On the Use of Greedy Shapers in Real-Time Embedded Systems

ERNESTO WANDELER, ALEXANDER MAXIAGUINE, and LOTHAR THIELE, ETH Zurich

Traffic shaping is a well-known technique in the area of networking and is proven to reduce global buffer requirements and end-to-end delays in networked systems. Due to these properties, shapers also play an increasingly important role in the design of multiprocessor embedded systems that exhibit a considerable amount of on-chip traffic. Despite the growing importance of traffic shapping in this area, no methods exist for analyzing shapers in distributed embedded systems and for incorporating them into a system-level performance analysis. Until now it was not possible to determine the effect of shapers on end-to-end delay guarantees or buffer requirements in such systems. In this work, we present a method for analyzing greedy shapers, and we embed this analysis method into a well-established modular performance analysis framework for real-time embedded systems. The presented approach enables system-level performance analysis of complete systems with greedy shapers, and we prove its applicability by analyzing three case study systems.

Categories and Subject Descriptors: C.3 [**Computer System Organization**]: Special Purpose and Application-Based Systems—*Real-time and embedded systems*

General Terms: Design, Performance, Verification

Additional Key Words and Phrases: Embedded systems, performance analysis, shapers

ACM Reference Format:
Wandeler, E., Maxiaguine, A., and Thiele, L. 2012. On the use of greedy shapers in real-time embedded systems. ACM Trans. Embedd. Comput. Syst. 11, 1, Article 1 (March 2012), 22 pages.
DOI = 10.1145/2146417.2146418 http://doi.acm.org/10.1145/2146417.2146418

1. INTRODUCTION

In the area of broadband networking, traffic shaping is a well-known and well-studied technique for regulating connections and avoiding buffer overflow in network nodes, see, for example, Gringeri et al. [1998] or Rexford et al. [1997]. A traffic shaper in a network node buffers the data packets of an incoming traffic stream and delays them such that the output stream conforms to a given traffic specification. A shaper may ensure, for example, that the output stream has limited burstiness or that packets on the output stream have a specified minimum interarrival time. A greedy shaper is a special instance of a traffic shaper that not only ensures an output stream that conforms to a given traffic specification but also guarantees that no packets get delayed longer than necessary.

Non-greedy shapers are able to keep packets longer than necessary in order to smoothen the output traffic. For example, besides a maximal number of output packets in a given time interval, one may also specify a minimal number in order to avoid a

This research has been funded by the Swiss National Science Foundation (SNF) under the Analytic Performance Estimation of Embedded Computer Systems project 200021-103580/1 and by ARTIST2.
Authors' addresses: E. Wandeler, A. Maxiaguine, and L. Thiele, Computer Engineering and Networks Laboratory, Swiss Federal Institute of Technology (ETH), CH-8092 Zurich, Switzerland; corresponding author's email: thiele@tik.ee.ethz.ch.

DOI 10.1145/2146417.2146418 http://doi.acm.org/10.1145/2146417.2146418

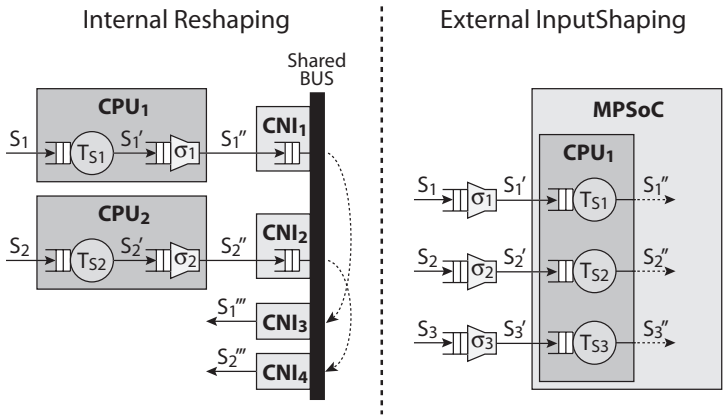

Fig. 1. Two systems with greedy shapers.

buffer underflow or to get a smoother output traffic. Therefore, shaping devices with this property are sometimes called smoothers. They may keep a number of packets as a reserve and send them in case of low input traffic. This way, a violation of the specified minimal packet rate can be avoided. This article concentrates on greedy shapers.

By limiting the burstiness of the output stream of a network node, shapers typically drastically reduce the buffer requirements on subsequent network nodes. In particular, if some sort of priority scheduling is used on a network node to share bandwidth among several incoming streams, then a limited burstiness of high-priority streams leads to better responsiveness of low-priority streams.

In addition, under some circumstances, shaping comes for free from a performance point of view. To be more specific, if the output stream of a node is shaped with a greedy shaper to conform again to the input traffic specification, and if the buffer of the shaper accesses the same memory as the input buffer of the node, then the end-to-end delay of the stream and the total buffer requirements on the network node are not affected by adding the shaper.

Due to these favorable properties, shapers also play an increasingly important role in the design of real-time embedded systems. Modern embedded systems are often implemented as multiprocessor systems with a considerable amount of on-chip traffic.

In this domain, we may identify two main application areas for traffic shaping. First, shapers may be used internally to reshape internal traffic streams to reduce global buffer requirements and end-to-end delays. Second, shapers may be added at the boundaries of a system to ensure conformant input streams, thereby preventing internal buffer overflows caused by malicious input. Figure 1 shows two simple example systems from these two application areas.

Within the framework of Network Calculus Le Boudec and Thiran [2001] present methods for analyzing the effects of traffic shapers in communication networks. But to our best knowledge, none of the existing frameworks for modular system-level performance analysis of real-time embedded system considers traffic shapers at this time, see, for example, Pop et al. [2003], Richter et al. [2003], González Harbour et al. [2001] or Chakraborty et al. [2003], or Thiele et al. [2000].

Only Richter et al. [2003] introduce a restricted kind of traffic shaping through so-called event adaption functions (EAFs). However, EAF's play a crucial role in the method's fundamental ability to analyze systems, and therefore, a designer has very limited freedom to place or omit or parameterize EAF's.

In this work, we will extend the framework presented in Chakraborty et al. [2003] and Thiele et al. [2000] to enable system-level performance analysis of real-time embedded systems with traffic shapers. The first (and preliminary) results on greedy shapers have been published in Wandeler et al. [2006]. It has to be noted that Le Boudec and Thiran [2001] challenge the ability of the methods presented by Thiele et al. [2000] to analyze traffic shapers, Schioler et al. [2005] even claim that it is not possible to analyze traffic shapers within the framework of modular performance analysis [Chakraborty et al. 2003; Thiele et al. 2000]. This article proposes a solution to these problems and challenges.

The contributions of this work can be summarized as follows.

—We present a method for analyzing greedy shapers in multiprocessor embedded systems.
—We embed this new analysis method into the well-established modular performance analysis framework of Chakraborty et al. [2003] and Thiele et al. [2000]. This enables system-level performance analysis of complete systems with greedy shapers, that is, we may analyze end-to-end delay guarantees and global buffer requirements of such systems.
—We prove the applicability of the presented methods by analyzing two small case study systems with greedy shapers.

2. MODULAR PERFORMANCE ANALYSIS

In the domain of communication networks, powerful abstractions have been developed to model flow of data through a network. In particular, Network Calculus [Cruz 1991] provides a means to deterministically reason about timing properties of data flows in queuing networks. Real-Time Calculus [Thiele et al. 2000] extends the basic concepts of Network Calculus to the domain of real-time embedded systems, and Chakraborty et al. [2003] proposes a unifying approach to Modular Performance Analysis with Real-Time Calculus. It is based on a general event and resource model, allows for hierarchical scheduling and arbitration, and takes computation and communication resources into account.

With Real-Time Calculus, hard upper and lower bounds can be computed to various performance measures in a real-time system, such as end-to-end delays of event streams or buffer requirements. Hence, real-Time Calculus qualifies to analyze hard real-time systems. This clearly distinguishes Real-Time Calculus from any probabilistic performance estimation methods or from performance estimation through simulation.

The central idea of Modular Performance Analysis is to first build an abstract performance model of the concrete system that bundles all information needed for performance analysis with Real-Time Calculus. The abstract performance model thereby unifies essential information the environment of the system, the available computation and communication resources, the application tasks (or dedicated HW/SW components), as well as the system architecture itself.

Within the abstract performance model, environment models describe how a system is being used by the environment: how often events (or function calls) will arrive; how much data is provided as input to the system; and how many events and how much data is generated in return by the system and fed back to the environment. Resource models provide information about the properties of the computing and communication resources that are available within a system, such as processor speed and communication bus bandwidth. Application task (or dedicated HW/SW component) models provide information about the processing semantics that is then used to execute the various application tasks or run the dedicated HW/SW components. Finally, the system model captures information about the applications and the available hardware architecture,

defines the mapping of tasks to computation or communication resources of the hardware architecture, and specifies the scheduling and arbitration schemes used on these resources.

Following, we introduce the basic concepts of Network and Real-Time Calculus.

2.1. Arrival Curves: A General Event Stream Model

A trace of an event stream can be described by means of a cumulative function $R(s,t)$, defined as the number of events seen on the event stream in the time interval $[s,t)$. While any R always describes one concrete trace of an event stream, a tuple $\alpha(\Delta) = [\alpha^u(\Delta), \alpha^l(\Delta)]$ of upper and lower *arrival curves* [Cruz 1991] provides an abstract event stream model, representing all possible traces of an event stream.

The upper arrival curve $\alpha^u(\Delta)$ provides an upper bound on the number of events seen on the event stream in any time interval of length Δ, and analogously, the lower arrival curve $\alpha^l(\Delta)$ denotes a lower bound on the number of events in a time interval Δ. R, α^u and α^l are related to each other such that

$$\alpha^l(t-s) \leq R(s,t) \leq \alpha^u(t-s) \quad \forall s < t, \tag{1}$$

with $\alpha^l(0) = \alpha^u(0) = 0$.

Arrival curves substantially generalize traditional event models, for example, as sporadic, periodic, periodic with jitter, or any other arrival pattern with deterministic timing behavior. For example, an event stream with a period p, a jitter j, and a minimum inter-arrival distance d can be modeled by the following arrival curves.

$$\alpha^l(\Delta) = \left\lfloor \frac{\Delta - j}{p} \right\rfloor; \quad \alpha^u(\Delta) = \min\left\{ \left\lceil \frac{\Delta + j}{p} \right\rceil, \left\lceil \frac{\Delta}{d} \right\rceil \right\}. \tag{2}$$

Besides being able to represent any message stream with known deterministic timing behavior, it is also possible to determine arrival curves corresponding to any finite-length message trace, obtained, for example, from observation or simulation.

Figure 2 shows some typical examples of arrival curves. The arrival curves in Figure 2(a) model a strictly periodic event stream, while the arrival curves in Figure 2(b) model a periodic event stream with jitter, and the arrival curves in Figure 2(c) model a periodic event stream with bursts. The arrival curves in Figure 2(d) model an event stream with more complex timing behavior. This event stream may have short steep bursts or longer-lasting less steep bursts, and the maximum long-term period does not equal the minimum long-term period. An event stream with such a complex timing behavior can not be represented accurately by any of the classical event arrival patterns.

2.2. Service Curves: A General Resource Model

Analogously to the cumulative function $R(s,t)$, the concrete availability of a computation or communication resource can be described by a cumulative function $C(s,t)$, defined as the number of available resources, for example, processor or bus cycles, in the time interval $[s,t)$. To provide an abstract resource model, we define a tuple $\beta(\Delta) = [\beta^u(\Delta), \beta^l(\Delta)]$ of upper, β^u and lower β^l, *service curves*. C, β^u, and β^l are related to each other such that

$$\beta^l(t-s) \leq C(s,t) \leq \beta^u(t-s) \quad \forall s < t, \tag{3}$$

with $\beta^l(0) = \beta^u(0) = 0$.

Note that the preceding definition of lower service curves corresponds to the definition of strict service curves in Network Calculus [Le Boudec and Thiran 2001], while the definition of upper service curves, as we have just given, is not used.

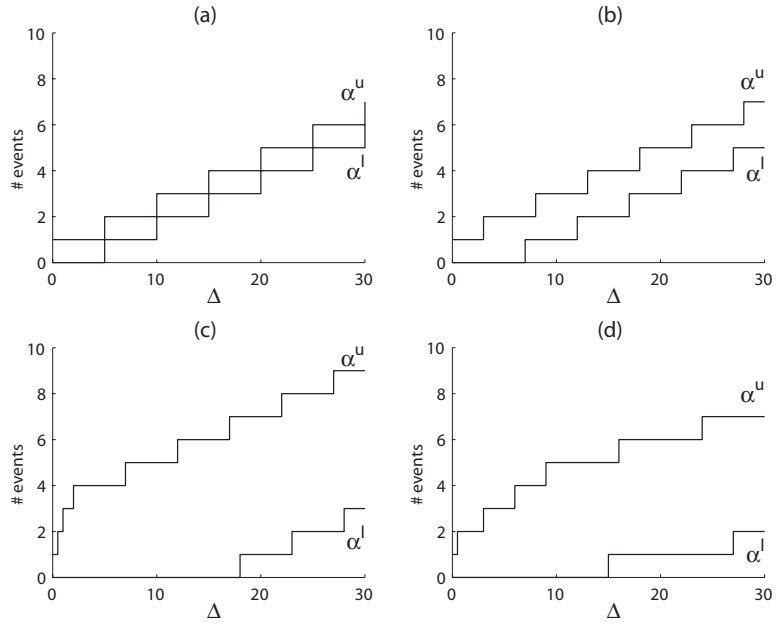

Fig. 2. Arrival curves $\alpha(\Delta)$ for standard event arrival patterns: (a) periodic; (b) periodic with jitter; (c) periodic with bursts; (d) general.

Figure 3 shows some examples of service curves that model the resource availability on processors or communication channels. The service curves in Figure 3(a) model a resource with full availability, while the service curves in Figure 3(b) model a bounded delay resource, as defined in Shin and Lee [2004]. The service curves in Figure 3(c) model the resource availability of one slot on a time division multiple access (TDMA) resource, and finally, the service curves in Figure 3(d) model a periodic resource, as defined in Shin and Lee [2003].

2.3. From Components to Abstract Components

In a real-time system, an incoming event stream is typically processed on a sequence of HW/SW components that we will interpret as tasks that are executed on possibly different hardware resources.

Figure 4 shows such a component. An event stream R enters the component and is processed using a hardware resource whose availability is modeled by C. After being processed, the events are emitted on the component's output, resulting in an outgoing event stream R', and the remaining resources that were not consumed are made available to other components and are described by an outgoing resource availability trace C'.

The relations between R, C, R', and C' depend on the component's processing semantics, and the outgoing event stream R' will typically not equal the incoming event stream R, as it may, for example, exhibit more or less jitter. Analogously, C' will differ from C.

For Modular Performance Analysis with real-time calculus, we model such an HW/SW component as an abstract component, as shown in Figure 5. Here, an abstract event stream $\alpha(\Delta)$ enters the abstract component and is processed using an abstract hardware resource $\beta(\Delta)$. The output is, again, an abstract event stream $\alpha'(\Delta)$,

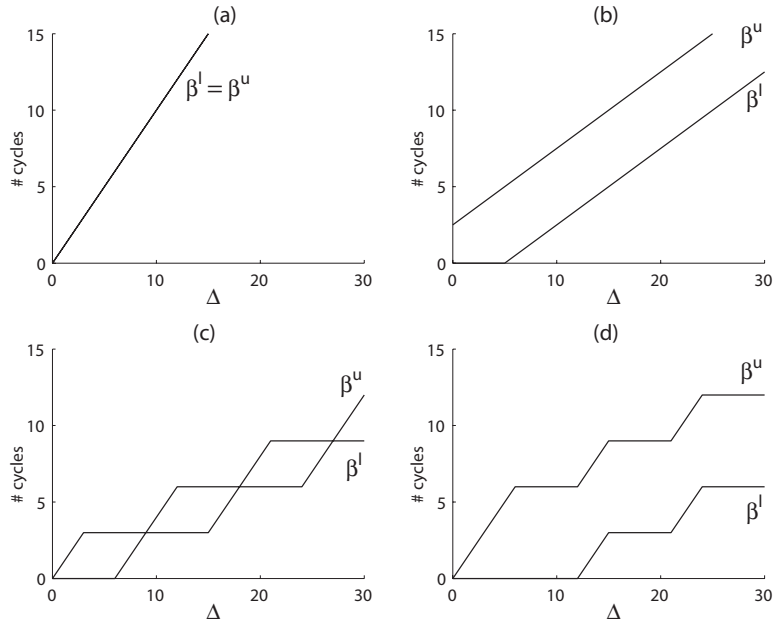

Fig. 3. Services curves $\beta(\Delta)$ for standard resource patterns: (a) complete; (b) bounded delay; (c) TDMA; (d) periodic resource.

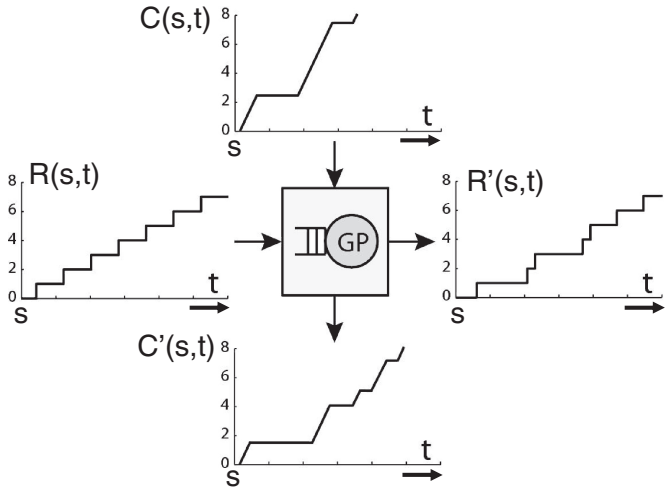

Fig. 4. A concrete component processing an event stream on a resource.

and the remaining resources are expressed, again, as an abstract hardware resource $\beta'(\Delta)$.

Internally, an abstract component is specified by a set of relations that relate the incoming arrival and service curves to the outgoing arrival and service curves, such that

$$\alpha' = f_\alpha(\alpha, \beta); \qquad \beta' = f_\beta(\alpha, \beta). \tag{4}$$

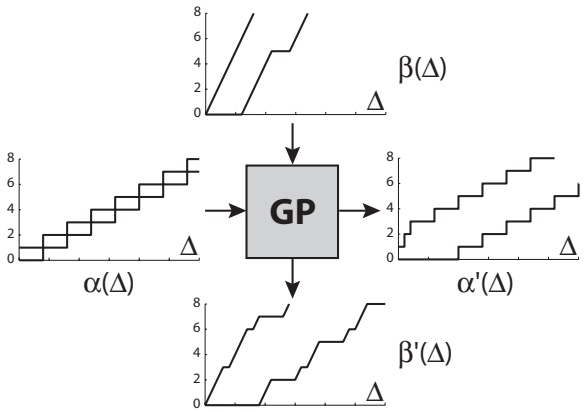

Fig. 5. An abstract component processing an abstract event stream on an abstract resource.

Again, these relations depend on the processing semantics of the modeled component and must be determined such that $\alpha'(\Delta)$ correctly models the event stream with event trace R' and $\beta'(\Delta)$ correctly models the resource availability C'.

As an example of an abstract component, consider a concrete component that is triggered by the events of an incoming event stream. A fully preemptable task is instantiated at every event arrival in order to process the incoming event, and active tasks are processed in a greedy fashion in first in, first out order while being restricted by the availability of resources. Such a component can be modeled as an abstract component with the following internal relations[1] [Chakraborty et al. 2003].

$$\alpha'^{u}_{GP} = \min\{(\alpha^u \otimes \beta^u) \oslash \beta^l, \beta^u\}. \quad (5)$$

$$\alpha'^{l}_{GP} = \min\{(\alpha^l \oslash \beta^u) \otimes \beta^l, \beta^l\}. \quad (6)$$

$$\beta'^{u}_{GP}(\Delta) = \inf_{\Delta \le \lambda}\{\beta^u(\lambda) - \alpha^l(\lambda)\}^+ \quad \forall \Delta \ge 0. \quad (7)$$

$$\beta'^{l}_{GP}(\Delta) = \inf_{0 \le \lambda \le \Delta}\{\beta^l(\lambda) - \alpha^u(\lambda) \quad \forall \Delta \ge 0\}. \quad (8)$$

Such components are very common in the area of real-time embedded systems, and we will refer to them as *Greedy Processing* (GP) components.

2.4. Abstract Performance Models

To analyze the performance of a concrete system, we need to capture its essential properties in an abstract performance model that consists of a set of interconnected abstract components. First, all concrete system components are modeled using their abstract representation (as described in the preceding section). Then, the arrival-curve inputs and outputs of these abstract components are interconnected to reflect the flow event streams through the system.

When several components of the concrete system are allocated to the same hardware resource, they must share this resource according to a scheduling policy. In the performance model, the scheduling policy on a resource can be expressed by the way the abstract resources β are distributed among the different abstract components.

[1]See the Appendix for a definition of $\otimes$ and $\oslash$. Note that these relations are only valid with arrival and service curves that are based on differential cumulative functions $R(s,t)$ and $C(s,t)$ that extend to the whole time domain. The preceding tight relations are not valid with only the arrival and service curves Cruz [1991] defined for the positive time domain ($R(0,t)$ and $C(0,t)$).

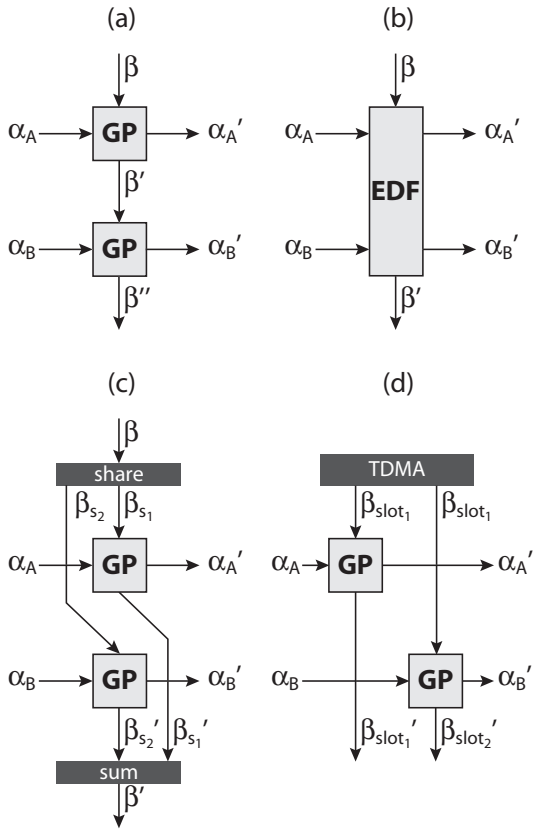

Fig. 6. Modeling of various scheduling and arbitration policies in the system performance model: (a) tasks with preemptive fixed-priority (FP) scheduling; (b) tasks with earliest deadline first (EDF) scheduling; (c) tasks with generalized processor sharing (GPS) scheduling; (d) tasks with time division multiple access (TDMA) scheduling.

For example, consider preemptive fixed-priority scheduling: Abstract component A with the highest priority may use all available resources on a hardware, whereas abstract component B with the second-highest priority only gets the resources that were not consumed by A. This is modeled by using the service curves β'_A that exit A as input to B. For some other scheduling policies, such as GPS or TDMA, resources must be distributed differently, while for some scheduling policies, such as EDF, different abstract components with tailored internal relations (see Equation (4)), must be used. Some examples of how to model different scheduling policies are depicted in Figure 6.

2.5. Analysis

In the performance model of a system, various performance measures can be computed analytically [Le Boudec and Thiran 2001; Chakraborty et al. 2003].

For instance, for a GP component, the maximum delay d_{max} experienced by an event is bounded by

$$d_{max} \leq \sup_{\lambda \geq 0}\{\inf\{\tau \geq 0 : \alpha^u(\lambda) \leq \beta^l(\lambda + \tau)\}\} \stackrel{def}{=} Del(\alpha^u, \beta^l). \quad (9)$$

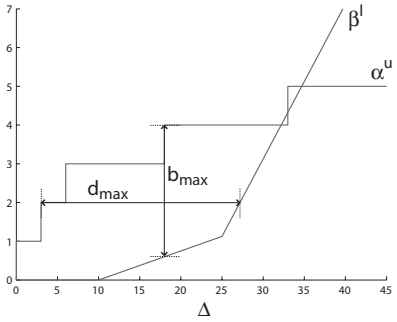

Fig. 7. Delay and backlog obtained from arrival and service curves.

When processed by a sequence of GP components, the total end-to-end delay experienced by an event is bounded by

$$d_{max} \leq Del\big(\alpha^u, \beta_1^l \otimes \beta_2^l \otimes \cdots \otimes \beta_n^l\big). \tag{10}$$

On the other hand, the maximum buffer space b_{max} that is required to buffer an event stream with arrival curve α in the input queue of a GP component on a resource with service curve β is bounded by

$$b_{max} \leq \sup_{\lambda \geq 0}\{\alpha^u(\lambda) - \beta^l(\lambda)\} \stackrel{def}{=} Buf(\alpha^u, \beta^l). \tag{11}$$

When the buffers of consecutive components access the same shared memory, the total buffer space is bounded by

$$b_{max} \leq Buf\big(\alpha^u, \beta_1^l \otimes \beta_2^l \otimes \cdots \otimes \beta_n^l\big). \tag{12}$$

In Figure 7, the relations between α, β, d_{max}, and b_{max} are depicted graphically. From this figure, we see that d_{max} and b_{max} are bounded by the maximum horizontal and maximum vertical distance between the upper arrival curve and the lower service curve, respectively. This corresponds to the intuition that d_{max} and b_{max} occur when the maximum load arrives at the time of minimum resource availability.

3. PERFORMANCE ANALYSIS OF GREEDY SHAPERS

From now on, we will consider one-sided cumulative functions only, that is, $R(t) \equiv R(0,t)$ and $C(t) \equiv C(0,t)$. In other words, we suppose that event streams start at $t = 0$ and we consider the positive time domain only.

In Figure 8, a greedy shaper component is depicted shaping an incoming event stream $R(t)$ with a shaping curve σ. After being shaped, the events are emitted on the component's output, resulting in a shaped outgoing event stream $R'(t)$.

To enable analysis of systems with greedy shapers in the Modular Performance Analysis framework, we need to introduce a new abstract component that models a greedy shaper, as depicted in Figure 9. Here, an abstract event stream $\alpha(\Delta)$ enters the abstract greedy shaper component to be shaped with shaping curve σ. Thus, the shaped output is, again, an abstract event stream $\alpha'(\Delta)$. Analogously to Equation (4), we consequently need to find relations that relate the incoming arrival curves to the outgoing arrival curves in order to specify such an abstract greedy shaper component, such that

$$\alpha' = f_{GS}(\alpha, \sigma). \tag{13}$$

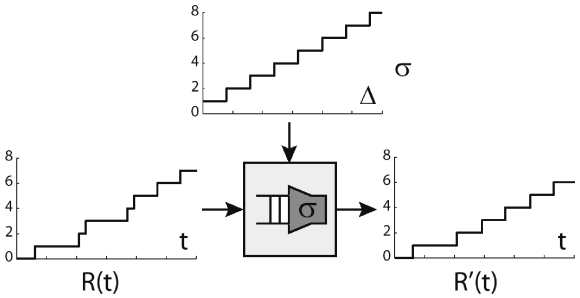

Fig. 8. A concrete greedy shaper that is shaping a concrete event stream.

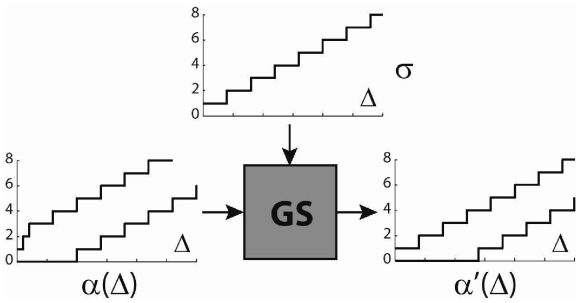

Fig. 9. An abstract greedy shaper that is shaping an abstract event stream.

Following, we will first explain the behavior and implementation of concrete greedy shapers. We will then introduce the internal relations that define an abstract greedy shaper component.

3.1. Concrete Greedy Shapers

A greedy shaper with a shaping curve σ delays events of an input event stream such that the output event stream has σ as an upper arrival curve. Additionally, a greedy shaper ensures that no events get delayed any longer than necessary.

Greedy shapers can therefore be used to ensure that an event stream is upper-bounded by an upper arrival curve α^u. For this, the event stream R is input to a greedy shaper with shaping curve $\sigma \leq \alpha^u$. The output event stream R' of the greedy shaper is then upper-bounded by α^u.

To analyze the behavior of a greedy shaper, consider a greedy shaper with shaping curve σ which is subadditive and with $\sigma(0) = 0$. Assume that the shaper buffer is empty at time 0 and that it is large enough so that there is no event loss. The meaning of subadditivity, in this context and in relation to whether this property restricts the class of shapers that can be analyzed, will be discussed in Section 3.3.

Le Boudec and Thiran [2001] prove that for an event trace R to input such a greedy shaper, the output event trace R' can be computed as

$$R' = R \otimes \sigma. \tag{14}$$

In practice, a greedy shaper with a shaping curve of

$$\sigma(\Delta) = \left\lfloor \min_{\forall i} \{(b_i) + r_i \Delta\} \right\rfloor \tag{15}$$

ALGORITHM 1: Leaky bucket greedy shaper with bucket size b and filling rate r.

Given:
A clock $c \in \mathbb{R}_{\geq 0}$ that is continuously running and that can be reset to $c = 0$.

Initialize:
reset clock $c = 0$;
reset bucket fill level $f = b$;

while true **do**
  blocking read of event e from FIFO input buffer;
  $f = \min\{f + c \cdot r, b\}$;
  send event e;
  $f = f - 1$;
  reset $c = 0$;
  wait $\max\{0, \frac{1-f}{r}\}$;
end while

and with $\sigma(0) = 0$ can be implemented using a cascade of so-called leaky bucket greedy shapers. Every leaky bucket greedy shaper is a greedy shaper with a shaping curve of $\sigma_i(\Delta) = \lfloor (b_i) + r_i \Delta \rfloor$ and can be implemented using some sort of leaky bucket, with bucket size b_i, that is filled (instead of emptied) at a constant rate of r_i. If an event arrives at such a leaky bucket greedy shaper, it can pass the shaper immediately if the fill level of the bucket is larger than or equal to 1. Otherwise, the event gets delayed until the bucket is filled enough. Finally, every event that is sent on the output of the shaper reduces the bucket fill level by 1.

At such a leaky bucket stage, the first $\lfloor b_i \rfloor$ events of a burst can pass without any delay. But further events of the burst will be delayed and can only pass at a rate of r_i. If no events arrive for some time, the bucket will eventually be full again, allowing another burst of $\lfloor b_i \rfloor$ events to pass.

Algorithm 1 shows the pseudocode to implement a leaky bucket greedy shaper. Initially, the bucket is full, and the clock is reset to $c = 0$. Then, the shaper does a blocking read on its first in, first out input buffer. If an event is available on the first in, first out input buffer, it is immediately sent to the shaper output, but before sending the event, the new bucket fill level f is computed as the sum of the last-computed bucket fill level and the amount $c \cdot r$ by which the bucket got filled since the last clock reset, and hence, the last update of the fill level. The fill level is thereby limited by the bucket size b. Immediately after sending the event, the bucket fill level is reduced by 1, and the clock is reset to $c = 0$. If necessary, the shaper then waits until the bucket fill level is larger than or equal to 1 again before doing the next blocking read on its first in, first out input buffer.

Figure 10 depicts a shaping curve that can be implemented by a cascade of two leaky buckets. The first leaky bucket has a bucket size of $b_1 = 1$ and a leaking rate of $r_1 = 1/4$, while the second leaky bucket has a bucket size of $b_2 = 2.8$ and a leaking rate of $r_2 = 1/15$.

It is also possible to implement greedy shapers with more complex shaping curves than that of Equation (15). For this, the greedy shaper needs to consider the history of sent events and compare it against the shaping curve to determine the next point in time when an event can pass the shaper. The implementation of such a greedy shaper typically would be more complex than a simple cascade of leaky bucket greedy shapers, and therefore, in practice, one would approximate a complex shaping curve with a shaping curve as defined in Equation (15).

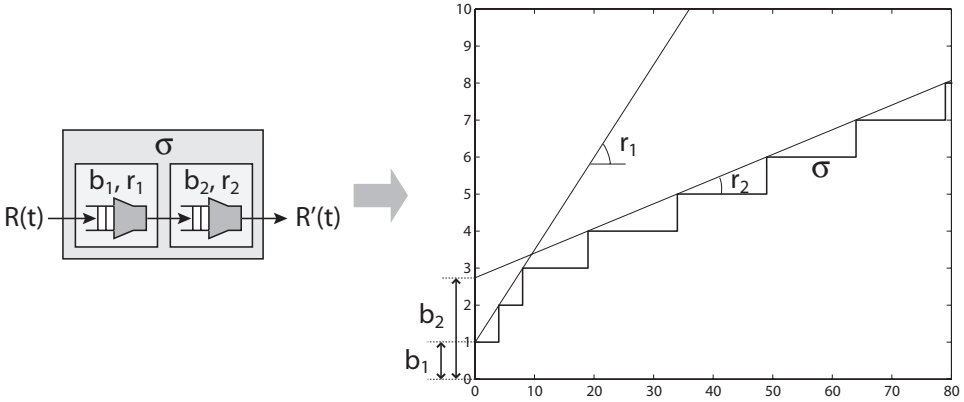

Fig. 10. A greedy shaper that is implemented by a cascade of two leaky bucket stages and the resulting total shaping curve.

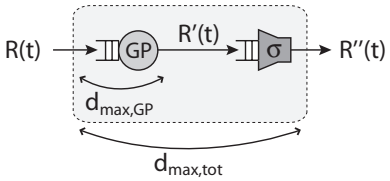

Fig. 11. Cascade of a processing component and a greedy shaper that reshapes the output event stream.

In practice, one of the simplest examples of a shaper is the play-out buffer that is read at a constant rate. Play-out buffers are widely used in multimedia processing, and can be implemented by a single leaky bucket greedy shaper.

3.2. Reshaping Comes For Free

An interesting property of greedy shapers is that reshaping does not increase delay or buffer requirements. That is to say, if an event stream $R(t)$ that is constrained by the upper arrival curve α^u is the input to a processing component, as depicted in Figure 11, and if a greedy shaper with shaping curve $\sigma \geq \alpha^u$ is used to reshape the output event stream, then the maximum delay experienced by any event is not increased by adding the greedy shaper, that is, $d_{max,\,tot} = d_{max,\,GP}$. Moreover, if the greedy shaper with shaping curve $\sigma \geq \alpha^u$ and the input buffer to the processing component access the same shared memory, then the total buffer requirement is also not increased by adding the greedy shaper, that is, $b_{max,\,tot} = b_{max,\,GP}$. These properties are sometimes referred to as "greedy shapers come for free" [Le Boudec and Thiran 2001].

3.3. Abstract Greedy Shapers

In order to embed greedy shapers into the Modular Performance Analysis framework, we need to determine the relation per Equation (13) that connects input and output arrival curves. This way, we can analyze distributed embedded systems that contain greedy shapers as components and determine worst-case bounds on end-to-end delays, throughput, and buffer sizes. Therefore, the challenges posed in Thiele et al. [2000] and Schioler et al. [2005] concerning the possibility of modeling traffic shapers in the framework of Modular Performance Analysis (MPA) are solved.

THEOREM 3.1 (ABSTRACT GREEDY SHAPERS). *Assume an event stream that is modeled as an abstract event stream with arrival curves* $[\alpha^u, \alpha^l]$ *serves as input to a greedy shaper with a subadditive shaping curve* σ *with* $\sigma(0) = 0$. *Then, the output of the greedy shaper is an event stream that can be modeled as an abstract event stream with arrival curves*

$$\alpha_{GS}^{u'} = \alpha^u \otimes \sigma, \tag{16}$$

$$\alpha_{GS}^{l'} = \alpha^l \otimes (\sigma \overline{\oslash} \sigma). \tag{17}$$

Further, the maximum delay and the maximum backlog at the greedy shaper are bounded by

$$d_{max,GS} = Del(\alpha^u, \sigma), \tag{18}$$

$$b_{max,GS} = Buf(\alpha^u, \sigma). \tag{19}$$

PROOF. To prove Equation (16), we use the fact that $R \oslash R$ is the minimum upper arrival curve of a cumulative function R, and we use the properties

$$(f \oslash g) \oslash h = f \oslash (g \otimes h),$$
$$(f \otimes g) \oslash g \leq f \otimes (g \oslash g),$$

proven in Le Boudec and Thiran [2001]. We can then compute

$$\begin{aligned} R' \oslash R' &= (R \otimes \sigma) \oslash (R \otimes \sigma) \\ &= ((R \otimes \sigma) \oslash R) \oslash \sigma \\ &= ((\sigma \otimes R) \oslash R) \oslash \sigma \\ &\leq (\sigma \otimes (R \oslash R)) \oslash \sigma \\ &\leq (\sigma \otimes \alpha^u) \oslash \sigma \\ &= (\alpha^u \otimes \sigma) \oslash \sigma \\ &= \alpha^u \otimes \sigma. \end{aligned}$$

To prove Equation (17), we use the fact that $R \overline{\oslash} R$ is the maximum lower arrival curve of a cumulative function R, and we use the property

$$(f \otimes g) \overline{\oslash} (h \otimes j) \geq (f \overline{\oslash} h) \otimes (g \overline{\oslash} j),$$

which we prove in the Appendix. We can then compute

$$\begin{aligned} R' \overline{\oslash} R' &= (R \otimes \sigma) \overline{\oslash} (R \otimes \sigma) \\ &\geq (R \overline{\oslash} R) \otimes (\sigma \overline{\oslash} \sigma) \\ &\geq \alpha^l \otimes (\sigma \overline{\oslash} \sigma). \end{aligned}$$

To prove Equation (18), we compute

$$\begin{aligned} d(t) &= \inf\{\tau \geq 0 : R(t) \leq R'(t+\tau)\} \\ &= \inf\{\tau \geq 0 : 0 \leq \inf_{0 \leq u \leq t+\tau} \{\sigma(t+\tau-u) + R(u) - R(t)\}\} \\ &\leq \inf\{\tau \geq 0 : 0 \leq \inf_{0 \leq u \leq t} \{\sigma(t-u+\tau) - \alpha^u(t-u)\}\} \\ &\leq \inf\{\tau \geq 0 : 0 \leq \inf_{0 \leq v} \{\sigma(v+\tau) - \alpha^u(v)\}\} \\ &= \inf\{\tau \geq 0 : \sup_{0 \leq v} \{\alpha^u(v) - \sigma(v+\tau)\} \leq 0\} \\ &= \sup_{0 \leq \Delta} \{\inf\{\tau \geq 0 : \alpha^u(\Delta) \leq \sigma(\Delta+\tau)\}\}. \end{aligned}$$

To prove Equation (19), we compute

$$\begin{aligned} b(t) &= R(t) - R'(t) = R(t) - (\sigma \otimes R)(t) \\ &= R(t) - \inf_{0 \le u \le t} \{\sigma(t-u) + R(u)\} \\ &= R(t) + \sup_{0 \le u \le t} \{-\sigma(t-u) - R(u)\} \\ &= \sup_{0 \le u \le t} \{R(t) - R(u) - \sigma(t-u)\} \\ &\le \sup_{0 \le u \le t} \{\alpha^u(t-u) - \sigma(t-u)\} \\ &= \sup_{0 \le v \le t} \{\alpha^u(v) - \sigma(v)\} \\ &\le \sup_{0 \le \Delta} \{\alpha^u(\Delta) - \sigma(\Delta)\}. \end{aligned}$$

□

Theorem 3.1 supposes that the shaping curve σ is subadditive. As defined in the Appendix, we have $\sigma(a)+\sigma(b) \ge \sigma(a+b)$ for all $a, b \ge 0$, in this case. In order to understand whether this is a restriction, let us consider a time interval of length $a + b$. If σ is not subadditive, then we could find some partition of this interval into two subintervals of length a and b such that $\sigma(a + b) > \sigma(a) + \sigma(b)$. Remember that $\sigma(\Delta)$ bounds the number of output events in any time interval of length Δ. Therefore, in any interval of size $a + b$, we can not have more events than $\sigma(a) + \sigma(b)$. As a result of this discussion, we conclude that $\sigma(a + b)$ can be lowered without changing the behavior of the shaper. In other words, the specification of the shaping curve σ was not tight, that is, it could be lowered without any change of the shaper behavior. This opens the question of how we can determine a "tight" subadditive shaping curve from a given one which is not subadditive. According to the Appendix, the tightest subadditive shaping curve is given by the subadditive closure $\overline{\sigma}$. If the given shaping curve σ is not subadditive, then it should be replaced by $\overline{\sigma}$ before applying Theorem 3.1.

Equations (16) and (17) can now be used as internal relations of an abstract greedy shaper, and Equations (18) and (19) can be used to analyze delay guarantees and buffer requirements of greedy shapers in a performance model.

4. APPLICATIONS AND CASE STUDIES

In this section, we analyze the two system designs depicted in Figure 1. The analysis results will clearly reveal the positive influence of greedy shapers on a system's performance and buffer requirements, when applied internally or on a system's robustness, when applied externally. We deliberately chose two small system designs that clearly focus on the influence of the greedy shapers and does not dilute the analysis results by any possibly hard recognizable influences of other system properties. Modular Performance Analysis with Real-Time Calculus have already been used several times to analyze bigger and more complex system designs [Chakraborty et al. 2003; Wandeler et al. 2006], and the abstract greedy shapers can seamlessly be integrated into bigger performance models.

4.1. Internal Shaping for System Improvement

Consider a distributed real-time system with two CPU's that communicate via a shared bus, as depicted on the left side in Figure 1. CPU_1 and CPU_2 both process an incoming event stream S_1 and S_2, respectively, and send the resulting event streams S_1'' and S_2'' via the shared bus to other components. The shared bus implements a fixed-priority protocol in which sending the events from CPU_1 has priority over sending the events from CPU_2. Events that are ready to be sent get buffered in the communication network

interfaces CNI_1 and CNI_2 that connect CPU_1 and CPU_2, respectively, with the shared bus.

In this system, S_1'' may differ considerably from S_1. For example, S_1'' may be bursty even when S_1 is a strictly periodic event stream. This may happen, for example, if other tasks, besides T_{S1}, are executed on CPU_1 using a TDMA scheduling policy, or if FP scheduling is used and T_{S1} has a low priority. In both cases, the processor may not be available to T_{S1} during some time interval in which all arriving events of S_1 get buffered, and it may be fully available to T_{S1} during a later time interval in which all the buffered events will be processed and emitted, leading to a burst on S_1''.

Now, suppose that event stream S_1'' is bursty. Whenever a burst of events arrives on S_1'', the shared bus gets fully occupied until all buffered events of S_1'' are sent. During this period, event stream S_2'' will receive no service, and S_2'' will experience a delay caused by the burstiness of S_1''. Moreover, the buffer demand in CNI_2 will increase with the increasing burstiness of S_1''.

In this system, it may be an interesting option to place a greedy shaper at the output of CPU_1 that shapes event stream S_1'. This greedy shaper will limit the burstiness of S_1'', and will, therefore, reduce the influence of CPU_1 and S_1 to the delay of S_2' and to the buffer requirements of CNI_2.

To investigate the effect of adding greedy shapers to the system with internal reshaping in Figure 1, we analyze it with Modular Performance Analysis, using the abstract greedy shaper component that we introduced in Section 3.

We assume that S_1 and S_2 are both strictly periodic with a period $p = 1ms$. Further, we model both CPU's as bounded delay resources. The CPU may not be available to process the tasks T_{Si} for up to $5ms$, but after this period of at most $5ms$, the processor is fully available and can process five events per ms ($\beta^u_{CPU1} = \beta^u_{CPU2} = 5\Delta[e/ms]$, $\beta^l_{CPU1} = \beta^l_{CPU2} = \max\{0, \Delta - 5\}[e/ms]$), and the bus can send 2.5 events per ms ($\beta^u_{BUS} = \beta^l_{BUS} = 2.5\Delta[e/ms]$).

With this specification, we analyze the four different system designs that are depicted in Figure 12. First, we analyze the system without greedy shapers (Figure 12(a)); secondly, we place a greedy shaper only at the output of CPU_1 to shape S_1' (Figure 12(b)); then, we place a greedy shaper only at the output of CPU_2 to shape S_2' (Figure 12(c)); and finally, we add two greedy shapers to shape both S_1', as well as S_2' (Figure 12(d)). We use the upper arrival curves α^u_{S1} and α^u_{S2} as shaping curves σ_1 and σ_2, respectively, and we assume that the buffers of the greedy shapers and the corresponding processing tasks access the same memory.

Using the four performance models, we analyzed the maximum required buffer spaces of the different buffers, as well as the end-to-end delays of both event streams S_1 and S_2. The results, computed with the Real-Time Calculus (RTC) Toolbox for Matlab [Wandeler and Thiele], are shown in Table I.

From the results, we learn that placing greedy shapers helps to reduce the total buffer requirements from 25 down to at most 14 events that need to be buffered. Moreover, the greedy buffers also reduce the end-to-end delay of both event streams, namely by 7.4% for S_1, and by a total of 40% for S_2.

When we look at the results, we also recognize the well-known property of greedy shapers that reshaping is for free. Since we use $\sigma_1 = \alpha^u_{S1}$ and $\sigma_2 = \alpha^u_{S2}$, the greedy shapers effectively only reshape S_1^1 and S_2^2, and therefore, the buffer requirements of CPU_1 and CPU_2 are not affected by adding the greedy shapers.

4.2. Input Shaping for Separation of Concerns

Typical large embedded systems often process several event streams in parallel. To achieve separation of concerns in such systems, they are often implemented using

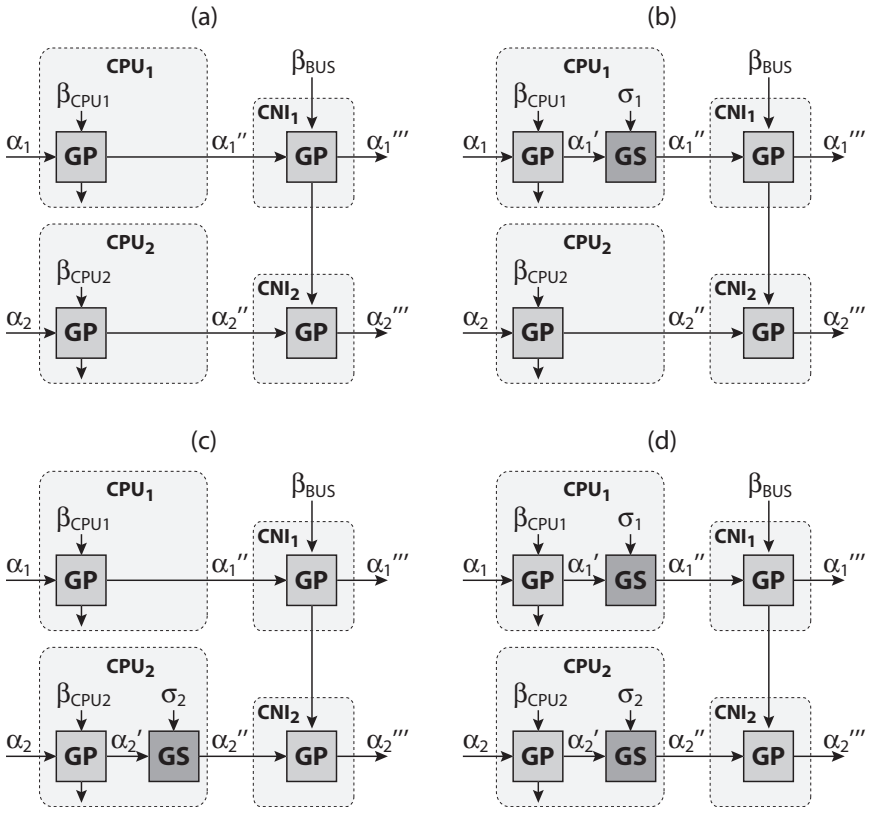

Fig. 12. Performance models for four system architecture scenarios with internal reshaping.

Table I. Effect of ReShaping

Shapers	Buffer					Delay	
	CPU_1	CPU_2	CNI_1	CNI_2	Tot	$S1$	$S2$
none	6	6	4	9	25	5.4	9
$S1$	6	6	1	6	19	5	5.8
$\Delta\%$	—	—	−75%	−33%	−24%	−7.4%	−36%
$S2$	6	6	4	4	20	5.4	8.6
$\Delta\%$	—	—	—	−56%	−20%	—	−4.4%
both	6	6	1	1	14	5	5.4
$\Delta\%$	—	—	−75%	−89%	−44%	−7.4%	−40%

time-triggered scheduling policies, or servers. While these scheduling policies help to decouple the influence of the various event streams to each other, they often do not use the available resources efficiently.

On the other hand, powerful methods were developed to analyze systems with event-triggered scheduling policies, such as RM or EDF. In these systems, resources are used efficiently, but on the downside, the various event streams may heavily influence each other. Slight changes in the timing behavior of a high-priority stream may increase the total delay of a lower-priority stream considerably, possibly leading to a missed deadline or to buffer overflows somewhere in the system.

To overcome this problem, greedy shapers may be placed at the input to such systems. Every incoming event stream S_i gets shaped with an individual shaping curve σ_i that

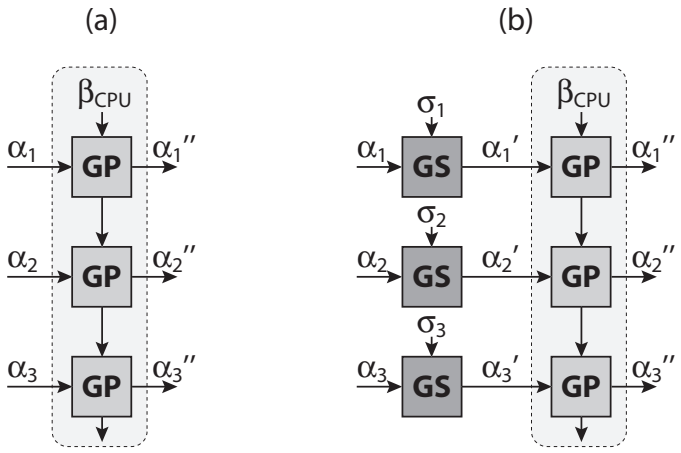

Fig. 13. Performance models with external input shaping.

Table II. Effect of Input Shaping

	Without Shaping			With Shaping		
	d_1	d_2	d_3	d_1	d_2	d_3
$j_1 = 0$	2.86	8.57	20	2.86	8.57	20
$j_1 = 0.1$	2.86	8.57	28.57	2.96	8.57	20
$\Delta\%$	0	0	+43%	+3.5%	0	0

corresponds to its design-time timing specification. The system can then be analyzed using the design-time timing specifications, and at runtime, non-adherence of S_i to its timing specification will have no influence on the delay of any other event streams but will, at most, increase the total delay of S_i itself. Moreover, no buffers will overflow inside the system. Instead, only the buffers of the greedy shapers themselves may overflow. But since these buffers are clearly localized at the boundary of the system, individual handling policies could easily be implemented.

Let's assume a real-time system, as shown on the right side of Figure 1. Here, a single CPU processes three event streams with a fixed-priority scheduling policy. The high-priority stream S_1 is strictly periodic with $p_1 = 5ms$; the medium-priority stream S_2 is strictly periodic with $p_2 = 10ms$; and the low-priority stream S_3 is strictly periodic with $p_3 = 20ms$. The CPU processes 0.35 events per ms.

To illustrate the influence of greedy shapers at the input of such a system, we add a jitter of $j_1 = 0.1ms$ to stream S_1. We then analyze the effect of this to the end-to-end delays of the three event streams, both without (as in Figure 13(a)) and with (as in Figure 13(b)) greedy shapers. The results, computed using Modular Performance Analysis, are shown in Table II.

Looking at the results, we clearly see the big influence of the little non-adherence of S_1 on the maximum delay of the completely independent stream S_3, if no input shaping is applied. On the other hand, we observe that input shaping effectively isolates the influence of the malicious input stream S_1 to the other present event streams. Now, only S_1 is affected by its own malbehavior.

4.2.1. Input Shaping for NRT Event Streams in Real-Time Systems. Inspired by the application of greedy shapers for input shaping for separation of concerns, one could also use greedy shapers to shape non-real-time event streams, as depicted in Figure 14. Here, the non-real-time event streams, are processed with the highest priority, guaranteeing a good reactivity and typically short delays, and the greedy shaper guarantees that

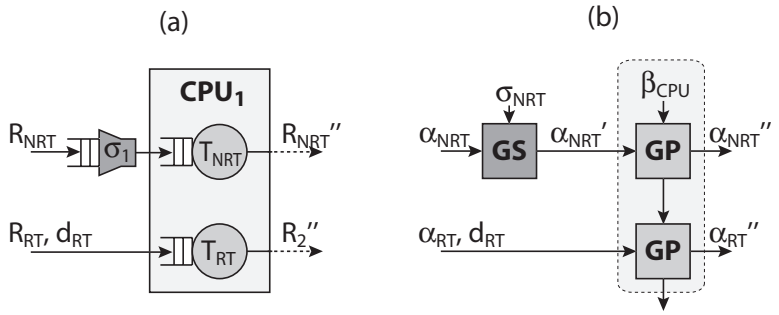

Fig. 14. System model (a) and performance model (b) with input shaping for NRTtraffic.

the load created by the non-real-time events is limited, such that the real-time event streams remain schedulable.

In such an application, the greedy shaper is an alternative to typical server implementations, such as the periodic [Sha et al. 1986] or the deferrable server [Strosnider et al. 1995], and in a certain respect, the greedy shaper also behaves like a server. The advantage of the greedy shaper, however, is its flexibility through the parameterizable shaping curve σ_{NRT}. With an appropriate shaping curve, a shaper can, for example, guarantee a reactive periodic service, such as the deferrable server, but additionally, it may also allow a burst of service from time to time. Moreover, a greedy shaper is typically easy to implement.

The main question that then arises in an application, as depicted Figure 14, is how to dimension the greedy shaper. How should σ_{NRT} be chosen such that the maximum possible service to the non-real-time events is provided without jeopardizing the schedulability of the real-time event streams?

In Wandeler and Thiele [2005], the theory of Real-Time Interfaces was first introduced, providing methods for finding the maximum allowable shaping curve σ_{NRT}. With Real-Time Interfaces, the maximum allowable non-real-time input load α'_{NRT} to the system in Figure 14 can be computed as

$$\alpha^u_{NRT,\,max} = RT^{-\alpha}\big(\alpha^u_{RT}(\Delta - d_{RT}),\, \beta^l_{CPU}\big), \tag{20}$$

with

$$RT^{-\alpha}(\beta',\,\beta)(\Delta) = \beta(\Delta+\lambda) - \beta'(\Delta+\lambda) \quad \text{for} \quad \lambda = \sup\{\tau \;:\; \beta'(\Delta+\tau) = \beta'(\Delta)\}, \tag{21}$$

where α^u_{RT} models the maximum real-time input load, d_{RT} denotes the relative deadline of the real-time events, and β^l_{CPU} models the minimum available service from the CPU.

The greedy shaper must then only enforce that the load on its output is limited by $\alpha^u_{NRT,\,max}$, which is easily achieved by setting the shaping curve equal to this upper bound. Without any loss, the shaping curve can even be ensured to be subadditive by setting it equal to the subadditive closure of the following upper bound (see Appendix).

$$\sigma_{NRT} = \overline{\alpha}^u_{NRT,\,max}. \tag{22}$$

The methods presented in Wandeler and Thiele [2005] also allow for the computation of σ_{NRT} for systems with more than one real-time event stream and for even more complex systems with mixed and hierarchical scheduling.

As an example, consider a system similar to the one depicted in Figure 14 but with three real-time event streams with decreasing priorities: $R_{RT,1}$ has a period of $100ms$, a jitter of $40ms$, and an execution demand of $25ms$; $R_{RT,2}$ has a period of $200ms$, a jitter

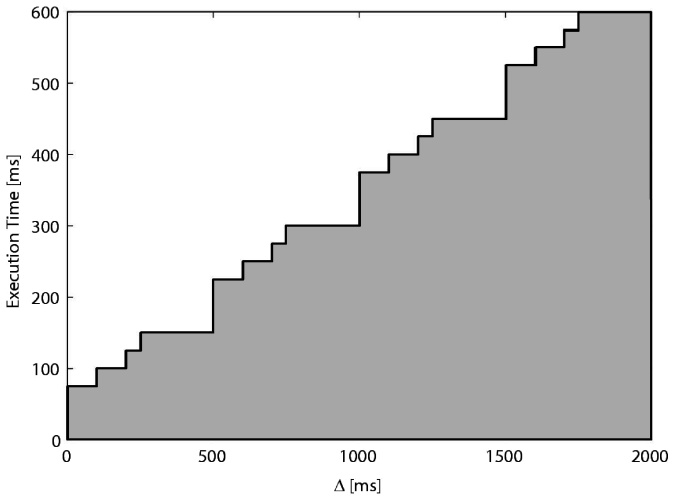

Fig. 15. Input shaping curve σ_{NRT} for the NRTtraffic in the example system.

of 150ms, and also an execution demand of 25ms; and $R_{RT,3}$ has a period of 500ms and an execution demand of 100ms. The relative deadlines of all real-time event streams equal their periods. To compute the shaping curve σ_{NRT} for the non-real-time event streams in this system, we use the methods presented in Wandeler and Thiele [2005] and Equation (22).

From the result depicted in Figure 15, we learn that within this system, any non-real-time event streams that are upper-bounded by σ_{NRT} will be served immediately with the highest priority, that is, they will not be delayed by the input shaper. This is the case if the input arrival curve $\alpha^u_{NRT,\,max}$ is completely within the grey shaded area. If there are events that contravene this upper bound, they will be delayed by the input shaper to ensure schedulability of the real-time streams. This is the case if the input arrival curve $\alpha^u_{NRT,\,max}$ is tight and reaches the white area. The corresponding delay can be computed using Equation (18).

5. CONCLUSIONS

We introduced a new method for analyzing greedy shapers and we embedded this method into the Modular Performance Analysis framework of Chakraborty et al. [2003] and Thiele et al. [2000], by introducing a new abstract component that models a greedy shaper. This approach enables system-level performance analysis of real-time systems with greedy shapers. We proved the applicability of the presented methods through performance analysis of two case study systems with greedy shapers. In these case study systems, we analyzed the detailed buffer requirements of all system components and provided end-to-end delay guarantees for the processed event streams. The analysis thereby clearly revealed the positive influence of greedy shapers on the system's performance and buffer requirements.

APPENDIX

Min-Max Algebra

The operators $\otimes$, $\oslash$, and $\overline{\oslash}$ are defined as

$$(f \otimes g)(\Delta) = \inf_{0 \le \lambda \le \Delta} \{f(\Delta - \lambda) + g(\lambda)\}, \tag{23}$$

$$(f \oslash g)(\Delta) = \sup_{\lambda \geq 0}\{f(\Delta+\lambda) - g(\lambda)\}, \tag{24}$$

$$(f \overline{\oslash} g)(\Delta) = \inf_{\lambda \geq 0}\{f(\Delta+\lambda) - g(\lambda)\}. \tag{25}$$

A curve f is subadditive if

$$f(a) + f(b) \geq f(a+b) \quad \forall a, b \geq 0. \tag{26}$$

The subadditive closure $\overline{f}$ of a curve f is the largest subadditive curve with $\overline{f} \leq f$ and is computed as

$$\overline{f} = \min\{f, (f \otimes f), (f \otimes f \otimes f), \ldots\}. \tag{27}$$

If f is interpreted as an arrival curve, then any trace R that is upper-bounded by f is also upper-bounded by the subadditive closure $\overline{f}$.

LEMMA 5.1. *Given nonnegative, monotonic increasing functions* $f, g, h, j : \mathbb{R} \mapsto \mathbb{R}$ *with* $f(t) = g(t) = h(t) = j(t) = 0$ *for all* $t \leq 0$. *Then*

$$(f \otimes g)\overline{\oslash}(h \otimes j) \geq (f \overline{\oslash} h) \otimes (g \overline{\oslash} j).$$

PROOF. We have, by definition of the operators,

$$(f \otimes g)\overline{\oslash}(h \otimes j) = \inf_{\lambda \geq 0} \sup_{0 \leq b \leq \lambda} \inf_{0 \leq a \leq \Delta+\lambda} [f(a) - h(b) + g(\Delta+\lambda-a) - j(\lambda-b)].$$

We will consider three cases depending on the value of a.

—$b \leq a \leq b + \Delta$:

$$\begin{aligned}
&\inf_{\lambda \geq 0} \sup_{0 \leq b \leq \lambda} \inf_{b \leq a \leq b+\Delta} [f(a) - h(b) + g(\Delta+\lambda-a) - j(\lambda-b)] \\
=&\inf_{\lambda \geq 0} \sup_{0 \leq b \leq \lambda} \inf_{0 \leq a' \leq \Delta} [f(a'+b) - h(b) + g(\Delta+\lambda-a'-b) - j(\lambda-b)] \\
\geq&\inf_{\lambda \geq 0} \inf_{0 \leq b \leq \lambda} \inf_{0 \leq a' \leq \Delta} [f(a'+b) - h(b) + g(\Delta+\lambda-a') - j(\lambda)] \\
\geq&\inf_{\lambda \geq 0} \inf_{0 \leq b} \inf_{0 \leq a' \leq \Delta} [f(a'+b) - h(b) + g(\Delta+\lambda-a') - j(\lambda)] \\
=&\inf_{0 \leq a' \leq \Delta} [\inf_{0 \leq b}(f(a'+b) - h(b)) + \inf_{\lambda \geq 0}(g(\Delta+\lambda-a') - j(\lambda))] \\
=&(f \overline{\oslash} h) \otimes (g \overline{\oslash} j).
\end{aligned}$$

Note that we used variable substitutions and the relation

$$\sup_a\{u(a) + v(a)\} \geq u(a_0) + \inf_a\{v(a)\}.$$

—$b + \Delta \le a \le \Delta + \lambda$:

$$\begin{aligned}
&\inf_{\lambda\ge 0}\sup_{0\le b\le\lambda}\inf_{b+\Delta\le a\le\Delta+\lambda}[f(a)-h(b)+g(\Delta+\lambda-a)-j(\lambda-b)]\\
=&\inf_{\lambda\ge 0}\sup_{0\le b\le\lambda}\inf_{0\le a'\le\lambda-b}[f(a'+b+\Delta)-h(b)+g(\lambda-b-a')-j(\lambda-b)]\\
=&\inf_{\lambda\ge 0}\sup_{0\le b'\le\lambda}\inf_{0\le a'\le b'}[f(a'+\Delta+\lambda-b')-h(\lambda-b')+g(b'-a')-j(b')]\\
\ge&\inf_{\lambda\ge 0}\inf_{0\le b'\le\lambda}\inf_{0\le a'\le b'}[f(a'+\Delta+\lambda-b')-h(\lambda-b')]\\
\ge&\inf_{0\le b}\inf_{0\le a'}[f(a'+\Delta+b)-h(b)]\\
=&\inf_{0\le b}[f(\Delta+b)-h(b)]\\
=&f\overline{\oslash}h.
\end{aligned}$$

—$0 \le a \le b$:

$$\begin{aligned}
&\inf_{\lambda\ge 0}\sup_{0\le b\le\lambda}\inf_{0\le a\le b}[f(a)-h(b)+g(\Delta+\lambda-a)-j(\lambda-b)]\\
=&\inf_{\lambda\ge 0}\sup_{0\le b\le\lambda}\inf_{0\le a'\le b}[f(b-a')-h(b)+g(\Delta+a'+\lambda-b)-j(\lambda-b)]\\
\ge&\inf_{\lambda\ge 0}\inf_{0\le b'\le\lambda}\inf_{0\le a'\le b}[g(\Delta+a'+\lambda-b)-j(\lambda-b)]\\
\ge&\inf_{0\le\lambda}\inf_{0\le a'}[g(\Delta+a'+\lambda)-j(\lambda)]\\
=&\inf_{0\le\lambda}[g(\Delta+\lambda)-j(\lambda)]\\
=&g\overline{\oslash}j.
\end{aligned}$$

Now we have

$$\begin{aligned}
(f\otimes g)\overline{\oslash}(h\otimes j)&\ge\min\{(f\overline{\oslash}h)\otimes(g\overline{\oslash}j),\ f\overline{\oslash}h, g\overline{\oslash}j\}\\
&=(f\overline{\oslash}h)\otimes(g\overline{\oslash}j).
\end{aligned}$$

□

REFERENCES

Chakraborty, S., Künzli, S., and Thiele, L. 2003. A general framework for analyzing system properties in platform-based embedded system designs. In *Proceedings of the 6th Design, Automation and Test in Europe (DATE)*. 190–195.

Chakraborty, S., Künzli, S., Thiele, L., Herkersdorf, A., and Sagmeister, P. 2003. Performance evaluation of network processor architectures: Combining simulation with analytical estimation. *Comput. Netw. 41,* 5, 641–665.

Cruz, R. 1991. A calculus for network delay. *IEEE Trans. Inform. Theory 37,* 1, 114–141.

González Harbour, M., Gutiérrez García, J., Palencia Gutiérrez, J., and Drake Moyano, J. 2001. MAST: Modeling and analysis suite for real time applications. In *Proceedings of the 13th Euromicro Conference on Real-Time Systems*. IEEE Computer Society, 125–134.

Gringeri, S., Shuaib, K., Egorov, R., Lewis, A., Khasnabish, B., and Basch, B. 1998. Traffic shaping, bandwidth allocation, and quality assessment for mpeg video distribution over broadband networks. *IEEE Networks 12,* 6, 94–107.

Le Boudec, J. and Thiran, P. 2001. *Network Calculus—A Theory of Deterministic Queuing Systems for the Internet*. Lecture Notes in Computer Science, vol. 2050, Springer-Verlag.

Pop, P., Eles, P., and Peng, Z. 2003. Schedulability analysis and optimization for the synthesis of multicluster distributed embedded systems. In *Proceedings of the 6th Design, Automation and Test in Europe (DATE'03)*. 184–189.

Rexford, J., Bonomi, F., Greenberg, A., and Wong, A. 1997. Scalable architectures for integrated traffic shaping and link scheduling in high-speed ATM switches. *IEEE J. Select. Areas Comm. 15,* 5, 938–950.

Richter, K., Jersak, M., and Ernst, R. 2003. A formal approach to mpsoc performance verification. *IEEE Comput. 36,* 4, 60–67.

Schioler, H., Jessen, J., Nielsen, J. D., and Larsen, K. G. 2005. CyNC—towards a general tool for performance analysis of complex distributed real-time systems. In *Proceedings of the WiP Session of the 17th EUROMICRO Conference on Real-Time Systems (ECRTS)*. IEEE, 61–64.

Sha, L., Lehoczky, J. P., and Rajkumar, R. 1986. Solutions for some practical problems in prioritized preemptive scheduling. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 181–191.

Shin, I. and Lee, I. 2003. Periodic resource model for compositional real-time guarantees. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2–13.

Shin, I. and Lee, I. 2004. Compositional real-time scheduling framework. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 57–67.

Strosnider, J. K., Lehoczky, J. P., and Sha, L. 1995. The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments. *IEEE Trans. Comput. 44,* 1, 73–91.

Thiele, L., Chakraborty, S., and Naedele, M. 2000. Real-time calculus for scheduling hard real-time systems. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*. Vol. 4. 101–104.

Wandeler, E., Maxiaguine, A., and Thiele, L. 2006. Performance analysis of greedy shapers in real-time systems. In *Proceedings of the Design, Automation and Test in Europe (DATE)*. 444–449.

Wandeler, E. and Thiele, L. Real-time calculus (RTC) toolbox. `http://www.mpa.ethz.ch/Rtctoolbox`.

Wandeler, E. and Thiele, L. 2005. Real-time interfaces for interface-based design of real-time systems with fixed priority scheduling. In *Proceedings of the 5th ACM Conference on Embedded Software (EMSOFT)*. 80–89.

Wandeler, E. and Thiele, L. 2006. Interface-based design of real-time systems with hierarchical scheduling. In *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 243–252.

Wandeler, E., Thiele, L., Verhoef, M., and Lieverse, P. 2006. System architecture evaluation using modular performance analysis—a case study. *Softw. Tools Technol. Transfer 8,* 6, 649–667.

Received January 2007; revised May 2007; accepted June 2007