Generating Event Traces Based on Arrival Curves

Simon Künzli* and Lothar Thiele

Computer Engineering and Networks Laboratory,
Swiss Federal Institute of Technology (ETH) Zurich, Switzerland
{kuenzli,thiele}@tik.ee.ethz.ch

Abstract. System-level performance-evaluation methods for computer and communication systems can be divided into two main areas: simulation and analytic methods. Analytic methods are often fast but rather coarse, whereas simulation is time-consuming but often leads to more accurate results. Therefore, there is the need to (a) determine analytic models from simulation results, actual measurements or formal specifications and (b) to generate representative event traces from analytic models. Whereas there are many results available in case of statistical analytic models, there are no methods known for other forms of variability characterizations. The method presented in this paper is suited for arrival curves, a widely accepted tool for traffic characterization. In addition, this class of event models has been successfully used to perform a modular performance, end-to-end delay and buffer size analysis of distributed computer and communication systems. In particular, we propose a new method to generate event traces for simulation or physical measurements, starting from a formal specification of event streams in form of arrival curves. In addition, a quality indicator is defined to evaluate the generated traces. Finally, experiments are described that show the applicability of the approach.

1 Introduction

Performance evaluation is an important task during all phases of system design. In the context of complex heterogeneous component-based systems, designers need to have models and methods at hand to validate the end-to-end system behavior in early stages of the design. As a result of this need, there exist many different approaches to system-level performance evaluation at all levels of abstraction. They can be divided into two main classes: analytic methods and simulation-based approaches. Analytic methods are often used in an early design phase as they lead to a fast performance estimation without the necessity of detailed system specifications. They often give rather coarse-grained results because of the high layer of abstraction they are applied to.

* Simon Künzli has been supported through the European Network of Excellence ARTIST2.

On the other hand, simulation-based methods are used on a lower level of abstraction, can combine several layers of abstraction and lead to more accurate results. Simulation techniques exist from low levels as gate-level simulation, over cycle-accurate (e.g. [1]) to high abstraction levels as in trace-based simulation (e.g. [2]). The main drawbacks of simulation-based validation techniques are the high run-time and the difficulty to determine expressive test cases and input loads. The results of a simulation hold only for the workload that has been used to drive the simulation. Designers have to carefully interpret the simulation results to draw conclusions about the system performance, especially if there are critical corner cases in the system behavior. Simulation is widely established for performance evaluation, the techniques are well-known and the results obtained via simulation are trusted by designers. Normally, the average-case behavior of a system can be captured well by simulation-based performance evaluation methods, but no statements about the worst-case behavior (e.g. in terms of end-to-end delay or memory requirement) are possible.

There are two major classes of analytic methods, namely statistical and worst-case approaches. In both cases, the non-functional system properties are formally analyzed for a whole set or a class of input loads. Therefore, it is possible to draw general conclusions about the system behavior in one single analysis, e.g. worst case end-to-end delay or average throughput for a certain system environment. Because of the high level of abstraction and the necessity to follow a formal modeling paradigm, the results of a performance analysis are usually coarse grained.

Given the above, there are important reasons to bridge the gap between formal event models and simulation and/or measurement methods. In particular, there is the need of (a) transferring simulation or measurement results into formal models that capture these traces and (b) to generate representative and expressive event streams given a formal event model. If these methods are available, performance analysis of complex computation and communication systems would benefit in the following respects:

- Given a formal specification of the input loads, e.g. in form of burstiness, period, average rate or statistical behavior, one could generate expressive system loads as well for simulation as for measurements on the actual system. This would help as well in the design phase as in a later validation phase.
- Part of the system or system environment could be simulated or measured and the resulting traces could be transferred into a formal model. These formal event stream specification could be used for an analytic performance analysis of (part of) the system. The resulting specifications of output streams could then be converted into event stream instances for further simulation or measurement studies on the rest of the system.

There exist many different approaches to traffic generation algorithms. Most of them are situated in the networking domain, where the goal is to generate artificial traffic traces that match statistical properties of measured traffic as good as possible. Examples for this approach and references for further reading

can be found in [3] or [4]. The approaches presented in [5] and [6] are closer related to ours as they are used in the embedded system domain. In the approach presented in [5], real application traces are recorded and then played back for simulation trace generation. In [6], traces for performance evaluation are generated at random using a uniform distribution or a two-state Markov chain, or also based on collected traces.

In this paper we are concerned with a completely different and orthogonal way of describing properties of event streams, namely arrival curves, see e.g. [7, 8]. This characterization enables to specify classes of event streams on different time scales. In particular, statements about the burstiness of the stream and its average behavior can be made. Moreover, the specification of event streams by arrival curves is a major prerequisite for the worst-case analysis of computer and communication systems in terms of end-to-end delay and memory requirements, see e.g. [8–10]. In addition, arrival curves are often used to specify formally the behavior of an environment to which the system is exposed. For example, it is possible to specify classical models such as periodic event streams, periodic with jitter and bursty behavior. In addition, it is equivalent to conventional service level agreements (SLA) where a client and a system supplier agree on a T-Spec [11] arrival curve describing the traffic which the networking device has to be able to support.

In the following, we inspect two situations in which the transformation between an abstract description of a workload and a simulator input event trace can be used: (1) In a service level agreement (SLA) a client and a system supplier agree on a T-Spec [11] arrival curve describing the traffic which the networking device has to be able to support. In addition, the designers provide a simulation model for the networking device that should be tested. They are interested whether or not the networking device is capable to support the traffic described with the T-Spec curve. To be able to check this, a simulator input trace has to be generated representing the traffic described with the curve. (2) Figure 1 shows a framework which can be used to measure properties of a networking device, such as packet queuing delays or the maximum supported throughput. The traffic is specified using a formal description, but the networking device under test is measured using a real packet trace. Therefore, we also need a method to generate packet traces based on a formal traffic specification.

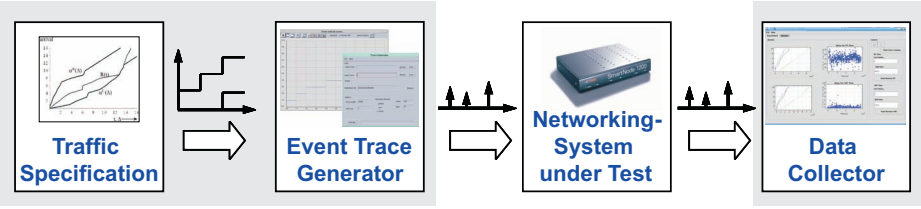

Fig. 1. Block diagram of framework for property-testing of networking systems.

Despite of the importance of arrival curves in terms of requirement specification and performance analysis, there are no methods available that allow to generate representative event streams that respect a given curve. In addition, it would be highly desirable to generate expressive event streams that challenge the subsequent system components by exploiting corner cases. But here, not even a well defined indicator is known that is able to measure the 'expressiveness' of an event stream. The paper contains the following new results:

- Definition of a quality measure that defines the expressiveness of event streams that respect a given arrival curve.
- A set of parameterizable methods for generating expressive event streams that respect a given arrival curve.
- Experimental validation by means of simulations and measurements.

As a result, the event trace generation approach presented in this paper helps to bridge the gap between formal input specifications in terms of service level agreements, analytic methods for worst-case performance analysis, simulation and measurement. In the next section, we will present a modular performance analysis framework as basis for the work presented here. Our contribution, a method for event trace generation and a quality indicator for the generated traces is presented in Sect. 3. In Sect. 4 we discuss implementation issues and present experimental results.

2 Arrival Curves

This section introduces shortly the well known concept of arrival curves which are used for requirement specification and analytic system analysis.

Definition 1. *The upper and lower arrival curves* $\alpha^u_E(\Delta)$ *and* $\alpha^l_E(\Delta)$ *give upper and lower bounds on the number of events that can be observed in any interval of length* Δ *in the event stream* E*.*

Therefore, the upper and lower arrival curves of an event stream bound the minimal and maximal number of events that can be observed in any interval of a specified length Δ. As the curves describe this behavior for all window sizes Δ, the characteristics of a class of event streams is captured on different time scales, e.g. burstiness (short term) and average rate (long term).

Arrival curves were initially used in the networking domain, see e.g. [7],[8]. The curves give a compact representation for a whole class of event streams. For example, the arrival curve on the right hand side of Fig. 2 represents not only the event trace on the left hand side but also an infinite number of 'similar' traces, i.e. traces that are compliant with the arrival curve according to Def. 1.

Recently, Chakraborty et al. have introduced a general framework for performance evaluation of embedded systems. The work presented in [8] was extended by means of introducing also a lower arrival curve, which in addition to the upper arrival curve gives a lower bound on the number of events in time intervals.

The framework is described in more detail in [10, 12]. Using the framework, we can combine different analytic components to an analytic model of the system in the same way, as we can compose different architectural blocks to a whole system. The framework has successfully been used for design space exploration of network processor architectures [13] and to derive optimal scheduling parameters for multimedia processors [14]. The performance analysis method is based on event models to characterize event streams and on resource models to characterize the processing units that are available to process these event streams. The event streams are described using the concept of arrival curves as given in Definition 1.

Arrival curves can be obtained in different ways. First, they can be computed from an existing event trace or a set of traces. According to Figure 2, we can traverse the event trace(s) with windows of different sizes Δ and keep track of the maximum and minimum number of events seen in this window. The value found for the maximum results in $\alpha^u(\Delta)$, the minimum leads to $\alpha^l(\Delta)$. The traces used can be the result of measurements or simulations. Note that the final arrival curves not only represent the traces that have been used for their construction but an infinite number of compliant event traces, too.

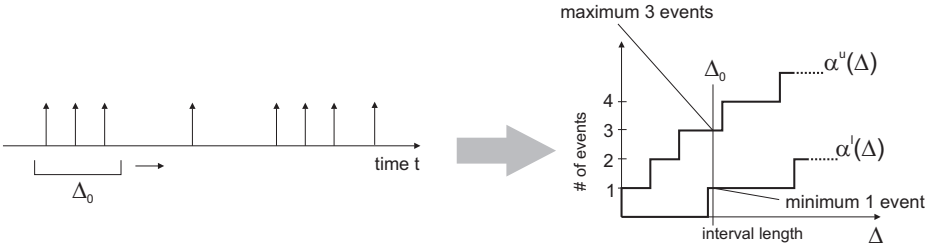

Fig. 2. Example event trace with corresponding upper and lower arrival curves.

Second, arrival curves are also used for contracts in the telecommunications domain. In service level agreements (SLA), the traffic allowed for a client to generate on a server is often characterized using T-Spec curves [11], which are just a special case of arrival curves. In a similar way, arrival curves may be constructed from known properties of the generating process, such as periodic with jitter or a certain degree of burstiness, see e.g. Fig. 3.

Arrival curves are defined for all $0 \leq \Delta \leq \infty$. In order to allow for a finite representation, we introduce a specification size W, up to which an arrival curve is defined. For arguments larger than W, arrival curves are periodically extended:

$$\alpha(\Delta) = \left\lfloor \frac{\Delta}{W} \right\rfloor \alpha(W) + \alpha(\Delta - \left\lfloor \frac{\Delta}{W} \right\rfloor W)$$

As has been described in the introduction, analytic methods are often not sufficient for a thorough performance analysis in the networking and embedded sys-

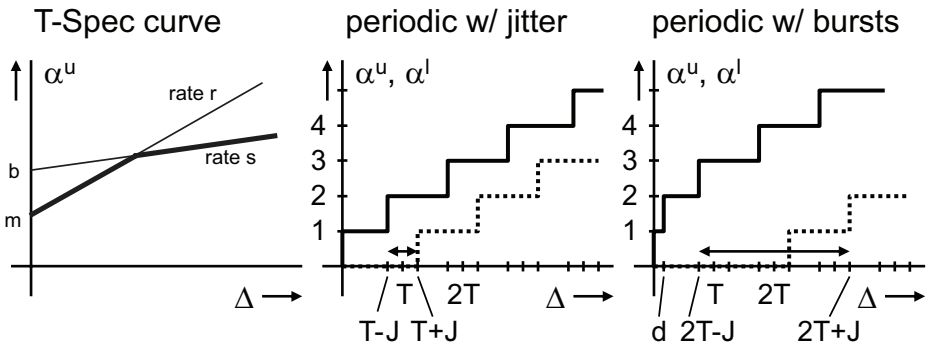

Fig. 3. T-Spec curve with burst rate r, long term rate s, maximum packet size m and burst b (left). Example upper and lower arrival curves for a periodic event stream with jitter (middle), and a periodic event stream with bursts (right).

tem domain. Therefore, a set of transformations is necessary to transform a formal event stream specification into an event stream for simulation/measurement and vice versa. While computing the arrival curves for an event stream based on a given set of event traces is straightforward, the generation of representative event traces is more involved. In the next section, we will define the requirements for event trace generation and propose a method to generate event traces starting from a specification with an upper and a lower arrival curve.

3 Event Trace Generation

In this section we describe a method to generate a representative and expressive event trace starting from an upper and a lower arrival curve $[\alpha^l(\Delta), \alpha^u(\Delta)]$. Figure 4 shows as an example the first part of a trace that was generated using the approach presented in the remainder of this paper. The upper and lower arrival curves that were used as specification curves for this generation are given in Figure 9 (top,left). In Fig. 4 we can identify bursts (marked with (a)), where the generated trace is as bursty as specified by the upper arrival curve. But we can also identify periods in which only a few events are generated (marked with (b) in the figure). In this case, the generated trace represents the lower specification arrival curve.

The generated event stream must respect the specification following Def. 1. But we also need to define the meaning of 'expressiveness' which is done in the next section.

3.1 Requirements and quality assessment

It is obvious, that the expressiveness of an event stream very much depends on the further use of it. In simulation and measurements, we are interested in corner case behavior. Therefore, it appears to be appropriate to require that the

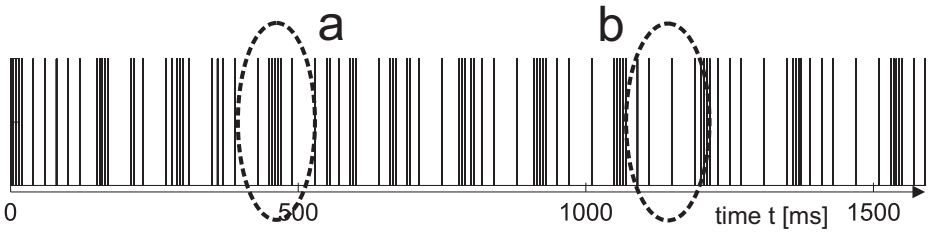

Fig. 4. Generated trace based on the arrival curves for specification 1 in Figure 9.

event stream follows as well the short term characteristics (bursts), i.e. $\alpha^l(\Delta)$ and $\alpha^u(\Delta)$ for small values of Δ as well as the long term characteristics (average case) for large values of Δ. In this sense, the generated trace should show "fractal" or self-similar behavior [15], i.e. in a short observation interval, the trace should be as bursty as allowed by the upper and the lower specification curve in a short time interval — whereas for a larger observation interval, the trace should represent the whole upper and lower specification curves.

In addition, as the behavior of different event streams interact on the system under observation, we should require that we find the characteristics of the arrival curves everywhere in the stream. For example, we would expect that bursts or silent intervals occur frequently in the generated event trace. Therefore, we are looking for a new quality indicator I that covers the property that the multi-scale representation of arrival curves can be seen frequently in the generated event stream.

To obtain this indicator value we first calculate the probability P_τ that the measured arrival curves of an arbitrarily selected trace snippet of length τ match the specified curves $[\alpha^l(\Delta), \alpha^u(\Delta)]$ for all $0 \leq \Delta \leq \frac{\tau}{2}$.

To compute this probability we have to traverse the full trace using a sliding window of size τ, then calculate the upper and lower arrival curves from the trace snippet seen in the sliding window and finally check whether these curves $[\alpha_c^l(\Delta), \alpha_c^u(\Delta)]$ equal the given curves $[\alpha^l(\Delta), \alpha^u(\Delta)]$ for all $0 \leq \Delta \leq \frac{\tau}{2}$.

We calculate the probability P_τ for all values of τ up to some observation window size L which can be chosen arbitrarily. The indicator I is then the minimum over all $\tau \leq L$ of these probabilities P_τ as we want to represent the arrival curves at all scales τ.

Formally, we can compute this indicator I with the following steps:

1. Select all trace snippets T_i of length τ in trace T.
2. Compute the upper and lower curve $[\alpha_c^l, \alpha_c^u]$ from each trace snippet T_i.
3. Set $Z(T_i) = 1$ if $\alpha_c^u(\Delta) = \alpha^u(\Delta)$ and $\alpha_c^l(\Delta) = \alpha^l(\Delta)$ for all $0 \leq \Delta \leq \frac{\tau}{2}$ and $Z(T_i) = 0$ otherwise.
4. Compute the probability $P_\tau = \frac{1}{N} \sum_{T_i \in T} Z(T_i)$ where N denotes the number of considered trace snippets T_i in trace T.
5. Set $I = \min_{\forall \tau \leq L} P_\tau$.

The larger the indicator value I, the better the trace exposes the desired fractal behavior. For the trace generation algorithm presented next we intend to maximize the indicator value I in order to generate a trace that represents the specification in a any short observation interval as well as in any large observation interval.

3.2 General event trace generation

The main idea underlying the proposed event trace generation algorithm is to use a ON/OFF traffic source as shown in Fig. 5. The traffic generator has two distinct states: ON and OFF. In existing traffic generation approaches (see e.g. [16],[17]) events are generated in the ON state, and in the OFF state no events are generated at all. In contrary to the classical approach, we generate events that conform with the upper specification arrival curve, if the generator is in the ON state. In this case, if we would compute the upper arrival curve for the generated trace, the upper specification curve would be obtained. In other words the generator creates events as soon as it is allowed by the upper specification curve. Similarly, the generator in the OFF state is as "lazy" as allowed by the lower curve, i.e. it generates an event only if the lower specification arrival curve would be violated otherwise. We also say that the generator "follows" the upper curve while it is in the ON state and "follows" the lower curve in the OFF state.

The basic event trace generation algorithm consists of three main steps:

1. Determine time stamp T at which to switch between ON- and OFF-states.
2. Generate events according to the state while $0 \leq t < T$.
3. If $t = T$ then switch the state, set $t = 0$, and go to step 1.

The time t denotes the time spent in a state. In Fig. 5(bottom) an event trace generated with this simple algorithm is given, the grey boxes denote the state of the generator.

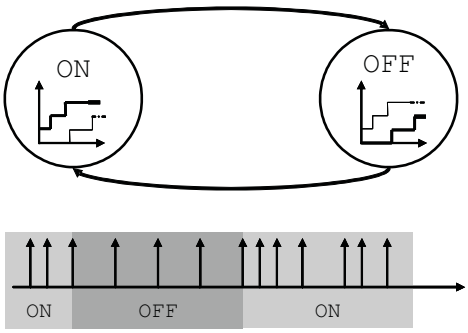

Fig. 5. (top) ON/OFF-automaton used for trace generator. (bottom) Example for a generated trace with boxes indicating whether generator in ON- (light grey) or OFF-state (dark grey).

The choice of the times at which the traffic source switches between states influences the generated trace and as a consequence also the indicator value I for the generated trace. See Figure 6 for two different traces that are based on the same specification arrival curves. The only difference for these traces are the distributions of the switching times between ON and OFF state of the generator. The second trace is more regular, e.g. we can observe the event pattern between 5000 and 8000 ms, during which no state switches occur at all.

We will next look at a simple example which will lead us to a deterministic algorithm to determine the state switching time. Then, we will randomize the algorithm to make the generated trace less predictable.

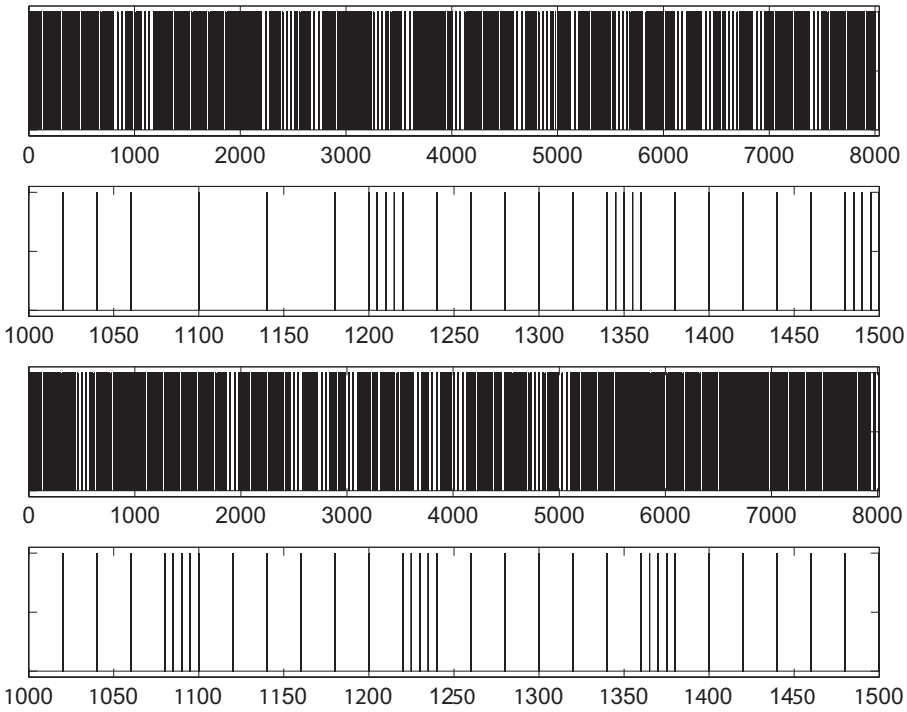

Fig. 6. Two generated traces (plots 1 and 3 from top) based on the same specification. They were generated using a different switching time estimation technique. Plots 2 and 4 from top are magnified snippets of the traces (the traces are shown between 1000 and 1500 ms).

Assume the upper and lower arrival curve specified in Fig. 7(top). For simplicity we only look at three different intervals of length A, B and C with $A = 2 \times B = 4C$. For the example, we use the ON/OFF-automaton from Fig. 5, but we restrict ourselves to switches between states after A, B or C time units only. If

we are in the ON state for A time units, we denote this by $A \uparrow$, if we are in the OFF state for B time units, we denote this by $B \downarrow$ accordingly.

Furthermore, we assume that after a state change we generate events according to the specification curve, i.e. if we are in the ON state for C time units, we can generate an event sequence that reproduces an upper arrival curve equal to the specification curve up to C time units as shown in Fig. 7(top).

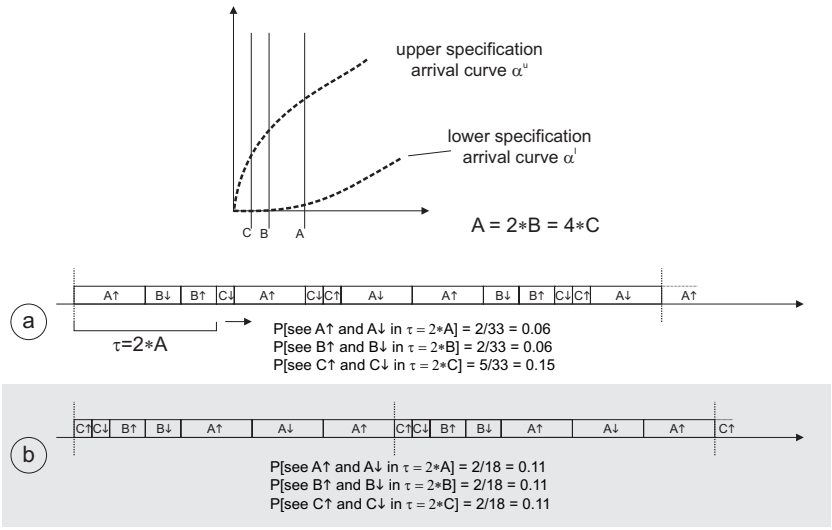

Fig. 7. Example for upper and lower specification arrival curves and two examples for generated traces with probabilities corresponding to the intervals A, B and C. (a) randomly generated trace and (b) deterministically generated trace. The dotted lines give the limits after which the generation pattern is repeated in the examples.

In the example, we check the match between the specification curve and the computed arrival curve at discrete times and the step size is set to C. In this setting, the probability to see the upper and lower specification curve in interval $2C$ is the number of times we can see $C \uparrow$ followed by $C \downarrow$ (or vice versa) divided by the total number of checks performed. As we want to achieve a large indicator value I, we have to ensure that for all the different interval sizes (A, B, C) the probability to see the corresponding upper and lower curve is maximized. In other words, we have to generate at least one trace snippet for the upper and the lower specification curve for each of the interval sizes in order to achieve non-zero probabilities for all the intervals. Because we do not know a priori the length of the generated trace, we want to generate all possible trace snippets as soon as possible.

The specification curves shown in Fig. 7(top) reveal that the upper curve of length C is contained in the upper curve of length B. This fact helps us to further improve the value of the indicator I: We start the trace generation in

the ON state, set the dwell time to the smallest possible interval and generate a trace snippet according to the upper specification curve. In the example, we therefore first generate a trace snippet $C \uparrow$. We then switch state and remain in the OFF state for again C time units, generating a trace snippet $C \downarrow$. Next, we increase the dwell time by the step size, thus in the example we generate a trace snippet $B \uparrow$. We again switch state to OFF, generate the corresponding trace snippet, switch to ON state, increase the dwell time, switch back to OFF state, and so on. Like this, we can not only see the upper and lower specification curve in the the sequence $C \uparrow C \downarrow$, but also in the sequence $C \downarrow B \uparrow$, as the upper curve of length C is contained in the upper curve of length B.

Having the above considerations in mind, we can propose a deterministic algorithm that generates an event trace that leads to a large indicator value I:

1. Set next switching time $T = 0$
2. Increase T by the chosen step size.
3. Generate events according to ON state while $t < T$.
4. If $t = T$ then switch to OFF state, set $t = 0$, and generate events while $t < T$.
5. If T exceeds window size L, go to 1.
6. Switch to ON state, set $t = 0$, and go to 2.

The above event generation optimizes two conflicting goals: (1) all the intervals should be generated as often as possible to increase the individual probability P_τ and (2) all the intervals should be generated as soon as possible, because we have no a priori knowledge about the desired length of the trace.

In Fig. 7(bottom,a) the example of a randomly generated trace is given. Below the probabilities to detect the upper and lower specification curve for A, B or C are depicted. In part (b) of the figure, we show a trace that was generated according to the deterministic algorithm. Again the probabilities for the interval sizes are given. Note that the largest interval (in the example A) has to be generated 3 times in order to equalize the probabilities P_τ.

The algorithm shown above generates a deterministic trace. To avoid this problem the deterministic algorithm is randomized. Instead of stepwise increasing the dwell times in the generator states deterministically, we determine the dwell times randomly, according to a uniform distribution of the dwell times between the minimum interval size and the window size L. With this procedure, the event trace becomes unpredictable and especially multiple event traces are independent and uncorrelated.

4 Implementation

4.1 Trace Generation

In this section we present an implementation of the algorithms described in Sect. 3. As discussed above, event traces can be generated starting from arrival curves using an ON/OFF state machine as traffic generator. The times at which

we switch from ON to OFF state and vice versa can be determined at random or deterministically.

We can use arbitrary specification arrival curves for the event generator. Of course, the specification curves have to represent a valid characterization, i.e. it must be possible to find an event stream that complies with both the upper and the lower specification curve. The only additional constraining factor for the curves is the memory demand. To keep the needed memory bounded, we use the periodically extended finite representation introduced in Section 2.

Algorithm 1 describes our general approach to generate event traces starting from upper and lower arrival curves $[\alpha^l(\Delta), \alpha^u(\Delta)]$. The algorithm contains calls to several functions which are further described in Tab. 1.

Algorithm 1 Algorithm for trace generation in pseudo-code

```
/* initialize variables */
t = 0;
generate = false;
state = 0;
swt = getNextSwitchingTime(t);
/* generate event at time t */
generateEvent(t);
while (!stopGeneration) {
  while ( t < swt ) {
    if (state == 0) {
      if ( canIGenerateNow(t) )
        generate = true;
    }
    else{
      if ( !canIStillWait(t) )
        generate = true;
    }
    if (generate) {
      /* generate event at time t */
      generateEvent(t);
      updateHistoryWithEvent(t);
    }
    t = t + timeStep;
    generate = false;
  }
  swt = getNextSwitchingTime(t);
  state = (state + 1) mod 2;
}
```

In the implementation, the algorithm to determine the next switching time between ON and OFF state is the only component that can be freely chosen by the user, whereas the generation pattern in the ON and the OFF state is

fixed by the upper and lower specification arrival curves. We have implemented the deterministic switching time generator as proposed in Sect. 3 and two randomized versions. One is based on a uniform distribution of the switching times between an upper and lower bound that can be set as a parameter. The generator with a uniform distribution represents a randomized version of the deterministic algorithm. The other one is based on a Weibull-distribution. We chose the Weibull-distribution here, because it is often used for traffic generation (e.g. in [16] and [17]). We use it in our experiments to have control runs that we can compare with the other runs. The Weibull probability distribution function of a random variable X,

$$P\{X \leq x\} = 1 - e^{-(x/\beta)^{\alpha}}, x \geq 0$$

has two parameters $\alpha > 0$ and $\beta > 0$.

Name	**Description**
`getNextSwitchingTime(`t`)`	Determines the next point in time based on the time stamp t at which the state of the generation mode should be changed.
`generateEvent(`t`)`	Generates event at time t
`canIGenerateNow(`t`)`	Returns a Boolean value denoting whether it is possible to generate event at time t without violating the specification curves.
`canIStillWait(`t`)`	Returns a Boolean value denoting whether it is possible to wait for `timeStep` time units after t with the event generation without violating the specification curves.
`updateHistoryWithEvent(`t`)`	Update the history of already generated events. Add the newest time stamp t and remove the oldest time stamp if the size of the history is > `WindowSize`.

Table 1. Functions used in Alg. 1

Note the difference between the event trace generation presented in Sect. 3 and the implementation. Although we stated that in the ON state, we will generate events as greedy as allowed by the upper arrival curve, this is not always possible without violating the specification $[\alpha^l(\Delta), \alpha^u(\Delta)]$.

Assume that the generator was in the ON state for some time t_0, the generated trace snippet represents the upper specification curve. In other words, the upper curve derived from the generated trace snippet matches the upper specification curve. Then, we switch for a very short time to the OFF state, and

do not generate events at all. After coming back to the ON state, we are not necessarily allowed to again generate events as specified in the upper curve up to t_0. This is because we then would generate the maximum number of events allowed by the specification curves in t_0 twice within an interval of length $2t_0$. This could potentially violate the specification. See Fig. 8 for an example. To solve this problem, we use in the algorithm the guard functions `CanIStillWait()` and `CanIGenerateNow()`. They ensure that a generated event does neither violate the upper nor the lower specification curve at any time.

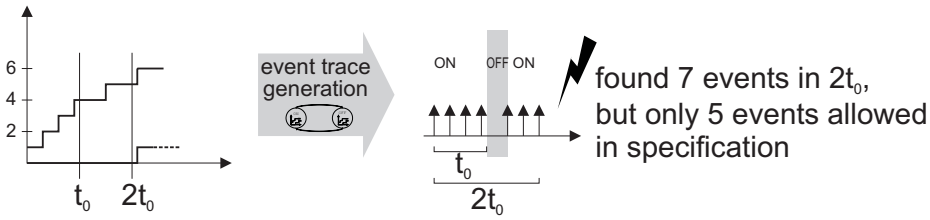

Fig. 8. Example in which greedy event generation leads to violation of the upper specification arrival curve.

4.2 Results

To compare different switching time estimation algorithms, we generated traces of 100'000 events each for different switching strategies. The results of the comparison are shown in Fig. 9. On the y-axis the indicator values I are given. To be able to compare the different algorithms for two different specifications, we use the indicator values achieved for the deterministic algorithm as a base line and give the relative indicator values for the other algorithms. On the x-axis, we show the result bars for different switching time algorithms.

We have performed the tests for two different pairs of specification curves leading to two bars for all different switching time algorithms. The parameters corresponding to the different switching time estimation algorithms are specified in Tab. 2. For the Weibull distribution, α was fixed and the parameter β was then calculated such that the expectation value of the dwell time was as desired.

The deterministic switching time algorithm (left-most bars) achieves the highest indicator value, as expected. It leads to the event trace that represents well the desired worst-case behavior. The randomized algorithms all perform worse than the deterministic algorithm. Nevertheless, the randomized algorithm using switching times uniformly distributed between 0 and $2L$, achieves also reasonably good results (within 65 % of the deterministic algorithm on average), while being non-predictable. Weibull-distributed switching times lead to worse indicator values and should not be used for our event trace generator as a consequence. The indicator value I for a uniformly distributed switching time with

expectation $\frac{L}{2}$ varies much between the two specifications (cf. Fig. 9, column 10). The reason for this is that intervals of length L are hardly generated using this switching strategy and that the specifications consist of two different arrival curve pairs, where specification 1 is more regular and therefore easier to fulfill than specification 2.

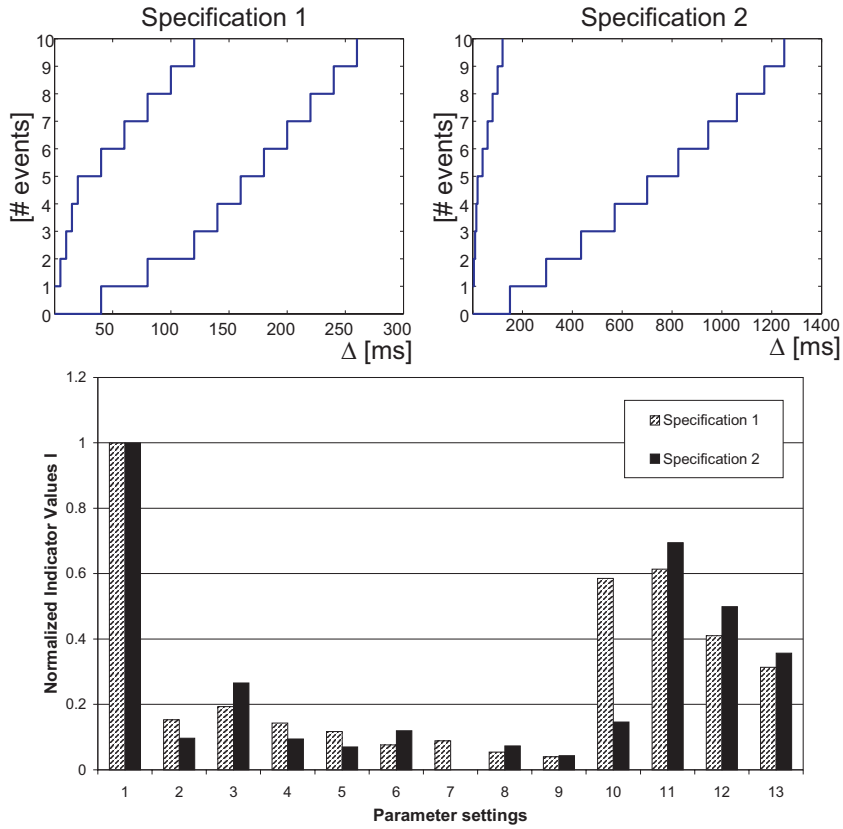

Fig. 9. Normalized indicator values I for different switching strategies between ON and OFF state specified in Tab. 2

4.3 Application for Measurement System

The event generation has been applied to an industrial case study. In particular, a tool chain for validating a complex packet processing device with several quality of service classes was implemented. The framework consists of an arrival curve based event trace generator as presented in this paper, a packet generator that generates IP (internet protocol) packets at exactly the times specified in the event

Nr.	Switching Time Determination
1	Deterministic algorithm as presented in Sect. 3 up to window size L
2	Weibull-distributed with expectation $\frac{L}{2}$ and $\alpha = 0.5$
3	Weibull-distributed with expectation L and $\alpha = 0.5$
4	Weibull-distributed with expectation $2L$ and $\alpha = 0.5$
5	Weibull-distributed with expectation $3L$ and $\alpha = 0.5$
6	Weibull-distributed with expectation $\frac{L}{2}$ and $\alpha = 0.3$
7	Weibull-distributed with expectation L and $\alpha = 0.3$
8	Weibull-distributed with expectation $2L$ and $\alpha = 0.3$
9	Weibull-distributed with expectation $3L$ and $\alpha = 0.3$
10	Uniformly distributed with expectation $\frac{L}{2}$
11	Uniformly distributed with expectation L
12	Uniformly distributed with expectation $2L$
13	Uniformly distributed with expectation $3L$

Table 2. Parameters for the different switching strategies for the event trace generator.

trace, and finally a data collector tool that collects the packets that have been processed by the system under test. In addition, a high precision synchronization device allowed us to measure end-to-end delays of the system up to a precision of 1μs.

The graphical components of the tool chain have been implemented in Java, the packet generator and the data collector tool as Linux kernel modules in order to achieve good timing accuracy.

The block diagram of the tool chain is shown in Fig. 1, a screenshot of the data collector tool is given in Fig. 10. In Fig. 10, the top row shows the measured arrival curve at the receiver (left) and the measured per packet delay (right) for a real-time packet flow (RT), the bottom row for a flow with no real-time requirements (NRT). Note that the maximum delay measured for the RT flow is 95μs, whereas for the NRT flow the maximum delay is ca. 60 ms. With this framework, we have a tool chain at hand that can be used to validate system properties such as packet throughput or processing delays, starting from a formal specification of the system input with arrival curves.

5 Conclusions

The event trace generation algorithms presented in this paper allow to bridge the gap between simulation/measurement and analytic methods for performance evaluation, as they define a way to translate a formal workload specification with arrival curves into a representative event trace. The proposed quality indicator measures the expressiveness of a generated trace represents. To show the applicability of the presented approach we (1) implemented the proposed event trace generation algorithms, (2) compared different generation algorithms, and (3) used the generated traces in a performance measurement framework depicted in Fig. 1.

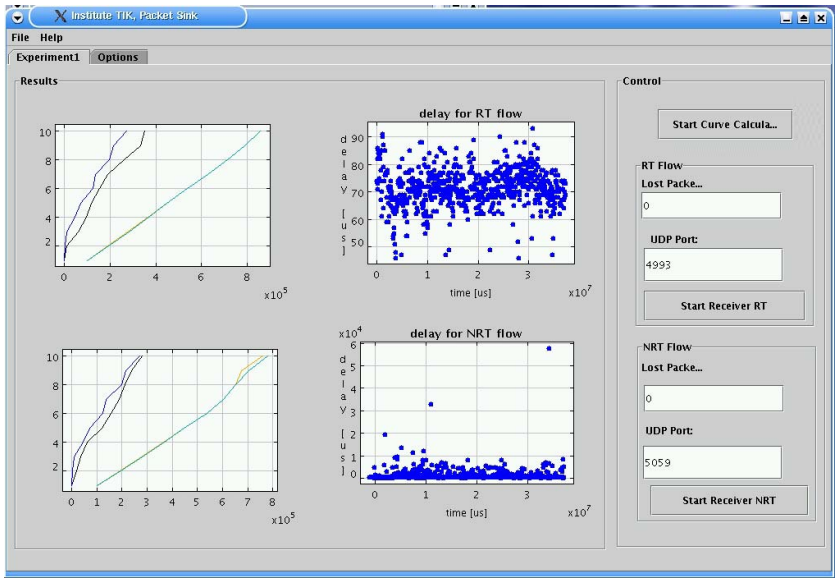

Fig. 10. Screenshot of the measurement tool.

References

1. Loghi, M., Angiolini, F., Bertozzi, D., Benini, L., Zafalon, R.: Analyzing on-chip communication in a MPSoC environment. In: Proceedings of the Design, Automation and Test in Europe Conference and Exhibition Volume II (DATE'04), IEEE Computer Society (2004) 752–757
2. Lieverse, P., Stefanov, T., van der Wolf, P., Deprettere, E.: System level design with spade: an m-jpeg case study. In: ICCAD '01: Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design, IEEE Press (2001) 31–38
3. Sommers, J., Barford, P.: Self-configuring network traffic generation. In: IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement, ACM Press (2004) 68–81
4. Emma, D., Pescape, A., Ventre, G.: Analysis and experimentation of an open distributed platform for synthetic traffic generation. In: FTDCS '04: Proceedings of the 10th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS'04), IEEE Computer Society (2004) 277–283
5. Mahadevan, S., Angiolini, F., Storgaard, M., Olsen, R.G., Sparso, J., Madsen, J.: A network traffic generator model for fast network-on-chip simulation. In: DATE '05: Proceedings of the Design, Automation and Test in Europe (DATE'05) Volume 2, IEEE Computer Society (2005) 780–785
6. Genko, N., Atienza, D., Micheli, G.D., Mendias, J.M., Hermida, R., Catthoor, F.: A complete network-on-chip emulation framework. In: DATE '05: Proceedings of the Design, Automation and Test in Europe (DATE'05) Volume 1, IEEE Computer Society (2005) 246–251
7. Cruz, R.: A calculus for network delay, part I: Network elements in isolation. IEEE Trans. on Information Theory **37** (1991) 114–131

8. Boudec, J.L., Thiran, P.: Network Calculus - A Theory of Deterministic Queuing Systems for the Internet. LNCS 2050, Springer Verlag (2001)
9. Chakraborty, S., Künzli, S., Thiele, L., Herkersdorf, A., Sagmeister, P.: Performance evaluation of network processor architectures: Combining simulation with analytical estimation. Computer Networks **41** (2003) 641–665
10. Chakraborty, S., Künzli, S., Thiele, L.: A general framework for analysing system properties in platform-based embedded system designs. In: Proc. 6th Design, Automation and Test in Europe (DATE), Munich, Germany (2003) 190–195
11. Wroclawski, J.: Specification of the controlled-load network element service (standard track rfc 2211) (1997)
12. Maxiaguine, A., Künzli, S., Thiele, L.: Workload characterization model for tasks with variable execution demand. In: Proc. 7th Design, Automation and Test in Europe (DATE), Paris, France (2004) 1040–1045
13. Thiele, L., Chakraborty, S., Gries, M., Künzli, S.: Design space exploration of network processor architectures. In Crowley, P., Franklin, M.A., Hadimioglu, H., Onufryk, P.Z., eds.: Network Processor Design: Issues and Practices. Volume 1. Morgan Kaufmann Publishers, San Francisco, CA, USA (2002) 55–90 A preliminary version of this paper appeared in the Proc. 1st Workshop on Network Processors, held in conjunction with the 8th International Symposium on High-Performance Computer Architecture, Cambridge, Massachusetts, 2002.
14. Maxiaguine, A., Zhu, Y., Chakraborty, S., Wong, W.F.: Tuning soc platforms for multimedia processing: identifying limits and tradeoffs. In: CODES+ISSS '04: Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, ACM Press (2004) 128–133
15. Leland, W.E., Taqqu, M.S., Willinger, W., Wilson, D.V.: On the self-similar nature of Ethernet traffic. In Sidhu, D.P., ed.: ACM SIGCOMM, San Francisco, California (1993) 183–193
16. Anastasi, G., Lenzini, L., Mingozzi, E.: Stability and performance analysis of hiperlan. In: Proceedings of the IEEE Conference on Computer Communications (INFOCOM '98). (1998)
17. Barford, P., Crovella, M.: Generating representative web workloads for network and server performance evaluation. In: Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems, ACM Press (1998) 151–160