Performance Analysis of Greedy Shapers in Real-Time Systems

Ernesto Wandeler Alexander Maxiaguine Lothar Thiele
Computer Engineering and Networks Laboratory
Swiss Federal Institute of Technology (ETH)
8092 Zürich, Switzerland
{wandeler,maxiagui,thiele}@tik.ee.ethz.ch

Abstract— 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 multi-processor embedded systems that exhibit a considerable amount of on-chip traffic. Despite their growing importance in this area, no methods exist to analyze shapers in distributed embedded systems, and to incorporate them into a system-level performance analysis. Hence it is until now not possible to determine the effect of shapers to end-to-end delay guarantees or buffer requirements in these systems. In this work, we present a method to analyze greedy shapers, and we embed this analysis method into a well-established modular performance analysis framework. The presented approach enables system-level performance analysis of complete systems with greedy shapers, and we prove its applicability by analyzing two case study systems.

1. Introduction

In the area of broad-band networking, traffic shaping is a well-known and well-studied technique to regulate connections and to avoid buffer overflow in network nodes, see e.g. [3] or [6]. 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 inter-arrival time. A greedy shaper is a special instance of a traffic shaper, that not only ensures an output stream stream that conforms to a given traffic specification, but that also guarantees that no packets get delayed any longer than necessary.

By limiting the burstiness of the output stream of a network node, shapers typically drastically reduce the buffer requirements on subsequent network nodes. And 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 lower-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. Particularly, since modern embedded systems are often implemented as multi-processor 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 re-shape internal traffic streams to reduce global buffer requirements and end-to-end delays, and secondly, shapers may be added at the boundaries of a system, to ensure conformant input streams and to thereby prevent internal buffer overflows caused by malicious input. Figure 1 shows two simple example systems from these two application areas.

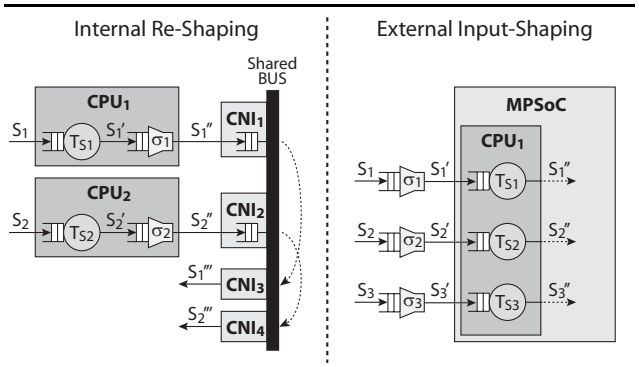

Figure 1. Two systems with greedy shapers.

The analysis of traffic shapers in communication networks is well-known [4]. But to our best knowledge, none of the existing frameworks for modular system level per-

formance analysis of real-time embedded system considers traffic shapers at this time, see e.g. [5], [7] or [1, 9].

Only [7] introduces a restricted kind of traffic shaping through so-called event adaption functions (EAF's). But EAF's play a crucial role in the fundamental ability of [7] to analyze systems, and a designer has therefore a very limited freedom to place or leave away, or to parameterize EAF's.

In this work, we will extend the framework presented in [1, 9], to enable system level performance analysis of real-time embedded systems with traffic shapers. It has to be noted here, that in [4], Le Boudec and Thiran challenge the ability of the methods in [9] to analyze traffic shapers, and in [8], Schiøler *et al.* even claim that it is not possible to analyze traffic shapers within the framework of [1, 9].

Contributions of this work:

- We present a method to analyze greedy shapers in the area of multi-processor embedded systems.
- We embed this new analysis method into the well established modular performance analysis framework of [1, 9]. This enables system-level performance analysis of complete systems with greedy shapers, i.e. amongst others, 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 [4] provides means to deterministically reason about timing properties of data flows in queuing networks. Real-Time Calculus [9] extends the basic concepts of Network Calculus to the domain of real-time embedded systems, and in [1] a unifying approach to Modular Performance Analysis with Real-Time Calculus has been proposed. It is based on a general event and resource model, allows for hierarchical scheduling and arbitration, and can take computation and communication resources into account. Following, we introduce some concepts of Network and Real-Time Calculus.

2.1. A General Event Stream Model

A trace of an event stream can conveniently be described by means of a cumulative function $R(t)$, defined as the number of events seen on the event stream in the time interval $[0, t]$. 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* [2] provides an abstract event stream model, representing *all* possible traces of an event stream.

$\alpha^u(\Delta)$ provides an upper bound on the number of events seen on the event stream in *any* time interval of length Δ, and analogously, $\alpha^l(\Delta)$ 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 as follows:

$$\alpha^l(t-s) \leq R(t) - R(s) \leq \alpha^u(t-s) \quad \forall s < t \qquad (1)$$

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

Arrival curves substantially generalize traditional event models such 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:

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

2.2. A General Resource Model

Analogously to the cumulative function $R(t)$, the concrete availability of a computation or communication resource can be described by a cumulative function $C(t)$, defined as the number of available resources, e.g. processor or bus cycles, in the time interval $[0, t]$. 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*. Then, C, β^u and β^l are related to each other as follows:

$$\beta^l(t-s) \leq C(t) - C(s) \leq \beta^u(t-s) \quad \forall s < t \qquad (3)$$

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

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.

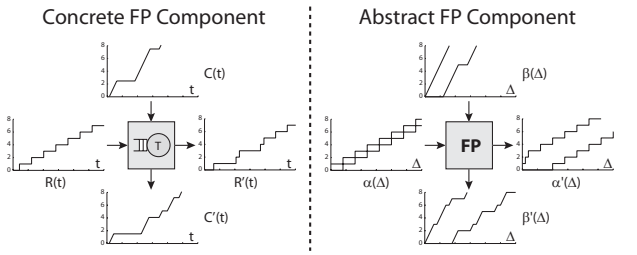

Figure 2. A component and its abstraction.

Figure 2 shows on the left side such a component. An event stream, described by $R(t)$, enters the component and

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

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

For modular performance analysis with real-time calculus, we model such a HW/SW component as an abstract component as shown on the right side of Fig. 2. Here, an abstract event stream $\alpha(\Delta)$ enters the abstract component and is processed using an abstract hardware resource $\beta(\Delta)$. The output is then again an abstract event stream $\alpha'(\Delta)$, 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:

$$\alpha' = f_\alpha(\alpha, \beta) \qquad \beta' = f_\beta(\alpha, \beta)$$

Again, these relations depend on the processing semantics of the modeled component, and must be determined such that $\alpha'(\Delta)$ and correctly models the event stream with event trace $R'(t)$ and that $\beta'(\Delta)$ correctly models the resource availability $C'(t)$.

As an example, consider a component modeling a task that greedily uses the resources offered to it. This component can be described by the relations f_α as follows[1] [1]:

$$\alpha'^{u}_{FP} = \min\{(\alpha^u \otimes \beta^u) \oslash \beta^l, \beta^u\} \quad (4)$$

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

Such a component is very common in the area of real-time embedded systems, and we will refer to it as a *Fixed Priority* (FP) component.

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 inter-connected abstract components. For this, first all concrete system components are modeled using their abstract representation (as described in the preceding section). And then, the arrival-curve inputs and outputs of these abstract components are inter-connected to reflect the flow event streams through the system.

1 See the Appendix for a definition of $\otimes$ and $\oslash$

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.

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 or non-preemptive scheduling, different abstract components, with tailored internal relations, must be used.

2.5. Analysis

In the performance model of a system, various performance measures can be computed analytically.

For instance, for an FP component the maximum delay d_{max} experienced by an event is bounded by [4, 1]:

$$\begin{aligned} d_{max} &\leq \sup_{\lambda \geq 0} \left\{ \inf\{\tau \geq 0 : \alpha^u(\lambda) \leq \beta^l(\lambda + \tau)\} \right\} \\ &\stackrel{def}{=} Del(\alpha^u, \beta^l) \end{aligned} \quad (6)$$

and when processed by a sequence of components, the total end-to-end delay experienced by an event is bounded by [4]:

$$d_{max} \leq Del(\alpha^u, \beta^l_1 \otimes \beta^l_2 \otimes \ldots \otimes \beta^l_n) \quad (7)$$

Similarly, the maximum buffer space b_{max} required to buffer an event stream in front of such an FP component is bounded by:

$$b_{max} \leq \sup_{\lambda \geq 0}\{\alpha^u(\lambda) - \beta^l(\lambda)\} \stackrel{def}{=} Buf(\alpha^u, \beta^l) \quad (8)$$

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

$$b_{max} \leq Buf(\alpha^u, \beta^l_1 \otimes \beta^l_2 \otimes \ldots \otimes \beta^l_n) \quad (9)$$

3. Performance Analysis of Greedy Shapers

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 Fig. 3. We will first explain the behavior and the implementation of concrete greedy shapers, and will then introduce the internal relations for abstract greedy shapers.

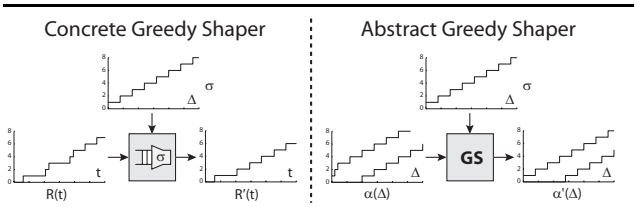

Figure 3. A greedy shaper and its abstraction.

3.1. Concrete Greedy Shapers

A *greedy shaper* with a shaping curve σ delays events of an input event stream, so that the output event stream has σ as an upper arrival curve, and it outputs all events as soon as possible.

Consider a greedy shaper with shaping curve σ, which is sub-additive 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. In [4], Le Boudec and Thiran proved that for an input event trace R to such a greedy shaper, the output event trace R' is given by:

$$R' = R \otimes \sigma \tag{10}$$

In practice, a greedy shaper with a shaping curve $\sigma(\Delta) = \min_{\forall i}\{b_i + r_i\Delta\}$ with $\sigma(0) = 0$ can be implemented using a cascade of leaky buckets. Every leaky bucket has a bucket size b_i and a leaking rate r_i, and the leaky buckets are arranged with decreasing leaking rate within the cascade. Initially all buckets are empty. When an event arrives at a leaky bucket stage, a token is generated. If there is enough space in the bucket, the token is put into the bucket and the event is sent to the next stage immediately. Otherwise, the event is buffered until the bucket emptied enough to put the token in.

3.2. Abstract Greedy Shapers

Theorem 1 (Abstract Greedy Shapers) *Assume an event stream that can be modeled as an abstract event stream with arrival curves $[\alpha^u, \alpha^l]$ serves as input to a greedy shaper with a sub-additive 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*

$$\alpha^{u'}_{GS} = \alpha^u \otimes \sigma \tag{11}$$
$$\alpha^{l'}_{GS} = \alpha^l \otimes (\sigma \overline{\oslash} \sigma) \tag{12}$$

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

$$d_{max,GS} = Del(\alpha^u, \sigma) \tag{13}$$
$$b_{max,GS} = Buf(\alpha^u, \sigma) \tag{14}$$

Proof: To prove (11) we use the fact that $R \oslash R$ is the minimum upper arrival curve of a cumulative function R, and we use the properties

$$\begin{array}{rcl}(f \oslash g) \oslash h & = & f \oslash (g \otimes h) \\ (f \otimes g) \oslash g & \leq & f \otimes (g \oslash g)\end{array}$$

that were proven in [4]. We can then compute

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

To prove (12) we use the fact that $R\overline{\oslash}R$ is the maximum lower arrival curve of a cumulative function R. We can then compute

$$R'\overline{\oslash}R' = (R \otimes \sigma)\overline{\oslash}(R \otimes \sigma)$$

$$= \inf_{\lambda \geq 0} \sup_{0 \leq v \leq \lambda} \inf_{v \leq u \leq v+\mu} \{R(u) - R(v) + \sigma(\mu + \lambda - u) - \sigma(\lambda - v)\}$$

When we separately evaluate this formula for $0 \leq u \leq v$, for $v \leq u \leq v + \mu$ and for $v + \mu \leq u \leq \lambda + \mu$, we get

$$\begin{array}{rcl}(R \otimes \sigma)\overline{\oslash}(R \otimes \sigma) & \geq & \min\{(R\overline{\oslash}R) \otimes (\sigma\overline{\oslash}\sigma), R\overline{\oslash}R, \sigma\overline{\oslash}\sigma\} \\ & = & (R\overline{\oslash}R) \otimes (\sigma\overline{\oslash}\sigma)\end{array}$$

The complete proofs for (13) and (14) are omitted here due to space restrictions, but they were deducted starting from the following relations:

$$\begin{array}{rcl}d(t) & = & \inf\{\tau \geq 0 : R(t) \leq R'(t+\tau)\} \\ & = & \inf\{\tau \geq 0 : 0 \leq \inf_{0 \leq u \leq t+\tau} \{\sigma(t+\tau-u) + R(u) - R(t)\}\} \\ b(t) & = & R(t) - R'(t) = R(t) - (\sigma \otimes R)(t) \\ & = & \sup_{0 \leq u \leq t} \{R(t) - R(u) - \sigma(t-u)\}\end{array}$$

□

Relations (11) and (12) can now be used as internal relations of an abstract greedy shaper, and (13) and (14) can be used to analyze delay guarantees and buffer requirements of greedy shapers in a performance model.

4. Applications & Case Studies

In this section, we analyze the two system designs depicted in Fig. 1. The analysis results will clearly reveal the positive influence of greedy shapers to a system's performance and buffer requirements when applied internally, or to 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 that do not dilute the analysis results by any possibly hard recognizable influences of other system properties. Modular Performance Analysis with Real-Time Calculus was however already used several times to analyze bigger and more complex system designs, 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 2 CPU's that communicate via a shared bus, as depicted on the left side in Fig. 1. CPU_1 and CPU_2 both process an incoming event stream S_1 and S_2, 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, where 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 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 besides T_{S1}, other tasks are executed on CPU_1 using a TDMA scheduling policy. Or also 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 arrive 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, also the buffer demand in CNI_2 will increase with 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 the buffer requirements of CNI_2.

To investigate the effect of adding greedy shapers to the system with internal re-shaping in Fig. 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 $p = 1ms$. In both CPU's, the CPU may not be available to process the tasks T_{Si} for up to $5ms$. After this period of at most $5ms$, the processor is fully available and can process 5 events per ms ($\beta^u_{CPU1} = \beta^u_{CPU2} = 5\Delta[e/ms]$, $\beta^l_{CPU1} = \beta^l_{CPU2} = \max\{0, \Delta - 5\}[e/ms]$). 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 four different system designs. First, we analyze the system without greedy shapers, secondly, we place a greedy shaper only at the output of CPU_1 to shape S_1', then, we place a greedy shaper only at the output of CPU_2 to shape S_2', and finally we will add two greedy shapers to shape both S_1' as well as S_2'. 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. On the left side of Fig. 4, the abstract performance model of the fourth system design is depicted.

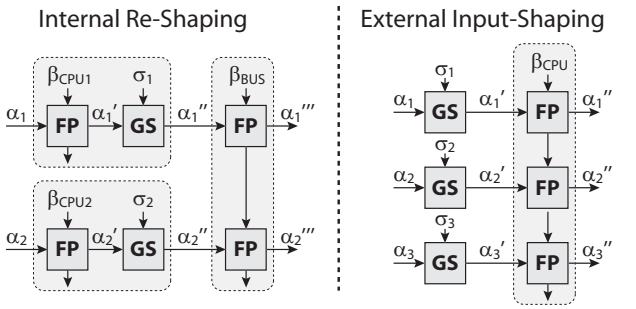

Figure 4. Performance models.

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 are shown in Table 1.

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%

Table 1. Effect of Re-Shaping.

From the results, we learn that placing greedy shapers helps to reduce the total buffer requirements from 25 down to 14 events that need to be buffered at most. 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 re-shaping is for free [4]. Since we use $\sigma_1 = \alpha^u_{S1}$ and $\sigma_2 = \alpha^u_{S2}$, the greedy shapers effectively only re-shape 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 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 corresponds to its design-time timing specification. The system can then be analyzed using the design-time timing specifications, and at run-time, non-adherence of S_i to its timing specification will have no influence to the delay of any other event streams, but will at most increase the total delay of S_i itself. And 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.

Lets assume a real-time system as shown on the right side of Fig. 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, and we then analyze the effect of this to the end-to-end delays of the three event streams, both with and without greedy shapers. The results, computed using Modular Performance Analysis are shown in Table 2.

	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

Table 2. Effect of Input-Shaping.

Looking at the results, we clearly see the big influence of the little non-adherence of S_1 to 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 from its own malbehavior.

5. Conclusions

We introduced a new method to analyze greedy shapers, and we embedded this method into the modular performance analysis framework of [1, 9], 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 could analyze the detailed buffer requirements of all system components, and we could provide end-to-end delay guarantees for the processed event streams. The analysis thereby clearly revealed the positive influence of greedy shapers to the system's performance and buffer requirements.

Appendix: Min-Max Algebra

The min-plus convolution $\otimes$, min-plus deconvolution $\oslash$ and max-plus deconvolution $\overline{\oslash}$ of f and g are defined as:

$$(f \otimes g)(\Delta) = \inf_{0 \le \lambda \le \Delta} \{f(\Delta - \lambda) + g(\lambda)\} \quad (15)$$

$$(f \oslash g)(\Delta) = \sup_{\lambda \ge 0} \{f(\Delta + \lambda) - g(\lambda)\} \quad (16)$$

$$(f \,\overline{\oslash}\, g)(\Delta) = \inf_{\lambda \ge 0} \{f(\Delta + \lambda) - g(\lambda)\} \quad (17)$$

A curve σ is sub-additive, if

$$\sigma(a) + \sigma(b) \ge \sigma(a + b) \quad \forall a, b \ge 0 \quad (18)$$

Acknowledgements

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.

References

[1] S. Chakraborty, S. Künzli, and L. Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *Proc. 6th Design, Automation and Test in Europe (DATE)*, pages 190–195, March 2003.

[2] R. Cruz. A calculus for network delay. *IEEE Trans. Information Theory*, 37(1):114–141, 1991.

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

[4] J. Le Boudec and P. Thiran. *Network Calculus - A Theory of Deterministic Queuing Systems for the Internet*. LNCS 2050, Springer Verlag, 2001.

[5] P. Pop, P. Eles, and Z. Peng. Schedulability Analysis and Optimization for the Synthesis of Multi-Cluster Distributed Embedded Systems. In *Design, Automation and Test in Europe (DATE 2003)*, pages 184–189, 2003.

[6] J. Rexford, F. Bonomi, A. Greenberg, and A. Wong. Scalable architectures for integrated traffic shaping and link scheduling in high-speed ATM switches. *IEEE Journal on Selected Areas in Communications*, 15(5):938–950, 1997.

[7] K. Richter, M. Jersak, and R. Ernst. A formal approach to mpsoc performance verification. *IEEE Computer*, 36(4):60–67, April 2003.

[8] H. Schioler, J. Jessen, J. D. Nielsen, and K. G. Larsen. 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 05)*, pages 61–64. IEEE, 2005.

[9] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 4, pages 101–104, 2000.