Performance Analysis of Asynchronous Circuits and Systems using Stochastic Timed Petri Nets

Aiguo Xie and Peter A. Beerel

Department of Electrical Engineering – Systems
University of Southern California, Los Angeles, CA 90089-2562
{aiguo, pabeerel}@eiger.usc.edu

Abstract. This paper describes and extends a recently developed approach for performance analysis of asynchronous circuits modeled with stochastic timed Petri nets (STPNs) with unique- and free-choice places and arbitrary delay distributions. The approach analyzes finite STPN executions to derive closed-form expressions for lower and upper bounds on the performance estimates that can be efficiently evaluated using standard statistical methods. The mean of the derived upper and lower bounds thus provides an estimate of the performance metric which has a well-defined error interval. Moreover, we can often make the error interval arbitrarily small by analyzing longer STPN executions at the cost of additional run-time. Experiments on several asynchronous systems demonstrate the high quality of our estimates and the efficiency of the technique. The experiments include the performance analysis of a full-scale Petri net model of Intel's asynchronous instruction length decoding and steering unit RAPPID containing over 900 transitions and 500 places.

1 Introduction

In hardware, an algorithm can be implemented using asynchronous circuits or synchronous circuits or some combination of the two. Synchronous circuits use a single global clock to sequence the operation of the datapath whereas asynchronous circuits use completion sensing logic to determine when a particular datapath unit is done and communicating control circuits to sequence the activation of different datapath units. Synchronous design techniques are by far the predominant methodology used today because the technology is mature and has significant computer-aided design tools to support it. Asynchronous circuits, however, can *sometimes* achieve significantly higher performance than synchronous counterparts because 1) under favorable input data distributions, well-designed asynchronous datapath units, such as adders and multipliers, can provide better performance 2) asynchronous micro-architectures provide finer-grain flexibility to optimize typical datapath operation sequences and 3) asynchronous circuits automatically adapt to changes in the input data rate, varying voltage supply levels, and chip temperatures [27, 41]. One dramatic design in

	Exact	Bounds
Analytical	√	√
Simulation/ Random Sampling	√	×

Fig. 1: A classification of performance analysis techniques for timed Petri nets.

which these potential advantages have been realized is an asynchronous x86 instruction length decoder which is 3X faster than the comparable synchronous state of the art [31].

The performance of any asynchronous micro-architecture depends on three factors: the delay distribution of individual datapath units, the overhead of completion sensing and control overhead, and the micro-architecture and control flow of the design. Optimizing the degree of pipelining and the extent of control synchronization in the micro-architecture is critical to obtaining high performance and is particularly challenging because it often trades off control complexity with synchronization performance bottlenecks. Currently, asynchronous designers typically rely on slow simulation of detailed designs (which can be performed only late in the design cycle) or on ad-hoc back-of-the-envelope analysis of high-level architectures (see e.g., [31]). This paper describes a recently developed automated technique aimed at helping better guide architectural design early in the design cycle and optimize control circuits for average performance later in the design cycle.

1.1 Previous work

Let us first review related efforts on performance analysis. To aid exposition, we classify existing work into four categories depending on the solution methods and the target quantities, as shown in Fig. 1. Within each category, works vary on the expressiveness and size of system models that can be analyzed.

Let us first consider techniques that use analytical methods to obtain exact measures of performance. In [30], Ramamoorthy et al used Karp's theorem to determine the average throughput of a timed marked graph with deterministic firing delays in polynomial time. In similar work, Burns [10] modeled asynchronous circuits using an event-rule system similar to marked graphs with fixed delays and used linear programming to find the average cycle time of events. Lee [24] extended Burns' work to handle OR-causality. Although computationally efficient, these techniques are severely limited in applicability because they cannot handle systems with choice and must model component delays using their mean values.

To handle choice, Kudva et al [23] proposed adopting Generalized Timed Petri Nets (GTPNs) [20] to model asynchronous circuits in which component delays are again modeled using their mean values. As usual, the model is then converted into a continues time Markov chain (CTMC) and solved for its stationary probability distribution. In other Petri-net based approaches, the delays are assumed to be either fixed (possibly zero) or exponentially distributed and the analytical tool is either a discrete or continuous time Markov chain (see e.g., [26] for more information). In addition to the restricted delay modeling, the size of the models that can be analyzed is limited by the state explosion problem.

More recently, Xie and Beerel [35, 34] proposed a discretized-time model that can handle arbitrarily distributed delays. The model is converted into a discrete-time Markov chain (DTMC) and solved for its stationary distribution. To mitigate the state-explosion problem, they developed a technique called state-compression [34] to speed up the symbolic computation of the stationary distribution of their DTMC models. Specifically, they reduced the state space of the DTMC to one of the feedback vertex sets of its state space. Other efforts to mitigate the state explosion problem include Buchholz's work on (exact and weak) lumpability [7–9]. Nevertheless, even with these advances, the state explosion problem has limited Markov chain based approaches to small models.

To circumvent the state explosion problem, Queuing models have been explored. For example, Greenstreet [17] studied the performance of closed asynchronous ring structures. Unlike Petri nets, however, queuing models are less expressive in specifying synchronization and consequently are applicable only to a relatively small special class of asynchronous circuits. Moreover, there is no simple solution to such models with general delay distributions.

Another approach to circumvent the computational challenges of the above techniques is to analytically compute performance bounds rather than exact performance estimates. For example, Campos et al's work [11] computes throughput bounds for a limited class of timed Petri nets. These bounds are defined by considering all possible delay distributions with fixed means. Unfortunately, these bounds are often too wide to be practically useful. In addition, there is significant research focused on asynchronous system models in which delays are expressed as intervals and bounds on the time separation between system events (TSEs) are obtained. Although typically targeting verification applications, such bounds may serve as upper and lower bounds of average performance metrics that are irrespective of the actual distribution of the delays. For example, Ebergen and Berks [15] studied extreme and amortized response times of asynchronous pipelines with tree structures. In addition, Hulgaard and Burns [21] give extreme TSEs for timed Petri nets with free-choice and/or unique-choice. For more general Petri nets, however, all known techniques must perform a computationally expensive timed state space analysis to compute extreme TSEs (e.g., [1, 2]).

As apposed to the above analytical techniques, several researchers have studied simulation-based approaches to finding exact performance estimates that can more readily be applied to models with general delay distributions. For example, Shelder et al [33] develops performance estimates for regenerative timed Petri nets using independent replicants and Glynn and Iglehart [16] proposed using standardized time series method. For non-regenerative nets with arbitrary delay distributions, Haas [19] recently proposed a technique for performance estimation using standardized timed series. Unfortunately, we are not aware of any

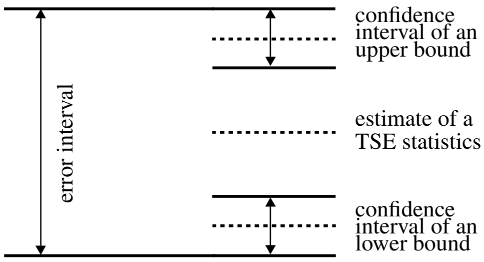


Fig. 2: Relationship among the estimates.

currently available tool which implements this technique and consequently its computational efficiency is unclear.

1.2 Overview of our approach: bounding performance estimates using random sampling

Lastly, we come to our approach in which simulation, or more accurately random sampling, is used to derive performance bounds. In particular, we use a combination of random sampling and longest path analytical analysis to efficiently derive performance bounds. The major contributions of this approach are the following:

- Development of a general framework of performance metrics (e.g., throughput, latency, response time) based on a unifying notion of time separation of events (TSEs).
- Definition of TSE statistics (i.e., average, variance, and distribution of TSEs) in stochastic timed Petri nets with unique- and free-choice and arbitrary distributed delays based on an infinitely long random simulation.
- Characterization of a large class of transition pairs for which these TSE statistics exist.
- Development of closed-form expressions for upper and lower bounds on these TSE statistics.
- Explanation of how our upper and lower bounds can often be made arbitrarily close to each other using more complicated expressions at the expense of additional run-time.
- Demonstration of how these closed-form expressions can be efficiently computed with arbitrary high accuracy and confidence using simple statistical methods. Consequently, the mean of the TSE bounds is an estimate of the TSE statistic with a probabilistically defined error interval, as illustrated in Fig. 2.
- Demonstration of the efficiency of the approach in a variety of examples, including a performance analysis of a full-scale stochastic timed Petri model of Intel's asynchronous instruction length decoding and steering unit, RAPPID, (with over 900 transitions) in less than 1.5 hours of CPU time.

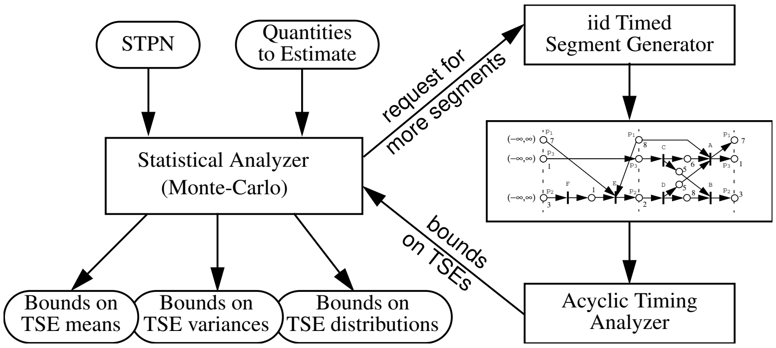


Fig. 3: Overview of UscPet.

The theoretical foundation to derive bounds on TSE statistics is based on a notion of segment. Roughly speaking, a segment is an untimed event graph corresponding to an untimed execution of the STPN that starts and ends with the same marking. For the STPN we are considering, a random time execution of the net can be partitioned into a sequence of independent and identically distributed segments. Within each segment, a lower and an upper bound on the TSE instance is derived using a simple longest path analysis. Consequently, one obtains a sequence of iid samples of the lower and upper bounds which can be statistically analyzed to obtain an unbiased estimate of the bounds. Fig. 3 depicts our fully automated tool that implements this approach.

1.3 Organization of the paper

The remainder of the paper is organized as follows. Section 2 describes the class of STPNs we consider in this paper, its relationship with other STPN models commonly referred to in the literature, and the definitions of TSE statistics. Section 3 formally introduces the concept of segments and their properties, and the algorithms to compute bounds on TSEs within individual segments. Section 4 describes the techniques to obtain bounds on various statistics of TSEs including their averages, variances and distributions. Experimental results are discussed in Section 6. Finally, we conclude the paper in Section 7. More formalized description of some of the key theorems (including proofs) can be found in earlier versions of this work [38, 39].

2 Performance Modeling of Asynchronous Systems using STPNs

2.1 Probabilistic Modeling of Asynchronous Systems

There have been developed many formalisms to specify asynchronous systems. Traditionally, these specifications target synthesis and verification and consequently do not need to be probabilistically quantified. For instance, non-fixed

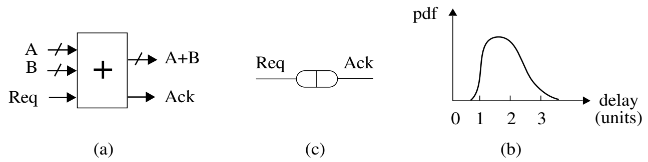


Fig. 4: Probabilistic modeling and time discretization: (a) a data-path component, (b) its delay distribution, (c) abstraction, and (d) time discretization.

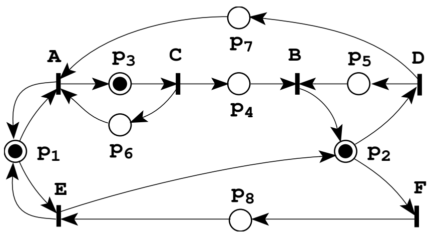
component delays may be specified solely in terms of upper and lower bounds. For some special classes of systems, techniques have been developed to evaluate system performance bounds directly from these specifications, e.g., [21, 15, 10]. However, in order to obtain system characteristics such as average throughput or latency, a system needs to be adequately modeled in a probabilistic sense.

We focus on the derivation of our performance models from an architectural description of the system. A similar derivation is also possible from more abstract behavioral or more detailed gate-level descriptions. At the architectural level, like most hardware systems, an asynchronous system contains interacting data-path components and controllers.

The operation of an asynchronous data-path component is typically sequenced by *go* and *done* signals that emanate from and connect to asynchronous controllers. Consequently, the delay of the done signal determines system performance. In principle, this delay could be modeled as a random variable which depends on the values of the inputs to the component. However, the size of such a model would likely grow exponentially with respect to the bitwidth of the component, making the analysis computationally intractable. In practice, one can often achieve useful performance models by abstracting the delay of a data-path component to a random variable with an estimated probability distribution that takes into account input statistics (e.g., common v.s. rare input combinations) [35, 14] obtained through high-level algorithmic simulation. Consequently, the actual data signals need not be modeled, thereby dramatically simplifying the complexity of the system model. As an example, Fig. 4 (c) plots the distribution of the delay of the adder shown in Fig. 4 (a). In some cases, such an abstraction may lead to significant absolute error in the performance estimates. One reason is that this abstraction does not account for the correlations between the delays of different data-path components. When necessary, however, these correlations can be explicitly accounted for by using extra random variables at the expense of a more complicated system model.

In addition, data-path components often generate signals that dictate subsequent control operations, such as the result of a comparator. These signals can also be modeled as random boolean variables whose distributions can be estimated using, for example, high-level algorithmic simulation (e.g., [14]).

The issues regarding modeling asynchronous controllers are made more obtuse because there is no universal specification language for asynchronous controllers. Regardless of the specification language, however, to model performance



$X(p_1) \sim \texttt{uniform}(5, 10)$,
$X(p_2) \sim \texttt{exp}(1.5)$,
$X(p_3) = 1$,
$X(p_4) \sim 1 + 5\beta(1, 3)$.
...
$\mu(p_2, D) = 0.6$,
$\mu(p_2, F) = 0.4 = 1 - \mu(p_2, D)$.

(a) The Petri net, (b) its stochastic assumption.

Fig. 5: A stochastic Petri net example.

the language must be augmented with delays and their distributions. For example, one performance model for asynchronous FSM controllers specifies the latencies of each state transition as a constant [35]. In order to incorporate the stochastic effects of the variation in chip operating temperatures and/or the manufacturing process, the relevant delays may be modeled probabilistically. Even under constant temperature and nominal process assumptions, some controllers do exhibit non-fixed delays and should be modeled probabilistically. For instance, the delay of an arbiter in metastability may be modeled as an exponential distributed random variable with proper parameters [13]. In addition, the probability distributions of input (environmental) choices can be modeled in the same manner as we model the data-path output signals (such as the result of a comparator) that dictate the controller behavior.

The above probabilistic specifications lead to a performance model consisting of a set of random variables which are related one to another by the system logic. To specify this system logic we use Petri nets which when annotated with delays and choice probabilities are known as stochastic timed Petri nets. Moreover, since most asynchronous system components have bounded delays, it is important to allow the specification of general delay distributions in the stochastic timed Petri net models.

2.2 Petri Net (PN) models

Because Petri nets have been widely and successfully used to specify asynchronous systems for both synthesis and verification, they are a natural model for performance analysis [40]. They easily capture the key aspects of concurrency, synchronization, and choice that are common in asynchronous circuits. In this section, we review the subclass of Petri nets that our technique can currently analyze. More general information on Petri nets and a description of other variants may be found in the survey paper [29].

A Petri net is a triple $N = (P, T, F)$ where P is the set of places, T the set of transitions, and $F \subseteq (P \times T) \cup (T \times P)$ the flow relation. N is a *marked graph* (also known as *event graph*) if every place has at most one input and output transition. A place p is a *choice* place if $|p\bullet| > 1$. N is *free-choice* (FC) net if $\forall p_1, p_2 \in P$, $p_1\bullet \cap p_2\bullet \neq \emptyset \Rightarrow |p_1\bullet| = |p_2\bullet| = 1$. Equivalently, in a FC net, if two transitions share an input place, they may not have any other input places. A Petri net is *extended free-choice* (EFC) if $\forall p_1, p_2 \in P$, $p_1 \bullet \cap p_2 \bullet \neq \emptyset \Rightarrow p_1 \bullet = p_2 \bullet$. An EFC net can be translated into a FC net [4]. A choice place is asymmetric if it is neither free-choice nor extended free-choice. An asymmetric choice place is *unique-choice* if at most one of its output transitions can be enabled at a time. Otherwise, it is an *arbitration*.

The class of underlying (untimed) nets of the performance models we consider here is the PNs with *free-choice* and/or *unique-choice*. That is, we restrict ourselves to PNs without arbitration choice. Fig. 5 shows an example of such a net, where p_1 is unique-choice and p_2 is free-choice. Note that tokens in p_1 never needs to choose which of its two output transitions to fire since p_6, p_7 and p_8 can never be simultaneously marked.

A marking is a mapping $M : P \rightarrow \{0, 1, 2, \cdots\}$. The number of tokens in place p under marking M is denoted by $M(p)$. A transition t is *enabled* at marking M if all its input places have at least one token, i.e., $M(p) \geq 1, \forall p \in \bullet t$. An enabled transition may fire. The firing of t removes one token from each place in its preset and deposits one token to each place in its poset, leading to a new marking M', denoted by $M[t\rangle M'$. A sequence of transitions $\sigma = t_0 t_1 \cdots t_{m-1}$ is a firing sequence from a marking M_0 iff $M_k[t_k\rangle M_{k+1}$ for $k = 1, \cdots, m-1$. In that case, we write $M_0[\sigma\rangle M_m$ and say σ has a length of m denoted by $|\sigma|$. We write $\boldsymbol{\sigma}$ to be the firing counter vector of σ, indicating the number of times each transition is fired in σ. It is called a *T-invariance* if there is a reachable marking M such that $M[\sigma\rangle M$.

A *marked* Petri net Σ is a tuple (N, M_0), where N is a Petri net and M_0 is its initial marking. The set of reachable markings of Σ is denoted by $R(M_0)$. Σ is *live* iff every transition of N will eventually be enabled from every $M \in R(M_0)$. It is *k-bounded* if no place is ever marked with more than k tokens, i.e., $M(p) \leq k$ for $\forall M \in R(M_0), \forall p \in P$. A 1-bounded net is also called *safe*. A live and bounded (LB) marked Petri net has no source or sink places and no source or sink transitions [3]. Thus, a LB Petri net can be partitioned into a set of strongly connected components each evolving independently of others. Below, we assume the Petri net is strongly connected. In particular, we restrict ourselves to LB Petri nets with free-choice and unique-choice places.

2.3 Timing and choice probabilities

For the purpose of performance analysis, Petri nets have been augmented with many different types of stochastic assumptions. For instance, time may be assigned to places, or transitions, or tokens, largely depending on personal taste.

For asynchronous circuits, we found it most convenient to associate time with places. That is, a token flowing into a place p experiences a random delay associated with p before it is *available* to be consumed by an output transition of p.The actual firing of a transition is assumed to be *instantaneous*. For Petri nets with free or unique choices, these assumptions imply there is no race condition among transitions that are in structural conflict (transitions in the poset of a choice place).

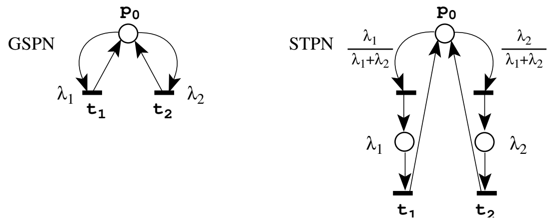


Fig. 6: Translating GSPNs to STPNs.

For every place p, we write $X(p)$ to be the random variable denoting the delay associated with it, and $F_{X(p)} : R \to [0, 1]$ the distribution function of $X(p)$, i.e., $F_{X(p)}(x) = \text{Prob}(X(p) \leq x)$. Unlike some other research in stochastic timed Petri nets, we do not put any restriction on the distribution functions except that they all have finite means and variance, i.e., $\mathbf{E}X(p) < \infty, \mathbf{Var}X(p) < \infty, \forall p \in P$.

To resolve a choice, we assume there is a probability mass function (p.m.f) μ associated with each free-choice place p. That is, if $t \in p\bullet$, $\mu(p, t)$ is the probability that t consumes the token in p each time p is marked. Of course, $\sum_{t \in p\bullet} \mu(p, t) = 1$.

Example 1. Fig. 5.(b) lists a possible stochastic assumption on delay distributions and choice p.m.f. on the Petri net in Fig. 5.(a). For example, the delay on place p_1 is uniformly distributed between 5 and 10, the delay on p_2 is exponentially distributed with a parameter 1.5, the delay on p_3 is 1, the delay on p_4 is β-shaped with a parameter of $(1, 3)$ which is located at 1 and scaled by a factor of 5, and so forth. In addition, with probability 0.6, D fires each time p_2 receives a token, and with the remaining probability, F fires. The other choice place p_1 is not assigned a p.m.f. since it is unique-choice.

One of the most general class of timed Petri nets is the GTTD_SPN where delays may be arbitrarily distributed [26]. A major difference between GTTD_SPNs and our STPN models is in the choice resolution semantics. In GTTD_SPNs, every choice, whether free-choice or asymmetric choice, is resolved through a transition race, i.e., the earliest firable transition in conflict consumes the tokens in the choice place. We call such a process as *dynamic* choice resolution. In contrast, our STPN models only support *static* choice resolution. Although GTTD_SPNs are very general, existing analytical solutions apply only to a subclass of them. GSPNs [25] fall into this subclass where the exponential GTTD_SPNs where delays are either exponentially distributed or zero. It can be shown that a GSPN without arbitration choice can be translated into a performance equivalent STPN of ours as shown in Fig. 6.

2.4 Timed executions

We call a possible run of an STPN a *timed execution* where choices are resolved and places are assigned random delays. In particular, we call a firing of a transition an *event*. A timed execution can be described as a sequence of events and their occurrence times. Alternatively, it can be depicted by a timed event graph which also describes the causality among events.

For example, Fig. 7 shows the event graph of a timed execution of the STPN in Fig. 5. In this execution, D fires after p_2 is marked for the 1st and 3rd times whereas F fires after p_2 is marked for the 2nd time. The numbers along the (instanced) places denote the delay values assigned to them. For convenience, we write $t^{(k)}$ and $p^{(k)}$ to denote the k-th event corresponding to transition t and the k-th instance of a place p, respectively. In the figure, these instance indices are dropped for brevity.

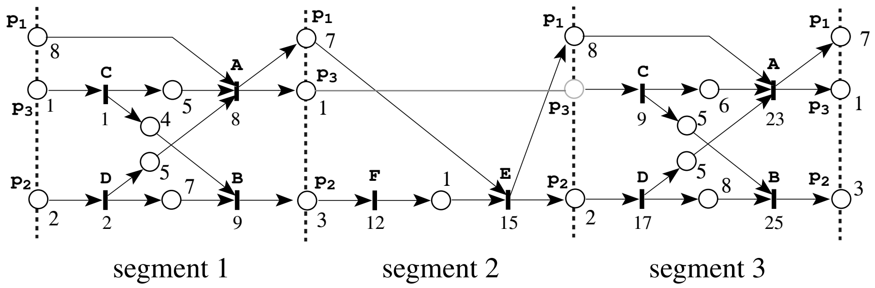


Fig. 7: A timed execution of the stochastic Petri net in Fig. 5.

Formally, a timed execution π of Σ is a triple (N_π, d_π, ℓ) where $N_\pi = (P_\pi, T_\pi, F_\pi)$ is an acyclic event graph (or marked graph), d is a function $P_\pi \to \mathbf{R}$ that denotes the (constant) delay value of each place in P_π and a labeling function $\ell : P_\pi \cup T_\pi \to P \cup T$ that maps each instanced places and transitions to their corresponding ones in Σ. We use a function τ to denote the occurrence times of events. For the timed execution shown in Fig. 7, the corresponding functions d and τ are illustrated along the instanced places and transitions. For a given timed execution, the occurrence time of event $t^{(k)}$ is determined as follows:

$$\tau(t^{(k)}) = \max_{\substack{(s^{(j)}, p) \in F_\pi, \\ p \in \bullet t^{(k)}}} \tau(s^{(j)}) + d(p) \tag{1}$$

where the term $\tau(s^{(j)}) + d(p)$ reduces to $d(p)$ if $\bullet p = \emptyset$ in π, i.e., if p is a source place of π.

Note that for Petri nets with only free-choice and unique-choice, the set of event graph structures resulting from all possible timed executions coincides with that from all possible untimed processes [22, 32]. This fact will be exploited later to decouple the choice decisions from the timing behavior.

2.5 TSEs and their statistics

Given a timed execution π, the Time Separation of an Event pair (TSE) is the time distance between their occurrences. More precisely, the TSE of event pair

$(s^{(k)}, t^{(k+\varepsilon)})$, denoted by, $\gamma^{(k)}(s,t,\varepsilon)$, is:

$$\gamma^{(k)}(s,t,\varepsilon) = \tau(t^{(k+\varepsilon)}) - \tau(s^{(k)}) \tag{2}$$

where τ is the function associated with π, and (s,t,ε) is called the corresponding *separation triple*. When considering π as a random time execution, we view the TSE $\gamma^{(k)}(s,t,\varepsilon)$ as a random variable. Consequently, the TSE sequence $\{\gamma^{(k)}(s,t,\varepsilon) : k = 1, 2, \cdots\}$ is a random process.

Definition 1. *The average TSE due to separation triple (s,t,ε) in a STPN, denoted by $\overline{\gamma}(s,t,\varepsilon)$, is the average of the corresponding TSEs of an infinite timed execution of the STPN. That is,*

$$\overline{\gamma}(s,t,\varepsilon) = \lim_{n\to\infty} \frac{1}{n} \sum_{k=1}^{n} \gamma^{(k)}(s,t,\varepsilon). \tag{3}$$

As we will show in Section 6, many system performance metrics such as average throughput and latency can be directly expressed as the average TSEs of some indicating event pairs, which is much easier than expressing them from the stationary distribution of system states.

Definition 2. *The variance of the TSE due to separation triple (s,t,ε) is:*

$$\sigma^2_{\gamma(s,t,\varepsilon)} = \lim_{n\to\infty} \frac{1}{n} \sum_{k=1}^{n} |\gamma^{(k)}(s,t,\varepsilon) - \overline{\gamma}(s,t,\varepsilon)|^2 \tag{4}$$

Equivalently, $\sigma^2_{\gamma(s,t,\varepsilon)} = \lim_{n\to\infty} \frac{1}{n} \sum_{k=1}^{n} {\gamma^{(k)}}^2(s,t,\varepsilon) - \overline{\gamma}^2(s,t,\varepsilon)$.

Definition 3. *The frequency function of the TSE due to separation triple (s,t,ε) is a mapping $F_{\gamma(s,t,\varepsilon)} : \mathbf{R} \to [0,1]$ such that*

$$F_{\gamma(s,t,\varepsilon)}(x) = \lim_{n\to\infty} \frac{1}{n} \sum_{k=1}^{n} \mathbf{1}_{\gamma^{(k)}(s,t,\varepsilon) \le x} \tag{5}$$

where the indicating function $\mathbf{1}_A$ evaluates to 1 if the subscript expression A is true, and 0 otherwise.

It is the above defined statistics that we want to characterize in this paper. Toward this end, it is natural to first question their existence. There are cases where the average TSE does not exist. For instance, let us consider transitions D and F of the Petri net in Fig. 5. The difference between the numbers of occurrences of D and F will be infinite almost surely as time progresses. Consequently, their average TSE diverges almost surely for any finite occurrence offset ε provided that the net has at least one place with a positive mean delay. However, for a wide class of TSE pairs, their average TSEs exist [38]. Condition 1 below formally characterizes such a class with the aid of the notion of *steady markings*. A steady marking is one that can be reached from all reachable markings. Since the reachability graph of a LB Petri net with only free-choice and unique-choice is irreducible [6], its unique terminal strongly connected component contains all the steady markings of the net.

Condition 1 *If M is a steady marking and σ is a firing sequence such that $M[\sigma\rangle M$, then $\boldsymbol{\sigma}(s) = \boldsymbol{\sigma}(t)$.*

The condition requires that the net fires both transitions the same amount of times in order to traverse any cycle of steady markings. In practice, we expect this condition to be guaranteed by the user. However, it can also be checked using structural analysis for free-choice nets and a reachability analysis [37] to determine the terminal components for nets with both free-choice and unique-choice.

The following theorem [38] verifies that when a separation triple satisfies Condition 1, its TSE sequence generated by a random timed execution is *weakly ergodic* and thus the average TSE exists.

Theorem 1. *Let Σ be a LB Petri net that has only free-choices and unique-choices and satisfies stochastic assumptions given in Section 2.3. For any transition pair (s, t) for which Condition 1 holds, its corresponding average TSE with a fixed occurrent offset ε is a finite constant $\overline{\gamma}(s, t, \varepsilon)$ almost surely and in mean. That is,*

$$\mathrm{Prob}(\lim_{n\to\infty} \frac{1}{n} \sum_{k=1}^{n} \gamma^{(k)}(s, t, \varepsilon) = \overline{\gamma}(s, t, \varepsilon)) = 1, \tag{6}$$

$$\lim_{n\to\infty} \frac{1}{n} \sum_{k=1}^{n} \mathbf{E}\gamma^{(k)}(s, t, \varepsilon) = \overline{\gamma}(s, t, \varepsilon). \tag{7}$$

In some cases, it may be of interest to compute average time distance between some special occurrences of transition pair (s, t) which does not satisfy Condition 1. We state the following two cases where the *average time distance* still exists and can be computed by the approach described in this paper.

Case 1. The interested average time distance is between an occurrence of s to the ε-th occurrence of t following the occurrence of s.

Such an average time distance may make sense if s occurs less frequently than t, although it does not satisfy the definition of average TSE set earlier. If one can skip the irrelevant occurrence of t, Condition 1 is met. For instance, the pair (t_3, t_4) in the above example can be considered within this case.

Case 2. The pair (s, t) is such that for every T-invariant $\boldsymbol{v}$, if $\boldsymbol{v}(s) > 0$, $\boldsymbol{v}(t) > 0$, and vice versa. Moreover, $\boldsymbol{v}(s)/\boldsymbol{v}(t)$ is a constant.

In this case, one may split the occurrences of each of the two transitions into the occurrences of some new transitions, and consider the average TSE of a pair of new transitions corresponding to s and t. The new transition pair satisfies Condition 1.

In the paper, we assume the interested transition pair satisfies Condition 1 which serves as the basis of the other two cases.

The existence of TSE variance for the STPNs considered in this paper requires more consideration. In practice, the delays on places are finite or their moments are finite up to a sufficiently high order. In such cases, we believe when a TSE sequence is weakly ergodic (e.g., as characterized by Theorem 1), the existence of its variance (and in fact, any of its finite-order moments) is guaranteed.

Similarly, when the TSE sequence $\{\gamma^{(k)}(s,t,\varepsilon), k = 1, 2, \cdots\}$ is weakly ergodic, we believe its frequency function, F_γ, defined as the limit in (5) exists for every fixed x almost surely and in mean. In that case, we will call F_γ the *distribution* of the TSE.

3 Segments and TSE bounds

The key idea of the technique described in this paper is to partition an infinite timed execution into a sequence of so called *segments*. These segments are independent and identically distributed (iid). The targeted TSEs are then analyzed within individual segments independent of the remaining segments. Since the analysis is done using limited history, it yields upper and lower bounds on the TSEs (via simple longest path analysis) instead of exact values of the TSEs. The iid property of the bounds inherited from that of the partitioned segments facilitates the estimation of the statistics of the bounds by basic statistical methods discussed in Section 5.

3.1 Partitioning infinite timed executions

Let $\pi = (N_\pi, d, \ell)$ be a timed execution of a STPN $\Sigma = (N, M_0)$. It is known that every (untimed) reachable marking of Σ induces a cut ξ of instanced places which partitions the event graph N_π into disjoint parts [5]. The portion of the event graph in between two different cuts due to the same reachable marking M is a *segment* [38]. Formally, if σ is a firing sequence that starts from a cut ξ and ends at another cut ξ' such that $\ell(\xi) = \ell(\xi') = M$, then the portion of N_π between ξ and ξ' is a segment denoted by $S(\xi, \sigma)$. The segment is *minimal* if it does not contain any other segment starting from ξ. For the timed execution shown in Fig. 7, there are three minimal segments corresponding to the reachable marking where places p_1 and p_3 are marked. In the sequel, when we write segments, we refer to minimal ones. For convenience, we will write $\mathcal{S}(M)$ to be the set of all possible segments starting from a cut due to marking M.

One simple property of a random time execution of a Petri net considered in this paper is that the structures of its segments are independent of each other. This is simply because the structure of a segment is determined by the choices made on the places inside the segment and choices made in different segments are independent. As a result, the sequence of segments generated by a random timed execution has a property that their structures are not determined by the location of the segment in the sequence. In fact, they are independent and identically distributed (iid). This simple fact will allow us to reason about an infinite execution by considering all possible finite executions of as little as one segment in length.

3.2 Bounding a single TSE instance

By ignoring the history of a segment, we obtain bounds on a single TSE belonging to a particular segment. In other words, we define the TSE bounds by assuming tokens in the source places of the segment can be available at any time within $(-\infty, \infty)$. Fig. 8 illustrates this through an example. We will now show the upper bound can be computed using longest path analysis in time linear in the size of the segment.

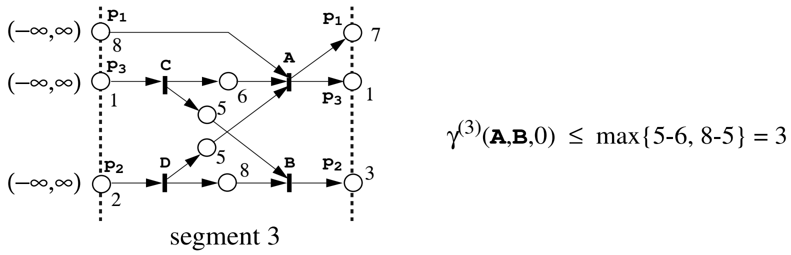


Fig. 8: Achieving bounds on TSE by using limited history by taking the token available time range to be $(-\infty, \infty)$for every source place.

The lower bound can be computed using duality. The reason is that, for a given ε, and any $j \in \mathbf{N}$ such that $j > -\varepsilon$, we have $-\gamma^{(j)}(s,t,\varepsilon) = -\tau(t^{(j+\varepsilon)}) + \tau(s^{(j)}) = \gamma^{(j+\varepsilon)}(t,s,-\varepsilon)$ from the definition of γ. Thus, if U is an upper bound on $\gamma(t,s,-\varepsilon)$, then $-U$ is a lower bound on $\gamma(s,t,\varepsilon)$. Because of this, we will discuss only the upper bounds on TSEs.

The idea to compute the upper bound is to identify a set of *reference events* that act as synchronization points for the targeted events. By assuming each of the synchronization points to be the critical one, one determines a set of time separations of the event pair. The upper bound is simply the largest separation obtained. To see this in more detail, let us consider bounding the TSEs due to separation triple $(s, t, \varepsilon = 0)$. Extension to the case where $\varepsilon \neq 0$ is not difficult.

Let ρ be a path of N_π which is the underlying event graph of a timed execution of $\pi = (N_\pi, d, \ell)$. The set of all paths leading from x to y is denoted by $\mathcal{P}(x,y)$ where $x, y \in P_\pi \cup T_\pi$. A *reference set* for event e in a timed execution π is a subset of events in π such that every path from a source place of N_π to e contains at least one event in R, and every event in R has a path to e. For example, for the segment shown in Figure 7, it can be checked that $\{C^{(1)}, D^{(1)}\}$ is a *minimal* reference set for event $B^{(1)}$ but not for $A^{(1)}$.

Let $\delta(\rho)$ be the sum of random delays assigned to all the places along path ρ, i.e., $\delta(\rho) = \sum_{p\in\rho} d(p)$. From the timing relation (1) (see Section 2.4), if there is a path from event x to y, y must occur after x by at least the maximum sum of delays on all the paths from x to y. That is, whenever $\mathcal{P}(x,y) \neq \emptyset$, we have

$$\tau(x) + \max_{\rho \in \mathcal{P}(x,y)} \delta(\rho) \leq \tau(y). \qquad (8)$$

Under a given delay assignment, event x is *critical* for event y if (8) holds in equality. In that case, there is a path $\rho^* \in \mathcal{P}(x,y)$ such that $\delta(\rho^*) = \tau(y) - \tau(x)$, and ρ^* is a critical path from x to y.

The significance of a reference set is that we can determine the occurrence time of an event e by knowing only the occurrence times of the events contained in a reference set of e plus the delay values of places following the events in the reference set. That is, the following timing relation holds for e, where R is a reference set of e.

$$\tau(e) = \max_{e' \in R}[\tau(e') + \max_{\rho \in \mathcal{P}(e',e)} \delta(\rho)]. \tag{9}$$

The term $\max_{\rho \in \mathcal{P}(e',e)} \delta(\rho)$ in (9) measures the maximum (random) delay on any path from event e' to e. For convenience, we write:

$$\delta^*(e', e) = \max_{\rho \in \mathcal{P}(e',e)} \delta(\rho), \tag{10}$$

where $\delta^*(e', e) \stackrel{\triangle}{=} -\infty$ if there is no path from e' to e, i.e., $\mathcal{P}(e', e) = \emptyset$. For completeness, we define an event e itself to be a path of delay 0, and consequently $\delta^*(e, e) = 0$.

Suppose the m-th TSE instance starts in the j-th segment of π, i.e., $S^{(l)}$. Since the set of source places of a segment is a cut of N_π, the poset of the source places of $S^{(l)}$ must contain a reference set for every event e in the segment (in fact, for every event in $S^{(l')}$ if $l' \geq l$). For convenience, if event $e \in S^{(l')} (l' \geq l)$, let us denote by $R(e, l)$ its reference set contained in the poset of the source places of segment $S^{(l)}$. An upper bound $U_\gamma{}^{(m)}(s, t, 0)$ on TSE $\gamma^{(m)}(s, t, 0)$ is determined by (11).

$$U_\gamma{}^{(m)}(s, t, 0) = \max_{e \in R(t^{(m)}, l)} [\delta^*(e, t^{(m)}) - \delta^*(e, s^{(m)})] \tag{11}$$

The critical fact concerning the above upper bound $U_\gamma{}^{(m)}(s, t, 0)$ on $\gamma^{(m)}(s, t, 0)$ is that it is independent of the occurrence time of the events in the reference set of $s^{(m)}$. In other words, it relies only on the structure of segment $S^{(l)}$ and the delays of the places within the segment. In particular, it does not depend on the history of the timed execution prior to this segment.

As an example, consider upper bounding the TSE $\gamma^{(3)}(A, B, 0)$ in the segment shown in Fig. 8. Since $\{C^{(3)}, D^{(3)}\}$ is a reference set for $B^{(3)}$, we know that $\gamma^{(3)}(A, B, 0)$ is upper bounded by $\max\{\delta^*(C^{(3)}, B^{(3)}) - \delta^*(C^{(3)}, A^{(3)}), \delta^*(D^{(3)}, B^{(3)}) - \delta^*(D^{(3)}, A^{(3)})\} = \max\{5-6, 8-5\}$=3.

Several possible algorithms may be used to compute the upper bound on a TSE. Since a segment structure is acyclic, one possible approach is to use the TSE approximate algorithm described in [12] by assigning a delay interval $(-\infty, \infty)$ to every source place of the segment being considered. The algorithm has a complexity no lower than $O(|T(S)|^2)$ where $|T(S)|$ is the number of events in segment S.

However, since all the internal places of a segment in a given timed execution has a fixed delay (only the delay values of source places whose delay values are unknown), the problem can be solved more efficiently. In particular, applying a simple longest path analysis from a fixed event e as discussed above, the term $\delta^*(e, t^{(m)})$ in (11) can be determined in $O(|T(S)| + |P(S)|)$ time where $|P(S)|$ is the number of places in segment S. Thus, $U_\gamma{}^{(m)}(s, t, 0)$ in (11) can be determined in $O((|T(S)|+|P(S)|)*|R|)$ time where $|R|$ is the size of the referent set of e, which is typically much faster than using the method in [12]. Alternatively, the longest path analysis can be conducted from each of the interested events back towards their reference events. This way, $U_\gamma{}^{(k)}$ is determined in $O(|T(S)| + |P(S)|)$ time.

4 Bounding TSE statistics

4.1 Grouping of TSEs

As pointed out earlier, the structures of segments are independent. However, multiple TSE instances of a transition pair (s, t) may start in one segment and they can be dependent on each other. For example, for the time execution shown in Fig. 7, there are two TSE instances of pair (t_4, t_2) starting from segment 3 and their TSE values are not independent. To overcome this dependency, we treat all the TSE instances starting from a same segment together as a *group*.

To formally define a group, consider a TSE sequence $\{\gamma^{(j)}(s, t, \varepsilon) : j \geq 1\}$ generated by a random timed execution π of $\Sigma = (N, M_0)$. The segment sequence $\{S^{(l)} \in \mathcal{S}(M_0) : l \geq 1\}$ of π partitions the TSE sequence into sub-sequences. Two TSEs $\gamma^{(i)}(s, t, \varepsilon)$ and $\gamma^{(j)}(s, t, \varepsilon)$ are contained in one sub-sequence if their corresponding source events $s^{(i)}$ and $s^{(j)}$ are in a same segment.

We write $\eta^{(k)}(s)$ the index of the last TSE in group k by $\eta^{(k)}(s)$, and define the number of TSEs due to the separation triple (s, t, ε) to be the *length* of the group, denoted by $\alpha^{(k)}(s)$. That is,

$$\alpha^{(k)}(s) = \eta^{(k+1)}(s) - \eta^{(k)}(s).$$

The sum of all TSEs in group k as k-th *grouped TSE* denoted by $\beta^{(k)}(s, t, \varepsilon)$. That is

$$\beta^{(k)}(s, t, \varepsilon) = \sum_{l=\eta^{(k-1)}(s)+1}^{\eta^{(k)}(s)} \gamma^{(l)}(s, t, \varepsilon).$$

We have shown that both the group length sequence and the grouped TSE sequence exhibit similar ergodicity property to that of the TSE sequence [38]. In other words,

$$\frac{1}{n} \sum_{k=1}^{n} \beta^{(k)}(s, t, \varepsilon) \to \overline{\beta}(s, t, \varepsilon), \tag{12}$$

$$\frac{1}{n} \sum_{k=1}^{n} \alpha^{(k)}(s) \to \overline{\alpha}(s). \tag{13}$$

The convergence in (12) and (13) takes almost surely and in mean.

Moreover, $\overline{\beta}(s, t, \varepsilon) = \overline{\alpha}(s)\overline{\gamma}(s, t, \varepsilon)$ where $\overline{\gamma}(s, t, \varepsilon)$ is the average TSE of the triple (s, t, ε). Note that since the labels of segment structures are iid for a random time execution, the average group length $\overline{\alpha}(s)$ is merely the average length of group k (k fixed) over all possible executions, i.e., $\overline{\alpha}(s) = \mathbf{E}\alpha^{(k)}(s)$. Section 5 uses simple statistical method to evaluated $\mathbf{E}\alpha^{(k)}(s)$ as well as the bounds on $\overline{\beta}(s,t,\varepsilon)$, and consequently we get bounds on $\overline{\gamma}(s,t,\varepsilon)$.

Similarly, we define the grouped TSE-squares as the sum of squares of all targeted TSEs in a group. Similar to the argument of the existence of the TSE variance, we expect that the sequence of the grouped TSE-squares also exhibits the weak ergodicity property.

4.2 Bounding average TSEs

Following the argument in the previous section, the problem of bounding the average TSE can be translated into bounding the average grouped TSE. Towards this end, an upper bound on a grouped TSE can be simply computed as the sum of the upper bounds on the TSEs in the group, which in turn can be obtained by the longest path analysis technique described in Section 3.2. In other words, for a grouped TSE $\beta^{(k)}(s,t,0)$, we have

$$\beta^{(k)}(s,t,0) \leq {U_\beta}^{(k)}(s,t,0) = \sum_{m=\eta^{(k-1)}+1}^{\eta^{(k)}} {U_\gamma}^{(m)}(s,t,0) \tag{14}$$

Consequently, the expectation $\mathbf{E}{U_\beta}^{(k)}(s,t,0)$, servers an upper bound on the average of $\beta^{(k)}(s,t,0)$ over all possible timed executions. Due to the weak ergodicity property of the grouped TSE, it is not difficulty to show that $\mathbf{E}\beta^{(1)}(s,t,0)$ is an upper bound on $\overline{\gamma}(s,t,0)$.

4.3 Bounding the averages for multiple TSEs

In some applications such as in performance-driven synthesis, it is desirable to find average TSEs for multiple transition pairs.

Towards this end, one approach is to apply the single-source longest path analysis from the targeted events as suggested in Section 4.2, once for each of the transition pairs. That way, for W pairs of transitions, the bounds on their TSEs due to a given segment can be computed in $O(|E| \times W)$ time.

For a large W, a more efficient approach is to apply the single-source longest path algorithm on the segment by taking each of the poset events of source places of the segment as the source. This way, the longest path analysis is performed $|R|$ times where R is the poset of the source places of the segment. Consequently, the TSE bounds can be computed in $O(|E| \times |R| + W)$ time.

4.4 Bounding TSE variance

The following inequality gives the bounds on $\sigma^2(s,t,\varepsilon)$:

$$\lim_{n\to\infty} \frac{1}{n} \sum_{k=1}^{n} \min\{{U_\gamma}^{(k)2}(s,t,\varepsilon), {L_\gamma}^{(k)2}(s,t,\varepsilon)\} - \max\{\overline{U_\gamma}^2(s,t,\varepsilon), \overline{L_\gamma}^2(s,t,\varepsilon)\}$$

$$\leq \sigma^2_{\gamma(s,t,\varepsilon)} \leq$$

$$\lim_{n\to\infty} \frac{1}{n} \sum_{k=1}^{n} \max\{{U_\gamma}^{(k)^2}(s,t,\varepsilon), {L_\gamma}^{(k)^2}(s,t,\varepsilon)\} - \min\{\overline{U_\gamma}^2(s,t,\varepsilon), \overline{L_\gamma}^2(s,t,\varepsilon)\}$$

where $\overline{U_\gamma}(s,t,\varepsilon)$ and $\overline{L_\gamma}(s,t,\varepsilon)$ are the means of the upper and lower bounds on the TSE, respectively. Should the above lower bound be negative, we take its trivial lower bound on the variance, zero.

By grouping the squares of the upper and lower bounds on individual TSEs (cf. Section 4.1), we can avoid the dependency among upper and lower bounds on the TSEs due to a same segment. Consequently, the grouped sum of squares of the bounds are iid due to iid property of the random segments.

4.5 Bounding TSE distributions

For the k-th TSE instance of the separation triple, the following relation holds:

$$\mathbf{1}_{{U_\gamma}^{(k)}(s,t,\varepsilon)\leq x} \leq \mathbf{1}_{\gamma^{(k)}(s,t,\varepsilon)\leq x} \leq \mathbf{1}_{{L_\gamma}^{(k)}(s,t,\varepsilon)\leq x} \tag{15}$$

where ${U_\gamma}^{(k)}(s,t,\varepsilon)$ and ${L_\gamma}^{(k)}(s,t,\varepsilon)$ are the upper and lower bounds on the TSE instance $\gamma^{(k)}(s,t,\varepsilon)$, respectively. The quantities $\mathbf{1}_{{U_\gamma}^{(k)}(s,t,\varepsilon)\leq x}$ and $\mathbf{1}_{{L_\gamma}^{(k)}(s,t,\varepsilon)\leq x}$ can be estimated again using longest path analysis as we did with the TSEs. Finally, using a technique similar to the one used in bounding the means of TSEs, we achieve bounds on $F_{\gamma(s,t,\varepsilon)}(x)$ for a fixed x. In practice, upper and lower bounds on $F_{\gamma(s,t,\varepsilon)}$ may be plotted at a sufficiently large number of different values of x, say, 20, yielding a discretized approximation of the TSE distribution.

5 Evaluating the bounds

In the previous section, closed-form expressions of the bounds on various statistics of TSEs are derived. These expressions take expectations of potentially complicated random variables. In some special cases, analytical methods are available to obtain the exact value of the expectation [36]. In the general case, the formula can be estimated using statistical methods to a sufficiently high accuracy in the probabilistic sense. In our current tool, we use Monte-Carlo sampling (e.g.,[28]).

5.1 Monte-Carlo sampling

For convenience, let us write W to denote the above random variable for which the expectation is taken, and let F_W denote its distribution function.

The intuition behind Monte-Carlo simulation approach is the Central Limit Theorem [18] which deals with the partial sum of iid random variables. In our case, if $W_1, W_2, \cdots, W_n$ are independent random variables all having the same distribution function F_W, then for a large n, the random variable $\frac{1}{n}S_n$ approaches to $\mathbf{E}W$ where $S_n = W_1+W_2+\cdots+W_n$. More precisely, $\frac{1}{n}S_n$ approaches a *normal* distribution with mean $\mathbf{E}W$ and variance $\frac{1}{n}\sigma_W^2$ where σ_W is the variance of W. Since $\frac{1}{n}\sigma_W^2$ decreases to 0 as n increases, any *realization* of the random variable $\frac{1}{n}S_n$ is a good (unbiased) estimate of $\mathbf{E}W$ when n is large. In fact, the following result is well-known (e.g.,[28]). For any given *relative error interval* I_e and a confidence level L_c, we have $P\{|(\frac{1}{n}S_n - \mathbf{E}W)/\mathbf{E}W| < I_e\} > L_c$ as long as

$$n > (\frac{z_{x/2}\sigma_W}{I_e\mathbf{E}W})^2, \tag{16}$$

where $x = 1 - L_c$ and $z_{x/2}$ is defined such that the tail probability to its right under the standard normal distribution is $x/2$. The procedure of this realization of $\frac{1}{n}S_n$ is commonly known as *Monte-Carlo* sampling. It involves a random realization of n iid random variables (i.e., W_1 through W_n) also called as a *random sample* of W with a size of n. (16) is referred to as the stopping criterion.

In practice, the true variance of W, i.e., σ_W^2 is not known and is commonly replaced by the variance of the sample, i.e., $S^2 = \frac{1}{n-1}\sum(W_i - \frac{1}{n}S_n)^2$. Similarly, $\mathbf{E}W$ in (16) is replaced by the mean of the sample, i.e., $\frac{1}{n}S_n$ itself. As a consequence, $z_{x/2}$ in (16) with $x = 1 - L_c$ is replaced by $t_{x/2}$ which is defined such that the tail probability to its right under the student t-distribution [28] is $x/2$.

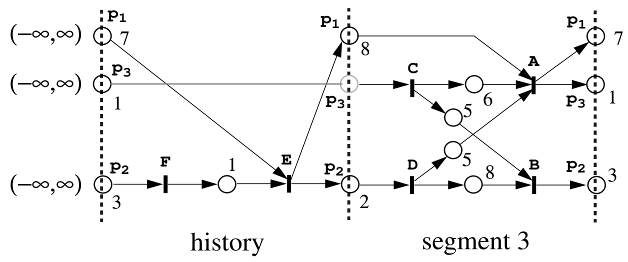
From a theoretical point of view, the validity of the Monte-Carlo approach depends on the finiteness of high order moments of the delays of the SPNs. We do not believe this is a significant restriction because system component delays in real designs are either finite or their moments are finite up to a sufficiently high order.

5.2 Improving the bounds

In many cases, the upper and lower bounds on the TSE statistics derived in the previous subsections are very close to each other, which combined yield very good estimates. As one of the distinct features of our technique, the above bounds can be made even closer to each other by slightly increasing the amount of (consecutive) segments used in the derivation of the bound. That is, we can often improve the bounds by slightly pushing the reference set further apart from the TSEs pairs being considered, thereby increasing the amount of history considered. Fig. 9 demonstrate this improvement as compared with the bound achieved without extra history in Fig. 8.

6 Experiments

The bounding techniques described in this paper has been fully automated. The resulting tool is named `UscPet`. Currently, it applies the simplest statistical method, i.e., the Monte-Carlo approach, to construct the error interval and confidence of the estimates. In this section, we apply `UscPet` to analysis of two scalable examples. The first example is a micropipeline where each stage exhibits choice, and the second a Petri net model of Intel's asynchronous instruction length decoding and steering unit called RAPPID [31].



$\gamma^{(3)}(\mathbf{A},\mathbf{B},0) \le \max\{5\text{-}6, (2+8)\text{-}8\} = 2$

Fig. 9: Improving TSE bounds by adding extra segments as history.

6.1 Micropipelines with choice

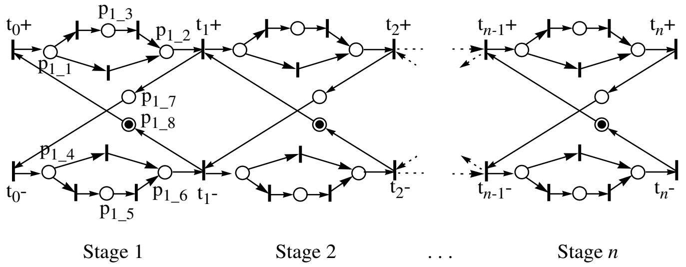


Fig. 10: A micropipeline where each stage exhibits choice.

Fig. 10 depicts the model of the micropipeline with which we experimented. For stage i, places p_{i_1} and p_{i_4} are assigned delays that are uniformly distributed between 0 and 10, and places p_{i_2} and p_{i_6} are assigned delays that are uniformly distributed between 1 and 2. Places labeled by p_{i_3} and p_{i_5} have a fixed of 2. The other two places p_{i_7} and p_{i_8} have a fixed delay of 0.5. Each transition out of all choice places is given a probability of 0.5.

Table 1 lists the estimated bounds on the mean and variance of the cycle time (i.e., $\gamma(\mathtt{t_i+}, \mathtt{t_i+}, 1)$ with i fixed) as a function of pipeline depth. It is interesting to note that the sample size needed in estimating the bounds of variance of the cycle time is remarkably larger ($3 \sim 5$ times larger) than that needed in estimating mean of the cycle time, and thus more CPU time is required. In this table, the amount of segment history is set to approximately 1.5 times the depth of the micropipeline. The results also show that the cycle time of the pipeline increases in mean but decreases in variance as the pipeline depth increases.

Fig. 11 shows 4 plots of the lower and upper bounds on the distribution function of the cycle time of an 8-stage micropipeline as the amount of history considered is increased. As in the case of bounding the mean and variance, these

# stages	# transitions	# places	Mean Lower	Mean Upper	Mean CPU secs	Variance Lower	Variance Upper	Variance CPU secs
2	18	16	19.35	19.35	2.8	12.69	12.69	3.9
4	32	32	21.78	21.78	4.3	7.05	7.21	9.2
8	66	64	22.67	22.68	8.1	5.76	5.90	43.5
16	130	128	23.11	23.12	34.0	5.41	5.58	113.2
32	258	256	23.55	23.55	148.8	4.58	5.31	781.8

Table 1: Bounds on the mean and variance of cycle time of the micropipeline. The relative error interval and confidence level are set to 1% and 99%, respectively, in the Monte-Carlo sampling.

plots show that the bounds on the distribution function quickly sharpens as the length of the considered history increases. Fig. 12 plots the estimated distribution function of the cycle time for different pipeline depth. As the depth increases, the distribution function shifts towards the right.

# stages	bounding the mean latency: through whole pipeline # TSEs	through whole pipeline CPU seconds	through every stage # TSEs	through every stage CPU seconds
2	1	3.29	2	3.44
4	1	9.22	4	11.61
8	1	28.5	8	32.8
16	1	111.4	16	118.5
32	1	446.7	32	460.3

Table 2: Computing bounds on multiple TSEs.

The efficiency of the tool in computing bounds on multiple TSEs is demonstrated by simultaneously estimating bounds on average latencies through all pipeline stages. Table 2 lists the required CPU times for both single TSE estimation (estimating the average latency through the whole micropipeline) versus those required for computing bounds on average latencies through all stages. The CPU times are reported by applying the longest path analysis from the poset events of the source places of segments. The result shows the the percentage of extra CPU time required to compute bounds on means of multiple TSEs is negligible. In addition, we observe that this percentage is almost in proportion to the number of transition pairs being analyzed for their TSEs.

6.2 A model of RAPPID

The RAPPID design is an aggressive asynchronous design to decode the length of x86 instructions and steer them to the subsequent instruction decoder with

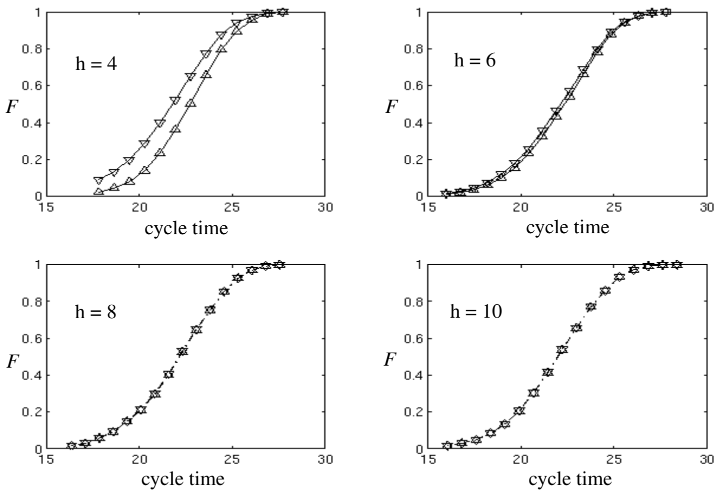


Fig. 11: Improvement of bounds on the distribution function of cycle time in an 8-stage micropipeline as the amount of the history considered (h is the number of history segments) increases. The relative error interval and confidence level are set to 1% and 99%, respectively, in the Monte-Carlo sampling.

a high average throughput. The instructions are fed into the design through a number of parallel byte units associated with columns. Each column speculatively decodes the byte under the assumption that an instruction starts at its associated byte. Each column has a number of tag units responsible for receiving a token indicating this column is the start of the next instruction and routing the token to a tag unit associated with the column which contains the start of the subsequent instruction. The tag units are arranged in rows each associated with an output buffer for the instruction. To alleviate a bottleneck in the output buffers, tag units in one row send its token to tag units in the next row. Both columns and rows are designed in a ring fashion so that they handle an infinite number of instructions. The real design has 16 columns, 4 rows, and is optimized to decode instructions of up to 7 bytes in length [31].

In our experiment, we examine a live and safe Petri net model of the RAPPID where the delays of places are only loosely related to the real delays in the design. Our model also ignores the handling of very rare instructions (e.g., length changing prefixes) and branches. The central driving units for the model is the actual length decoding logic in the columns. These units are optimized for the common instruction lengths and consequently have longer delay for less common instructions, as previously described in [14]. In our Petri net model, depicted in Fig. 13, this logic is modeled with a free-choice place with probability mass function that matches the relative frequency of instruction lengths given in [14]. Even for a given instruction length the decoding time can vary, thereby motivating the use of stochastic delay models for the decoding of each length.

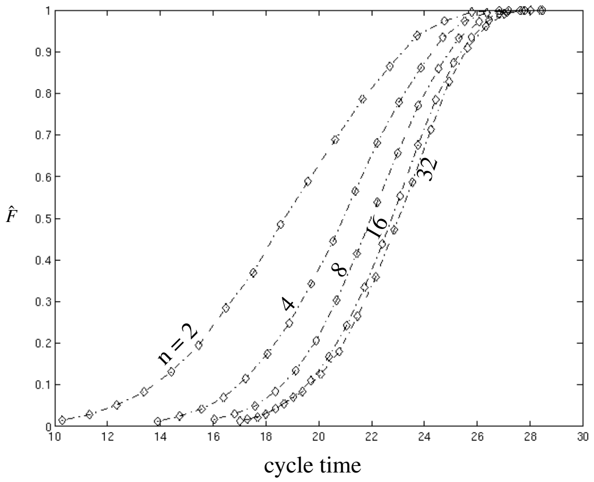


Fig. 12: The estimated distribution functions (the average of the upper and lower bounds) plotted for pipelines of different depths (i.e., n in number of stages). The relative error interval and confidence level are set to 1% and 99%, respectively, in the Monte-Carlo sampling.

Once decoded, the column broadcasts a signal to multiple tag units associated with its column indicating its instruction is ready to be transmitted to the output buffer associated with the tag unit. In our Petri net model, the tag unit in the top row of the first column is initially marked with a token to indicate that the first instruction will start at the first column and should be routed to the first output buffer. Once all bytes of the first instruction are available, this tag unit sends a token to a downstream tag unit in the second row of the column where the next instruction starts. We use unique-choice places in the model to route tokens to downstream tag units as well as the consumption of tokens representing all available bytes of an instruction.

We specified uniform delays of between 0.1 and 0.6 time units for pGotBuff, between 0.5 and 1.5 time units for pLen1, pLen2, and pLen3, between 1.0 and 2.5 time units for pLen4 and pLen5, and between 2 and 4 time units for pLen6 and pLen7. The pRowReq was given a fixed delay of 0.5 time units and all other places where given fixed delays of 0.1 time units.

We experimented with a number of different RAPPID-based architectures by varying the number of columns, rows, and the maximum instruction lengths handled. For each architecture, the mean and variance of the time separation between consecutive firings of transition `tRowReq0`, i.e., $\overline{\gamma}(\texttt{tRowReq0}, \texttt{tRowReq0}, 1)$. The bounds on the mean of this TSE divided by the number of rows provides an estimate of the average decoding cycle time and thus the throughput. The bounds on its variance give an estimate of the performance deviation. The reported variance bounds, when divided by the number of rows, suggest that increasing the the number of rows helps reduce the variance of the overall decoding cycle time.

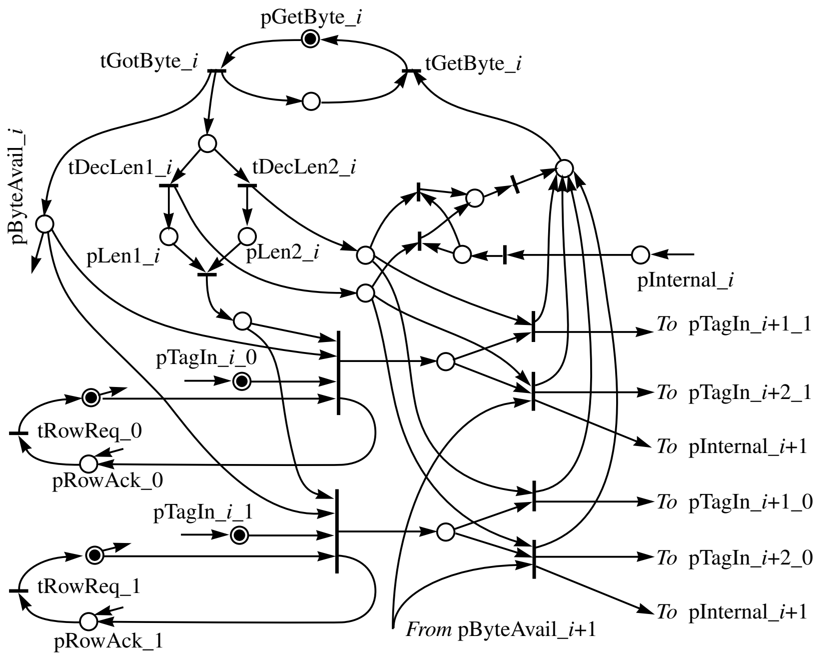


Fig. 13: A Petri net model of 1 column of the RAPPID instruction length decoding and steering unit. This column has 2 Tag Unit rows and can handle instructions of length up to 2 bytes.

Experimental results are tabulated in Table 3. The CPU times listed are for bounding the variance, which includes the times for bounding the averages. The length of the history used in the experiments (i.e., the number of segments prior to the one considered) was set between 8 to 40 segments depending on the sizes of the individual RAPPID models.

Since the delay models do not match with the real design, we would rather not stress the significance of the performance trends. Instead, we prefer to stress the low run-time of the tool and the accuracy of the estimates. In all the examples tested, the width of the derived error bounds are less than 5% and are typically less than 1%. The largest model contains over 900 transitions and 500 places and yet can be analyzed with 5% error tolerance and 99% confidence in 1.5 hours. These experiments demonstrate that the tool may be used to trade off the silicon area of the design with the system throughput. If possible, we hope to incorporate more accurate component delay estimates to obtain a more realistic analysis of competing architectures.

7 Conclusions

This paper describes our recent developments on the performance analysis of asynchronous circuits and systems using stochastic timed Petri nets. We use a

# Columns	# Rows	Length	# Trans.	# Places	Mean Lower	Mean Upper	Variance Lower	Variance Upper	CPU seconds
2	1	2	29	34	1.315	1.327	0.219	0.249	40.6
2	2	2	36	40	2.643	2.644	0.437	0.442	46.9
2	3	2	43	46	3.987	3.991	0.592	0.594	55.16
2	4	2	50	52	5.295	5.304	0.753	0.754	66.89
4	1	2	57	66	0.684	0.703	0.157	0.214	156.2
4	2	2	70	76	1.374	1.376	0.222	0.231	254.3
4	3	2	83	86	2.062	2.063	0.334	0.337	153.6
4	4	2	96	96	2.744	2.746	0.361	0.363	359.5
8	1	7	273	210	0.957	0.976	0.778	0.871	411.1
8	2	7	338	228	1.925	1.926	1.420	1.428	630.5
8	3	7	403	246	2.893	2.893	2.106	2.106	990.5
8	4	7	468	264	3.843	3.843	2.689	2.690	1,385
16	1	7	545	418	0.548	0.554	0.328	0.348	558.9
16	2	7	674	452	1.056	1.056	0.821	0.824	2,189
16	3	7	803	486	1.588	1.589	1.170	1.175	3,419
16	4	7	932	520	2.122	2.123	1.355	1.358	5,426

Table 3: Bounds on the mean and variance of the decoding cycle time of the RAPPID model. The relative error interval and confidence level are set to 5% and 99%, respectively, in Monte-Carlo sampling.

novel random sampling approach to give precise estimates of the mean, variance and distribution function of time separations of events for a wide class of transition pairs. The estimates are achieved by computing the lower and upper bounds on the targeted quantities based on iid segment generation and simple longest path analysis. Experiments on large-scale STPN models demonstrated the efficiency and capability of the tool.

We are currently studying the possibility of extending the tool to handle STPNs with arbitration. The integration of this technique into performance-driven synthesis tools is also being examined.

Acknowledgements

This work is partially funded by NSF Grant CCR-9812164. We would like to thank the anonymous reviewers of this and previous versions of this paper for insightful feedback and suggestions.

References

1. R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

2. W. Belluomini and C. J. Myers. Verification of timed systems using POSETS. In *Proc. International Workshop on Computer Aided Verification*, pages 403–415, 1998.
3. E. Best. Some classes of live and safe Petri nets. In *Concurrency and nets,* Special volume in the series, "Advances in Petri nets", pages 71–94. Springer-Verlage, 1987.
4. E. Best. Structural theory of Petri nets: The free choice hiatus. In *Lecture Notes in Computer Science, LNCS 254*, pages 168–206. Springer-Verlage, 1987.
5. E. Best. Partial order behavior and structure of Petri nets. *Formal Aspects of Computing*, 2:123–138, 1990.
6. E. Best and K. Voss. Free choice systems have home states. *Acta Informatica*, 21:89–100, 1984.
7. P. Buchholz. Exact and ordinary lumpability in finite Markov chains. *Journal of Applied Probability*, 31:59–75, 1994.
8. P. Buchholz. Lumpability and nearly-lumpability in hierarchical queueing networks. In *Proc. IEEE Int. Computer Performance and Dependability Symp. (IPDS)*, pages 82–91, 1995.
9. P. Buchholz. Hierarchical structuring of superposed gspns. In *International Workshops on Petri Nets and Performance Modelds*, pages 81–90, June 1997.
10. S. M. Burns. *Performance Analysis and Optimization of Asynchronous Circuits.* PhD thesis, California Institute of Technology, 1991.
11. J. Campos, G. Chiola, and M. Silva. Properties and performance bounds for closed free choice synchronized monoclass queueing networks. *IEEE Transactions on Automatic Control*, 36(12):1368–1382, Dec. 1991.
12. S. Chakraborty and D. L. Dill. Approximate algorithms for time separation of events. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, pages 190–194, November 1997.
13. T. J. Chaney and C. E. Molnar. Anomalous behavior of synchronizer and arbiter circuits. *IEEE Transactions on Computers*, C-22(4):421–422, April 1973.
14. W.-C. Chou, P. A. Beerel, R. Ginsoar, R. Kol, C. J. Myers, S. Rotem, K. Stevens, and K. Y. Yun. Average-case optimized technology mapping of one-hot domino circuits. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pages 80–91. IEEE Computer Society Press, March 1998.
15. J. Ebergen and R. berks. Reponse-time of asynchronous linear pipelines. *Proceedings of the IEEE*, 87(02):308–318, February 1999.
16. P. W. Glynn and D. L. Iglehart. Simulation output analysis using standardized time series. *Math. Oper. Res.*, 3:1–16, 1990.
17. M. R. Greenstreet and K. Steiglitz. Bubbles can make self-timed pipelines fast. *Journal of VLSI Signal Processing*, 2(3):139–148, November 1990.
18. G. R. Grimmett and D. R. Stirzaker. *Probability and Random Processes (2nd Edition).* Oxford Science Publications, 1992.
19. P. J. Haas. Estimation methods for stochastic Petri nets based on standardized time series. In *International Workshops on Petri Nets and Performance Modelds*, pages 194–204, June 1997.
20. M. A. Holliday and M. Y. Vernon. A generalized timed Petri net model for performance analysis. *IEEE Transactions on Software Engineering*, 13(12):1297–1310, December 1987.
21. H. Hulgaard and S. M. Burns. Bounded delay timing analysis of a class of CSP programs with choice. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pages 2–11. IEEE Computer Society Press, November 1994.

22. H. Hulgaard and S. M. Burns. Efficient timing analysis of a class of Petri nets. In *Proc. International Workshop on Computer Aided Verification*, pages 423–436, 1995.
23. P. Kudva, G. Gopalakrishnan, E. Brunvand, and V. Akella. Performance analysis and optimization of asynchronous circuits. In *Proc. International Conf. Computer Design (ICCD)*, pages 221–225, October 1994.
24. T. Lee. *A General Approach to Performance Analysis and Optimization of Asynchronous Circuits*. PhD thesis, California Institute of Technology, 1995.
25. M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. John Wiley & Sons, 1995.
26. M. Ajmone Marsan, A. Bobbio, and S. Donatelli. Petri nets in performance analaysis: An introduction. In *Lectures on Petri nets I: Basic Models*, Lecture Notes in Computer Science. Springer-Verlag, 1998.
27. A. J. Martin, S. M. Burns, T. K. Lee, D. Borkovic, and P. J. Hazewindus. The design of an asynchronous microprocessor. In Charles L. Seitz, editor, *Advanced Research in VLSI: Proceedings of the Decennial Caltech Conference on VLSI*, pages 351–373. MIT Press, 1989.
28. I. R. Miller, J. E. Freund, and R. Johnson. *Probability and Statistics for Engineers*. Prentice Hall, 1990.
29. T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77:541–580, April 1989.
30. C. V. Ramamoorthy and G. S. Ho. Performance evaluation of asynchronous concurrent systems using Petri nets. *IEEE Transactions on Software Engineering*, 6(5):440–449, September 1980.
31. S. Rotem, K. Stevens, B. Agapiev, C. Dike, M. Roncken, R. Ginosor, R. Kol, P. A. Beerel, K. Y. Yun, and C. J. Myers. Rappid: An asynchronous instruction length decoding and steering unit. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, April 1999.
32. A. Semenov and A. Yakovlev. Verification of asynchronous circuits using time Petri-net unfolding. In *Proc. ACM/IEEE Design Automation Conference*, pages 59–63, 1996.
33. G. S. Shedler. *Regenerative Stochastic Simulation*. Academic Press, New York, 1993.
34. A. Xie and P. A. Beerel. Accelerating Markovian analysis of asynchronous systems using state compression. to appear in *IEEE Transactions on Computer-Aided Design*, vol. 18, No. 7, July 1999.
35. A. Xie and P. A. Beerel. Symbolic techniques for performance analysis of asynchronous systems based on average time separation of events. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pages 64–75. IEEE Computer Society Press, April 1997.
36. A. Xie and P. A. Beerel. Bounding average time separations of events in stochastic timed marked graphs. Technical report, EE-Systems Department, University of Southern California, Oct 1998.
37. A. Xie and P. A. Beerel. Efficient state classification of finite state Markov chains. *IEEE Transactions on Computer-Aided Design*, 17(12):1334–1338, December 1998.
38. A. Xie, S. Kim, and P. A. Beerel. Bounding average time separation of events in stochastic timed Petri nets with choice. to appear in *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, April 1999.

39. A. Xie, S. Kim, and P. A. Beerel. Bounding average time separation of events in stochastic timed Petri nets with choice (w/ proofs), November 1998. See http://jungfrau.usc.edu/publications.html.
40. A. Yakovlev and A. M. Koelmans. Petri nets and digital hardward design. In *Lectures on Petri nets II: Basic Models*, Lecture Notes in Computer Science. Springer-Verlag, 1998.
41. K. Y. Yun, P. A. Beerel, V. Vakilotojar, A. E. Dooply, and J. Arceo. A low-control-overhead asynchronous differential equation solver. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pages 210–223. IEEE Computer Society Press, April 1997.