Proceedings of the 2012 Winter Simulation Conference
C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose, and A. M. Uhrmacher, eds.

ARRIVAL AND DELAY CURVE ESTIMATION FOR SLA Calculus

Sebastian Vastag

Technische Universität Dortmund
Informatik 4
Martin-Schmeisser-Weg 18
D-44227 Dortmund, GERMANY

ABSTRACT

An algorithm and selection method to estimate Network Calculus arrival bounds for systems with concurrent arrivals is presented. Concurrent job arrivals are common for Service-Oriented Architectures. Their performance is described in Service Level Agreements including quantitative requirements for load and response times. SLA Calculus, a variant of Network Calculus, can be used for service performance modeling and validation with SLAs. Functions called curves are used to bound job arrivals as well as their delay. Due to the concurrent nature of job arrivals curve estimation methods used for successive packet arrivals in Network Calculus cannot be applied in SLA Calculus. We present a method to estimate unknown SLA Calculus arrival and delay bounds from input and output traces. This paper introduces an algorithm for the estimation of the curves. Optimal selection of a curve model based on several fitting criteria is performed using candidates from trace sets.

1 INTRODUCTION

When Service-Oriented Architectures (SOA) are built basic components in form of services are combined to complex systems. In early design phases analytical models can be used to derive system properties. Often quantitative requirements like system response time have to be considered.

Performance guarantees of a service component, a web service for example, is often committed by the service provider to the customer in form of a Service Level Agreement (SLA). SLAs may include guarantees on the response time if a certain bound on the job arrival rate is met. Hard deadlines are rarely used since strict realtime behavior is difficult to implement in systems communicating over the Internet. Interestingly, SLAs are often the only system performance description available since many service providers do not provide the service capacity of their systems.

For modeling quantitative requirements of systems with components described by SLAs, a domain-specific extension of Network Calculus (Le Boudec and Thiran 2004) has been developed. Network Calculus limits arrival rates by so called arrival curves that impose bounds on the cumulated number of arrivals over time (arrival flow). SLA Calculus (Vastag 2011; Vastag 2012) also uses delay curves to bound the system delay over time. As curves are functions burst phases for arrivals or relaxed deadlines can be defined in a flexible way. The combination of arrival and delay curves reflect the two-sided contract between customer (maximum load) and provider (response time). This fact is employed by SLA Calculus to characterize worst-case system performance with SLA Delay Properties (Vastag 2012). They can be used for modeling with arrival and delay limits and without any knowledge of the available processing capacity.

Ideally the bounding curves in SLA Delay Properties can be derived from given SLAs. If the SLA of a component is unknown or the service only exists as a simulation model there is the option to fit curves to recorded traces. Such curve estimation is done for Network Calculus (Künzli et al. 2006; Chakraborty et al. 2003). Packet arrival traces are inspected for packet sizes, sequences and the mean arrival rate. Network properties like maximum packet size (MTU) and correlation of equally sized packets are exploited.

When we tried to use these estimation schemes to fit SLA Delay Properties to components several problems surfaced. Job arrivals to SOA services are not necessarily sequential as packet arrivals to a network interface. A second job can arrive while the first one is still in transmission. The arrival characteristic becomes concurrent as no minimum interarrival times determined by bit rate and packet size are calculable. When several small jobs arrive at the same or almost same time in parallel they can induce more system load than a larger single arrival. Additionally, maximum job sizes equal to a network MTU are often unknown. Fitting delay curves to delay flows yields additional difficulties. When a service finishes a job the processing duration is considered as arrival event in a delay flow. Delay arrivals can also be in parallel if the system can handle multiple jobs at the same time. Their size is measured in time units and it is very unlikely to find two delay arrivals of the same size in sequence, so exploiting the correlation of arrival sizes as in Network Calculus is not possible.

As a consequence, parameter estimation for SLA Calculus is not comparable to arrival curve estimation for Network Calculus based on packet traces. Considering these requirements several open questions have to be answered:

- How to estimate parameters of arrival curves for (parallel) arrivals in SLA Calculus?
- Delay curves limit delay time. Is there a difference in curve estimation for packets or jobs?
- If arrival and delay curves can be estimated, can their fitting quality be quantified?

To address these issues an arrival curve estimation method for general arrival traces has been developed. It is independent of the unit of the measured trace, so it can be used to find bounding functions for service call arrivals, network packet arrivals and also for the system delay arrivals in SLA Calculus. Traces from simulation models or real world systems can be used.

Several function types can be used to define bounds in Network Calculus. Compositions of affine functions is one of the most commonly used function class to define piecewise a bounding function. Since every affine function can be described by rate r and axis intercept b the curve model only needs a small number of parameters.

An algorithm that is able to determine the required number of affine functions as well as their parameters for a given arrival trace will be presented. It is based on the idea that an arrival flow is a superposition of several subflows that can be extracted by a Network Calculus model element called shaper. The extracted subflows are reflected in the number of required affine functions. We will introduce the version for concave curves as upper bounds and briefly discuss the convex variant for lower bounds.

A curve fitted to a single short trace may not be sufficient for bounding a system property in general. Exact bounds for one trace might over- or underestimate the limits for other system runs. Therefore a model selection method is used. We use hold-out validation as a simple form of cross-validation. To calibrate models a small training set is used to generate curves. They are evaluated by criteria defined on boundary violation, time of violation and mean distance to the trace. The best curve is selected and validated on a validation trace set. The method was implemented in a tool and used to generate SLA Delay Properties for simulation models.

In literature several results on estimating arrival curve parameters for a given arrival flow can be found. In (Heckmann et al. 2001) an efficient algorithm for dimensioning an affine function (token bucket) for a given arrival flow is presented. In the second part of the work a method for the Token Bucket Reallocation Problem (TBRP) is given. Under constraints of a cost function the bucket parameters are reconfigured at different points in time. Although the efficient algorithm for a single bucket is interesting it does not dimension parameters for arrival curves with two or more buckets.

A direct approach to calculate arrival curves as input into a simulator is used in (Künzli et al. 2006). Deconvolution (dual operation to convolution introduced in Section 2) of input traces with themselves gives an arrival curve. This equals the minimum arrival curve in (Le Boudec and Thiran 2004, Chapter 1.2.4). A disadvantage is that the curve will be a step function if the flow is a step function. Regarding SLA contract modeling a characterization with less parameters is preferred.

To evaluate routers as a form of embedded systems with Network Calculus Chakraborty et al. proposed a method to derive T-SPEC arrival curves for packet traffic traces (Chakraborty et al. 2003). Traffic specifications originate from modeling variable bitrate IP-traffic (Shenker and Wroclawski 1997) and are thus often found in Network Calculus models. They allow definitions of a short-term burst arrival rate as well as a long-term rate with two affine functions. Measurements in (Chakraborty et al. 2003) included size and inter-arrival times of packets to a network interface and their extrema for curve estimation. Parameters for T-SPEC upper arrival curves are extracted by examining the traces. The maximum packet size is given by the largest arrival (in bytes) found in the trace. When these largest packets are found in a sequence the peak rate p is determined. The long term rate is calculated as the cumulated bytes of all arrivals divided by the measurement interval.
The estimation method proposed in (Chakraborty et al. 2003) is suitable for traces of packets in computer networks. However, it does not give valid values for general arrival traces with parallel arrivals. We consider the case of parallel arrival of n small jobs of size w_1 and the arrival of a single job of size w_2. We assume $w_1 < w_2$. The method of Chakraborty et al. would select w_2 as job size to determine p because this is the largest increment to the arrival flow. This is only true as long $n \cdot w_1 < w_2$, otherwise the bulk of small packets would be the biggest contribution to the arrival flow.

The remaining paper is structured as follows: A short introduction to delay curves in SLA Calculus and its foundation on Network Calculus will be given in Section 2. The main results for arrival and delay curve estimation for SLA Calculus are presented in section 3. Section 4 describes the curve selection criteria. They are applied to an example in Section 5.

2 SLA Calculus

SLA Calculus (Vastag 2011; Vastag 2012) is based on the ideas of Network Calculus. While Network Calculus is primarily intended to derive worst-case bounds on network response times based on known system properties SLA Calculus is used for modeling quantitative requirements in systems. Boundaries are seen as contracts or commitments between service provider and customer.
Network Calculus uses (min,+) algebra replacing addition with the minimum function and multiplication with addition. An extensive overview of the theory can be found in (Baccelli et al. 1992). Network Calculus can be described as linear time invariant filtering theory using wide-sense increasing functions as characteristic description of a system. A function is said to be wide-sense increasing if $f(a) \leq f(b)$ for all $a \leq b$. An important operator in Network Calculus is the (min,+) equivalent to convolution. When f and g are two wide-sense increasing functions their (min,+) convolution (notation $f \underline{\otimes} g$) is the function (c.f. (Le Boudec and Thiran 2004))

$$(f \underline{\otimes} g)(t) = \inf_{0 \leq s \leq t} \{f(t-s) + g(s)\} \text{ if } t \geq 0 \tag{1}$$

Arrivals to a system can be network packets or processing jobs measured in bits, packets or any other quantitative unit to describe data transmissions. In Network Calculus these arrivals are captured with cumulative functions. Let $r(t)$ be the number of arrivals of an arrival process at time t. Then an arrival flow $R(t)$ is the cumulative sum of arrivals in time interval $[0,t]$, thus $R(t) = \int_0^t r(x)\,\mathrm{d}x$ (Figure 1).
To characterize arrival functions and to set bounds on arrival flows, Network Calculus abstracts arrival functions with functions called arrival curves (Le Boudec and Thiran 2004).
When an arrival flow R is processed in a system it will leave as outgoing flow R^*. In general we can assume $R \geq R^*$ is true and R^* also fulfills the arrival curve property. For this reason it is often referred as outgoing arrival flow. Figure 2 shows both flows (Le Boudec and Thiran 2004).

A function α^U is said to be an upper arrival curve for arrival function $R(t)$ when $R \leq R \underline{\otimes} \alpha^U$ holds. The arrival curve property expresses that for all possible time intervals R does not grow faster than α^U. This is the time invariance filtering property of Network Calculus. A lower curve α^L is given by $R \geq R \underline{\otimes} \alpha^L$.

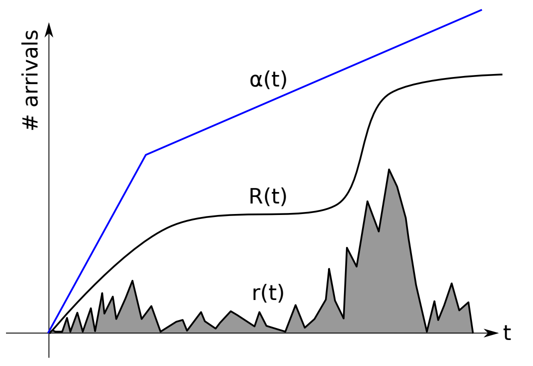

Figure 1: Arrival function $r(t)$, arrival flow $R(t)$ and arrival curve $\alpha(t)$.

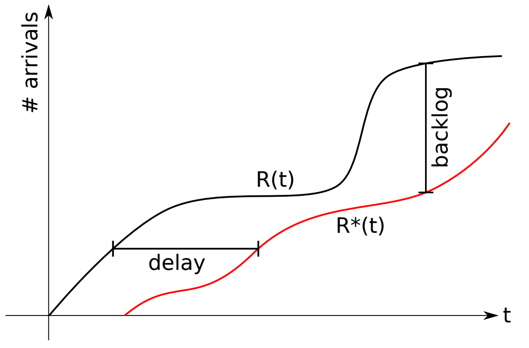

Figure 2: Horizontal and vertical deviations between $R(t)$ and $R^*(t)$.

Important results in Network Calculus can be obtained from the horizontal and vertical deviations between functions. The horizontal deviation between two non-decreasing functions f, g in t is

$$\delta(f,g)(t) = \inf\{\tau \geq 0 : f(t) \leq g(t+\tau)\} \tag{2}$$

The value of $\delta(f,g)(t)$ is the additional time $g(t)$ needs to reach the level $f(t)$. In Network Calculus the maximum horizontal deviation (Fig. 2) between input and output gives the maximum system latency (Le Boudec and Thiran 2004; Chang 2001).
The vertical deviation of two wide-sense increasing functions f and g, $f(t) \geq g(t)$ is $x(t) = f(t) - g(t)$. Vertical deviation $x(t)$ describes the backlog (buffer content) used by a system to buffer arrivals until they are processed and sent to the output (Fig. 2).

To set bounds on system response times, an arrival curve variant was introduced for SLA Calculus in (Vastag 2011). When an arrival flow traverses a system it is subject to delays. For each arrival leaving the system its delay is emitted as delay arrival into a delay flow. With this system model a delay flow can be interpreted as an arrival flow with a different unit and therefore it can be bounded with delay curves similar to arrival curves.
For an arrival flow R and its corresponding departure flow R^* the system delay in t is $\delta(R,R*)(t)$. The unit of delay is a unit of time. Since $\delta(R,R*)(t)$ is an arrival function of delay events, a delay flow can be formed. Delay flow $D(t)$ is the cumulative sum of delays in time interval $[0,t]$.

$$D(t) = \int_0^t \delta(R,R^*)(x)\,\mathrm{d}x \tag{3}$$

Delay flows can be bounded with delay curves. A lower delay curve Ψ^L or upper delay curve Ψ^U for delay functions $D(t)$ satisfy the relations $D \geq D \underline{\otimes} \Psi^L$ and $D \leq D \underline{\otimes} \Psi^U$. $\Psi^L(t)$ and $\Psi^U(t)$ are lower and upper bounds of delays occurring in the interval $[0,t]$. For further details we refer the reader to (Vastag 2012).

Every wide-sense increasing function is suitable as arrival curve. In modeling practice a small set of affine or step functions or combinations of them is used. (Le Boudec and Thiran 2004, 3.1.3) includes a catalog of wide-sense increasing functions. A simple but very common instance of an arrival curve is given by an affine function $\gamma_{r,b} = \max(r \cdot t + b, 0)$ for $t \geq 0$. As shown in (Le Boudec and Thiran 2004), $\gamma_{r,b}$ describes a leaky-bucket with bucket size b and long-term rate r. An intuitive interpretation of a leaky bucket is a system that accepts items (arrivals), processes and emits them. Figure 3 shows two leaky buckets. Items of size w can only be processed if w fluid from an attached bucket with maximum available capacity b is taken. The tandem halts if one station halts. Parameter b is also referred as maximum burst size as the bucket content can be immediately consumed. The bucket is refilled by a constant inflow ('leaky') with rate r. This rate is known as the mean rate.

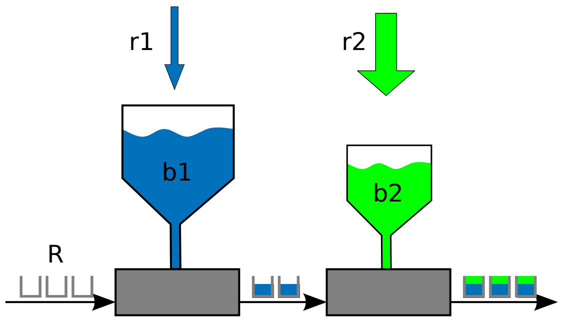

Figure 3: Two leaky buckets with bucket capacities $b1, b2$ and refill rates $r1, r2$.

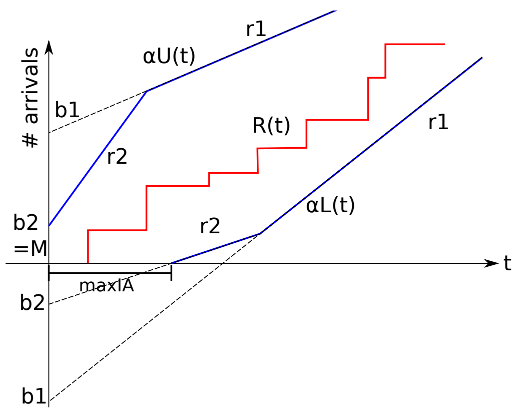

Figure 4: Upper (α^U) and lower (α^L) arrival curve.

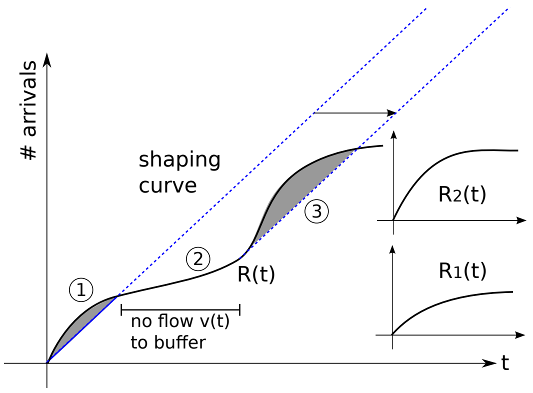

Figure 5: Effects of a shaper processing a flow. The flow $v(t)$ towards the buffer forms two new arrival flows.

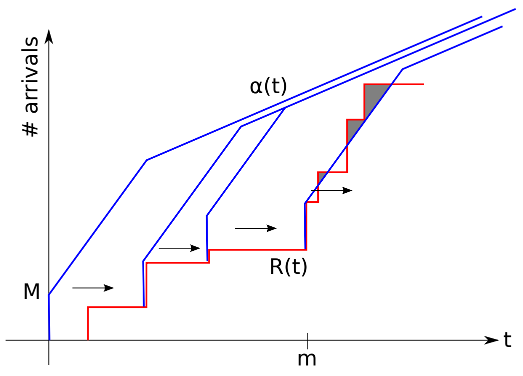

Figure 6: Curve violations and time invariance for arrival curves. For estimation curves are shifted by event arrival times.

When buffer content b is used in a burst the output rate of the leaky bucket system is infinite. This situation can be avoided when the burst rate is limited with a second leaky bucket system $\gamma_{r2,b2}$ connected to the first one (Fig. 3). An arrival can only pass the tandem system if every bucket contains enough fluid to be consumed, otherwise the system halts. Bursts emitted by bucket $\gamma_{r1,b1}$ can be managed by bucket $\gamma_{r2,b2}$. The burstiness of the arrival flow can be controlled at a more detailed level by adding additional leaky buckets to bound the curve f: $f(t) = \min_i(\gamma_i(t))$ with $\gamma_i(t) = r_i \cdot t + b_i$ and $r_{i+1} > r_i, b_i > b_{i+1}$ for concave curves (Fig. 4).
Regarding arrival lower envelopes convex curves are used: $f(t) = \max_i(\gamma_i(t), 0)$ with $r_{i+1} < r_i$ and $b_{i+1} < b_i \leq 0$. y-intercepts b_i are used to model quantities on the time axis (c.f Figure 4). b_i is the lag to an ideal arrival curve $r_i \cdot t$.

2.1 Curve Contract Modeling and SLA Delay Properties

Arrival curves can be used to specify and model a contract on the arrival flow. SLA Delay Properties (Vastag 2012) in SLA Calculus reflect contracts or commitments between service providers and customers for quantitative requirements found in SLAs. Bounds on the workload (request flow R) sent to service

providers by the customer are expressed with upper arrival curves α^U and optionally lower curves α^L. The arrival flow sent by the customer is conform to the SLA Delay Property if $R \leq R \underline{\otimes} \alpha^U$ (optional $R \geq R \underline{\otimes} \alpha^L$). From the perspective of SLAs it is the customers part of the contract.
The contract part of the service provider is expressed with delay curves Ψ^U and Ψ^L. They limit the cumulated system delay D generated by the processing of R within all time intervals : $D \leq D \underline{\otimes} \Psi^U$. A minimum delay can be required with Ψ^L. Delay curves Ψ are dependent on their complementary arrival curves α, so SLA Delay Properties are sets $\{\alpha^L, \alpha^U, \Psi^L, \Psi^U\}$ of bounding curves ($\alpha^L \leq \alpha^U, \Psi^L \leq \Psi^U$). If it is not necessary to use lower boundaries the abbreviation $\{\alpha^U, \Psi^U\}$ can be used.

SLA Delay Properties only virtually regulate request flows, service users are free to produce workloads above limits but in those cases they cannot claim any guarantees on delays. However, if the limits set by arrival and service curves are exceeded by measured arrival and delay flows the overload has to be quantified. We will define the overload with the help of a Network Calculus buffered shaper.
A shaper takes an incoming arrival flow and modifies it in a way that it satisfies an arrival curve constraint. A greedy shaper (Le Boudec and Thiran 2004) uses a buffer to store excessive arrivals and outputs them as soon as arrival curve constraint δ is fulfilled. Its output flow is given by $R^* = R \underline{\otimes} \delta$ (identity used here). Figure 5 shows shaping with a linear curve $\lambda_r(t) = r \cdot t$ applied to an arrival flow. Every arrival above the limit has to be buffered (1). If the arrival flow throttles no shaping is applied (2), but at the beginning of the next burst phase the shaping curve is applied immediately (3).
When shaping is applied we are interested in the function $v(t)$ of excessive arrivals towards the buffer (Figure 5). An arrival flow passing a shaper with arrival curve α^U with zero buffer size ($v(t) = 0 \ \forall t \geq 0$) conforms to arrival curve α^U. If the system has to buffer ($v(t) > 0, t \geq 0$) a contract violation occurred. The buffer occupation $x(t)$ of a greedy shaper is given by (Le Boudec and Thiran 2004, 1.5.2):

$$x(t) = \sup_{0 \leq s \leq t} \left\{ R(t) - R(s) - \alpha^U(t-s) \right\} \tag{4}$$

This is the maximum difference between real input $R(s)$ and system output $R(t)$ and the output the shaping curve allows in interval $t - s$. The curve estimation algorithm presented in the following section uses buffer flow $v(t)$ to separate an arrival flow into subflows with different rates and $x(t)$ to determine the bucket size for a leaky bucket.

3 CURVE ESTIMATION FOR SLA Calculus

Curve Estimation for SLA Calculus can be applied to arrival and delay flows as delay flows are just arrival flows with a different unit. This allows to estimate SLA Delay Properties from measurements. However, when estimating arrival curves from arbitrary traces situations will occur when a second measurement under same conditions does not conform to the estimated curves. An option to avoid such violations is to formulate very loose-fitting traffic contracts. This over-fitting might lead to very pessimistic assumptions on system performance in a following system analysis. The road taken here is to accept curve contract violations, quantify them and select the 'best' fitting curve with respect to several measurements. The primary goal of the following curve estimation is to parametrize a curve in a way that violations are minimized.

To analyze a trace we use a discrete arrival model based on event traces: Events e_i with index $1 \leq i \leq n$ are captured by measurement from a system. Request events are discrete arrivals to systems weighted by their request size or processing complexity (Bause et al. 2009). Thus an event $e_i = (t_i, w_i, a_i)$ has an arrival time t_i, a weight w_i (default $w_i = 1$) and an identifier a_i. The weight w for an event in a delay flow has the value of the measured delay. For simplicity of notation we use $t(e_i) = t_i, w(e_i) = w_i$ and $id(e_i) = a_i$.
An event trace E is a set of events $e_1, \ldots, e_n$ with $t_i \leq t_j$ for $i < j$. $|E|$ is the number of events in E. Arrival flow $R(E,t)$ is a step-function indicating the cumulated weight: $R(E,t) = \sum_{e \in E, t(e) \leq t} w(e)$.
If the system delay per arrival is not recorded directly it can be constructed from traces measured at the input and the output of a system. When A is the arrival trace and B is the departure trace, delay trace D is formed by computing the horizontal distance (c.f. Section 2) of matching input and output events. The new events $(t_i, w_i, a_i) \in D$ are given by $(t_i = t(b), w_i = t(b) - t(a), a_i = id(a))$.

3.1 Curve Estimation: The Onion Bucket Algorithm

The basic idea of our arrival curve estimation algorithm is that all arrival flows are a superposition of any number of sub-flows clearly distinguishable by their rate. The algorithm extracts these sub-flows with a shaper and assigns a mean rate r_i and buffer requirement b_i to each sub-flow i, hence every sub-flow corresponds to an affine function $\gamma_i(t) = r_i \cdot t + b_i$ in the concave bounding function $\alpha^U = min_i(\gamma_i)$.

This algorithm is called the onion bucket algorithm as the flow is shaped and cut like onion bulbs. For technical reasons the algorithm works with elementary traces E instead of flows $R(E,t)$. Listing 1 contains the main loop generating affine functions and Listing 2 contains a shaper for a set of traces.
Initial input of Listing 1 is a measured arrival trace E. It is included in the first bulb, a data structure containing the sub-traces of each loop iteration (line 2). The output will be stored in variable *curve*, a set of affine functions with no initial content.

The main part of the algorithm runs in an infinite loop whose iterations are identified by counter i: For all traces in *bulb* the mean rate is computed and its maximum is saved in r_i (line 8). Also the maximum buffer requirement b_i for a shaper processing flow $R(E,t)$ with a linear shaping curve of rate r_i is estimated for all flows in the bulb. Buffer estimation function $x(trace, rate)$ is discussed below.
Now the parameters for the first ($i = 1$) upper boundary $\gamma_i(t) = r_i \cdot t + b_i$ are known, the affine function can be added to data structure *curve* (line 16). γ_i does limit flow $R(E,t)$ in the long term (for the duration of the trace) but does not in short term. Arrival bursts phases with rate r_b and $r_i \leq r_b \leq \infty$ and cumulated weight up to b_i are still possible. To find rate limits for these short phases we have to extract those sub-flows whose superposition forms R and that have a higher rate than r_i. Line 22 prepares the trace set for the shaping by removing all traces whose buffer requirement is smaller than maximum arrival size M.
The shaping itself is performed in function SHAPEANDCUT (Listing 2): Input is the set *onion* of arrival traces to be shaped and the rate parameter r for a linear shaping curve. Return value *bulb* is initialized as empty set of traces. For all traces E in *onion* the following procedure is repeated: First some variables are set: i is the iteration counter and *startTime* marks the beginning of a backlogging phase. Although the shaping curve is linear its y-axis intercept is variated with b.
At the level of every single event $e_j \in E$ cumulated weight $R_i(E, t(e_j))$ is compared to the value of shaping curve $\gamma(t(e_j))$ to decide if backlogging is required. If it is the beginning of a backlogging phase a new trace data structure $lossFlow$ is created (line 4). It will contain the arrivals in E that are non-compliant for the shaper and thus have to be backlogged in a buffer. The offset *startTime* is set to the event arrival time of the last event e_{j-i} before backlogging started (line 14). This is necessary to preserve the rate induced by e_j in the new trace. All succeeding events in the backlogging phase are transformed to new events e' using offset *startTime* and their full weight (line 18). If there is no backlog the shaping curve is lowered by adjusting b to simulate the effect of a (min,+) convolution as described in section 2.1 and Fig. 5.
We return to Listing 1. With the new set of traces generated by the shaper in line 23 the next loop iteration can be started. It will produce an affine bounding function that limits the loss flows towards the shaper buffer in terms of rate and bucket size. The loop runs as long as there are traces in set *bulb* whose corresponding flows can be bound by an affine function and shaped to subflows (line 24). Maximum arrival size M was saved at the beginning to set all b_i to at least M. We decided to do so because when there is an arrival of weight M the minimum bucket size of all leaky buckets bounding the flow has to be greater or equal to M. Otherwise the arrival with weight M could never pass the system. For this reason condition *endPhase* is set in line 21 to ensure that no buffer requirement is smaller than M. It decides in line 15 if a new affine curve is added or the last one is just updated with a new rate.

The maximum buffer size estimation $x(E,r)$ function is based on equation 4 but takes care of the step function nature of R and the discrete points of time $t(e_i)$ resulting from the event arrivals.

$$x(E,r) = \max_{t>0}\{ \sup_{0\leq s\leq t} \{R(E,t) - R(E,s) - w(s) - r(t-s)\}\} \tag{5}$$

Algorithm 1 Fitting upper (concave) arrival curves.

```
1: $set\ of\ affine\ functions:\ curve \leftarrow \{\}$
2: $set\ of\ traces:\ bulb \leftarrow \{E\}$
3: $M \leftarrow maxWeight(bulb)$
4: $endPhase \leftarrow$ false
5: $loopActive \leftarrow$ true
6: $i \leftarrow 1$
7: repeat
8:     $r_i = \max(rate(bulb))$
9:     $b_i \leftarrow t \leftarrow 0$
10:    for all $E \in bulb$ do
11:        $myBuffer \leftarrow x(E, r_i)$
12:        $b_i \leftarrow \max(myBuffer, b_i)$
13:    end for
14:    $b \leftarrow \max(b_i, M)$
15:    if $\neg endPhase$ then
16:        $curve \cup (r_i, b_i)$
17:        $t \leftarrow i$
18:    else
19:        $r_t \leftarrow r_i$
20:    end if
21:    $endPhase \leftarrow b_i \leq M$
22:    $bulb \leftarrow \{E : E \in bulb, x(E, r_i) \geq M\}$
23:    $bulb \leftarrow$ SHAPEANDCUT$(bulb, r_i)$
24:    $loopActive \leftarrow elements(bulb) > 0$
25:    $i \leftarrow i + 1$
26: until $\neg loopActive$
27: return $curve$
```

Algorithm 2 Shaper implementation.

```
1: function SHAPEANDCUT$(onion, r)$
2:     $set\ of\ traces:\ bulb \leftarrow \{\}$
3:     for all $E \in onion$ do
4:         $trace:\ lossFlow \leftarrow \{\}$
5:         $startTime \leftarrow 0$
6:         $b \leftarrow 0$
7:         $i \leftarrow 0$
8:         $backLogging \leftarrow$ false
9:         for all $e_i \in E$ do
10:            if $R(E, t(e_i)) > r \cdot t(e_i) + b$ then
11:                if $\neg backLogging$ then
12:                    $backLogging \leftarrow$ true
13:                    $i \leftarrow i + 1$
14:                    $startTime \leftarrow t(e_{i-1})$
15:                    $lossFlow_i \leftarrow \{\}$
16:                    $bulb \leftarrow bulb \cup lossFlow_i$
17:                end if
18:                $e' = (t(e_i) - startTime, w(e_i))$
19:                $lossFlow \leftarrow lossFlow \cup e'$
20:            else
21:                $backlogging \leftarrow$ false
22:                $b \leftarrow R(E, t(e_i)) - r \cdot t(e_i)$
23:            end if
24:        end for
25:    end for
26:    return $bulb$
27: end function
```

When evaluating $x(E,r)$ at $t(e_i)$ only the left beginning of each step in $R(E,t)$ is considered. To include the time interval between two succeeding events e_i, e_{i+1} term $w(s)$ is subtracted (Fig. 6). If an event has to be buffered it is buffered as a whole, thus the maximum of the required buffer and the event is used.

A convex version of the algorithm for lower arrival curves can be implemented by using minimum instead of maximum operations for rates and replacing bucket sizes b_i with time-axis intercepts. Instead of maximum arrival M the maximum interarrival time can be used (Fig. 4).

4 MODEL SELECTION

To select characteristic arrival and delay curves for a system we apply Hold-Out Validation (Kohavi 1995) to a sets of traces. The method can be divided into calibration and validation phases. The first phase calibrates our model (arrival curve) for a small and exclusive calibration set using the estimation algorithm from Section 3. Typically 10% of the traces are used for calibration. The second task of the calibration phase is to select a curve from the estimated curves that has the fewest curve contract violations with the other calibration traces. We use a cost function described in the following to quantify the error between a curve and a flow. The curve with the lowest error value is selected.
The second phase tests the curve against the remaining traces. For every combination of the selected curve candidate and a trace the error value is computed with the same conditions applied to the calibration set.

A selected curve is accepted if the confidence interval of its validation results shows significant distance to error values of unselected curve candidates.

A first attempt for a cost function is to average the buffer occupation $x(t)$ (eqn. 4) for a curve and all flows under test. For a good estimation the mean buffer size has to be minimized. The average buffer usage does indicate curve contract violations but leaves out some aspects that are interesting for modeling quantitative properties for SLAs. It gives no information on the duration of the contract violation and it does not prevent loosely fitted curves. Instead we developed three criteria for violation duration, violation area and the mean squared distance between curve and flow. For better comparison independent of trace lengths their results are normalized in interval $[0,1]$

The fulfillment of an arrival curve contract is tested for every arrival in the criteria to consider the time invariance of arrival curves (see Section 2.1). Figure 6 shows a situation for upper curves where only one computation of the fitting criteria would ignore a contract violation. Starting the curve at $t = 0$ will indicate no contract violation as $R(t) < \alpha^U(t)\ \forall t$. If we apply the arrival curve at a later point in time arrivals following position m clearly violate the curve contract. To preserve time-invariance in fitting criterion computation a shifting operator S is applied to event traces E: $S(E,\Delta) = \{e' | e' = [t(e) - \Delta, w(e), a(e)], e \in E, t(e) - \Delta \geq 0\}$ Let $\omega(E) = \max_{e \in E}(t(e))$ return the arrival time oft the last event in E. The average and time-invariant value of every criterion $c \in \{it, ia, mse\}$ can be computed with

$$AVG(c,f,E) = \sum_{e \in E} \frac{1}{|S(E,t(e))|} \frac{1}{\omega(S(E,t(e)))} c(f, S(E,t(e))) \tag{6}$$

Indicator function $h(f,g)(x) = 1_{(0,\infty)}(g(x) - f(x))$ shows curve contract violations. The criteria are normalized to a reference (worst-case) value p for better comparability between different trace lengths.
The first criterion is the mean intersection time of arrival flow and bounding curve. It gives the percentage how long the arrival curve contract within a time interval is violated (Figure 7):

$$it(\alpha,E) = \int_0^{\omega(R_e)} x \cdot h(\alpha,E)(x)\,\mathrm{d}x \tag{7}$$

For normalization we use the worst-case intersection time $p = \omega(R)$.
The second criterion, the mean intersected area, includes the area between arrival function and arrival curve (Figure 7) spanned by a curve contract violation:

$$ia(\alpha,E) = \int_0^{\omega(E)} (R(E,x) - \alpha(x)) \cdot h(\alpha,E)(x)\,\mathrm{d}x \tag{8}$$

Worst-case intersected area for scaling is given by $p = \int_0^{\omega(E)} R(E,x)\mathrm{d}x$. This criterion can be seen as an approximation for the omitted mean buffer occupation.
The last criterion is on the distance between curve and flow. It is intended as an antagonist to intersection time and area by giving a reward on a tight fit between curve and flow. The first two criteria based on intersection will return a value of 0 if there is no curve contract violation. This is independent on the distance between flow and curve, using only intersection might result in very loose bounds. To include and reduce the distance the mean squared distance is added to the cost function set: $\mathrm{mse}(\alpha,E) = \sum_{e \in E} (R(E,t(e)) - \alpha(t(e)))^2$. To normalize the result we use $p = \mathrm{mse}(\alpha, R_Z)$ with $R_Z(t) = 0$ and event arrival times $E_{R_Z} = \{0, x_i, \omega(R)\}$. Time x_i is the abscissa value of the intersection points of curve segments γ_i. A flow with no arrivals will give the worst-case fitting value for any curve.

5 EXAMPLE

The Onion Bucket Algorithm and the model selection described in Section 3 and 4 were implemented in an SLA Delay Property fitting tool. To generate input and output traces the *OMNeT++* simulator (Varga

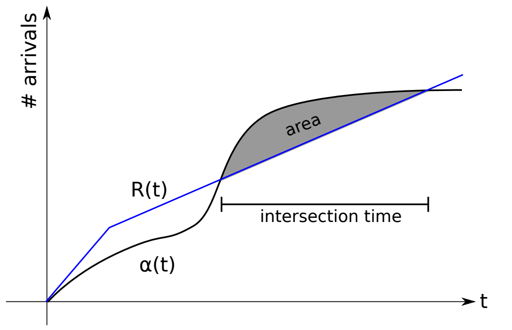

Figure 7: Application of intersected area and intersection time criteria.

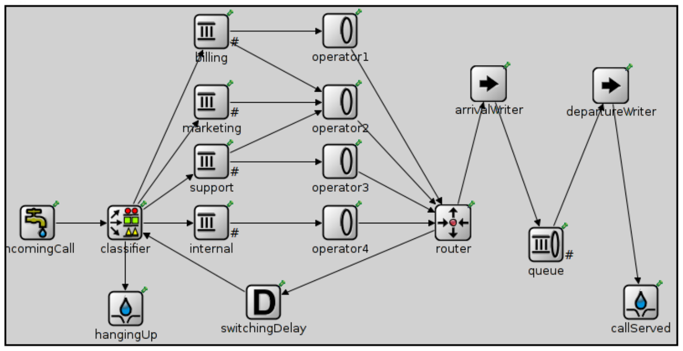

Figure 8: Modified *OMNeT++* callcenter model with trace output.

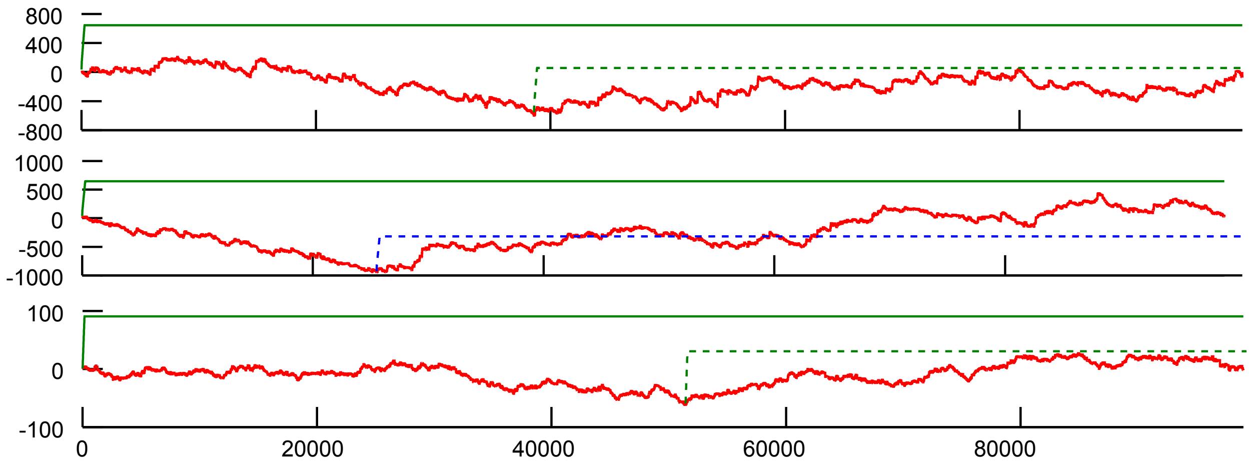

Figure 9: Top: Arrival flow with estimated α^U (curve #8). Middle: α^U with worst fitting flow #1. Bottom: Delay flow with estimated Ψ^U (curve #2). Transformation $f(t) - r_1 \cdot t$ applied to all plots f.

and Hornig 2008) was used, delay traces are computed by the estimator as described in Section 3.
Here we present the estimation of a SLA Delay Property for a system based on the callcenter model from the *OMNeT++* queueing library examples (Fig. 8). The output of the callcenter model acts as a service call generator for a simple server system. We have chosen this model configuration to include a source with a non-trivial interarrival time distribution in the system. The original callcenter ends in the router component in the right part of the model (Fig. 8). Arrivals are recorded at the connected arrival trace writer and forwarded to the server component, its output is delivered to the departure trace.
Incoming calls to the callcenter have exponential ($\mu = 15s$) distributed interarrival times. The additional server has a service rate given by to a truncated normal distribution ($\mu = 5s, \sigma = 2s$). For the experiment we generated 100 pairs of traces with identical parameters but different seeds for 100000s of model time each. To avoid the transient phase of the simulation runs the arrivals of the first 2000s were dropped.

For calibration we used the first 10 pairs of traces to estimate arrival and delay curves. Table 1 shows the estimated curve parameters with 3 and 4 buckets for each arrival flow. In the first bucket of all curves the average rate $r_1 \approx \frac{1}{15} = 0.0\bar{6}$ is determined by the arrival rate of incoming calls to the callcenter. For bucket b_3 (b_4) the buffer size equals the default arrival weight 1 (Sec. 3).
The mean results of each criteria applied to a curve and the nine remaining flows of the calibration set are also given in Table 1. Based on the overall mean value on the right curve #8 was selected as characteristic

Table 1: Estimated leaky bucket parameters for upper arrival curve α^U.

	Bucket Parameter								Intersected Area		Intersection Time		Mean Squared Error		Overall
Curve #	b_4	r_4	b_3	r_3	b_2	r_2	b_1	r_1	mean	std.	mean	std.	mean	std.	mean
1	1	705.582	1.57781	297.893	4.98795	0.197502	74.094	0.0627804	0.00322	0.00468	0.16297	0.14161	0.00170	0.00023	0.05597
2	-	-	1	2065.46	2.14333	4.28695	82.5346	0.0649811	0.00001	0.00003	0.00326	0.00978	0.00302	0.00069	0.00210
3	1	537.008	1.98774	6.58638	16.6349	0.0777795	98.7674	0.0636619	0.00045	0.00116	0.03173	0.06840	0.00235	0.00060	0.01151
4	-	-	1	978.786	2.7071	1.09763	58.8953	0.0640002	0.00100	0.00225	0.06521	0.11510	0.00134	0.00032	0.02252
5	-	-	1	385.448	3.87977	0.258886	56.8743	0.0639058	0.00124	0.00263	0.07732	0.12668	0.00119	0.00028	0.02658
6	-	-	1	727.938	3.71806	0.374306	72.6915	0.06342046	0.00147	0.00292	0.08164	0.12112	0.00156	0.00028	0.02822
7	-	-	1	2572.86	2.84389	0.699311	107.222	0.0640278	0.00014	0.00041	0.01565	0.04464	0.00350	0.00064	0.00643
*8	-	-	1	1229.6	3.1065	0.593122	90.6407	0.0655885	0.00000	0.00000	0.00000	0.00000	0.00401	0.00087	0.00134
9	-	-	1	1125.13	3.31477	0.767517	55.816	0.0639301	0.00124	0.00264	0.07786	0.12754	0.00120	0.00028	0.02677
10	-	-	1	230.345	2.84519	0.874648	74.8787	0.064284	0.00033	0.00091	0.02735	0.06549	0.00211	0.00048	0.00993

Table 2: Estimated leaky bucket parameters for upper delay curve Ψ^U.

	Bucket Parameter								Intersected Area		Intersection Time		Mean Squared Error		Overall
Curve #	b_4	r_4	b_3	r_3	b_2	r_2	b_1	r_1	mean	std.	mean	std.	mean	std.	mean
1	-	-	28.4966	693.853	93.7272	2.19688	902.419	0.39801	0.00224	0.00372	0.08412	0.11258	0.00438	0.00064	0.03024
*2	-	-	35.4441	215.749	106.168	2.47372	645.733	0.42198	0.00000	0.00001	0.00072	0.00217	0.00554	0.00151	0.00209
3	28.2806	26.7737	36.6414	9.63921	246.768	0.55034	950.943	0.40987	0.00006	0.00015	0.00917	0.02369	0.00647	0.00163	0.00523
4	-	-	30.6942	45.1363	82.0811	1.57013	461.898	0.40217	0.00480	0.00662	0.17138	0.16479	0.00171	0.00025	0.05930
5	-	-	31.4197	58.2452	116.512	1.36264	539.618	0.40394	0.00292	0.00468	0.11699	0.14593	0.00221	0.00043	0.04071
6	-	-	34.9433	189.499	93.1401	3.17001	958.717	0.40184	0.00097	0.00191	0.04912	0.08179	0.00535	0.00092	0.01848
7	-	-	29.2407	438.891	100.577	2.66975	912.093	0.40642	0.00038	0.00086	0.02915	0.05894	0.00549	0.00110	0.01168
8	-	-	29.8506	36.4559	55.6746	2.56809	980.164	0.41886	0.00000	0.00000	0.00000	0.00000	0.00847	0.00180	0.00282
9	-	-	39.1513	56.6582	392.27	1.08789	775.873	0.40641	0.00076	0.00157	0.04478	0.07918	0.00423	0.00090	0.01659
10	-	-	29.9802	89.2779	41.8839	5.22601	592.197	0.40665	0.00155	0.00284	0.07586	0.11574	0.00287	0.00065	0.02676

Table 3: Validation findings.

		Intersected Area		Intersection Time		Mean Squared Error				
	#	mean	std.	mean	std.	mean	std.	mean	std.	95% conf.
α^U	2	0.00000	0.00000	0.00004	0.00036	0.00418	0.00112	0.00141	0.000368	[0.001399, 0.0014152]
Ψ^U	8	0.00002	0.00013	0.00283	0.01292	0.00605	0.00203	0.00297	0.004205	[0.002878, 0.0030627]

model for the delay flow. Table 3 includes the validation results when curve #8 is applied to the validation set. When the 95% confidence interval is compared to the calibration results, selected curve #8 has a decent distance to the mean error of other curves in the calibration set. In Figure 9 curve #8 is plotted together with the flow it is based on. The estimated curve always stays above the arrival flow independent of the starting point s and we have $R(t) \leq \inf_{0 \leq s \leq t}\{R(t-s) + \alpha^U(s)\} = (R \underline{\otimes} \alpha^U)(t)$. Indeed α^U is an upper arrival curve for flow R. The dashed line in Fig. 9 depicts the situation where bucket size b_1 is used to fulfill this relationship. A flow from the calibration set with the most curve contract violations is plotted below for comparison.

To complete the SLA Delay Property an upper delay curve Ψ^U was also estimated. Table 2 shows estimated parameters and comparison results of the calibration phase, curve #2 was selected for its low overall error value. Validation results are shown in Table 3, a plot is also included in Figure 9. Again the confidence interval does not overlap with other cost values from the calibration phase.

All three criteria were equally weighted for curve selection. An interesting point is the impact of the mean squared distance to curve selection: On the one hand delay curve #8 has virtually no curve violations for the calibration set. Still it was not selected because for its high distance to the flows and curve #2 was preferred despite found curve violations. On the other hand arrival curve #8 was selected with no curve violations and a high distance to all calibrations flows. The explanation is the low variance in the arrival traces, so the estimated long-term rate r_1 becomes important. Delay curve #8 has the highest r_1 of all curve candidates, additionally r_2 is also high. This bucket combination results in less local curve violations, but if a flow has a low average rate the mean squared distance increases.

6 CONCLUSION

We presented a method to estimate arrival curves from measurements or simulation traces. It can be used for systems with parallel arrivals in SOA where other estimation schemes from Network Calculus for sequential arrivals are unsuitable. SLA Calculus Delay curves to bind delays can also be estimated.
The algorithm extracts subflows with higher than average rate and bounds them with affine functions. The corresponding bucket size is computed from buffer occupation of a shaper used to separate the subflows. Output of the estimation is a upper or lower arrival curve given by several leaky buckets. For model selection hold-out validation is used. The method was implemented and applied to a simulation trace set.

REFERENCES

Baccelli, F., G. Cohen, G. Olsder, and J. Quadrat. 1992. *Synchronization and Linearity*. Wiley New York.

Bause, F., P. Buchholz, J. Kriege, and S. Vastag. 2009, December. “Simulation Based Validation of Quantitative Requirements in Service Oriented Architectures”. In *Proceedings of the 2009 Winter Simulation Conference*, edited by M. D. Rossetti, R. R. Hill, B. Johansson, A. Dunkin, and R. G. Ingalls, 1015–1026. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Chakraborty, S., S. Künzli, L. Thiele, A. Herkersdorf, and P. Sagmeister. 2003. “Performance evaluation of network processor architectures: Combining simulation with analytical estimation”. *Computer Networks* 41 (5): 641–665.

Chang, C. 2001. “Performance guarantees in communication networks”. *European Transactions on Telecommunications* 12 (4): 357–358.

Heckmann, O., F. Rohmer, and J. Schmitt. 2001, December. “The Token Bucket Allocation and Reallocation Problems (MPRASE Token Bucket)”. Technical Report TR-KOM-2001-12, Technische Universität Darmstadt, Multimedia Communications Lab (KOM).

Kohavi, R. 1995. “A study of cross-validation and bootstrap for accuracy estimation and model selection”. In *International joint Conference on artificial intelligence*, Volume 14, 1137–1145.

Künzli, S., F. Poletti, L. Benini, and L. Thiele. 2006. “Combining simulation and formal methods for system-level performance analysis”. In *Design, Automation and Test in Europe, 2006. DATE’06. Proceedings*, Volume 1, 1–6. IEEE.

Le Boudec, J.-Y., and P. Thiran. 2004, May. *Network Calculus - A Theory of Deterministic Queuing Systems for the Internet*, Volume 4 of *LNCS*. Springer Verlag.

Shenker, S., and J. Wroclawski. 1997. “RFC 2215”. *General Characterization Parameters for Integrated Service Network Elements*.

Varga, A., and R. Hornig. 2008. “An overview of the OMNeT++ simulation environment”. In *Simutools ’08: Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, 1–10. Brussels, Belgium: ICST.

Vastag, S. 2011, May 17th-20th. “Modeling quantitative requirements in SLAs with Network Calculus”. In *Proceedings of the 5th International ICST Conference on Performance Evaluation Methologies and Tools (ValueTools)*. ENS, Cachan, France: ICST.

Vastag, S. 2012. “A Calculus for SLA Delay Properties”. In *Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance*, edited by J. Schmitt, Volume 7201 of *Lecture Notes in Computer Science*, 76–90. Springer Berlin / Heidelberg. 10.1007/978-3-642-28540-0_6.

AUTHOR BIOGRAPHY

SEBASTIAN VASTAG is a research assistant at the chair for quantitative techniques in computer science at the TU Dortmund, Germany. He received the Diploma degree in computer science in 2006. His research topics are modeling, analysis and validation of systems with Network Calculus. His e-mail address is sebastian.vastag@udo.edu.