Efficient Service Allocation in Hardware Using Credit-Controlled Static-Priority Arbitration

Benny Akesson[1], Liesbeth Steffens[2], Kees Goossens[2,3]
[1]Eindhoven University of Technology
[2]NXP Semiconductors Research
[3]Delft University of Technology

***Abstract*—Resources in contemporary systems-on-chip (SoC) are shared between applications to reduce cost. Access to shared resources is provided by arbiters that require a small hardware implementation and must run at high speed. To manage heavily loaded resources, such as memory channels, it is also important that the arbiter minimizes over allocation. A Credit-Controlled Static-Priority (CCSP) arbiter comprised of a rate regulator and a static-priority scheduler has been proposed for scheduling access to SoC resources. The proposed rate regulator, however, is not straight-forward to implement in hardware, and assumes that service is allocated with infinite precision.**

In this paper, we introduce a fast and small hardware implementation of the CCSP rate regulator and formally prove its correctness. We also show an efficient way of representing the allocated service in hardware with finite precision. Based on this representation, we define and evaluate two allocation strategies, and derive tight bounds on their respective over allocations. We show that increasing the precision of the implementation results in an exponential reduction in maximum over allocation at the cost of a linear increase in area. We demonstrate that the allocation strategy has a large impact on the allocation success rate for use cases with high load. Finally, we compare CCSP to traditional frame-based approaches and conclude that having a fine allocation granularity that is decoupled from latency is essential to manage highly loaded resources in real-time systems.

***Index Terms*—real-time; arbitration; SoC; implementation; CCSP; over allocation**

I. Introduction

A contemporary system-on-chip (SoC) consists of a large number of intellectual property components, such as hardware accelerators and processors. These systems have many use cases, comprised of concurrently executing applications with real-time requirements [1]. We consider hard real-time applications, requiring a *guaranteed minimum service rate and a bounded maximum latency* that can be analytically verified at design time to guarantee the functional correctness of the SoC [2], [3].

Resources, such as memory channels, are shared between applications to reduce system cost. We refer to users of these resources as *requestors*, corresponding to communication channels of an application mapped on the SoC. Some resources, such as memory channels, are scarce and loaded close to maximum capacity [3]. To manage such resources, it is imperative that the arbiter provides the guaranteed service rate to a requestor without reserving more capacity than required, referred to as *over allocation*. The over allocation of an arbiter depends on its allocation granularity, which is limited by the precision of the service allocation mechanism in the hardware implementation.

A Credit-Controlled Static-Priority (CCSP) arbiter comprised of a rate regulator and a static-priority scheduler has been proposed in [4] for scheduling access to shared SoC resources. This arbiter has two properties that are essential to satisfy our requirements: 1) it belongs to the class of latency-rate ($\mathcal{LR}$) servers and guarantees an allocated service rate and a bounded maximum latency to each requestor, and 2) the static-priority scheduler decouples latency and rate, such that low latency can be provided to any requestor, regardless of its allocated rate. However, the work in [4] only presents a model of the proposed rate regulator that assumes service allocation with infinite precision, and does not explain how to implement it in hardware.

The contribution of this paper is a fast and small hardware implementation of the CCSP rate regulator that allows over allocation to become negligible, making it useful for SoC resources with very high loads. We furthermore explore how to efficiently represent the service allocation in hardware, explore the trade-off between over allocation and area of the implementation, and investigate how over allocation affects the provided service.

The rest of this paper is organized as follows. First, we review related work in Section II and explain why existing arbiters do not satisfy our requirements. We then introduce a formal service model in Section III, before recapitulating the CCSP arbiter in Section IV. In Section V, we present an efficient way of representing the allocated service in hardware using finite precision. Two allocation strategies based on this representation are then evaluated and we derive tight bounds on their respective over allocations. We then derive a simple implementation of CCSP's accounting mechanism that is suitable for hardware implementation and formally prove its correctness. We also present synthesis results and show that increasing the precision of the rate regulator causes an exponential reduction in maximum over allocation at the cost of a linear increase in area. In Section VI, we experimentally compare CCSP using the two allocation strategies to traditional frame-based approaches and show that having a fine allocation granularity that is decoupled from latency is essential to manage resources with high loads in real-time systems. Lastly, we present conclusions in Section VII.

II. Related work

Resource arbitration has been extensively researched in different contexts during the past half century. Still, emerging technologies like SoCs continues to change the requirements. Existing arbiters are unsuitable for the SoC context for at least one of the following three reasons: 1) they cannot run at *high clock speed* with a *small implementation*, 2) *allocation granularity is coupled to latency*, resulting in long latencies or over allocation due to discretization, or 3) *latency is coupled to rate*, preventing low latency from being provided to requestors with low rate requirements without over allocating. We proceed by discussing these problems in more detail.

Much work has been carried out in the real-time community concerning server-based processor scheduling of aperiodic and sporadic requestors [5]. These schedulers, however, are designed to be implemented in software, and are often not suitable for hardware implementation. The sporadic server [6], for example, uses a complex accounting mechanism that is difficult to implement efficiently in hardware. Another example is the Constant Bandwidth Server [2] that uses an earliest-deadline-first (EDF) scheduler, which requires a sorted priority queue. The hardware implementation of an EDF scheduler in [7] uses a tree of multiple-bit comparators to compare deadlines in the priority queue, which is a relatively slow operation that may make it difficult to keep up with some SoC resources.

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 [8] and Deficit Round-Robin [9] are extensions that guarantee each requestor a minimum service, proportional to an allocated rate, in a common periodically repeating frame of fixed size. This type of *frame-based* rate regulation is similar to the Deferrable Server [10], and suffers from an *inherent coupling between allocation granularity and latency*, where allocation granularity is inversely proportional to the frame size [11]. Larger frame size results in finer allocation granularity, reducing over allocation, but at the cost of increased latencies for all requestors. Another common example of frame-based scheduling disciplines is time-division multiplexing that suffers from the additional disadvantage that it requires a schedule to be stored for each configuration, which is very costly if the frame size or the number of use cases are large.

The granularity issue is addressed in [12]–[14] with hierarchical framing strategies that accomplish exact allocation over multiple frames. However, these algorithms, just as the family of Fair Queuing algorithms [11], 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 is inversely proportional to the allocated rate. Requestors with low rate requirements hence suffer from long latencies unless their rates are increased, resulting in over allocation.

Four approaches using static-priority scheduling are presented in [15]–[18]. Static-priority schedulers have the benefit of decoupling latency and rate and are cheap to implement in hardware. However, the arbiters in [15], [17], [18] have significant shortcomings, as the rate regulators are frame based and couple allocation granularity and latency. In [16], 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 latency. Moreover, at most 84% of the resource capacity can be allocated to the requestors as guaranteed service.

A priority-based arbiter is presented in [19] for resource scheduling in SoCs. The rate regulator uses an accounting mechanism based on integers that is easily implemented in hardware, and inspired the implementation in this paper. The arbiter, however, does not meet our requirements, as no results are presented on latency, over allocation, or area.

In this paper, we present a fast and small hardware implementation of the CCSP rate regulator and derive its allocation properties. Unlike any of the previously mentioned works, we explore how to efficiently represent the service allocation in hardware, present a trade-off between over allocation and area of the implementation, and investigate how over allocation affects the provided service. We furthermore show that our implementation allows over allocation to become negligible, which is essential for scarce SoC resources with very high loads, such as memory channels.

III. Formal service model

In this section, we introduce the formal service model used in this paper. This is a compacted version of the model in [4] that is sufficient to recapitulate the CCSP arbiter, and to derive new results required by the hardware implementation of the rate regulator. We use capital letters (A) to denote sets, hats to denote upper bounds ($\hat{a}$), and checks to denote lower bounds ($\check{a}$). We adopt an abstract resource view, where a *service unit* corresponds to the access granularity of the resource. Time is discrete and a time unit, referred to as a *service cycle*, is defined as the time required to serve such a service unit. Cumulative service curves are used to model the interaction between the resource and the requestors. We let $\xi(t)$ denote the value of a service curve ξ at service cycle t. We furthermore use $\xi(\tau, t) = \xi(t+1) - \xi(\tau)$ to denote the difference in values between the endpoints of the closed interval $[\tau, t]$.

A requestor generates requests of variable but bounded size, as stated in Definition 1. Requests arrive in separate buffers per requestor at the resource. This is captured by the requested service curve, w, defined in Definition 2.

Definition 1 (Request): The k:th request ($k \in \mathbb{N}$) from a requestor $r \in R$ is denoted $\omega_r^k \in \Omega_r$. The size of ω_r^k in service units is denoted $s(\omega_r^k) : \Omega_r \to \mathbb{N}^+$.

Definition 2 (Requested service curve): The requested service curve of a requestor $r \in R$ is denoted $w_r(t) : \mathbb{N} \to \mathbb{N}$, where $w_r(0) = 0$ and

$$w_r(t+1) = \begin{cases} w_r(t) + s(\omega_r^k) & \omega_r^k \text{ arrived at } t+1 \\ w_r(t) & \text{no request arrived at } t+1 \end{cases}$$

The provided service curve, w', reflects the amount of service units provided by the resource to a requestor. The provided service curve is defined in Definition 3, where $\gamma(t) : \mathbb{N} \to R \cup \{\varnothing\}$ denotes the scheduled requestor at time t. The backlog of a requestor corresponds to the amount of requested service that has not yet been served at a particular time, as defined in Definition 4. An illustration of a requested service curve and a provided service curve along with their corresponding bounds and related concepts is provided in Figure 1.

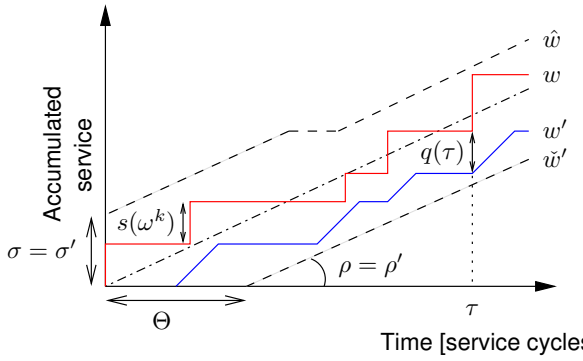

Fig. 1. A requested service curve and a provided service curve with corresponding bounds and related concepts.

Definition 3 (Provided service curve): The provided service curve of a requestor $r \in R$ is denoted $w'_r(t) : \mathbb{N} \to \mathbb{N}$, where $w'_r(0) = 0$ and

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

Definition 4 (Backlog): The backlog of a requestor $r \in R$ at a time t is denoted $q_r(t) : \mathbb{N} \to \mathbb{N}$, and is defined as $q_r(t) = w_r(t) - w'_r(t)$.

We use the (σ, ρ) model [20] to characterize the requested service curve. This model uses a linear function to express a burstiness constraint. The bounding function is determined by two parameters, σ and ρ, corresponding to burstiness and average request rate, respectively. The graphical interpretation of these parameters is shown in Figure 1. In this paper, we assume that all requestors have been characterized according to Definition 5. An example of how to perform this characterization is presented in [21].

Definition 5 (Requestor): A requestor $r \in R$ is characterized by $(\sigma_r, \rho_r) \in \mathbb{R}^+ \times \mathbb{R}^+$, such that for any interval $[\tau, t]$: $\hat{w}_r(\tau, t) = \sigma_r + \rho_r \cdot (t - \tau + 1)$.

IV. Recapitulation of CCSP Arbitration

The CCSP arbiter was originally proposed in [4]. In this section, we recapitulate the most important aspects of the arbiter, starting with an overview in Section IV-A. We proceed in Section IV-B by explaining the model behind CCSP's active period rate regulation. This model is not straight-forward to implement in hardware, and assumes that service is allocated with infinite precision. In Section V, we introduce a fast and small hardware implementation of the model that uses finite precision, and formally prove the equivalence to the regulation presented in this section.

A. Overview

A CCSP arbiter consists of a rate regulator and a scheduler. The regulator provides *accounting* and *enforcement* and determines which requests are *eligible* for scheduling at a particular time, considering their allocated service.

The service allocated to a requestor consists of two parameters, as defined in Definition 6. These are the allocated burstiness, σ', and allocated service rate, ρ', 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 maintain 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 service unit. The last condition is required for the latency bound of the arbiter to be valid. Note that the allocated service uses real values, possibly requiring infinite precision to be accurately represented.

Definition 6 (Allocated service): The service allocation of a requestor $r \in R$ is defined as $(\sigma'_r, \rho'_r) \in \mathbb{R}^+ \times \mathbb{R}^+$. 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 $\forall r \in R : \sigma'_r \geq 1$.

CCSP uses a static-priority scheduler, because it decouples latency and rate and has a simple hardware implementation. Each requestor is assigned a unique priority level, where a lower level indicates higher priority. In this paper, we consider a non-work-conserving static-priority scheduler that is preemptive on the granularity of a single service unit. The effects of work-conservation and non-preemptive scheduling are discussed in [22].

It is shown in [4] that CCSP belongs to the class of $\mathcal{LR}$ servers [23], which is a general framework for analyzing scheduling algorithms. A $\mathcal{LR}$ server is defined by two parameters, Θ and ρ', being service latency and allocated rate, respectively. In essence, a $\mathcal{LR}$ server guarantees a requestor service according to its allocated rate, after a waiting time maximally equal to its service latency, as illustrated by the lower bound on provided service, $\check{w}'$, in Figure 1. The service latency hence intuitively corresponds to the maximum interference from other requestors. It is proven in [4] that the service latency of a requestor r_i using CCSP is given by Equation (1), where $R^+_{r_i}$ denotes the set of requestors with higher priority

than r_i. Note in Equation (1) that service latency and rate of a requestor are decoupled by the priorities through the set of higher priority requestors. Low service latency can hence be provided to any requestor, regardless of its allocated rate, by assigning it a high priority.

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

B. Active period rate regulation

CCSP regulates provided service based on the notion of *active periods*. Definition 7 states that a requestor is active at t if it is either live at t (Definition 8), backlogged at t, or both. Definition 8 states that a requestor must on average have requested service according to its allocated rate since the start of the active period to be considered live at a time t. We denote the set of requestors that are active at t and live at t with R_t^a and R_t^l, respectively.

Definition 7 (Active period): An active period of a requestor $r \in R$ is defined as the maximum interval $[\tau_1, \tau_2]$, such that $\forall t \in [\tau_1, \tau_2] : w_r(\tau_1 - 1, t - 1) \geq \rho'_r \cdot (t - \tau_1 + 1) \vee q_r(t) > 0$. Requestor r is active $\forall t \in [\tau_1, \tau_2]$.

Definition 8 (Live requestor): A requestor $r \in R$ is defined as live at a time t during an active period $[\tau_1, \tau_2]$ if $w_r(\tau_1 - 1, t - 1) \geq \rho'_r \cdot (t - \tau_1 + 1)$.

Figure 2 illustrates the relation between being live, backlogged and active. Note, however, that the lower bound on provided service, w', has been omitted for clarity. Three requests arrive starting from τ_1, keeping the requestor live until τ_3. The requestor is initially both live and backlogged, but the provided service curve catches up with the requested service curve at τ_2. This puts the requestor in a live and not backlogged state until τ_3. The requestor is neither 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 not enough service is provided to remove the backlog. The requestor is hence backlogged, but not live at τ_5, and remains such until the end of the shown interval. The requestor in Figure 2 is active between τ_1 and τ_3 and from τ_4 and onwards, according to Definition 7. Note from this example that a live requestor is not necessarily backlogged, nor vice versa.

The enforced upper bound on provided service, $\hat{w}'$, is defined according to Definition 9. The intuition behind the definition is that the bound of an active requestor increases according to the allocated rate every service cycle, as shown in Figure 2. Conversely, for an inactive requestor, the bound is limited to $w'(t) + \sigma'$, a value that depends on the allocated burstiness. This prevents that a requestor that has been inactive for an extended period of time increases its bound, possibly resulting in starvation of other requestors once it becomes active again.

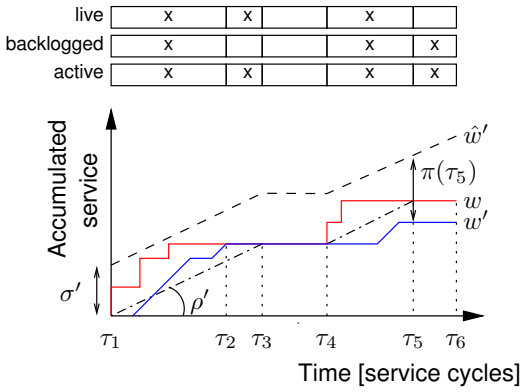

Fig. 2. Illustration of the relation between being live, backlogged, and active.

Definition 9 (Provided service bound): The enforced upper bound on provided service of a requestor $r \in R$ is denoted $\hat{w}'_r(t) : \mathbb{N} \rightarrow \mathbb{R}^+$, where $\hat{w}'_r(0) = \sigma'_r$ and

$$\hat{w}'_r(t+1) = \begin{cases} \hat{w}'_r(t) + \rho'_r & r \in R_t^a \\ w'_r(t) + \sigma'_r & r \notin R_t^a \end{cases} \tag{2}$$

It is not possible to perform accounting and enforcement in hardware based on $\hat{w}'$, since $\lim_{t\rightarrow\infty} \hat{w}'(t) = \infty$, which cannot be represented in an implementation with finite precision. Instead, the accounting is based on the potential of a requestor, defined as $\pi_r(t) = \hat{w}'_r(t) - w'_r(t)$. The potential of a requestor is bounded since the arbiter guarantees a lower bound on provided service. We arrive at the potential-based accounting mechanism in Definition 10 by subtracting $w'_r(t+1)$ from both sides in Equation (2) and applying Definition 3, as shown in [22]. The graphical interpretation of potential is shown in Figure 2.

Definition 10 (Potential-based accounting): The accounted potential of a requestor $r \in R$ is denoted $\pi_r(t) : \mathbb{N} \rightarrow \mathbb{R}$, where $\pi_r(0) = \sigma'_r$ and

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

Enforcement in the rate regulator takes place before the accounting is updated in a service cycle, and is performed by determining if a request from a requestor is eligible for scheduling. A request is defined as eligible if the following three conditions are satisfied: 1) all previous requests from the requestor are served, 2) the requestor is backlogged, and 3) the requestor has at least enough potential to serve one service unit, including the service earned when the accounting is updated, i.e. $\pi(t) \geq 1 - \rho'$. The eligibility information is used by the static-priority scheduler that schedules the highest priority eligible requestor every service cycle.

V. Regulator implementation

Having recapitulated the CCSP arbiter, we proceed in this section by deriving the hardware implementation of the rate regulator. While doing so, we also present the main theoretical contributions of this paper. First in Section V-A, we discuss how to represent the allocated service, (σ', ρ'), in hardware using finite precision and present two allocation strategies that address different aspects of over allocation. We proceed in Section V-B by providing tight bounds on over allocation of the two strategies and comparing these results to those of a frame-based arbiter. An implementation of the CCSP rate regulator based on simple integer arithmetic is derived in Section V-C, and we formally prove the equivalence between this implementation and the model of active period rate regulation from Section IV-B. Synthesis results are shown in Section V-D, indicating that our implementation provides an exponential reduction in maximum over allocation at the cost of a linear increase in area.

A. Service representation

The hardware implementation of the rate regulator only offers finite precision for representing the service allocation of a requestor, potentially causing it to be discretized. We hence associate each requestor with a discrete service allocation, denoted (σ'', ρ''), that *conservatively approximates* the real-valued allocation in Definition 6. The discrete allocated rate is represented as a fraction of integers, as proposed in [19], whose maximum size is limited by the number of bits used to represent them in the implementation. This provides a design time trade-off between precision and area, as we will see in Section V-D. The discrete allocated rate and burstiness are formally defined in Definition 11 and Definition 12, respectively.

Definition 11 (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 $n_r, d_r \in \mathbb{N}^+$ and $n_r \leq d_r < 2^\beta$.

Definition 12 (Discrete allocated burstiness): The discrete allocated burstiness of a requestor $r \in R$ is denoted $\sigma''_r \in \mathbb{Q}^+$, and is defined as $\sigma''_r = \frac{\lceil \sigma'_r \cdot d_r \rceil}{d_r}$.

The conservative approximation of the allocated service may cause the allocated rate and burstiness to be over allocated. We define the over-allocated rate of a requestor according to Definition 13. This definition shows us how much of the resource capacity is lost when service is allocated to a requestor. We are also interested in the over-allocated burstiness, defined in Definition 14, since the service latency of CCSP in Equation (1), depends on both the allocated rate and the allocated burstiness. This allows us to study how over allocation impacts the service latency of the arbiter. It follows from these definitions that the total over-allocated rate and burstiness are obtained by summing over the set of requestors sharing the resource.

Definition 13 (Over-allocated rate): The over-allocated rate of a requestor $r \in R$ is denoted $o_\rho(\rho''_r, \rho'_r) : \mathbb{Q}^+ \times \mathbb{R}^+ \rightarrow \mathbb{R}$, and is defined according to $o_\rho(\rho''_r, \rho'_r) = \rho''_r - \rho'_r$.

Definition 14 (Over-allocated burstiness): The over-allocated burstiness of a requestor $r \in R$ is denoted $o_\sigma(\sigma''_r, \sigma'_r) : \mathbb{Q}^+ \times \mathbb{R}^+ \rightarrow \mathbb{R}$, and is defined according to $o_\sigma(\sigma''_r, \sigma'_r) = \sigma''_r - \sigma'_r$.

There are multiple strategies when selecting the n and d of a requestor to allocate its service. It follows directly from Definition 12 and Definition 14 that $o_\sigma(\sigma'', \sigma') = \frac{\lceil \sigma' \cdot d \rceil}{d} - \sigma' < \frac{1}{d}$, and hence that a large d reduces the over-allocated burstiness. This may, however, not provide the closest approximation of the allocated rate, resulting in wasted resource capacity. Considering this, we present two *allocation strategies*. The first strategy, called *Closest Rate Approximation* (CRA), involves approximating the allocated rate as closely as possible to reduce wasted resource capacity, with a secondary objective to reduce the over-allocated burstiness. Conversely, the second strategy, referred to as *Closest Burstiness Approximation* (CBA), attempts to reduce the service latency by closely approximating the allocated burstiness, and reducing the over-allocated rate as a secondary objective.

CRA chooses the n and d, such that ρ'' is the minimum rate that satisfies $\rho'' \geq \rho'$. If there are multiple n and d pairs providing equal approximations of the allocated rate (e.g. $\frac{1}{2} = \frac{2}{4}$), the one with the largest d is preferred to improve the approximation of the allocated burstiness. CBA, on the other hand, picks the largest possible d to reduce the over-allocated burstiness. To provide the best possible conservative approximation of the allocated rate, given the selected d, this implies $n = \lceil \rho' \cdot d \rceil$. Next, we derive the allocation properties of these strategies.

B. Allocation properties

In this section, we analytically examine the properties of the CRA and CBA allocation strategies and compare them to those of a frame-based arbiter. We start in Lemma 1 by bounding the over-allocated rate of both strategies, revealing that it reduces exponentially with the number of bits, β, used to represent n and d.

Lemma 1: The over-allocated rate of a requestor in a CCSP arbiter with a precision of β bits is upper bounded according to $o^{ccsp}_\rho(\rho'', \rho') < \frac{1}{2^\beta - 1}$.

Proof: The over-allocated rate is defined as $o_\rho(\rho'', \rho') = \rho'' - \rho'$, according to Definition 13. We know from Definition 11 that $\rho'' = n/d$. For CBA, it holds that $d = 2^\beta - 1$ and $n = \lceil d \cdot \rho' \rceil$. CRA also falls back on this allocation, unless there is another n, d pair that yields a tighter approximation. By substituting these results into Lemma 1 and performing basic algebraic manipulation, we arrive at $o^{ccsp}_\rho(\rho'', \rho') < \frac{\lceil d \cdot \rho' \rceil}{d} - \frac{d \cdot \rho'}{d} < 1/d$. The proof is concluded by substituting $d = 2^\beta - 1$. □

The derived bound on over-allocated rate is tight for CBA. For CRA, however, it is only tight for requestors with $\rho' = \varepsilon$, where ε is close to zero. In this case, the closest approximation for both strategies is given by choosing $n = 1$ and $d = 2^\beta - 1$, which is the smallest rate that can be represented, given a particular precision. This results in $o_\rho^{ccsp}(\rho'', \rho') = \frac{1}{2^\beta - 1} - \varepsilon \approx \frac{1}{2^\beta - 1}$, corresponding to the worst case. The worst-case reoccurs for CBA when $\forall k \in \mathbb{N}^+, 1 < k < 2^\beta - 1 : \rho' = \frac{k}{2^\beta - 1} + \varepsilon$, as it results in a discrete allocation according to $\rho'' = \frac{k+1}{2^\beta - 1}$. By now, CRA is guaranteed to find a solution that results in a tighter approximation. CRA hence results in a smaller over-allocated rate than suggested by the bound for larger allocated rates, as we will show experimentally in Section VI. However, no tighter bound exists for this strategy that supports arbitrary allocations.

We proceed by bounding the over-allocated burstiness for a requestor using the two allocation strategies. Since a requestor always uses the largest possible d under the CBA strategy, it follows directly that $o_\sigma^{cba}(\sigma'', \sigma') < \frac{1}{d} < \frac{1}{2^\beta - 1}$. The over-allocated burstiness of a requestor using the CRA strategy is derived in Lemma 2.

Lemma 2: The over-allocated burstiness of a requestor using the CRA strategy in a CCSP arbiter with a precision of β bits is upper bounded according to $o_\sigma^{cra}(\sigma'', \sigma') < \frac{2}{2^\beta - 1}$.

Proof: We know that the over-allocated burstiness for CCSP is upper bounded by $\frac{1}{d}$. The CRA strategy uses the n and d pair with the largest d that provides the tightest approximation of ρ'. We note that for any $d < \frac{2^\beta - 1}{2}$ there exists a $k \in \mathbb{N}^+, k > 1$ such that $\frac{k \cdot n}{k \cdot d}$ is an equivalent allocation of ρ' with a larger d. We hence get that $d \geq \frac{2^\beta - 1}{2}$ and that $o_\sigma^{cra}(\sigma'', \sigma') < \frac{2}{2^\beta - 1}$. □

The bounds computed in this section show that the *over-allocated rate and burstiness monotonically reduce with increased precision* for both strategies. Hence, increasing precision cannot result in more resource capacity being wasted or increased service latency. This property is essential for effective design-space-exploration and optimization algorithms. We compare this result to that of an arbiter with a typical frame-based regulator, such as [8], [15], [18], together with a static-priority scheduler. We refer to this combination as frame-based static-priority (FBSP) in this paper.

FBSP allocates service to a requestor by assigning it a number of slots, ϕ, proportional to the allocated rate, in a frame of size f, which is the same for all requestors. The arbiter only has a single allocation parameter and thus cannot allocate rate and burstiness separately. Instead, the allocated burstiness follows implicitly from the allocated rate and the frame size. The number of slots allocated to a requestor is assigned according to $\phi = \lceil \rho' \cdot f \rceil$. This implies that FBSP allocates service in the same way as CBA if $f = 2^\beta - 1$, and hence that the over-allocated rate is $o_\rho^{fbsp}(\rho'', \rho') < 1/f$. We observe that the maximum over-allocated rate of a requestor is inversely proportional to the frame size, implying that a large frame size is required to provide an efficient allocation. However, the service latency of this arbiter equals $\Theta_{r_i}^{fbsp} = 2 \cdot \sum_{\forall r_j \in R_{r_i}^+} \phi_{r_j} \geq 2 \cdot f \cdot \sum_{\forall r_j \in R_{r_i}^+} \rho'_{r_j}$, and is thus proportional to the frame size. Increasing the frame size to reduce the over-allocated rate increases the implicitly allocated burstiness, and results in a trade-off between low service latency and over allocation. The over-allocated rate and service latency do hence not monotonically reduce with increased frame size for FBSP, which is typical for frame-based arbiters.

C. Credit-based rate regulation

In this section, we derive a simple hardware implementation of the rate regulator model in Section IV-B, based on the discrete representation of the allocated service in Definition 11 and Definition 12. The main difficulty in efficiently implementing the potential-based accounting in Definition 10 lies in knowing if a requestor is active or not. To accomplish this, Definition 7 states that we need to know if a requestor is backlogged or live during a particular service cycle. It is easy to determine if a requestor is backlogged in hardware by checking if there are any requests waiting to be served. Knowing how the requested service during an active period relates to the allocated rate, on the other hand, is more challenging, especially considering that CCSP enforces an upper bound on *provided service* and is only aware of the request at the head of the request buffer of each requestor. Although this design has a number of benefits, as explained in [4], it complicates the hardware implementation, since the regulator cannot directly observe the requested service.

We proceed by showing a simple way of determining if a requestor is live by only looking at its backlog and potential. To do this, we require Lemma 3 and Lemma 4, of which proofs can be found in [22]. The key idea is based on two observations. The first observation is that the provided service curve of a requestor satisfies $w'(\tau_1 - 1, t - 1) \geq \rho' \cdot (t - \tau_1 + 1)$, the condition that has to be satisfied by the requested service curve for a requestor to be live, if $\pi(t) \leq \sigma' - \rho'$. This is shown in Lemma 5. The second observation is that the requested service equals the provided service for non-backlogged requestors, which follows from Definition 4. Combined with the first observation, this implies that it is possible to determine liveness of a non-backlogged requestor by looking at its potential, as shown in Lemma 6. Similarly, that means that there is a lower bound on the potential of an inactive requestor, as shown in Lemma 7. We can now easily determine if a requestor is active or not, solving the previously mentioned problem. We know per definition that a backlogged requestor is active, and we can determine if a non-backlogged requestor is active by looking at the potential. Next, we use these results to derive a simple hardware implementation of the CCSP rate regulator.

Lemma 3: If τ_1 is the start of an active period then $w(\tau_1) > w(\tau_1 - 1) = w'(\tau_1) = w'(\tau_1 - 1)$.

Lemma 4: During an active period $[\tau_1, \tau_2]$, it holds that $\forall t \in [\tau_1, \tau_2] : \ \pi(t) = \hat{w}'(\tau_1) - w'(\tau_1) + \hat{w}'(\tau_1, t - 1) - w'(\tau_1, t - 1)$.

Lemma 5: 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 that the equation in Lemma 4 holds during an active period $[\tau_1, \tau_2]$. Definition 9 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 the equation in Lemma 4 yields $\pi(t) = \sigma' + (t - \tau_1) \cdot \rho' - w'(\tau_1, t-1) \leq \sigma' - \rho'$. The proof is concluded by solving for $w'(\tau_1, t-1)$. □

Lemma 6: 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 5, 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 4 states that $w'_r(t) = w_r(t)$, since $q_r(t) = 0$. From Lemma 3, we additionally know that $w'_r(\tau_1) = w_r(\tau_1 - 1)$. We conclude the proof by substituting these results into the result from Lemma 5. □

Lemma 7: For a requestor $r \notin R^a_t \Rightarrow \pi_r(t) > \sigma'_r - \rho'_r$.

Proof: By negating Definition 7, 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 4, 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 3 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 5 implies that $\pi_r(t) > \sigma'_r - \rho'_r$. □

The hardware implementation of the rate regulator uses a credit-based accounting mechanism. Credits are a discrete representation of potential of a requestor, based on the service representation in Section V-A. The proposed accounting mechanism is presented in Definition 15, and the formal proof of correctness is provided in Theorem 1. The proposed accounting mechanism is simple and only needs the current credit state of each requestor, if they are backlogged or not, and which requestor was scheduled in the service cycle when updating the state. It furthermore only uses integer arithmetic, making it suitable for hardware implementation. Note that the underlying ideas behind this mechanism, as well as the efficient integer representation of the allocated rate and the allocation strategies, are useful to implement other rate regulators that use continuous replenishment and enforce linear bounds, such as (σ, ρ) regulators [20].

Definition 15 (Credit-based accounting): The number of credits of a requestor $r \in R$ is denoted $c_r(t) : \mathbb{N} \to \mathbb{N}$, where $c_r(0) = \sigma''_r \cdot d_r$ and

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

Theorem 1: The credit-based accounting is an implementation of potential-based accounting, where the service allocation of a requestor $r \in R$ equals (σ''_r, ρ''_r), and it holds that $\forall t :\ c_r(t) = \pi_r(t) \cdot d_r$.

Proof: We rewrite the equation in Definition 10 by splitting the second case, where $r \in R^a_t$, in two, according to Definition 7. 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 8 and Lemma 6, $r \in R^l_t$ and $q(t) = 0$ implies that $\pi_r(t) \leq \sigma''_r - \rho''_r$. We use the results from Lemma 7 to rewrite the case where $r \notin R^a_t$, resulting in

$$\pi_r(t+1) = \begin{cases} \pi_r(t) + \rho''_r - 1 & \gamma(t) = r \\ \pi_r(t) + \rho''_r & \gamma(t) \neq r \wedge q_r(t) > 0 \\ \pi_r(t) + \rho''_r & (\gamma(t) \neq r \wedge q_r(t) = 0 \wedge \\ & \ \pi_r(t) \leq \sigma''_r - \rho''_r) \\ \sigma''_r & (\gamma(t) \neq r \wedge q_r(t) = 0 \wedge \\ & \ \pi_r(t) > \sigma''_r - \rho''_r) \end{cases} \quad (3)$$

Multiplying both sides of Equation (3) 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 Definitions 11, 12, and 15 yields

$$c_r(t+1) = \begin{cases} c_r(t) + n_r - d_r & \gamma(t) = r \\ c_r(t) + n_r & \gamma(t) \neq r \wedge q_r(t) > 0 \\ c_r(t) + n_r & (\gamma(t) \neq r \wedge q_r(t) = 0 \wedge \\ & \ c_r(t) \leq c_r(0) - n_r) \\ c_r(0) & (\gamma(t) \neq r \wedge q_r(t) = 0 \wedge \\ & \ c_r(t) > c_r(0) - n_r) \end{cases} \quad (4)$$

To simplify the accounting, we merge the two last cases in Equation (4) into $c_r(t+1) = \min(c_r(t) + n_r, c_r(0))$, where the third case in Equation (4) 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 simple credit-based accounting mechanism in Definition 15. □

The introduction of the credit-based accounting mechanism also affects the enforcement. Similarly to the proof of Theorem 1, we multiply the eligibility criterion in Section IV-B with d and use that $c(t) = \pi(t) \cdot d$ and $n = \rho'' \cdot d$, which results in that a requestor requires $c(t) \geq d - n$ to be considered eligible at t.

D. Synthesis results

The CCSP arbiter has been implemented in RTL VHDL according to the architecture presented in [4]. Synthesis in a 90 nm CMOS process with six ports using eight bits to represent each of n, d, $c(t)$ and $c(0)$ results in a total cell area of 0.0175 mm^2 with a clock frequency of 200 MHz. In [4], we illustrated the scalability of the arbiter by showing that the area increases linearly with the number of requestors. In this paper, we consider synthesis from a different perspective by varying the number of bits used to represent the values in

the register bank to show how the bound on over-allocated rate is traded for area. Figure 3 presents this trade-off for an instance with six ports as the bit widths of n, d, $c(t)$ and $c(0)$ are uniformly changed. The figure shows the bound on over-allocated rate for six requestors and hence corresponds to the bound in Lemma 1 multiplied by six. Note that the exponential reduction in the bound on over-allocated rate comes at a linear increase in area.

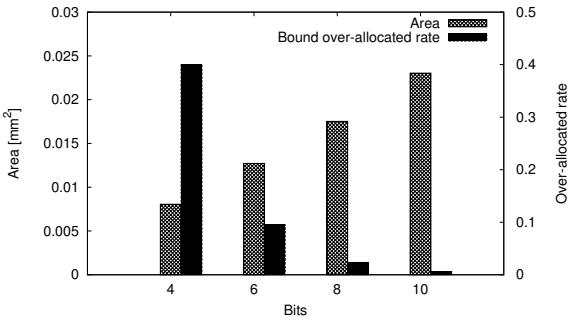

Fig. 3. The over allocation / area trade-off.

VI. Experimental results

In this section, we experimentally evaluate the allocation properties of CCSP. First, we present the experimental setup in Section VI-A. Then, we proceed in Section VI-B by comparing the CRA and CBA allocation strategies. We conclude in Section VI-C by experimentally demonstrating that in contrast to a frame-based static-priority arbiter (FBSP), the allocation behavior of CCSP improves monotonically with increased precision.

A. Experimental setup

The experimental setup consists of a SystemC simulation model of a predictable multi-processor SoC. The processing elements are represented by traffic generators that generate requests that arrive at the resource according to a normal distribution with a programmable average time and variance. We have integrated the CCSP arbiter into the Predator DDR2 SDRAM controller [24], providing predictable access to a 16-bit DDR2-400 SDRAM with a minimum guaranteed bandwidth of 660.9 MB/s. The size of a service unit for this resource is 64 byte and the length of a service cycle is approximately 80 ns. The processing elements communicate with the memory through guaranteed throughput channels provided by the Æthereal [25] network-on-chip.

B. Comparison of allocation strategies

We start by comparing the closest rate approximation (CRA) and closest burstiness approximation (CBA) strategies by looking at how the average and maximum *measured* over-allocated rates and burstinesses relate to each other and to the analytical bounds computed in Section V-B. For each number of requestors in 2, 4, 6, and 8, we randomly generate 1000 synthetic use cases with uniformly distributed loads in the interval [0, 100]%. We are interested in the total over allocation of all requestors and hence add their individual over-allocated rates and burstinesses. Similarly, all derived bounds are multiplied with the number of requestors in the use case. Five bits of precision ($\beta = 5$) are used for both strategies, and σ'_r are real numbers in the range [1, 5] service units. The over-allocated rate is shown in Figure 4.

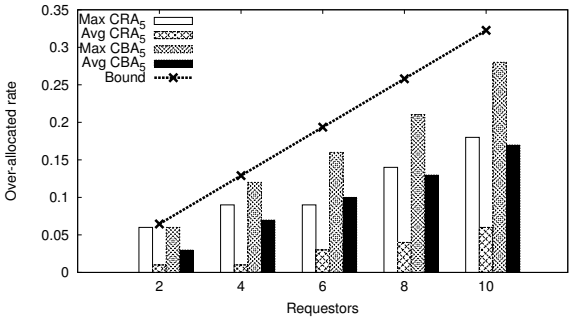

Fig. 4. Over-allocated rate for the CRA and CBA strategies.

We see in the figure that the CRA strategy indeed results in lower average over-allocated rate than CBA. In fact, *the CRA strategy reduces the average over-allocated rate with a factor three compared to CBA*. The maximum measured over-allocated rate is close to the analytical bound for both strategies for use cases with two requestors, although the difference increases with the number of requestors. This reflects that the worst-case over allocation becomes increasingly unlikely as the number of requestors increase. In particular, we note that the difference between the maximum over allocation and the bound becomes very large for CRA, as it is extremely unlikely that a generated use case, much like a realistic one, only contains requestors with allocated rates close to zero.

The over-allocated burstinesses of the two strategies are shown in Figure 5. We see that the CBA representation does reduce the average over-allocated burstiness, although the difference between the two strategies is less significant than for the over-allocated rate. We conclude that *reducing the average over-allocated rate by a factor three using CRA comes at the cost of a 25% increase in the average over-allocated burstiness.* The maximum over-allocated burstiness is close to the bound for CBA, but not for CRA, reflecting the unlikeliness that $d = \frac{2^\beta - 1}{2}$ for all requestors, which is required for its worst-case.

Next, we compare the behavior of the CRA and CBA strategies for use cases with high load and hard service latency requirements. The use cases all have six requestors and are randomly generated with the total load divided in a number of bins (91%, 93%, 95%, 97%, and 99%, respectively). In this experiment, we generate 1000 use cases for each bin. The service latency requirements of the requestors are uniformly distributed in the interval [0, 10000] ns. This range is chosen as it provides requirements that are feasible to satisfy with the considered SDRAM controller and loads. The requirements are then transformed from ns to service cycles using the

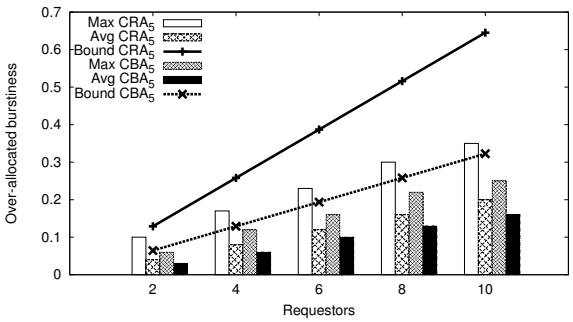

Fig. 5. Over-allocated burstiness for the CRA and CBA strategies.

inverse of the latency functions of the SDRAM controller, presented in [24]. This results in requirements that vary in the range [0, 120] service cycles. In addition to allocating service for the use case, priorities are assigned in an attempt to satisfy the service latency requirements of the requestors. For this purpose, we use the optimal priority assignment algorithm presented in [26]. We compare the two allocation strategies by measuring the percentage of use cases in which the rate requirements of all requestors are satisfied and the total allocated rate is less than 100%, indicating successful allocation. Additionally, we compare the percentage of use cases where the service latency requirements of all requestors are satisfied. Lastly, we study the total success rate, being the percentage of use cases where both service allocation and priority assignment are successful, indicating that both rate and latency requirements are satisfied. The results of this experiment are shown in Figure 6.

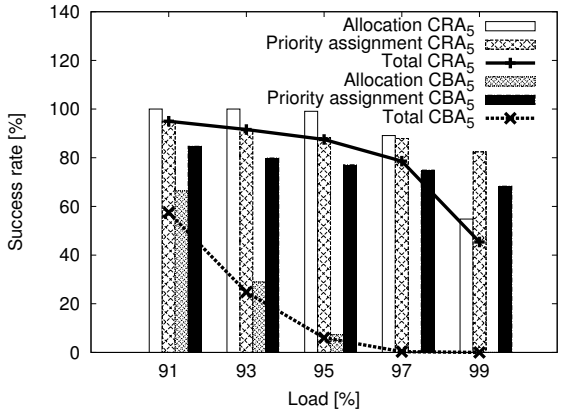

Fig. 6. Successful allocations and priority assignments for CRA and CBA.

We note that all use cases with up to 93% load, and 99.1% of the use cases with 95% load, are successfully allocated when using CRA. The success rate is reduced to 89.1% and 54.8% for use cases with 97% and 99% loads, respectively. As expected, CBA performs worse, and only allocates 66.4% of the use cases with 91% load successfully. The success rate is significantly reduced for higher loads and reaches zero for loads higher than 95%. We see that CRA also performs better when priorities are assigned to satisfy the service latency requirements. The latency requirements are satisfied for 95% of the use cases with 91% load and drops towards 82.4% for use cases with 99%. The trend is similar when using CBA, although it starts at 84.7% for 91% load and ends at 68.3% for loads of 99%. The answer to why CRA is better at satisfying latency requirements, even though CBA provides a closer approximation of the allocated burstiness, is found in Equation (1). We note in the equation that over allocating the burstiness results in a linear increase of the service latency, while over allocating the rate causes a faster increase, favoring the CRA strategy. The total success rate shows that the CRA strategy performs better than CBA for all tested loads, primarily because the smaller over-allocated rate allows more use cases to be successfully allocated. On average, CRA results in more than four times as many use cases with high loads having both their service and latency requirements satisfied. We conclude from this experiment that having a close approximation of the allocated rate is essential to manage heavily loaded resources.

C. Increasing precision

In the next experiment, we study the effects of increasing the precision to achieve a finer allocation granularity. Use cases are randomly generated according to the previous experiment, but we now compare CCSP with five and six bits, respectively, using the CRA strategy.

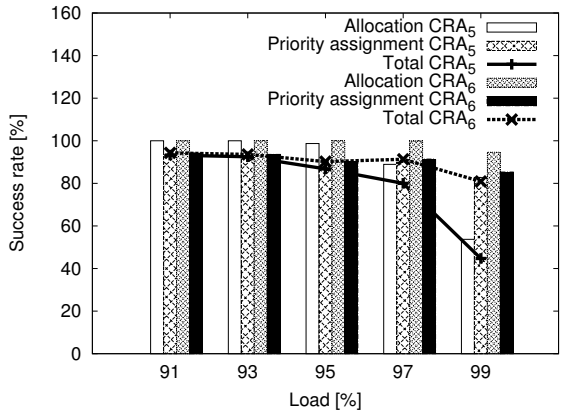

Fig. 7. Increasing precision with CRA.

As seen in Figure 7, increasing precision improves both the number of successful allocations and priority assignments. This is because both the over-allocated rate and burstiness of CCSP are monotonically reduced with increased precision, as explained in Section V-B. We experimentally compare this behavior to that of FBSP in Figure 8, where the frame size, f, is increased from 31 to 63. These particular frame sizes are chosen, as they provide the same bounds on over-allocated rate as for CCSP with five and six bits of precision, used in Figure 7. We first note that the percentage of successful allocations is much lower for FBSP than for CCSP using the CRA strategy. Using a frame size of 31, only 63.7% of the use cases with a load of 91% are successfully allocated and the

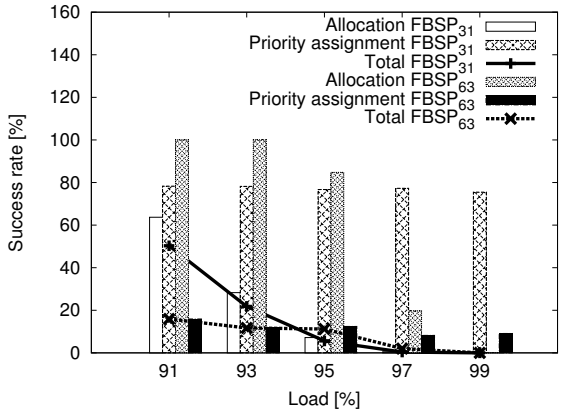

Fig. 8. Increasing precision with FBSP.

success rate drops dramatically for higher loads, approaching zero at loads of 97%. The percentage of successful priority assignments, however, is stable at about 80% across all loads. Doubling the frame size to increase precision results in a significant improvement in the percentage of successful allocation for loads up to 95%, all being above 80%. This, however, causes the percentage of successful priority assignments to be less than 20% for all loads. This is because both the allocation granularity and the service latency depend on the frame size, causing one to be traded for the other. This shows that the coupling between allocation granularity and latency makes FBSP unsuitable for highly loaded resources in the presence of applications with real-time requirements.

VII. Conclusions

In this paper, we derive a fast and small hardware implementation of the rate regulator of a Credit-Controlled Static-Priority (CCSP) arbiter, and formally prove its correctness. The implementation is based on simple integer arithmetic with finite precision, and we show that increasing the precision results in an exponential reduction in maximum over allocation at the cost of a linear increase in area. This allows over allocation to become negligible, which is essential for system-on-chip resources with very high loads, such as memory channels.

We analytically and experimentally compare closest rate approximation (CRA) and closest burstiness approximation (CBA) allocation strategies. We experimentally demonstrate that CRA reduces the average over-allocated rate with a factor of three at the cost of a 25% increase in over-allocated burstiness over CBA. This results in that more than four times as many use cases with high loads have both their rate and latency requirements satisfied. We furthermore show that the allocation behavior of CCSP, unlike most traditional frame-based arbiters, improves monotonically with increased precision, which is essential for effective design-space-exploration and optimization algorithms. From our experiments, we conclude that having a fine allocation granularity that is decoupled from latency is essential to manage resources with high loads in real-time systems.

References

[1] P. Kollig *et al.*, "Heterogeneous Multi-Core Platform for Consumer Multimedia Applications," in *Proc. DATE*, 2009.

[2] L. Abeni and G. Buttazzo, "Resource Reservation in Dynamic Real-Time Systems," *Real-Time Systems*, vol. 27, no. 2, 2004.

[3] K. Goossens *et al.*, "Interconnect and memory organization in SOCs for advanced set-top boxes and TV — Evolution, analysis, and trends," in *Interconnect-Centric Design for Advanced SoC and NoC*. Kluwer, 2004, ch. 15.

[4] B. Akesson *et al.*, "Real-Time Scheduling Using Credit-Controlled Static-Priority Arbitration," in *Proc. RTCSA*, Aug. 2008.

[5] G. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Springer, 2004.

[6] B. Sprunt *et al.*, "Aperiodic task scheduling for Hard-Real-Time systems," *Real-Time Systems*, vol. 1, no. 1, 1989.

[7] J. Rexford *et al.*, "A router architecture for real-time point-to-point networks," in *Proc. ISCA*, 1996.

[8] M. Katevenis *et al.*, "Weighted round-robin cell multiplexing in a general-purpose ATM switch chip," *IEEE J. Sel. Areas Commun.*, vol. 9, no. 8, Oct. 1991.

[9] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round robin," in *Proc. SIGCOMM*, 1995.

[10] J. Strosnider *et al.*, "The Deferrable Server Algorithm for Enhanced Aperiodic Responsiveness in Hard Real-Time Environments," *IEEE Trans. Comput.*, 1995.

[11] H. Zhang, "Service disciplines for guaranteed performance service in packet-switching networks," *Proceedings of the IEEE*, vol. 83, no. 10, Oct. 1995.

[12] C. R. Kalmanek and H. Kanakia, "Rate controlled servers for very high-speed networks," *Proc. GLOBECOM*, 1990.

[13] 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*, 2001.

[14] D. Saha *et al.*, "Carry-over round robin: a simple cell scheduling mechanism for ATM networks," *IEEE/ACM Trans. Netw.*, vol. 6, no. 6, 1998.

[15] S. Hosseini-Khayat and A. Bovopoulos, "A simple and efficient bus management scheme that supports continuous streams," *ACM TOCS*, vol. 13, no. 2, 1995.

[16] T. Bjerregaard and J. Sparsø, "A scheduling discipline for latency and bandwidth guarantees in asynchronous network-on-chip." in *ASYNC*, 2005.

[17] S. Heithecker and R. Ernst, "Traffic shaping for an FPGA based SDRAM controller with complex QoS requirements," in *Proc. DAC*, 2005.

[18] F. Harmsze *et al.*, "Memory arbitration and cache management in stream-based systems," in *Proc. DATE*, 2000.

[19] C. Otero Pérez *et al.*, "Resource reservations in shared-memory multiprocessor SOCs," in *Dynamic and Robust Streaming In And Between Connected Consumer-Electronics Devices*. Springer, 2005.

[20] R. Cruz, "A calculus for network delay. I. Network elements in isolation," *IEEE Trans. Inf. Theory*, vol. 37, no. 1, 1991.

[21] L. Steffens *et al.*, "Real-time analysis for memory access in media processing socs: A practical approach," *Proc. ECRTS*, 2008.

[22] B. Akesson *et al.*, "Real-Time Scheduling of Hybrid Systems using Credit-Controlled Static-Priority Arbitration ," NXP Semiconductors, Tech. Rep., 2007, http://www.es.ele.tue.nl/~kakesson/publications/pdf/NXP-TN-2007-00119.pdf.

[23] D. Stiliadis and A. Varma, "Latency-rate servers: a general model for analysis of traffic scheduling algorithms," *IEEE/ACM Trans. Netw.*, vol. 6, no. 5, 1998.

[24] B. Akesson *et al.*, "Predator: a predictable SDRAM memory controller," in *Proc. CODES+ISSS*, 2007.

[25] K. Goossens *et al.*, "The Æthereal network on chip: Concepts, architectures, and implementations," *IEEE Des. Test. Comput.*, vol. 22, no. 5, Sep. 2005.

[26] N. Audsley, "Optimal priority assignment and feasibility of static priority tasks with arbitrary start times," *Real-Time Systems*, 1991.