$ from Lemma 3 and Corollary 1, respectively. A (σ, ρ) characterization of w^* is derived in Lemma 18.

Definition 39 (Eligible service curve). *The eligible service curve of a requestor $r \in R$ is denoted $w_r^*(t) : \mathbb{N} \to \mathbb{N}$, and is defined as $w_r^*(t) = \min(\hat{w}_r'(t), w_r(t))$.*

Lemma 18. *w_r^* of a requestor $r \in R$ is (σ, ρ) constrained with parameters $(\sigma_r' + \rho_r' \cdot \Theta_r, \rho_r')$.*

Proof. We calculate the upper bound on w_r^* during an active period $[\tau_1, \tau_2]$, since an inactive requestor is not backlogged, according to Definition 20, and hence cannot be scheduled. From Definition 2 and Definition 39, we derive that $\forall \tau', t \in [\tau_1, \tau_2], \tau' < t$

$$w_r^*(\tau', t) = \min(\hat{w}_r'(t+1), w_r(t+1)) - \min(\hat{w}_r'(\tau'), w_r(\tau')) \tag{5.11}$$

The equation is maximized at t by maximizing the first term and minimizing the second, which happens if $w_r(\tau') = \check{w}_r(\tau')$ and $w_r(t+1) \geq \hat{w}_r(t+1)$. This results in $w_r^*(\tau', t) = \hat{w}_r'(t+1) - \check{w}_r(\tau')$. We know from Definition 2 that $\hat{w}_r'(t+1) = \hat{w}_r'(\tau') + \hat{w}_r'(\tau', t)$, and conclude from Corollary 1 that $\check{w}_r(\tau') = \check{w}_r'(\tau')$. Substituting these results into Equation (5.11) results in

$$\hat{w}_r^*(\tau', t) = \hat{w}_r'(\tau') + \hat{w}_r'(\tau', t) - \check{w}_r'(\tau') \tag{5.12}$$

Lemma 8 gives us that $\hat{w}_r'(\tau', t) \leq \rho_r' \cdot (t - \tau' + 1)$. We also know from Lemma 17 that $\hat{w}_r'(\tau') - \check{w}_r'(\tau') \leq \sigma_r' + \rho_r' \cdot \Theta_r$ for an active requestor. By substitution into Equation (5.12), we get

$$\hat{w}_r^*(\tau', t) \leq \sigma_r' + \rho_r' \cdot \Theta_r + \rho_r' \cdot (t - \tau' + 1)$$

According to Definition 17, this implies that w_r^* is (σ, ρ) constrained with parameters $(\sigma_r' + \rho_r' \cdot \Theta_r, \rho_r')$. □

We continue by deriving a (σ, ρ) characterization of w' in Lemma 19. This characterization is useful when computing latencies in networks of servers, as the output of one server in a network may be the input of another.

Lemma 19. *w_r' of a requestor $r \in R$ is (σ, ρ) constrained with parameters $(\sigma_r' + \rho_r' \cdot \Theta_r, \rho_r')$ if the CCSP arbiter is non-work-conserving.*

Proof. We calculate the upper bound on w_r' during an active period $[\tau_1, \tau_2]$, since an inactive requestor is not backlogged, according to Definition 20, and hence cannot be scheduled.

From Lemma 3 and the definition of a non-work-conserving static-priority scheduler in Definition 34, we get that $\forall \tau', t \in [\tau_1, \tau_2], \tau' < t$

$$w_r'(\tau', t) = w_r'(t+1) - w_r'(\tau') \leq \hat{w}_r'(t+1) - \check{w}_r'(\tau')$$

According to Definition 2, this is equivalent to the right-hand side of Equation (5.12). We hence conclude from Lemma 18 that w_r' is (σ, ρ) constrained with parameters $(\sigma_r' + \rho_r' \cdot \Theta_r, \rho_r')$. □

Next, we combine the results from Lemma 18 and Lemma 19, to show that CCSP admits burstier traffic than a SRSP arbiter without resulting in a burstier characterization of w'. Assume that both arbiters are configured with parameters (σ', ρ') for a particular requestor. A rate regulator that enforces a (σ, ρ) constraint on requested service causes w_{srsp}^* to be characterized

as (σ', ρ'), since all service in the request buffer is considered eligible. A CCSP arbiter admits burstier traffic, since Lemma 18 states that w^*_{ccsp} is characterized as $(\sigma' + \rho' \cdot \Theta, \rho')$. Lemma 19 shows that w'_{ccsp} is characterized as $(\sigma' + \rho' \cdot \Theta, \rho')$, which is identical to the characterization of w'_{srsp}, according to [16, 18, 21]. This follows from that the service latencies of the arbiters are identical, even though the CCSP regulator admits burstier traffic. This is possible since the worst-case interference in an interval occurs when all interfering requestors start their active periods simultaneously at τ_1, as shown in Lemma 16. At this time $\pi(\tau_1) = \sigma'$, which is less than $\hat{\pi}(t)$, as derived in Lemma 17. The key reason behind this is that Lemma 13 states that all interfering requestors cannot reach their maximum potential at the same time, as their aggregate potential cannot increase as long of one of them is scheduled.

We proceed by looking at the difference between regulating provided service based on active periods and enforcing a (σ, ρ) constraint on the requested service curve according to Definition 17. The latter implies increasing $\hat{w}_{srsp}$, which is equal to $\hat{w}'_{srsp}$ for this kind of regulator, when a requestor is in a *busy period* [18], defined in Definition 40. Notice that the only difference between being busy at t and being live at t, as defined in Definition 21, is that τ_1 is required to be the start of an active period in the latter case.

Definition 40 (Busy period). *A busy period of a requestor $r \in R$ is defined as a maximum interval $[\tau_1, \tau_2]$, such that $\forall t \in [\tau_1, \tau_2]$. $w_r(\tau_1 - 1, t - 1) \geq \rho'_r \cdot (t - \tau_1 + 1)$.*

Figure 5.1 illustrates the difference between the two kinds of regulation. The requestor in the figure is not busy between τ_1 and τ_2, causing both $\hat{w}'_{srsp}$ and $\check{w}'_{srsp}$ to shift $\tau_2 - \tau_1$ steps to the right when enforcing a (σ, ρ) constraint. This shift does not happen with $\hat{w}'_{ccsp}$, since the requestor is backlogged in the interval, and hence active. The requestor is busy again between τ_2 and τ_4, causing $\hat{w}'_{srsp}$ to increase in that interval. However, the requestor is not backlogged at τ_3, and is inactive until τ_5, shifting $\hat{w}'_{ccsp}$ and $\check{w}'_{ccsp}$ $\tau_5 - \tau_3$ steps to the right. This example shows that there are cases where the two methods of regulation provide different bounds on the amount of provided service.

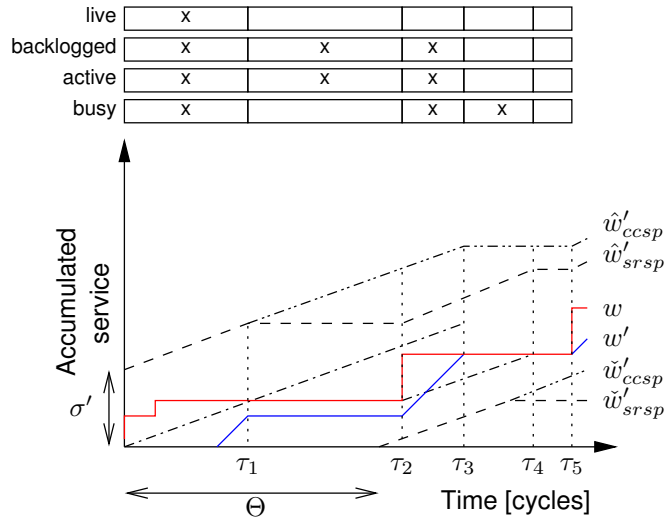

Figure 5.1: Some differences between enforcing a (σ, ρ) constraint on requested service and active period rate regulation.

We proceed by deriving an upper bound on backlog in Lemma 20.

Lemma 20. *The maximum backlog of a requestor $r \in R$ equals $\hat{q}_r(t) = \sigma_r + \Theta_r \cdot \rho_r$.*

Proof. Definition 20 defines that $q_r(t) = 0$ if r is inactive. The maximum backlog is hence encountered during an active period $[\tau_1, \tau_2]$, and is defined according to Equation (5.13). Note that the equation accounts for that arrivals are detected in the beginning of a cycle by subtracting one from the endpoints of the interval.

$$\hat{q}_r(t) = \max_{\forall t \in [\tau_1 - 1, \tau_2 - 1]} (\hat{w}_r(\tau_1 - 1, t) - \check{w}'_r(\tau_1 - 1, t)) \tag{5.13}$$

From Definition 17, we get that $\hat{w}_r(\tau_1 - 1, t) = \sigma_r + \rho_r \cdot (t + 1 - (\tau_1 - 1))$. We know from Theorem 2 that $\check{w}'_r(\tau_1 - 1, t) = \rho'_r \cdot \max(0, t + 1 - (\tau_1 - 1) - \Theta_r)$. We substitute these results into Equation (5.13), yielding

$$\hat{q}_r(t) = (\sigma_r + \rho_r \cdot (t - \tau_1) + 2) - (\rho'_r \cdot \max(0, t - \tau_1 + 2 - \Theta_r))$$

We substitute ρ'_r with ρ_r and eliminate, which is possible since Definition 19 states that $\rho'_r \geq \rho_r$. We conclude the proof by solving for intervals satisfying $t - \tau_1 + 2 \geq \Theta_r$, as this maximizes the expression. □

The lower bound on provided service from Theorem 2 causes us to arrive at the bound on maximum backlog from [18] that is valid for any requestor with (σ, ρ)-constrained arrivals in a $\mathcal{LR}$ server. Considering that CCSP has the same parameters and service latency bound as an SRSP arbiter, we conclude that our approach does not require additional request buffering.

We proceed by using the bound on service latency to derive a bound on maximum delay. The delay bound is only valid for requestors that are configured according to Section 5.2, such that they are not slowed down by the rate regulation. According to [18], the maximum delay of a request from a (σ, ρ)-constrained requestor $r \in R$ in a $\mathcal{LR}$ server is guaranteed to be bounded according to $\hat{\delta}(\omega_r^k) = \Theta_r + \sigma'_r / \rho'_r$. This is seen in Figure 3.2 as the horizontal distance between $\hat{w}_r$ and $\check{w}'_r$. This general bound, however, does not account for the fact that a requestor in a static-priority scheduler is free from interference from lower priority requestors after the service latency, as long as it does not exceed the burstiness constraint enforced by the rate regulator. It hence receives service at a rate of $1 - \sum_{\forall r_j \in R^+_{r_i}} \rho'_{r_j}$, rather than ρ'_r, resulting in a delay bound according to Equation (5.14).

$$\hat{\delta}(\omega_{r_i}^k) = \Theta_{r_i} + \frac{\sigma'_{r_i}}{1 - \sum_{\forall r_j \in R^+_{r_i}} \rho'_{r_j}} = \frac{\hat{b}_{r_i} + \sum_{\forall r_j \in R^+_{r_i}} \sigma'_{r_j} + \sigma'_{r_i}}{1 - \sum_{\forall r_j \in R^+_{r_i}} \rho'_{r_j}} \tag{5.14}$$

The maximum delay bound in Equation (5.14) is the same as those derived for static-priority schedulers in [16, 14]. However, a benefit of our result is that we distinguish different maximum blocking terms depending on whether or not the arbiter is preemptive, non-preemptive and non-work-conserving, or non-preemptive and work-conserving, as opposed to just using the largest blocking term for work-conserving arbiters. The maximum blocking for preemptive arbiters is derived in Section 6.1, and for work-conserving arbiters in Section 6.2.

In conclusion, we showed in this section that although CCSP admits burstier traffic than an SRSP arbiter, it still provides the same upper bounds on service latency, output burstiness, backlog and delay.

Section 6

Extensions

In this section, we extend the arbiter and its analysis to include preemption and work conservation. Preemption is first discussed in Section 6.1, followed by work conservation in Section 6.2.

6.1 Preemption

As mentioned in Section 4.1, preemption allows an arbiter that regulates provided service to efficiently handle requests without knowing their size. It furthermore allows latency to be reduced for critical requestors. We assume that the resource is preemptive on the granularity of a single service unit, and that there is no associated latency cost due to context switching. This allows us to treat a request ω^k with size $s(\omega^k) > 1$ as $s(\omega^k)$ requests of size one.

Introducing preemption as a tool for handling unknown request sizes requires the definitions of eligibility and eligibility time in Section 4.2 to be slightly altered, as it may no longer possible to know if a request has completely arrived or if a requestor has enough potential to serve a request. Definition 41 presents a definition of eligibility time that is suitable for preemptive arbiters. The definition states that a backlog of one service unit and the potential required to serve it is needed at the eligibility time. Changing the definition of eligibility time, according to Definition 41, implies that a requestor is no longer considered eligible in the entire interval $[t_e(\omega_r^k), t_\mathrm{f}(\omega_r^k) - 1]$, as stated in Definition 29. Instead, the preemptive version in Definition 42 only considers a requestor eligible in this interval it is backlogged and has enough potential to serve the next service unit. These changes impact the proof of Lemma 12, although it is easily verified that the lemma still holds in both cases since $\pi(t) > \sigma' - \rho' > 1 - \rho'$.

Definition 41 (Eligibility time). *The eligibility time of a request ω_r^k from a requestor $r \in R$ in a preemptive arbiter is denoted $t_e^{pr}(\omega_r^k)$, and is defined as the smallest t for which the following conditions apply:*

1. $\forall i < k.\ t \geq t_f(\omega_r^i)$, *and*
2. $w_r(t) > w'_r(t)$, *and*
3. $\pi_r(t) \geq 1 - \rho'_r(t)$

Definition 42 (Eligible requestor). *r is defined as eligible in a preemptive arbiter* $\forall k \in \mathbb{N}, t \in [t_e^{pr}(\omega_r^k), t_f(\omega_r^k) - 1].\ \pi_r(t) \geq 1 - \rho'_r(t) \wedge w_r(t) > w'_r(t)$.

A preemptive non-work-conserving static-priority scheduler, defined in Definition 43, schedules the highest priority eligible requestor. Since the scheduler is non-work-conserving, just like in Definition 34, no requestor is scheduled if there are no eligible requestors. We examine the case of work-conserving static-priority schedulers in Section 6.2.

Definition 43 (Preemptive non-work-conserving static-priority scheduler). *The scheduled requestor at a time t in a preemptive non-work-conserving static-priority scheduler is denoted $\gamma_{pr}^{nwc}(t) : \mathbb{N} \rightarrow R_t^e \cup \{\varnothing\}$, and is defined as*

$$\gamma_{pr}^{nwc}(t) = \begin{cases} \{r_i \mid r_i \in R_t^e \wedge \nexists r_j \in R_t^e.\ p_{r_j} < p_{r_i}\} & R_t^e \neq \varnothing \\ \varnothing & R_t^e = \varnothing \end{cases}$$

Preemption on the granularity of a service unit eliminates blocking, as shown in Lemma 21. This allows preemptive arbitration to offer lower latency to critical requestors.

Lemma 21. *A requestor $r_i \in R$ in a preemptive arbiter can maximally be blocked during any interval $[\tau_1, \tau_2]$, for which it holds that $\forall t \in [\tau_1, \tau_2].\ r_i \in R_t^e$, according to $\hat{b}_{r_i}^{pr} = 0$.*

Proof. Follows immediately from the fact that the scheduler is preemptive on a granularity of a service unit. □

We conclude this section by establishing a relation between the eligibility time and starting time of a request in Lemma 22, which we use to derive the finishing time of a request in a preemptive arbiter in Lemma 23. The derived finishing time assumes that the requestor is active in the interval $[t_e^{pr}(\omega_r^k), t_{\mathrm{f}}^{pr}(\omega_r^k) - 1]$, and hence that the request arrives at least according to the allocated rate after the eligibility time. Note that this requires the definitions of arrival time and the requested service curve in Definition 7 and Definition 8 to account for each arrived service unit separately.

Lemma 22. *For a request ω_r^k from a requestor $r \in R$ it holds that $t_s(\omega_r^k) \leq t_e(\omega_r^k) + \Theta_r$.*

Proof. We know from Theorem 1 that a requestor in an active period $[\tau_1, \tau_2]$ receives service according to $\forall t \in [\tau_1, \tau_2].\ w'_{r_i}(\tau_1, t) \geq \rho'_r \cdot (t - \tau_1 + 1 - \Theta_r)$. We know from Lemma 2 that $\forall t' \in [t_e(\omega_r^k), t_s(\omega_r^k)].\ r \in R_{t'}^a$. According to Definition 2 and the definition of starting time in Definition 10, we know that the maximum starting time of ω_r^k equals the minimum t for which it holds that $w'_{r_i}(t_e(\omega_r^k), t) > 0$. We hence get that

$$\rho'_r \cdot (t - t_e(\omega_r^k) + 1 - \Theta_r) > 0$$

Solving for t results in $t > t_e(\omega_r^k) + \Theta_r - 1$, which implies that $t_s(\omega_r^k) \leq t_e(\omega_r^k) + \Theta_r$. □

Lemma 23. *The finishing time of a request ω_r^k from a requestor $r \in R$, for which it holds that $\forall t \in [t_e(\omega_r^k), t_f(\omega_r^k) - 1].\ r \in R_t^a$, is denoted $t_f(\omega_r^k) : \Omega_r \rightarrow \mathbb{N}$, and is defined according to*

$$t_f(\omega_r^k) \leq t_s(\omega_r^k) + \frac{s(\omega_r^k)}{\rho'_r}$$

Proof. We know from Theorem 1 that a requestor in an active period $[\tau_1, \tau_2]$ receives service according to $\forall t \in [\tau_1, \tau_2].\ w'_{r_i}(\tau_1, t) \geq \rho'_r \cdot (t - \tau_1 + 1 - \Theta_r)$. The finishing time of ω_r^k equals $t + 1$ for the minimum t for which it holds that $w'_{r_i}(t_e(\omega_r^k), t) \geq s(\omega_r^k)$. We hence get that

$$\rho'_r \cdot (t - t_e(\omega_r^k) + 1 - \Theta_r) \geq s(\omega_r^k)$$

Solving for t results in $t \geq t_e(\omega_r^k) + \Theta_r + \frac{s(\omega_r^k)}{\rho'_r} - 1$, which implies that $t_{\mathrm{f}}(\omega_r^k) \leq t_e(\omega_r^k) + \Theta_r + \frac{s(\omega_r^k)}{\rho'_r}$. We conclude the proof by using that $t_s(\omega_r^k) \leq t_e(\omega_r^k) + \Theta_r$, according to Lemma 22. □

6.2 Work conservation

A work-conserving arbiter is never idle if there is a backlogged requestor, as mentioned in Section 4.1. A work-conserving static-priority scheduler hence schedules a backlogged requestor according to some slack management strategy if there are no eligible requestors. Any second-level scheduler can be used to distribute the slack, as long as it does not affect the guaranteed service. A preemptive work-conserving static-priority scheduler is presented in Definition 44, and its non-preemptive counterpart in Definition 45.

Definition 44 (Preemptive work-conserving static-priority scheduler). *The scheduled requestor at a time t in a preemptive work-conserving static-priority scheduler is denoted $\gamma_{pr}^{wc}(t) : \mathbb{N} \rightarrow R_t^q \cup \{\varnothing\}$, and is defined as*

$$\gamma_{pr}^{wc}(t) = \begin{cases} \{r_i \mid r_i \in R_t^e \wedge \nexists r_j \in R_t^e.\ p_{r_j} < p_{r_i}\} & R_t^e \neq \varnothing \\ \textit{any } r_i \in R_t^q & R_t^e = \varnothing \wedge R_t^q \neq \varnothing \\ \varnothing & R_t^e = \varnothing \wedge R_t^q = \varnothing \end{cases}$$

Definition 45 (Non-preemptive work-conserving static-priority scheduler). *The scheduled requestor at a time t in a non-preemptive work-conserving static-priority scheduler is denoted $\gamma_{np}^{wc}(t) : \mathbb{N} \rightarrow R_t^q \cup \{\varnothing\}$, and is defined as*

$$\gamma_{np}^{wc}(t) = \begin{cases} \{r_i \mid \exists k \in \mathbb{N}.\ t_s(\omega_{r_i}^k) < t < t_f(\omega_{r_i}^k)\} & \exists \omega_{r_i}^k.\ t_s(\omega_{r_i}^k) < t < t_f(\omega_r^k) \wedge R_t^e \neq \varnothing \\ \{r_i \mid r_i \in R_t^e \wedge \nexists r_j \in R_t^e.\ p_{r_j} < p_{r_i}\} & \nexists \omega_{r_i}^k.\ t_s(\omega_{r_i}^k) < t < t_f(\omega_r^k) \wedge R_t^e \neq \varnothing \\ \textit{any } r_i \in R_t^q & R_t^e = \varnothing \wedge R_t^q \neq \varnothing \\ \varnothing & R_t^e = \varnothing \wedge R_t^q = \varnothing \end{cases}$$

To prevent the second-level scheduler from affecting the guaranteed service and to create a separation of concerns, we propose to keep it separated from the primary static-priority scheduler. A logical separation is created by asserting that the second-level scheduler does not alter the potential of any requestor. This is accomplished by adapting the provided service bound in Definition 25. A work-conserving definition of this bound is shown in Definition 46 and the corresponding accounting mechanism is presented in Definition 47. Note that the only difference between the work-conserving and non-work-conserving accounting mechanisms in Definition 27 and Definition 47 is that the latter is unaffected if an active but non-eligible requestor is scheduled due to work-conservation. It is straight-forward to show that the accounting in Definition 47 is consistent with the definition of potential in Definition 26 using the method in Lemma 1, although no such proof is supplied here.

Definition 46 (Work-conserving provided service bound). *The enforced upper bound on provided service of a requestor $r \in R$ is denoted $\hat{w}'_r(t) : \mathbb{N} \rightarrow \mathbb{R}^+$, and is defined according to*

$$\hat{w}'_r(t+1) = \begin{cases} \hat{w}'_r(t) + \rho'_r & r \in R_t^a \wedge r \in R_t^e \\ \hat{w}'_r(t) + \rho'_r & r \in R_t^a \wedge r \notin R_t^e \wedge \gamma(t) \neq r \\ \hat{w}'_r(t) + \rho'_r + 1 & r \in R_t^a \wedge r \notin R_t^e \wedge \gamma(t) = r \\ w'_r(t) + \sigma'_r & r \notin R_t^a \wedge \gamma(t) \neq r \end{cases}$$

where $\hat{w}'_r(0) = \sigma'_r$.

Definition 47 (Work-conserving accounting). *The accounted potential of a requestor $r \in R$ is denoted $\pi^*_r(t) : \mathbb{N} \rightarrow \mathbb{R}$, and is defined according to*

$$\pi^*_r(t+1) = \begin{cases} \pi^*_r(t) + \rho'_r - 1 & r \in R^a_t \wedge r \in R^e_t \wedge \gamma(t) = r \\ \pi^*_r(t) + \rho'_r & r \in R^a_t \wedge r \notin R^e_t \wedge \gamma(t) = r \\ \pi^*_r(t) + \rho'_r & r \in R^a_t \wedge \gamma(t) \neq r \\ \sigma'_r & r \notin R^a_t \wedge \gamma(t) \neq r \end{cases}$$

where $\pi^*_r(0) = \sigma'_r$.

Work conservation requires a redefinition of blocking, as stated in Lemma 24, since it allows higher priority requestors to be scheduled without reducing their aggregate potential. Definition 48 supersedes that in Definition 35, as it is more general and covers both the non-work-conserving and the work-conserving case. A bound on blocking for a work-conserving arbiter is derived in Lemma 24.

Definition 48 (Blocking). *Blocking of a requestor $r_i \in R$ in a non-preemptive arbiter during any interval $[\tau_1, \tau_2]$, for which it holds that $\forall t \in [\tau_1, \tau_2].\ r_i \in R^e_t$, is denoted $b_{r_i} : R \rightarrow \mathbb{N}$, and is defined as the number of interfering service units a requestor $r_j \in R, r_i \neq r_j$ is provided, without reducing the aggregate potential of $R^+_{r_i}$.*

Lemma 24. *A requestor $r_i \in R$ in a non-preemptive work-conserving arbiter can maximally be blocked during any interval $[\tau_1, \tau_2]$, for which it holds that $\forall t \in [\tau_1, \tau_2].\ r_i \in R^e_t$, according to*

$$\hat{b}^{wc}_{r_i} = \max_{\forall r_j \in R, r_j \neq r_i} \hat{s}_{r_j} - 1 \tag{6.1}$$

Proof. Definition 45 states that any backlogged requestor can be scheduled if $R^e_t = \varnothing$. The maximum blocking of a requestor $r_i \in R$ in a non-preemptive work-conserving arbiter hence occurs when a request, $\omega^k_{r_i}$, becomes eligible at $t^{np}_e(\omega^k_{r_i}) = \tau_1$, and a maximally sized request from the requestor with the largest request size was scheduled at $t^{np}_e(\omega^k_{r_i}) - 1$. □

Work conservation affects the characterization of w' since the allocated rate and allocated burstiness are no longer enforced by the rate regulator. The characterization of w' is instead determined by the rate and burstiness of the requestor, defined in Definition 18, as shown in Lemma 25. This may result in a much burstier characterization of w', since σ may be much larger than σ' for soft real-time requestors. The rate, however, is unaffected since Definition 19 states that $\forall r \in R.\ \rho' \geq \rho$.

Lemma 25. *w'_r of a requestor $r \in R$ is (σ, ρ) constrained with parameters $(\sigma_r + \rho_r \cdot \Theta_r, \rho_r)$ if the CCSP arbiter is work conserving.*

Proof. The second case in Definition 47 states that π^*_r does not decrease at $t+1$ if $r \in R^a_t \wedge r \notin R^e_t \wedge \gamma(t) = r$. According to Definition 29, this implies that the burstiness of w'_r is constrained by $\hat{w}_r$ rather than $\hat{w}'_r$. The burstiness of w'_r is hence determined by the difference between $\hat{w}_r$ and $\check{w}'_r$. This corresponds to the maximum backlog, which equals $\hat{q}_r(t) = \sigma_r + \Theta_r \cdot \rho_r$, according to Lemma 20. The rate of w'_r is determined by ρ_r since Definition 18 states that this is the rate at which $\hat{w}_r$ increases. By combining these results with Definition 17, we get that w^*_r is (σ, ρ) constrained with parameters $(\sigma_r + \rho_r \cdot \Theta_r, \rho_r)$ if the arbiter is work conserving. □

The effects of work conservation are illustrated in Figure 6.1. The figure shows a bursty soft real-time requestor, $r_i \in R$, with $\sigma'_{r_i} < \sigma_{r_i}$ and $\rho'_{r_i} = \rho_{r_i}$. The requestor does not have enough potential to be scheduled at τ_1. The requestor is, however, scheduled anyway between τ_1 and τ_2 due to work conservation. The requestor is active in the interval and the second-level scheduler is not allowed to affect the potential. The potential of the requestor hence increases with a rate ρ' in the interval, as stated by the second case in Definition 47. The requestor is eligible at τ_2 since it has enough potential to serve the next request, and is scheduled again between τ_2 and τ_3. However, this time the potential of the requestor is reduced while it is receiving service, as stated by the first case in Definition 47. If a lower priority requestor, $r_j \in R$, becomes eligible between τ_1 and τ_2 then it gets blocked by r_i, according to Definition 35, since the aggregate potential of R^+_{rj} is not reduced. The figure also shows that $\hat{w}'$ approaches $\hat{w}$, as r_i receives service due to work conservation between τ_1 and τ_2. This illustrates that work conservation implies that the maximum burstiness of w' is determined by the maximum backlog, $\hat{q}$, as shown in Lemma 25.

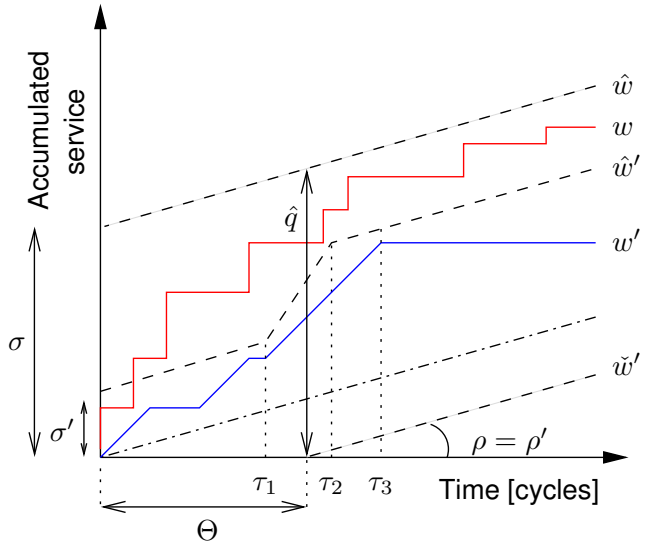

Figure 6.1: A bursty soft real-time requestor that is provided service between τ_1 and τ_2 due to work conservation.

Section 7

Hardware implementation

In this section, we present an efficient hardware implementation of a non-preemptive CCSP arbiter. First, a discrete version of the rate regulator that features negligible over-allocation is introduced in Section 7.1. We then proceed by presenting the architecture and synthesis results of the arbiter in Section 7.2.

7.1 Discrete rate regulation

To maximize the applicability of the arbiter in the context of SoCs, the hardware implementation requires a low-cost implementation that runs at high speeds. For this reason, we developed a discrete implementation of the accounting and enforcement in rate regulator that has finite precision and avoids elements like multipliers and comparators, where possible, to reduce area and increase speed.

The accounting mechanism used by the hardware implementation of the CCSP rate regulator is presented Definition 52. It is based on the mechanism presented in [22], modified to implement a discrete version of the potential-based accounting in Definition 27. The discrete service allocation, defined in Definition 49, tries to approximate the non-discrete allocation in Definition 19 as closely as possible, given a particular precision. The discrete allocated rate of a requestor, ρ'', is expressed as a fraction of the available service capacity, as defined in Definition 50. Resource access is based on credits that are a discrete representation of the potential. The amount of initial credits, $c(0)$, is determined by the discrete allocated burstiness of a requestor, defined in Definition 51, according to $c(0) = \sigma'' \cdot d$. Credits are updated at the end of every cycle according to Definition 52. Note how the credit mechanism reduces managing fractions to simple integer arithmetic suitable for efficient hardware implementation.

Definition 49 (Discrete service allocation). *The service allocation of a requestor $r \in R$ in the discrete implementation is defined as (ρ''_r, σ''_r).*

Definition 50 (Discrete allocated rate). *The discrete allocated rate of a requestor $r \in R$ in an arbiter with a precision of β bits is denoted $\rho''_r \in \mathbb{Q}^+$, and is represented as $\rho''_r = n_r/d_r$, where $n_r, d_r \in \mathbb{N}^+ < 2^\beta$. The values of n_r and d_r are chosen such that ρ''_r is the minimum rate that satisfies $\rho''_r \geq \rho'_r$.*

Definition 51 (Discrete allocated burstiness). *The discrete allocated burstiness of a requestor $r \in R$ is denoted $\sigma''_r \in \mathbb{R}^+$, and is defined as $\sigma''_r = \frac{\lceil \sigma' \cdot d_r \rceil}{d_r}$.*

Definition 52 (Credits). *The amount of credits of a requestor $r \in R$ is denoted $c_r(t) : \mathbb{N} \rightarrow \mathbb{N}$, and is defined as*

$$c_r(t+1) = \begin{cases} c_r(t) + n_r - d_r & \gamma(t) = r \\ c_r(t) + n_r & \gamma(t) \neq r \wedge q_r(t) > 0 \\ \min(c_r(t) + n_r, c_r(0)) & \gamma(t) \neq r \wedge q_r(t) = 0 \end{cases} \tag{7.1}$$

where $c_r(0) = \sigma''_r \cdot d_r$.

A lower bound on the potential of an inactive requestor is derived in Lemma 26. We use this result in Theorem 3 to prove that the credit mechanism in Definition 52 is a discrete implementation of the accounting based on potential in Definition 27 with discrete service allocations according to Definition 49.

Lemma 26. *For a requestor $r \notin R^a_t \Rightarrow \pi_r(t) > \sigma'_r - \rho'_r$.*

Proof. By negating Definition 20, we know that iff $r \notin R^a_t$ then $q_r(t) = 0$ and $w_r(\tau_1 - 1, t - 1) < \rho'_r \cdot (t - \tau_1 + 1)$, where τ_1 is the start of the last active period. From Definition 14, we get that $q_r(t) = 0$ implies $w_r(t) = w'_r(t)$. Substituting this into the expression results in $w'_r(\tau_1 - 1, t-1) < \rho'_r \cdot (t - \tau_1 + 1)$. Lemma 4 states that $w'_r(\tau_1 - 1, t-1) = w'_r(\tau_1, t-1)$, giving us $w'_r(\tau_1, t-1) < \rho'_r \cdot (t - \tau_1 + 1)$, which according to Lemma 6 implies that $\pi_r(t) > \sigma'_r - \rho'_r$. □

Theorem 3 (Discrete accounting). *The accounting mechanism in Definition 52 is a discrete implementation of Definition 27 with discrete service allocations according to Definition 49, where it holds that $\forall t.\ c_r(t) = \pi_r(t) \cdot d_r$.*

Proof. We rewrite the equation in Definition 27 by splitting the second case, where $r \in R^a_t$, in two, according to Definition 20. In the first case $q_r(t) > 0$ and in the other $q_r(t) = 0$ and $r \in R^l_t$. According to Definition 23 and Lemma 7, $r \in R^l_t$ and $q(t) = 0$ implies that $\pi_r(t) \leq \sigma''_r - \rho''_r$. We use the results from Lemma 26 to rewrite the case where $r \notin R^a_t$, resulting in

$$\pi_r(t+1) = \begin{cases} \pi_r(t) + \rho''_r - 1 & \gamma(t) = r \\ \pi_r(t) + \rho''_r & \gamma(t) \neq r \wedge q_r(t) > 0 \\ \pi_r(t) + \rho''_r & \gamma(t) \neq r \wedge q_r(t) = 0 \wedge \pi_r(t) \leq \sigma''_r - \rho''_r \\ \sigma''_r & \gamma(t) \neq r \wedge q_r(t) = 0 \wedge \pi_r(t) > \sigma''_r - \rho''_r \end{cases} \tag{7.2}$$

Multiplying both sides of Equation (7.2) with d_r and substituting $c_r(t) = \pi_r(t) \cdot d_r$, $n_r = \rho''_r \cdot d_r$ and $c_r(0) = \sigma''_r \cdot d_r$, according to Definition 50 and Definition 52 yields

$$c_r(t+1) = \begin{cases} c_r(t) + n_r - d_r & \gamma(t) = r \\ c_r(t) + n_r & \gamma(t) \neq r \wedge q_r(t) > 0 \\ c_r(t) + n_r & \gamma(t) \neq r \wedge q_r(t) = 0 \wedge c_r(t) \leq c_r(0) - n_r \\ c_r(0) & \gamma(t) \neq r \wedge q_r(t) = 0 \wedge c_r(t) > c_r(0) - n_r \end{cases} \tag{7.3}$$

To simplify the accounting, we merge the two last cases in Equation (7.3) into $c_r(t+1) = \min(c_r(t) + n_r, c_r(0))$, where the third case in Equation (7.3) is covered by the first operand and the fourth case by the second operand. This concludes the proof, as we have now arrived at the accounting mechanism proposed in Definition 52. □

Corollary 4. *It follows from Definition 29 and Theorem 3 that a requestor $r \in R$ requires $c_r(t) \geq s(\omega_r^k) \cdot d_r - n_r$ to become eligible for scheduling of ω_r^k at t.*

We proceed by examining the over-allocation properties, defined in Definition 53. The proposed accounting mechanism is effective as it has an accuracy that is limited only by the number of bits, β, used to represent n and d. Greater precision limits the over-allocation due to discretization errors, as shown in Lemma 27. This is beneficial as high precision potentially results in a significant increase in resource utilization if the number of requestors is large [10].

Definition 53 (Over-allocation). *The over-allocation of a requestor $r \in R$ in a CCSP arbiter is denoted $\varepsilon(\rho_r'', \rho_r') : \mathbb{Q}^+ \times \mathbb{R}^+ \rightarrow \mathbb{R}$, and is defined according to*

$$\varepsilon(\rho_r'', \rho_r') = \rho_r'' - \rho_r' \tag{7.4}$$

Lemma 27. *The over-allocation of a requestor in a CCSP arbiter with a precision of β bits is upper bounded according to*

$$\varepsilon(\rho'', \rho') < \frac{1}{2^\beta - 1} \tag{7.5}$$

Proof. Over-allocation is defined as $\varepsilon(\rho'', \rho') = \rho'' - \rho'$, according to Definition 53. We know from Definition 50 that $\rho'' = n/d$. The definition furthermore states that n and d are chosen such that ρ'' is the minimum rate satisfying $\rho'' \geq \rho'$. This means that $d = 2^\beta - 1$ and $n = \lceil d \cdot \rho' \rceil$, unless there is another n, d pair that yields a tighter approximation. By substituting this result into Equation (7.4) and perform basic algebraic manipulation, we hence arrive at

$$\varepsilon(\rho'', \rho') < \frac{\lceil d \cdot \rho' \rceil}{d} - \frac{d \cdot \rho'}{d} < 1/d$$

The proof is concluded by substituting $d = 2^\beta - 1$. □

Lemma 27 shows that the maximum over-allocation reduces exponentially with increasing precision. This can be compared to the over-allocation of a DRR [5] scheduler that over-allocates according to $\varepsilon^{DRR}(\rho'', \rho') < 1/F$, where F is the frame size of the scheduler. We note that the maximum over-allocation of a requestor is inversely proportional to the frame size, implying that a large frame size is required to provide an efficient allocation. However, we also know from Equation (5.10) that the service latency is proportional to the frame size, resulting in a trade-off between low service latency and over-allocation. The CCSP arbiter does not suffer from this limitation.

7.2 Architecture

The proposed arbiter, shown in Figure 7.1, has been implemented in VHDL and integrated into the Predator DDR2 SDRAM controller [23]. This controller is used in the context of a multiprocessor SoC that is interconnected using the Æthereal NoC [24]. Requests arrive at a network interface (NI) [25] on the edge of the network, where they are buffered in separate request buffers per requestor.

The rate regulator is implemented according to Section 7.1. A register bank contains four registers for every requestor: n, d, $c(t)$, and $c(0)$. These registers are programmable using memory mapped IO for run-time (re)configuration.

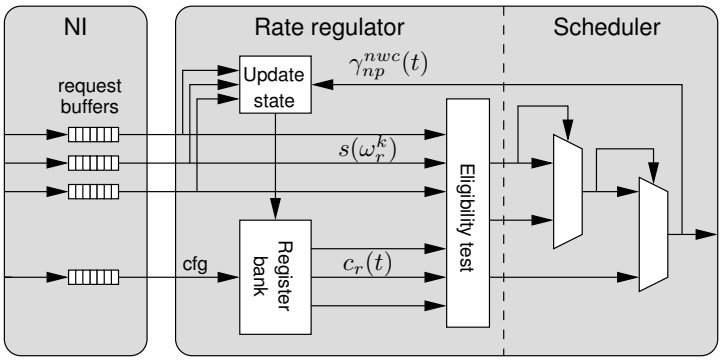

Figure 7.1: A non-preemptive non-work-conserving CCSP arbiter without priority switch supporting three requestors.

The static-priority scheduler is implemented by a tree of multiplexers that simply grant access to the highest priority requestor that is eligible, an operation that is faster than comparing deadlines, as done in [12]. The scheduled requestor is output from the arbiter, but also fed back to a unit that updates the register bank to reflect the new credit state, as discussed in Section 4.1. Work conservation is implemented by connecting the request pending signals to another multiplexer tree. A final multiplexer decides to use the second tree if no eligible requestor is found on the first, as shown in Figure 7.2. This means that static-priority scheduling is also used as a second-level scheduler to distribute slack. Configurable priorities can be implemented in two different ways. The intuitive way is to use a programmable priority switch that maps the request buffers to the scheduler according to its priority level. The switch is combined with a look-up table (LUT) that remaps the index of the scheduled requestor, as shown in Figure 7.2. Another option is to use the reconfiguration abilities of Æthereal [26] to change how the network connections of the requestors are mapped to the buffers in the NI. The first option requires additional hardware to implement the priority switch, but reconfigures faster since it does not require network connections to be flushed, torn down, and re-setup.

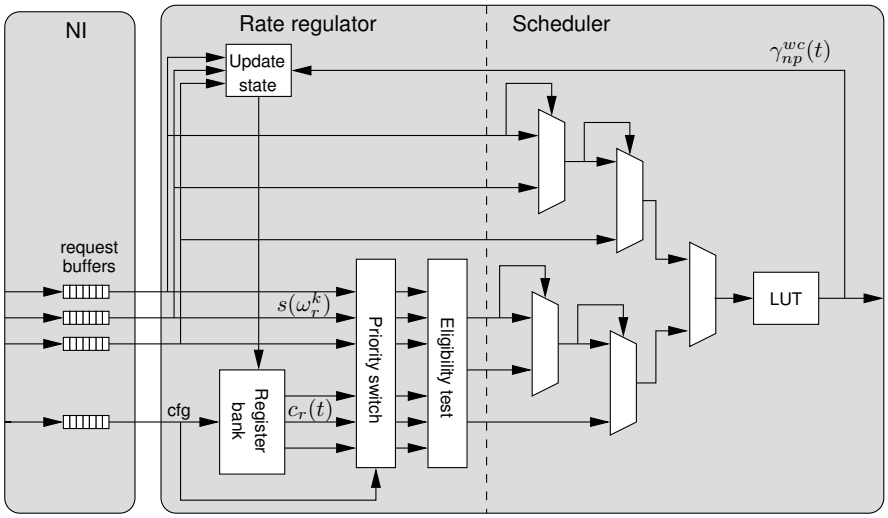

Figure 7.2: A non-preemptive work-conserving CCSP arbiter with priority switch supporting three requestors.

Synthesis of the arbiter in CMOS090LP with six ports using eight bits to represent each of n, d, $c(t)$ and $c(0)$ results in a total area of 0.0175 mm^2. This synthesized instance is non-preemptive and non-work-conserving, and does not contain the priority switch. This instance of the arbiter synthesizes at a frequency of 250 MHz, which is above 200 MHz required for a DDR2-400 SDRAM device. Figure 7.3 shows how the area of the arbiter scales with and without the switch with a speed target of 200 MHz. We see that the arbiter scales linearly with the number of ports without the switch. The arbiter is expected to scale quadratically when the switch is included. However, this trend is not visible in the results. A possible explanation for this is that the switch is not large enough for this trend to be clearly visible unless the number of ports is bigger.

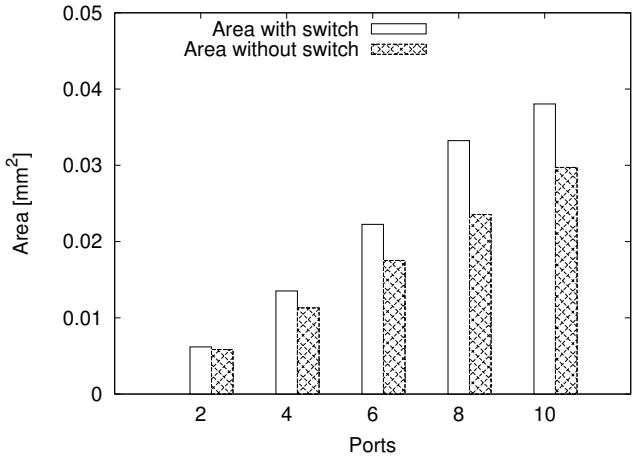

Figure 7.3: The area of the arbiter with and without the priority switch.

We experimented by varying the number of bits used to represent the values in the register bank to show how allocation granularity is traded for area. Figure 7.4 illustrates the trade-off between allocation granularity and area for an arbiter with six ports as the bit widths of n, d, $c(t)$ and $c(0)$ are changed. Notice that the exponential reduction in maximum over-allocation from Equation (7.5) comes at a linear increase in area.

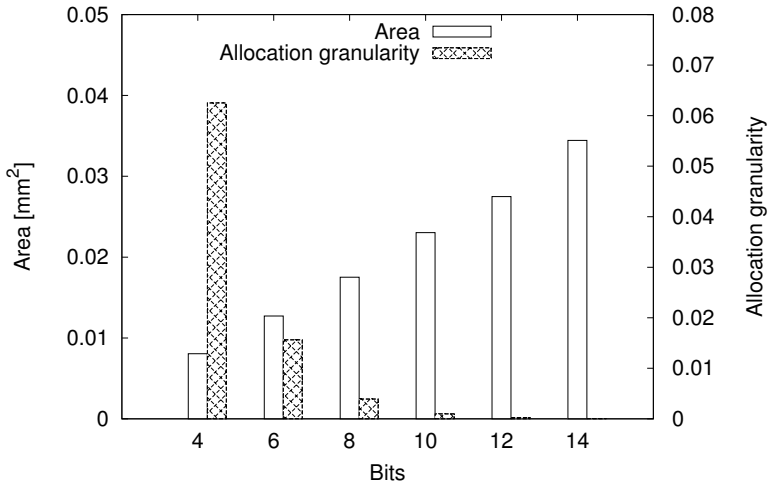

Figure 7.4: The allocation granularity/area trade-off.

Section 8

Experimental results

We have experimented with SystemC simulation of a use-case involving an H.264 video decoder. The H.264 decoder contains a number of requestors communicating through external memory. Access to a DDR2-400 SDRAM is provided by a Predator SDRAM controller [23]. A benefit of this controller is that the arbiter schedules memory accesses of 64 B to the requestors, as opposed to scheduling time, which means that the amount of work associated with a request is always known. This allows us to use the setup to experiment with both CCSP and SRSP. The time required by the memory controller to serve a service unit corresponds to approximately 80 ns on average. An equation is presented in [23] that computes the worst-case service latency in ns, given a number of interfering service units.

The use-case contains a file reader that reads an encoded image and stores it in external memory. This requestor works on the granularity of 64 KB blocks. These blocks are distributed over 1000 requests of 64 B each that are essentially issued back-to-back, making the requestor extremely bursty. The decoder software is running on a TriMedia 3270 [27]. The TriMedia uses separate read (TM_{rd}) and write connections to communicate with external memory through an L1 cache with a line size of 128 B. Finally, a display controller reads the decoded image in blocks of 128 B and shows it on a display. For the purpose of this document, this is considered as a soft real-time application that has deadlines on the granularity of decoded frames. We add two hard real-time requestors (HRT_1, HRT_2), mimicked by traffic generators, to create a hybrid system. These issue read and write requests of 128 B to external memory. High priority is assigned to the soft real-time requestors and lower priorities to the hard real-time requestors, according to the assignment strategy in [11]. The assignment within these classes is determined depending on the criticality of the requestors. The allocation parameters (σ' and ρ') of the hard real-time requestors are chosen such that the rate regulator never slows them down and violates their bounds on service latency. For the soft real-time requestors, ρ' is chosen based on measurements such that $\rho' \geq \rho$. Finding a structured methodology for deriving these parameters, such that soft real-time requestors meet their task-level deadlines, is still an active research question that is outside the scope of this document. Table 8.1 lists the configuration parameters of the requestors in the use-case. A total of 600 MB/s is allocated to the requestors, corresponding to a load of 90.7% of the capacity offered by the memory controller for a 16-bit DDR2-400 device after taking access overhead into account [23]. The total over-allocation is 0.0294%, corresponding to a waste of mere 0.19 MB/s, when 8 bits are used to represent n and d of the requestors. Table 8.1 presents average service latencies, $\bar{\Theta}$, and the maximum measured service latencies, $\max \Theta$, for all requestors after $2 \cdot 10^8$ ns of simulation with both a non-work-conserving and a work-conserving instance of CCSP. The corresponding service latency bounds, $\hat{\Theta}$, obtained

Table 8.1: Requestor configuration and results for use-case.

Requestor	σ'	ρ'	$\hat{s}$	p	$\bar{\Theta}^{nwc}$	$\max \Theta^{nwc}$	$\hat{\Theta}^{nwc}$	$\bar{\Theta}^{wc}$	$\max \Theta^{wc}$	$\hat{\Theta}^{wc}$
TriMedia (read)	2.0	0.151	2	0	5.29	13	N/A	1.10	11	N/A
TriMedia (write)	2.0	0.151	2	1	3.37	13	N/A	2.88	14	N/A
Display Controller	2.0	0.047	2	2	36.46	46	N/A	3.54	42	N/A
File Reader	2.0	0.077	1	3	11.26	17	N/A	2.03	14	N/A
Hard real-time 1	3.4	0.242	2	4	0.30	7	15	0.32	7	15
Hard real-time 2	3.5	0.242	2	5	2.47	10	34	2.46	11	37

using Equation (5.5), are also listed for hard real-time requestors.

We note that the average latencies of the soft real-time requestors are significantly reduced when work-conservation allows the arbiter to distribute the 9.3% of unallocated resource capacity. Hard real-time requestors do not benefit from work-conservation, since the rate regulator is configured to give them all the service they require, as mentioned in Section 5.2. We also observe that the implications of work-conservation on the service latency bounds of the hard real-time requestors are very small (+1 for HRT_2). This makes work-conservation an interesting option for single servers, where an increased burstiness of the provided service does not matter. The measured service latencies are lower than the bounds for both hard real-time requestors with both work-conserving, and non-work-conserving arbitration, as expected. However, we note that the difference between the maximum measured value and the bound gets larger with lower priorities. A reason for this is that the risk of simultaneous maximum interference from all higher priority requestors becomes increasingly unlikely with lower priorities.

We proceed by looking at how changing the service allocation, (σ', ρ'), affects the results. We begin by studying the impact of changing ρ'. The non-work-conserving use-case in Table 8.1 is used as a starting point, but HRT_2 is removed from the system to free capacity, and ρ' of TM_{rd} is used as a variable in the range $[0.10, 0.45]$ in increments of 0.05. The results of this experiment, shown in Figure 8.1, indicate that the service latency of TM_{rd} is reduced with approximately 40% per increment. We also see an increase in both the service latency bound and maximum measured service latency for HRT_1. The gap between these two lines become slightly wider for larger values of ρ', suggesting that the tightness of the bound is somewhat reduced.

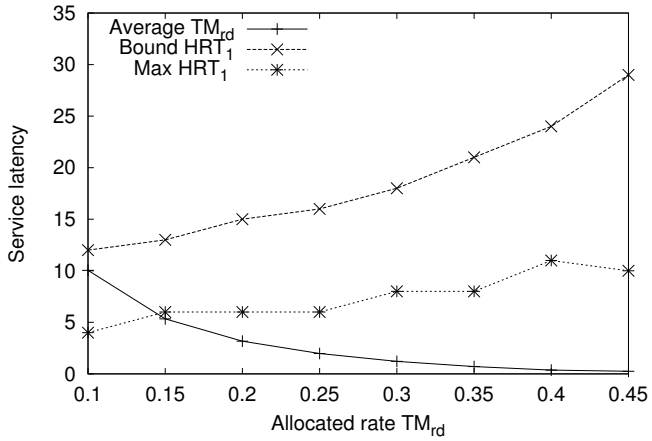

Figure 8.1: The average service latency of TM_{rd} is significantly reduced as its allocated rate is increased. This results in an increased latency bound and maximum measured latency for HRT_1.

A similar experiment is conducted to examine the impact of σ'. Again, we start from the non-work-conserving use-case in Table 8.1 and remove HRT_2. We let σ' for TM_{rd} be variable in the range $[2, 256]$ that doubles for every increment. The result of this experiment is shown in Figure 8.2. The average latency of TM_{rd} is slightly reduced with an average of 11% per increment. The bound on service latency for HRT_1, however, increases significantly, and requires a separate y-axis in the graph with a range that is almost two orders of magnitude larger than that of TM_{rd}. The maximum measured service latency is just increased from 6 to 11 in the interval, resulting in a rapidly growing gap between the maximum service latency and the bound.

From these experiments with service allocation, we conclude that increasing ρ' is a very powerful way of reducing average service latency while maintaining reasonable bounds, as long as the resource has unused capacity. We furthermore conclude that σ' should be fitted as tight as possible to produce meaningful bounds. This suggests that $\sigma' = \sigma$ for hard real-time requestors, and $\sigma' = \hat{s}$ for soft real-time requestors may be a good burstiness allocation strategy.

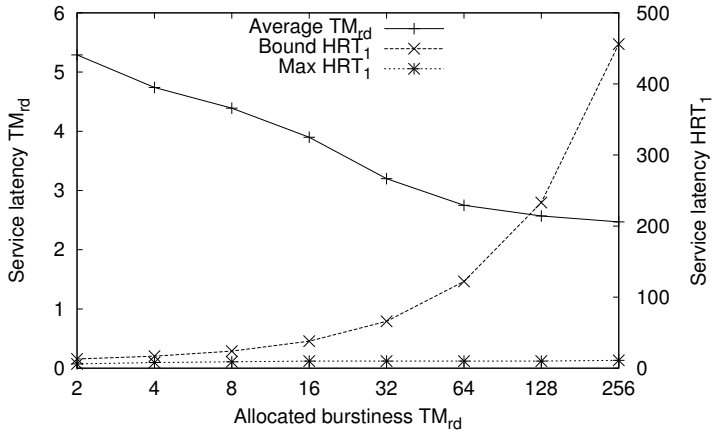

Figure 8.2: Increasing the allocated burstiness of TM_{rd} results in a slight decrease in service latency, at the cost of a significant increase in the latency bound of HRT_1.

We continue by experimenting with the allocation properties of CCSP. We create five use-cases by permuting the allocated rates of the requestors in the use-case in Table 8.1. Figure 8.3 shows the average and maximum over-allocation for these use-cases, along with the corresponding bounds, for a varying number of bits to representing n and d. We use the bound from Equation (7.5) and multiply that with the number of requestors, in this case six, to bound the total over-allocation. As seen in the figure, seven bits are required to get an allocation bound below 5%, and eight bits to get below 2.5%. The maximum measured over-allocation, however, is below 0.5% already when using six bits, demonstrating the efficiency and scalability of our approach to service allocation. We observe that the maximum over-allocation is consistently significantly smaller than the bound. This is because the bound assumes that $d = 2^{\beta} - 1$ and $n = \lceil d \cdot \rho' \rceil$, although in most cases a fraction can be found that yields a much tighter approximation of ρ'. This indicates that it may be possible to further improve the over-allocation bound for CCSP.

All simulations have been repeated with an SRSP arbiter, and the latency results proved to be identical for every single request for all configurations. A possible explanation for this is that the CCSP rate regulator admits more service than SRSP only when it is backlogged, and that any latency benefits of admitting burstier traffic may be canceled out when the request waits for

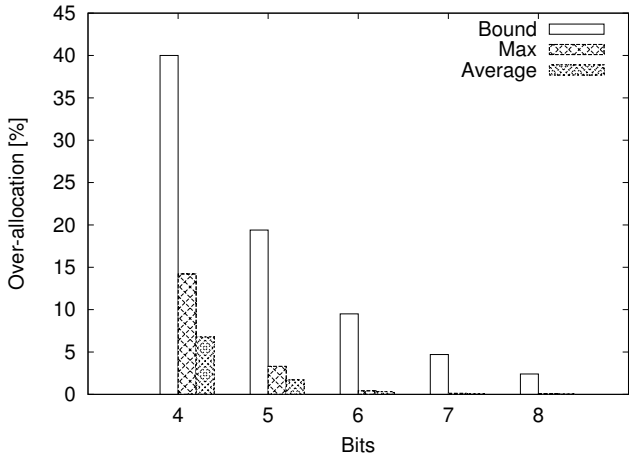

Figure 8.3: The average and maximum over-allocation, along with the corresponding bounds, for a varying number of bits representing n and d.

previous requests from the same requestor to be scheduled. This result, however, indicates that CCSP has all the benefits of regulating provided service, as mentioned in Section 4.1, without introducing additional latency, compared to an SRSP arbiter.

Section 9

Conclusions and future work

In this document, we present an arbiter called Credit-Controlled Static-Priority (CCSP) for scheduling access to resources, such as interconnect and memories in systems-on-chip. CCSP resembles an arbiter with a rate regulator that enforces a (σ, ρ) constraint on requested service together with a static-priority scheduler. However, instead of enforcing an upper bound on requested service, CCSP enforces an upper bound on provided service. Regulating provided service, as opposed to regulating requested service has two benefits: 1) the implementation of the regulator is less complex, and 2) the amount of work associated with a particular request does not have to be known. We show that CCSP enjoys these benefits, without increasing worst-case latency or buffering, compared to an arbiter regulating requested service. We furthermore show that CCSP belongs to the class of latency-rate ($\mathcal{LR}$) servers and guarantees the allocated service rate within a maximum latency, required by hard real-time requestors.

We present an implementation of the arbiter in the context of a DDR2 SDRAM controller that has been efficiently integrated into the network interface of a network-on-chip. The area of implementation is small and scales linearly with the number of requestors. An instance with six ports runs at 250 MHz and requires 0.0175 mm^2 in CMOS090LP. The efficiency of the service allocation is demonstrated in a use-case involving an H.264 decoder, where only 0.0294% of the resource capacity is lost due to over-allocation.

Future work involves creating a methodology to automatically derive service allocations and find a suitable priority assignment, such that the latency and service requirements of all requestors are met. This is a challenging problem that may require a heuristic approach, as the latency of the CCSP arbiter is computed by a non-linear equation that does not lend itself to integer-linear programming, and the configuration space is furthermore likely to be too large to allow exhaustive searches.

References

[1] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. *Real-Time Systems Symposium, 1998. Proceedings., The 19th IEEE*, pages 4–13, 1998.

[2] Kees Goossens, Om Prakash Gangwal, Jens Röver, and A. P. Niranjan. Interconnect and memory organization in SOCs for advanced set-top boxes and TV — Evolution, analysis, and trends. In Jari Nurmi, Hannu Tenhunen, Jouni Isoaho, and Axel Jantsch, editors, *Interconnect-Centric Design for Advanced SoC and NoC*, chapter 15, pages 399–423. Kluwer, 2004.

[3] C.M. Aras, J.F. Kurose, D.S. Reeves, and H. Schulzrinne. Real-time communication in packet-switched networks. In *Proceedings of the IEEE*, volume 82, pages 122–139, January 1994.

[4] M. Katevenis, S. Sidiropoulos, and C. Courcoubetis. Weighted round-robin cell multiplexing in a general-purpose ATM switch chip. *IEEE Journal on Selected Areas in Communication*, 9(8):1265–1279, October 1991.

[5] M. Shreedhar and George Varghese. Efficient fair queueing using deficit round robin. In *SIGCOMM*, pages 231–242, 1995.

[6] Hui Zhang. Service disciplines for guaranteed performance service in packet-switching networks. *Proceedings of the IEEE*, 83(10):1374–96, October 1995.

[7] C. R. Kalmanek and H. Kanakia. Rate controlled servers for very high-speed networks. *GLOBECOM*, pages 12–20, 1990.

[8] S. J. Golestani. A stop-and-go queueing framework for congestion management. In *SIGCOMM '90: Proceedings of the ACM symposium on Communications architectures & protocols*, pages 8–18, New York, NY, USA, 1990. ACM Press.

[9] S. S. Kanhere and H. Sethu. Fair, efficient and low-latency packet scheduling using nested-deficit round robin. *High Performance Switching and Routing, 2001 IEEE Workshop on*, pages 6–10, 2001.

[10] Debanjan Saha, Sarit Mukherjee, and Satish K. Tripathi. Carry-over round robin: a simple cell scheduling mechanism for ATM networks. *IEEE/ACM Trans. Netw.*, 6(6):779–796, 1998.

[11] S. Hosseini-Khayat and A.D. Bovopoulos. A simple and efficient bus management scheme that supports continuous streams. *ACM Transactions on Computer Systems (TOCS)*, 13(2):122–140, 1995.

[12] Jennifer Rexford, John Hall, and Kang G. Shin. A router architecture for real-time point-to-point networks. In *ISCA '96: Proceedings of the 23rd annual international symposium on Computer architecture*, pages 237–246, New York, NY, USA, 1996. ACM Press.

[13] B.K. Kim and K.G. Shin. Scalable Hardware Earliest-Deadline-First Scheduler for ATM Switching Networks. *Proceedings of the Real-time Systems Symposium*, pages 210–218, 1997.

[14] H. Zhang and D. Ferrari. Rate-controlled service disciplines. *Journal of High-Speed Networks*, 3(4):389–412, 1994.

[15] Tobias Bjerregaard and Jens Sparsø. A scheduling discipline for latency and bandwidth guarantees in asynchronous network-on-chip. In *ASYNC*, pages 34–43, 2005.

[16] R.L. Cruz. A calculus for network delay. I. Network elements in isolation. *IEEE Transaction on Information Theory*, 37(1):114–131, 1991.

[17] Jean-Yves Le Boudec and Patrick Thiran. *Network calculus: a theory of deterministic queuing systems for the internet*. Springer-Verlag New York, Inc., New York, NY, USA, 2001.

[18] Dimitrios Stiliadis and Anujan Varma. Latency-rate servers: a general model for analysis of traffic scheduling algorithms. *IEEE/ACM Transaction on Networking*, 6(5):611–624, 1998.

[19] Abhay K. Parekh and Robert G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM Trans. Netw.*, 1(3):344–357, 1993.

[20] Maarten H. Wiggers, Marco J. G. Bekooij, and Gerard J. M. Smit. Modelling run-time arbitration by latency-rate servers in dataflow graphs. In *SCOPES '07: Proceedingsof the 10th international workshop on Software & compilers for embedded systems*, pages 11–22, New York, NY, USA, 2007. ACM.

[21] Rajeev Agrawal and Rajendran Rajan. Performance bounds for guaranteed and adaptive services. Technical Report RC20649 (91385), IBM Research Division, May 1996.

[22] Clara Otero Pérez, Martijn Rutten, Jos van Eijndhoven, Liesbeth Steffens, and Paul Stravers. Resource reservations in shared-memory multiprocessor SOCs. In Peter van der Stok, editor, *Dynamic and Robust Streaming In And Between Connected Consumer-Electronics Devices*, chapter 5, pages 109 – 137. Springer, 2005.

[23] Benny Akesson, Kees Goossens, and Markus Ringhofer. Predator: a predictable sdram memory controller. In *CODES+ISSS '07: Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, pages 251–256, New York, NY, USA, 2007. ACM.

[24] Kees Goossens, John Dielissen, and Andrei Rădulescu. The Æthereal network on chip: Concepts, architectures, and implementations. *IEEE Design and Test of Computers*, 22(5):414–421, Sept-Oct 2005.

[25] Andrei Rădulescu *et al.* An efficient on-chip network interface offering guaranteed services, shared-memory abstraction, and flexible network programming. *IEEE Transaction on CAD of Integrated Circuits and Systems*, 24(1):4–17, January 2005.

[26] Andreas Hansson, Martijn Coenen, and Kees Goossens. Undisrupted quality-of-service during reconfiguration of multiple applications in networks on chip. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 954–959, April 2007.

[27] Jan-Willem van de Waerdt, Stamatis Vassiliadis, Sanjeev Das, Sebastian Mirolo, Chris Yen, Bill Zhong, Carlos Basto, Jean-Paul van Itegem, Dinesh Amirtharaj, Kulbhushan Kalra, Pedro Rodriguez, and Hans van Antwerpen. The TM3270 Media-Processor. In *MICRO 38: Proceedings of the 38th annual IEEE/ACM International Symposium on Microarchitecture*, pages 331–342, Washington, DC, USA, 2005. IEEE Computer Society.

