Towards Performance Prediction of Compositional Models in Industrial GALS Designs

Nicolas Coste[1,2], Holger Hermanns[1,3], Etienne Lantreibecq[2], and Wendelin Serwe[1]

[1] INRIA Grenoble – Rhône-Alpes, VASY project team
[2] STMicroelectronics Grenoble
[3] Universität des Saarlandes

Abstract. Systems and Networks on Chips (NoCs) are a prime design focus of many hardware manufacturers. In addition to functional verification, which is a difficult necessity, the chip designers are facing extremely demanding performance prediction challenges, such as the need to estimate the latency of memory accesses over the NoC. This paper attacks this problem in the setting of designing globally asynchronous, locally synchronous systems (GALS). We describe foundations and applications of a combination of compositional modeling, model checking, and Markov process theory, to arrive at a viable approach to compute performance quantities directly on industrial, functionally verified GALS models.

1 Introduction and motivation

Systems and networks on chip (NoC) are becoming more complex. Because mono-clocked designs are reaching their limits, globally asynchronous, locally synchronous (GALS) designs with multiple locally synchronous clock domains are prevalent in many hardware design labs. Due to high masking and production costs, their functional verification is a major concern, and concurrency phenomena pose additional challenges for the designers.

A complex NoC may be functionally verified but still not compliant with its targeted performances. Similar observations are well-known in the networked distributed systems community [12], but three difficulties characterize the NoC domain: *(i)* functionality and performance are much more deeply intertwined, *(ii)* prototype costs are prohibitive in the hardware domain, and *(iii)* isolated components are not available for physical experimentation or post-design tuning.

Therefore it is imperative to carry out performance evaluation as early as possible, i.e., on system models before having first prototypes, and even before having precise descriptions of the architecture. Until now, industrial performance evaluation of architectural specifications have been very approximative and based on back-of-the-envelope (or spreadsheet) calculations and sometimes rough simulations. This notoriously results in over-dimensioned communication networks. Therefore, STMicroelectronics is investigating a sound methodology to integrate performance evaluation and functional verification, which fits into their established design flow. This paper is a result of these activities.

Of specific interest are performance results concerning system utilization, latency, and throughput. For instance, one may want to study the utilization of a FIFO queue, the latency between entry and exit of a shared resource part, or the throughput of an operation. In this paper, we focus on the study of latencies. Notice that throughput problems can be seen as latency problems: the average throughput corresponds to the inverse of the average latency.

In a GALS design, hardware delays local to a single synchronous clock domain are discrete and can be precisely expressed as a number of clock steps. Generalizing the expression of a delay to a discrete probability distribution allows one to incorporate cumulative effects of system parts that are outside the scope of — but interfering with — the current model (e.g., arbitration strategies, memory latencies). In this way probabilities enter the modeling as an abstraction aid, possibly also to represent drifts between different clock domains. Figure 1 shows an example distribution, where a delay takes either one, two, or three steps. Modeling delays in this way, performance measures can be obtained by analyzing the underlying stochastic process, often a Markov chain (MC).

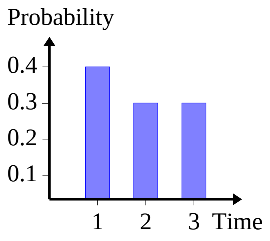


Fig. 1. Example delay

Due to the inherent complexity of asynchronous designs, existing functional verification approaches (mostly based on simulation) require long execution times to ensure sufficient coverage. Therefore formal methods are being applied to ensure functional correctness and detect errors. STMicroelectronics currently invests in the use of LOTOS [6] to formally model and validate their designs.

Performance evaluation studies for the same system usually require the development of another model. To avoid the cost of two different formal models, and to allow for joint considerations, we are working on a compositional approach allowing designers to enrich the available functional models with probabilistic time information. After first experiments with interactive Markov chains [4] the architects urged us to develop a synchronous variant with similar properties, and the result is presented in this paper. In our analysis so far, we are focusing on studying latencies of network components. As a matter of fact, though the notion of latency is often used in practical investigations, it turned out to be difficult to make that notion precise in a probabilistic timed setting. We define it as a Cesáro limit over the distribution of time elapsing between pairs of events.

In summary, the genuine contributions of this paper are: *(i)* the formalization of latency distributions on Markov chains, *(ii)* a compositional approach using a single model for functional verification and performance evaluation in a probabilistic discrete time setting, and *(iii)* the illustration of this approach on an industrial case study.

The remainder of this paper is organized as follows: Section 2 formally defines latency distributions of MC. Section 3 introduces the model of interactive probabilistic chains (IPCs). Section 4 describes how to extract latency distributions from an IPC via transformation to a MC. Section 5 reports on an industrial case study. Section 6 presents related work. Finally, Section 7 presents our conclusions.

2 Latency Distribution of Markov Chains

This section discusses latency distributions for time-homogeneous discrete time Markov chains (MC). After recalling the definition of an MC, we define the latency as a random variable denoting the number of time steps to reach a set of states from another set. Finally, we present how to compute latency distributions.

2.1 Markov Chains and Latencies

Definition 1. *A Markov chain is a tuple $\langle C, P, \hat{c}\rangle$ where C is a countable set of states, $P : C \times C \to [0,1]$ is a matrix of transition probabilities that satisfies $\sum_{c' \in C} P(c,c') = 1$ for all $c \in C$, and $\hat{c} \in C$ is a unique initial state.*

For a given MC $M = \langle C, P, \hat{c}\rangle$, $\mathit{Paths}(c)$ denotes the set of infinite paths starting in $c \in C$ where an infinite path is an infinite sequence $\sigma = c_0, c_1, c_2, \ldots$ satisfying $P(c_t, c_{t+1}) > 0$ for all $t \in \mathbb{N}$. A prefix $c_0, \ldots, c_n$ of an infinite path σ is called finite path, and in this case we write $\sigma = c_0, \ldots, c_n \cdot \star$.

For each state $c \in C$, a unique probability measure Pr^c is induced via the sigma-algebra generated from the finite prefixes of the paths in $\mathit{Paths}(c)$, by the cylinder set construction:

$$\mathsf{Pr}^c\{\sigma \in \mathit{Paths}(c) \mid \sigma = c_0, \ldots, c_t \cdot \star\} = \prod_{0 \le i < t} P(c_i, c_{i+1})$$

Applying this construction to the initial state $\hat{c}$ leads to the usual discrete-time stochastic process $X = \langle X_t, t \in \mathbb{N}\rangle$ associated with M, where the random variable X_t denotes the state occupied at time t. We write Pr instead of $\mathsf{Pr}^{\hat{c}}$, and use X^c for the stochastic process induced by $\langle C, P, c\rangle$, i.e., the MC where the initial state is c instead of $\hat{c}$.

In this work we focus on evaluating the latency of a NoC. To make this notion precise we identify two sets of states α and ω. The latency then corresponds to the number of time-steps required to reach a state in ω from a state in α. This is a time dependent quantity, since it may differ for different start times.

We include the case where α and ω are not disjoint: For a state $c \in \alpha \cap \omega$, one may imagine two possibilities to define the latency: either as zero or as the number of steps to reach ω after at least one step. We chose the latter approach, since the former seems irrelevant in practice: studying latencies is interesting only for operations incurring a non-zero delay, and it will become apparent in Sect. 4 that the case $\alpha \cap \omega$ is indeed a relevant one.

The time-dependent latency between states in α and ω will be defined in two steps: first, we introduce a notion of observation corresponding to the number time steps required to reach (from the current state) a state in ω. In a second step, we incorporate the set α. We fix a chain $M = \langle C, P, \hat{c}\rangle$ in the sequel.

Definition 2. *Let $O_{t_0}(\omega)$ be the random variable describing the number of steps, starting from time point t_0, before the first observation of a state in ω, defined as $O_{t_0}(\omega) = \min\{t \mid t > 0 \ \wedge \ X_{t_0+t} \in \omega\}$.*

The minimum over an empty set is defined to be ∞, which corresponds to the situation where a state of ω is never reached from X_{t_0}. If we consider the above definition for a different initial state c and thus X^c, we write $O^c_{t_0}(\omega)$.

For each time point t_0, we now define the latency to be $O_{t_0}(\omega)$ under the assumption that the chain currently resides in a state of α. In other words, the latency is defined as the number of time-steps between a state of α (start state) and a state of ω (stop state).

Definition 3. *The time-dependent latency $L_{t_0}(\alpha, \omega)$ from α to ω at time point t_0 is $L_{t_0}(\alpha, \omega) = O_{t_0}(\omega)$ if $X_{t_0} \in \alpha$ and 0 otherwise.*

When designing a system it is natural to look for the long-run behavior of the system, such as to take out the influence of initialization effects. One is thus usually interested not only in the time-dependent latency, but also in the latency perceived in the steady state equilibrium, thus the steady-state latency. We use the Cesáro limit construction to avoid periodicity considerations.

Definition 4. *The latency $L(\alpha, \omega)$ from α to ω is*

$$L(\alpha, \omega) = \lim_{t \to \infty} \frac{1}{t} \sum_{t_0=0}^{t} L_{t_0}(\alpha, \omega)$$

The Cesáro limit always exists for time-homogeneous MCs. The standard steady state limit $\lim_{t\to\infty} L_t(\alpha, \omega)$ may not exist, but agrees with the Cesáro limit if it does.

2.2 Computing latencies

We now discuss how to calculate the above measures for a given MC $\langle C, P, \hat{c} \rangle$.

Lemma 1. *For $t \in \mathbb{N}_{>0}$ the distribution of $O_{t_0}(\omega)$ is given by*

$$\mathsf{Pr}\big(O_{t_0}(\omega) = t\big) = \sum_{c \in \omega} \mathsf{Pr}\big(X_{t_0+t} = c \wedge (\forall i \in \{1, \ldots, t-1\})\ X_{t_0+i} \notin \omega\big)$$

The probability distribution of the latency $L_{t_0}(\alpha, \omega)$ can be derived from the probability distribution of $O_{t_0}(\omega)$. We only consider nonzero time points.

Lemma 2. *For all $t > 0$, the distribution of $L_{t_0}(\alpha, \omega)$ is given by:*

$$\begin{aligned}\mathsf{Pr}\big(L_{t_0}(\alpha, \omega) = t\big) &= \sum_{c_b \in \alpha} \mathsf{Pr}\big(X_{t_0} = c_b \mid X_{t_0} \in \alpha\big)\ \mathsf{Pr}\big(O_{t_0}(\omega) = t \mid X_{t_0} = c_b\big) \\ &= \sum_{c_b \in \alpha} \mathsf{Pr}\big(X_{t_0} = c_b \mid X_{t_0} \in \alpha\big)\ \mathsf{Pr}\big(O^{c_b}_0(\omega) = t\big)\end{aligned}$$

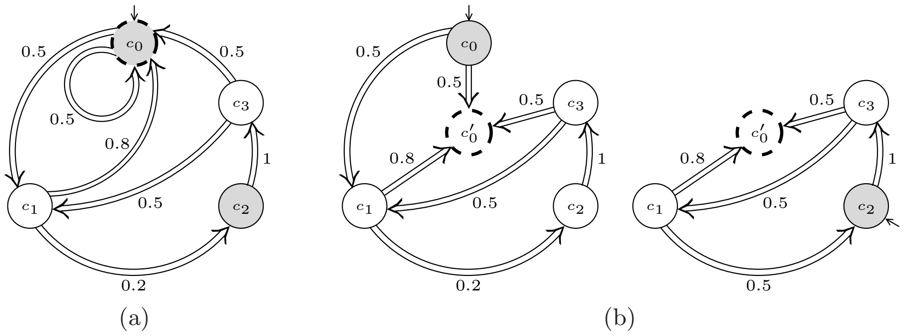


Fig. 2. Example of an MC and extracted sub-chains

The last equality is justified by the fact that in a time-homogeneous MC, each time point is a renewal point, and thus the probability of $O_{t_0}(\omega)$ is independent from the time t_0, if we fix a start state c_b. Therefore, we can compute it as $\mathsf{Pr}(O_0^{c_b}(\omega) = t)$. This, in turn, can be computed by considering the sub-chain consisting of all paths in $\mathit{Paths}(c_b)$.

Lemma 3. *For each state $c \in \alpha$, we have:*

$$\mathsf{Pr}\big(O_0^c(\omega) = t\big) = \sum_{\sigma \in \mathit{Paths}(c)} \mathsf{Pr}^c \left\{ \sigma \;\middle|\; \begin{array}{l} \sigma = c_0, \ldots, c_t \cdot \star \;\wedge\; c_t \in \omega \;\wedge \\ \big(\forall i \in \{1, \ldots, t-1\}\big)\; c_i \notin \omega \end{array} \right\}$$

The function associating $\mathsf{Pr}(O_0^{c_b}(\omega) = t)$ to each value of t is the distribution of the time to reach a state in ω given that the initial state is $c_b \in \alpha$. For our MC M, let $M^c(\omega)$ be the sub-chain spanning of all finite paths of M starting in c and ending at the first occurrence of a state $c' \in \omega$. By Lemma 3, the probability to reach an absorbing state of $M^c(\omega)$ in t time steps equals $\mathsf{Pr}(O_0^c(\omega) = t)$.

Example 1. For the MC of Fig. 2(a) and sets $\alpha = \{c_0, c_2\}$ (depicted as filled states) and $\omega = \{c_0\}$ (depicted with a dashed border), the two sub-chains $M^{c_0}(\omega)$ and $M^{c_2}(\omega)$ are shown in Fig. 2(b).

Now consider $L\big(\{c_b\}, \omega\big)$, the latency if starting in $c_b \in \alpha$. Since $\{c_b\}$ is a singleton, we have by Lemma 2 that $\mathsf{Pr}\big(L(\{c_b\}, \omega) = t\big) = \mathsf{Pr}(O_0^{c_b}(\omega) = t)$. Hence, this latency is time-independent and can be computed by reachability analysis of the absorbing states of $M^{c_b}(\omega)$. On the other hand, the probability of residing in a state $c_b \in \alpha$ under the assumption that M is in some state in α is a time-dependent quantity.

Lemma 4. *For each state $c_b \in \alpha$, we have:*

$$\mathsf{Pr}(X_{t_0} = c_b \mid X_{t_0} \in \alpha) = \frac{\mathsf{Pr}(X_{t_0} = c_b)}{\sum_{c \in \alpha} \mathsf{Pr}(X_{t_0} = c)}$$

The probability distribution of the steady-state latency $L(\alpha,\omega)$ (Def. 4) can now be computed using long-run averages of the states in M (steady-state may not exist, but if it does, it agrees with long-run average if existing). Let $\pi(c)$ denote the long run fraction of time spent in state $c \in C$. In vector-matrix notation, vector π is the unique probability distribution satisfying $\pi P = \pi$ (which always exists).

Lemma 5.

$$\mathsf{Pr}\big(L(\alpha,\omega)=t\big) = \sum_{c_b \in \alpha} \frac{\pi(c_b)}{\sum_{c\in\alpha}\pi(c)}\ \mathsf{Pr}\big(L(\{c_b\},\omega)=t\big)$$

The factor $\pi(c_b)/\sum_{c\in\alpha}\pi(c)$ is a normalization of the steady state probability of c_b over the set of states α. Consequently, the distribution of the latency $L(\alpha,\omega)$ is the sum of the normalized steady state probabilities of states of α, weighted by the distributions of the latencies starting in the states of α.

Example 2. For the MC chain shown in Fig. 2(a), with $\alpha = \{c_0, c_2\}$ (depicted as filled states) and $\omega = \{c_0\}$ (depicted with a dashed border), the latency distributions computed from the two extracted sub-chains starting in c_0 and c_2 shown in Fig. 2(b) are shown in Fig. 3. The distribution of $L(\alpha,\omega)$ (hashed bars on Fig. 3) is the sum of the normalized steady state probabilities of c_0 (9/10) and c_2 (1/10) weighted by those distributions.

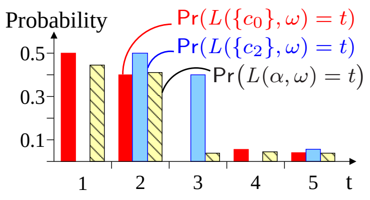


Fig. 3. Latency distributions

3 Compositional Modeling Approach

In this section, we introduce a compositional modeling approach merging functional and timed aspects, using probabilistic discrete-time distributions.

3.1 Interactive Probabilistic Chain

Our aim is to enable performance evaluation by enriching an existing formal model with timing information, following an approach similar to interactive Markov chains [4]. For the purpose of this paper, we use a probabilistic process calculus using the set of actions $\mathcal{A}$ (including the internal action τ). We assume that actions are taken instantaneously, and that every probabilistic choice takes exactly one time step.

A behavior is described using the following grammar (with $A \subset \mathcal{A} \setminus \{\tau\}$):

$$B \ ::= \ \delta \ \big|\ a\,;B \ \big|\ \textstyle\sum_i p_i :: B_i \ \big|\ B_1 \,[\,]\, B_2 \ \big|\ B_1 \,|[\,A\,]|\, B_2 \ \big|\ \text{hide } A \text{ in } B \ \big|\ \widetilde{B}$$

The operators will be referred to as: termination (δ), sequential composition (;), probabilistic choice ($\sum$, with the constraint $\sum_i p_i = 1$), non-deterministic choice ([]), LOTOS-style parallel composition with synchronization set ($|[\,A\,]|$), hiding of actions (hide A in $\cdots$), and process calls. A possibly recursive process is defined by a rule of the form $\widetilde{B} = B$. We write $\mathcal{B}$ to denote the set of all B.

$$\frac{}{\delta \xRightarrow{1} \delta} \qquad \frac{}{a;B \xrightarrow{a} B} \qquad \frac{}{a;B \xRightarrow{1} a;B} \qquad \frac{}{\textstyle\sum_i p_i{::}B_i \xRightarrow{p_i} B_i}$$

$$\frac{B_1 \xrightarrow{a} B_1'}{B_1[\,]B_2 \xrightarrow{a} B_1'} \qquad \frac{B_2 \xrightarrow{a} B_2'}{B_1[\,]B_2 \xrightarrow{a} B_2'} \qquad \frac{B_1 \xRightarrow{p_1} B_1' \quad B_2 \xRightarrow{p_2} B_2'}{B_1[\,]B_2 \xRightarrow{p_1\,p_2} B_1'[\,]B_2'}$$

$$\frac{B_1 \xrightarrow{a} B_1' \quad a \notin A}{B_1|[A]|B_2 \xrightarrow{a} B_1'|[A]|B_2} \qquad \frac{B_2 \xrightarrow{a} B_2' \quad a \notin A}{B_1|[A]|B_2 \xrightarrow{a} B_1|[A]|B_2'} \qquad \frac{B_1 \xrightarrow{a} B_1' \quad B_2 \xrightarrow{a} B_2' \quad a \in A}{B_1|[A]|B_2 \xrightarrow{a} B_1'|[A]|B_2'}$$

$$\frac{B_1 \xRightarrow{p_1} B_1' \quad B_2 \xRightarrow{p_2} B_2'}{B_1|[A]|B_2 \xRightarrow{p_1\,p_2} B_1'|[A]|B_2'} \qquad \frac{\widetilde{B}=B \quad B \xrightarrow{a} B'}{\widetilde{B} \xrightarrow{a} B'} \qquad \frac{\widetilde{B}=B \quad B \xRightarrow{p} B'}{\widetilde{B} \xRightarrow{p} B'}$$

Fig. 4. Operational semantics of the modeling language

Example 3. Consider the four processes $\widetilde{B}_1$, $\widetilde{B}_2$, $\widetilde{B}_3$, and $\widetilde{B}_4$, defined over the set of actions $\mathcal{A} = \{a_1, a_2, \tau\}$ by:

$$\widetilde{B}_1 = a_1\,;\widetilde{B}_2 \quad \widetilde{B}_2 = \sum \begin{array}{l} 0.5 :: \widetilde{B}_3 \\ 0.5 :: \widetilde{B}_4 \end{array} \quad \widetilde{B}_3 = a_2\,;\widetilde{B}_1 \quad \widetilde{B}_4 = \sum \begin{array}{l} 0.8 :: \widetilde{B}_3 \\ 0.2 :: \big(a_1\ ;\textstyle\sum 1 :: \widetilde{B}_2\big) \end{array}$$

The semantics of this language is defined in a structured operational semantics style as a probabilistic extension of a labeled transition system, called Interactive Probabilistic Chain (IPC).

Definition 5. *An IPC is a quintuple $D = \langle S, \mathcal{A}, \longrightarrow, \Longrightarrow, \hat{s}\rangle$ where S is a finite set of states, $\mathcal{A}$ is a finite set of actions including the internal action τ, $\longrightarrow \subset S \times \mathcal{A} \times S$ is a set of interactive transitions, $\Longrightarrow \subset S \times]0,1] \times S \to \mathbb{N}$ is a multi-set of probabilistic transitions, and $\hat{s} \in S$ is the initial state.*

We write $\mathcal{D}$ to denote the set of all IPCs over $\mathcal{A}$.

Definition 6. *The operational semantics of a behavior B_0 over the set of actions $\mathcal{A}$ is defined as the IPC $D = \langle \mathcal{B}, \mathcal{A}, \longrightarrow, \Longrightarrow, B_0\rangle$ where $\longrightarrow$ and $\Longrightarrow$ are defined by the inference rules of Fig. 4.*

The third rule expresses the *arbitrary waiting* property: a process may be blocked waiting for a synchronization that is arbitrarily long (even infinitely) while still letting time advance. All these rules are partly inspired by Hansson [3], and as a whole enforce that time advances synchronously and that it may always advance.

We also require the property of *minimal delay* or *maximal progress* [9]: a process cannot delay an internal transition. In other words, if we have a probabilistic transition in competition with a τ-transition, the probabilistic transition incurs a time-step, while the τ-transition is possible immediately and will not be delayed. So a τ-transition has precedence over any probabilistic transition. However, this is not integrated into the semantics, but is taken care of by the bisimulation equivalences we are defining below.

Example 4. The IPC corresponding to Example 3 is shown in Fig. 5(a).

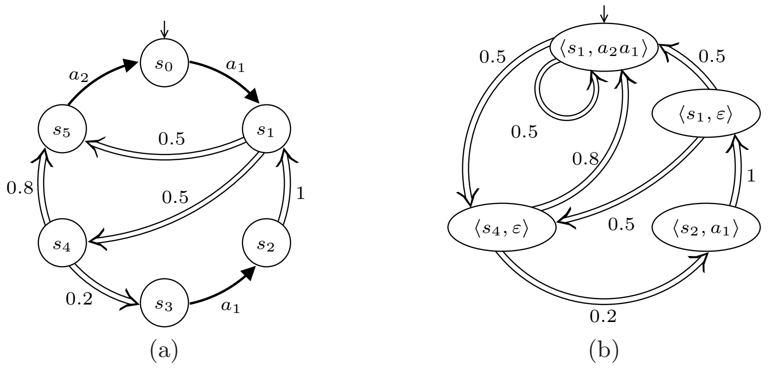

s_0
a_1
a_2
s_5
s_1
0.5
0.8
0.5
1
s_4
s_2
0.2
s_3
a_1
(a)
$\langle s_1, a_2a_1\rangle$
0.5
0.5
$\langle s_1, \varepsilon\rangle$
0.5
0.8
1
$\langle s_4, \varepsilon\rangle$
0.5
$\langle s_2, a_1\rangle$
0.2
(b)

Fig. 5. Example of an IPC and its associated MC

3.2 Probabilistic Bisimulations

For a regular expression e over $\mathcal{A}$, let $s_0 \xrightarrow{e} s_n$ denote that there exists a sequence of transitions $s_0 \xrightarrow{a_1} \dots \xrightarrow{a_n} s_n$ where $a_1 \dots a_n$ is a word in the language over $\mathcal{A}$ defined by e. We define a predicate $\gamma_0(s, a, S')$ which holds if and only if there is an $s' \in S'$ such that $s \xrightarrow{a} s'$. Moreover, we define a function $\gamma_P : S \times 2^S \mapsto [0, 1]$, that cumulates the probability of reaching a set of states S' from a single state s, by $\gamma_P(s, S') = \sum_{(n,p)\in M(s,S')} n\, p$, where $M(s, S')$ is the largest set satisfying $(n', p') \in M(s, S')$ if and only if $|\{s \overset{p'}{\Longrightarrow} s' \mid s' \in S'\}| = n'$. We use $s \overset{\tau}{\not\longrightarrow}$ to abbreviate $\neg\gamma_0(s, \tau, \mathcal{D})$. Finally, we write $\mathcal{D}/_E$ to denote the set of equivalence classes of $\mathcal{D}$ with respect to relation E. These ingredients are needed to define bisimulations along the lines of [8, 5].

Definition 7. *Strong probabilistic bisimulation equivalence ($\sim$) is the coarsest equivalence relation on $\mathcal{D}$ such that $s_1{\sim}s_2$ implies for all $a \in \mathcal{A}$ and all $C \in \mathcal{D}/_{\sim}$:*

- $\gamma_0(s_1, a, C) \;\Rightarrow\; \gamma_0(s_2, a, C)$,
- $s_1 \overset{\tau}{\not\longrightarrow} \;\Rightarrow\; (\; s_2 \overset{\tau}{\not\longrightarrow} \wedge \gamma_P(s_1, C) = \gamma_P(s_2, C)\;)$.

When relating MCs and IPCs, we need to abstract from internal computation. Thus, we use a weaker notion of equivalence, branching bisimulation, here lifted to IPC [14, 5].

Definition 8. *Branching probabilistic bisimulation equivalence ($\approx$) is the coarsest equivalence relation on $\mathcal{D}$ such that $s_1 \approx s_2$ implies for all $a \in \mathcal{A}$ and all $C \in \mathcal{D}/_{\approx}$:*

- $\gamma_0(s_1, a, C) \;\Rightarrow\; \big((\exists s_2')\; s_2 \xrightarrow{\tau^*} s_2' \wedge s_1{\approx}s_2' \wedge \gamma_0(s_2', a, C)\big) \vee (a, s_2) \in \{\tau\}{\times}C$,
- $s_1 \overset{\tau}{\not\longrightarrow} \;\Rightarrow\; \big((\exists s_2' \overset{\tau}{\not\longrightarrow})\; s_2 \xrightarrow{\tau^*} s_2' \wedge s_1 \approx s_2' \wedge \gamma_P(s_1, C) = \gamma_P(s_2', C)\big)$.

Strong and branching bisimulation equivalence enjoy all the usual properties of bisimulation-style relations. In particular, branching probabilistic bisimulation is a congruence with respect to the parallel composition operator. This is exploited in our model construction process, where we replace components of a large system by equivalent, but smaller ones. In addition, strong and branching bisimulation equivalence preserve the probabilistic behavior, in the sense that whenever two IPCs s_1, s_2 are equivalent, then for each resolution of the nondeterminism in s_1 there is a resolution of the nondeterminism in s_2 (and vice versa) such that the resulting MCs have the same transient and steady-state behavior, if one only looks at probabilities of the equivalence classes. The relation between determinized IPC and MC is the topic of the following section.

4 Performance Analysis of a deterministic IPC

In general, an IPC is a non-deterministic object, and non-determinism is an essential aid in building interacting systems from components. The resulting system however often shows deterministic behavior modulo branching bisimulation equivalence. Indeed, abstracting from all functional information (i.e., renaming them into τ) and minimizing with respect to branching bisimulation equivalence, *always* yields a deterministic system, though a very simple one, namely a single state with a probabilistic transition with probability 1. This is a trivial MC, and, of course, not very a insightful one. We will describe below how we keep precisely the information needed. Though this is only a partial solution, we focus in the analysis on deterministic non-zeno IPCs (dIPCs), and discuss the general case towards the end of this section.

Definition 9. *A deterministic IPC is an IPC $\langle S, \mathcal{A}, \longrightarrow, \Longrightarrow, \hat{s}\rangle$ satisfying:*

$$(\forall s, s', s'' \in S, a, a' \in \mathcal{A}) \quad (s \xrightarrow{a} s') \wedge (s \xrightarrow{a'} s'') \text{ implies } (s' = s'') \wedge (a = a')$$
$$(\forall s, s' \in S, a \in \mathcal{A}) \quad (s \xrightarrow{a} s') \text{ implies } (\exists s'', s''' \in S, w \in \mathcal{A}^*)\; s' \xrightarrow{w} s'' \overset{p}{\Longrightarrow} s'''$$

The above conditions guarantee that only finite, linear sequences of interactive transitions appear in a dIPC. When analyzing a complete system, we consider it *closed*, not interacting with the environment. Under this assumption, the maximal progress property generalizes to all actions, and is called *urgency* [9].

Definition 10. *An IPC $D = \langle S, \mathcal{A}, \longrightarrow, \Longrightarrow, \hat{s}\rangle$ is said to be urgency-cut iff*

$$(\forall s, s' \in S, a \in \mathcal{A}) \quad s \xrightarrow{a} s' \text{ implies } (\nexists s'' \in S)\; s \overset{p}{\Longrightarrow} s''$$

Example 5. The IPC shown in Fig. 5(a) is deterministic and urgency-cut.

We apply the urgency property by moving from a dIPC to the largest urgency-cut dIPC contained therein. Transforming the resulting dIPC into an MC must allow us to keep information about some of the interactive transitions. To this end, we enrich states of the MC with a word (over $\mathcal{A}$) representing the (possibly empty) sequence of actions executed since the last time step, i.e., last probabilistic transition. We let ε stand for the empty word, representing two successive probabilistic transitions.

Definition 11. *Let $D = \langle S, \mathcal{A}, \longrightarrow, \Longrightarrow, \hat{s}\rangle$ be an urgency-cut dIPC over $\mathcal{A}$. Let k be the length of the longest sequence of interactive transitions in D. The associated MC $M(D) = \big\langle C, P, \langle \hat{s}, \varepsilon\rangle\big\rangle$ is given by:*

– $C = \big\{s \in R(D) \mid (\exists s')\ s \overset{p}{\Longrightarrow} s'\big\} \times \mathcal{A}^{\leq k}$*, where $R(D)$ is the set of reachable states in D, and $\mathcal{A}^{\leq k} = \{\mathsf{w} \mid \mathsf{w} \in \mathcal{A}^* \wedge length(\mathsf{w}) \leq k\}$.*
– $P\big(\langle s, \mathsf{w}\rangle, \langle s', \mathsf{w}'\rangle\big) = \sum_{\{i \mid (\exists s'') s \overset{p_i}{\Longrightarrow} s'' \overset{\mathsf{w}'}{\longrightarrow} s'\}} p_i$

Notice that the transition probabilities from a state $\langle s, \mathsf{w}\rangle$ to a state $\langle s', \mathsf{w}'\rangle$ are accumulated from all possibilities to move from IPC state s — after one time step — to IPC state s' according to interactive transitions forming the word w'. The urgency cut ensures that in each state either one can spend time or one can engage in interactive actions, but not both.

Example 6. Consider the IPC D of Example 4. The associated MC $M(D)$ is shown in Fig. 5(b) and corresponds to the one of Example 1.

4.1 Computing Latency Distribution for dIPC

For a dIPC D, we define a latency $L_{IPC}(\mathsf{start}, \mathsf{stop})$ as the number of time steps between an action start (beginning of the latency) and stop (end of the latency). Considering the sets of states where a transition start (respectively stop) is possible, we can define the latency using the MC $M(D)$ and Def. 4.

Definition 12. *Let $D = \langle S, \mathcal{A}, \longrightarrow, \Longrightarrow, \hat{s}\rangle$ be a dIPC. The latency between two actions $\mathsf{start}, \mathsf{stop} \in \mathcal{A}$ is defined on the associated MC $M(D)$:*

$$L_{IPC}(\mathsf{start}, \mathsf{stop}) = L(\alpha, \omega)$$

where $\alpha = \big\{\langle s, \mathsf{w}\rangle \mid \mathsf{w} = (A')^*.\mathsf{start}.\mathcal{A}^*\big\}$, $\omega = \big\{\langle s, \mathsf{w}\rangle \mid \mathsf{w} = (A')^*.\mathsf{stop}.\mathcal{A}^*\big\}$, and $A' = \mathcal{A} \setminus \{\mathsf{start}, \mathsf{stop}\}$.

Notice that, similar to the definitions of Sect. 2, Def. 12 does not consider zero-latencies, i.e., a start followed by an stop in the same time step. Indeed, for each state $c = \langle s, \mathsf{w}\rangle$ of $M(D)$ with w of the form $(A')^*.\mathsf{start}.\mathcal{A}^*.\mathsf{stop}.\mathcal{A}^*$, we have $c \notin \omega$. However, for each state $c' = \langle s, \mathsf{w}\rangle$ of $M(D)$ with w of the form $(A')^*.\mathsf{stop}.\mathcal{A}^*.\mathsf{start}.\mathcal{A}^*$, we have $c' \in \alpha \cap \omega$: this allows to take into account latencies that follow each other immediately.

Example 7. Consider the dIPC and its associated MC of Fig. 5. We are interested in the latency between actions a_1 and a_2. The associated MC with sets $\alpha = \{\langle s_1, a_2 a_1\rangle, \langle s_2, a_1\rangle\}$ and $\omega = \{\langle s_1, a_1 a_2\rangle\}$ is identical to the MC of Fig. 2(a) with sets α and ω respectively depicted as filled states and dashed border states.

The restriction to dIPCs is a partial solution. But that does not mean that we cannot make use of the non-deterministic language constructs (choice, interleaving parallel composition) in our modeling. In practice, we construct an

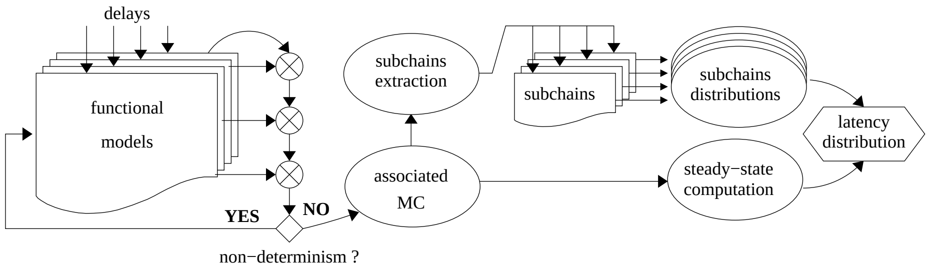


Fig. 6. Performance Evaluation Flow

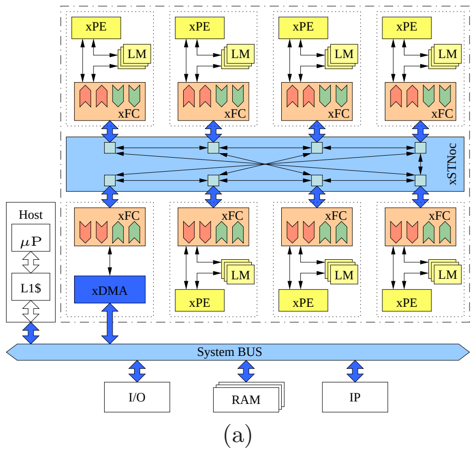
IPC using the non-deterministic constructs to let the system components interact as intended. Once the specification is complete, we identify the start and stop actions, and regard any other action as τ. We then move to the branching bisimulation quotient. In many cases this quotient is deterministic. The reason is that, intuitively, the non-deterministic branches are confluent up to branching bisimulation. In this case we can proceed with the resulting dIPC as suggested above. Otherwise, we need to (manually) refine the model, in order to resolve that non-determinism. This approach to handling nondeterminism via a quotient construction is akin to the one used in IMC modeling via CTMCs [4].

4.2 Implementation considerations

Following these ideas, we implemented the flow depicted in Fig. 6 to compute the distribution of a latency between two actions start and stop of a system consisting of several sub-components.

First, functional models of sub-components of our system are enriched with delays, modeled as discrete-time probabilistic distributions. The sub-components are then composed according to the semantic rules given in Sect. 3.1 to get an IPC (the state space of which is minimized wrt branching bisimulation using the `bcg_min` tool of the CADP toolbox [2]). Before computing performance results, we ensure the IPC is deterministic. The associated MC is generated from the dIPC according to Def. 11.

Thus, an implementable way to compute the distribution of the latency $L_{IPC}(\mathsf{start}, \mathsf{stop})$ is provided by Def. 12 and Lemma 5. For each state $c_b \in \alpha$ (where α is defined according to Def. 12): on one hand, the distribution of the latency $L\big(\{c_b\}, \alpha\big)$ can be obtained by extracting the sub-chain $M^{c_b}(\omega)$ and computing the distribution of the number of time steps needed to reach an absorbing state; on the other hand, steady state (actually long-run average) analysis yields $\pi(c_b)/\sum_{c\in\alpha}\pi(c)$. The distribution of the latency $L_{IPC}(\mathsf{start}, \mathsf{stop})$ is then computed as the weighted sum given by Lemma 5.



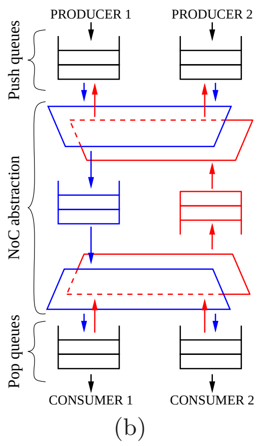


Fig. 7. xSTream architecture and the model of two parallel streams

5 Case study: the xSTream architecture

In the context of the Multival project [1], we investigated the xSTream architecture. xSTream is a multiprocessor data-flow architecture for high-performance embedded multimedia streaming applications designed at STMicroelectronics. xSTream supports stream-oriented programming, where an application performs several computation steps, called *filters*, on a stream of data. Each filter is mapped to a *processing element* (xPE) (i.e., a processor with some local memory), communicating over a *NoC* (xSTNoc). Several filters might be mapped to a single processing element according to its workload. To absorb traffic bursts, each processing element is linked to the NoC through buffering hardware queues for input flows (called *pop queues*) and for output flows (called *push queues*). A *flow controller* (xFC) manages the different push and pop queues for each processing element. The xSTream architecture is depicted on Fig. 7(a).

Operations on xSTream queues are blocking: a Push operation (insertion of an element in a queue) is blocked until there is a free place and a Pop operation (removal of an element from a queue) is blocked until an element is available. A queue stores its elements either in its dedicated hardware or in a *backlog* (LM) (i.e., the memory of the processing element) — the latter being much less efficient. Due to xSTream specific operations on these hardware queues, they are significantly more complex than standard FIFO queues: Each of them has between 1300 and 2400 states, for a capacity of up to three elements.

We focussed on the interaction between pairs of xSTream queues. Therefore, we developed a LOTOS [6] model of two parallel streams between two processing elements using two shared queues as an abstraction of the NoC, according to the virtual channel concept as defined in xSTream. This model is depicted on Fig. 7(b). This application scenario (two consumer-producer pairs) is not unrealistic, since xSTream is targeted at streaming applications.

Using the CADP toolbox [2], we analyzed the correct functioning of the model, highlighting some issues. For example, an under-specification concerning a credit protocol has been found. This protocol aims to avoid deadlocks in the NoC and was claimed optional by architects. Analysis of our model showed that it was mandatory. This protocol ensures that a push queue does not send more elements than there are free places in the corresponding pop queue. It is based on counters local to queues and a stream from pop queues to push queues (opposite to data streams, see Fig. 7(b)) used to update the counters according to a defined threshold. As this protocol adds communication, it may influence performance.

The LOTOS model is composed of several sub-components: queues, a NoC abstraction, network interfaces between queues and NoC. Performance evaluation of this kind of system depends on applications running on it. We added producers and consumers for modeling these applications. The sub-components have been enriched with delays for insertion and removal of elements in queues and delays modeling the application.

We focus on the study of the mean latency of a Pop operation, i.e., the mean time needed to get an element from the pop queue. The study of the distribution of the Pop operation latency could give us more information concerning bounds and borderline behaviors. In this case-study, we only consider mean values which can be (and have been) confirmed by simulation.

The Pop operation latency has a theoretical lower bound corresponding to the time physically needed to access the pop queue's head. In the case of an empty pop queue, the Pop operation is blocked until an element is available. In a streaming application, the mean duration of a Pop operation should be close to its minimum value. Indeed, queues are dedicated to absorb traffic variations, and to hide communication latencies. A mean duration of a Pop operation that is much greater than its minimum value indicates that pop queues are often in a starvation context and do not fulfill their task.

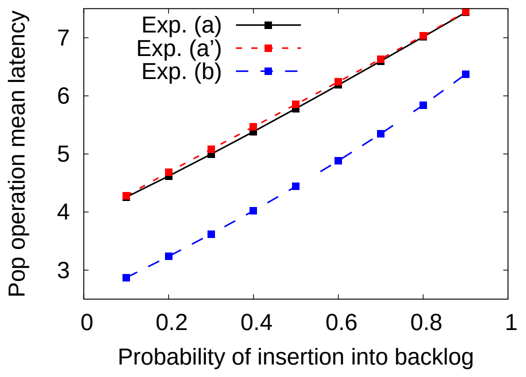
Using our prototypical tool-chain, we performed three experiments varying the applications (production and consumption rates) and the credit protocol threshold. The first two experiments use the worst possible threshold (i.e., a threshold equal to the pop queue size): hence the push queue is often blocked (it has no more credit left). Experiments (a) and (a′) differ in the delay between the generation and consumption of two packets: Exp. (a) uses the distribution shown in Fig. 1, whereas Exp. (a′) uses a similar distribution, but with higher mean value. Experiment (b) differs from Exp. (a) by using a larger pop queue and a threshold strictly smaller than the pop queue size. For all three experiments, the delay to insert an element in the pop queue was abstracted by a probabilistic distribution: insertion of an element took either one time step (insertion in the queue) or five time steps (insertion in the backlog). Figure 8(a) gives the sizes of IPCs and MCs, and Fig. 8(b) shows the mean latency of the three experiments for different probability values to insert in the backlog.

Experiments (a) and (a′) are similar: the throughput of applications has indeed no real influence on the Pop operation latency. However, using the worst case for the threshold impacts the Pop latency (its theoretical lower bound is 1.5

Exp.	IPC size	MC size
(a)	9,238,425 39,741,107	207,071 822,445
(a′)	20,362,426 82,846,047	379,656 1,487,138
(b)	19,719,588 92,093,649	from 232,976 1,173,701 to 235,203 1,186,162

sizes in states/transitions

(a)



(b)

Fig. 8. Experimental results

packets every time step on average). Experiment (b) confirms that using a lower threshold reduces the latency. Finally, for all experiments, we see that the lower the probability to insert in the backlog, the better the performance.

The results are showing trends that one would indeed expect for a single pair of queues. But it is not obvious that these trends are actually observed for a system consisting of two pairs of queues sharing the same virtual channels. Thus, the results confirm the architectural choices.

6 Related Work

There is a plethora of probabilistic process calculi around in the literature [7], none of which served our purposes, because we needed a synchronous time model as in ATP [10], which incorporates probabilistic branching. The calculus is most inspired by the alternating model of Hansson [3]. For this model a branching bisimulation has been studied in [13], which is similar in spirit to ours, but can amalgamate sequences of probabilistic steps. This is not compatible with our setting where probabilistic steps are time steps.

Our maximal progress and compositionality considerations are akin to those in IMC [4]. One might consider IPC as the discrete time analogon of IMC, and indeed we are re-using for IPC the IMC branching bisimulation minimizer of CADP.

Looking superficially at the example case one may think of (discrete time) queueing networks and related models as a possible alternative. However, standard queueing networks are usually too abstract to incorporate the complex behavior of the circuit. If one indeed invests in extending the queueing network setting to incorporate this hardware behavior, there seems to be no obvious advantage over the approach presented by us.

7 Conclusion

The industrial practitioners are asking for new recipes to study the performance of SoCs and NoCs. In this paper we introduced interactive probabilistic chains to enable compositional modeling, functional verification, and performance evaluation, the latter via a translation into Markov chains. This is a partial solution approach, because in general the resulting model is a Markov decision process. We applied this approach to compute latencies in an industrial case study.

Though the theory is solid and prototype tool support is available, a lot remains to be achieved before the issue of model-based performance evaluation can be closed. As a concrete future challenge, we need to improve the efficiency of our prototypical performance evaluation flow, and we plan to apply it to further examples and performance measures. We are also thinking of combining it with simulative approaches. It appears possible to extend our analysis approach to the general IPC setting, because the principal algorithmic building blocks, working on Markov decision processes, are known [11].

References

1. N. Coste, H. Garavel, H. Hermanns, R. Hersemeule, Y. Thonnart, and M. Zidouni. Quantitative evaluation in embedded system design: Validation of multiprocessor multithreaded architectures. In *DATE*, Mar. 2008.
2. H. Garavel, F. Lang, R. Mateescu, and W. Serwe. CADP 2006: A toolbox for the construction and analysis of distributed processes. In *CAV*, LNCS 4590, pp. 158–163, 2007.
3. H. A. Hansson. *Time and Probability in Formal Design of Distributed Systems*. Elsevier Science, 1994.
4. H. Hermanns. *Interactive Markov Chains and the Quest for Quantified Quality*, LNCS 2428. Springer, 2002.
5. H. Hermanns and M. Lohrey. Priority and maximal progress are completely axiomatisable (extended abstract). In *CONCUR*, LNCS 1466, pp. 237–252, 1998.
6. ISO/IEC. LOTOS — a formal description technique based on the temporal ordering of observational behaviour. International Standard 8807, 1989.
7. B. Jonsson, W. Yi, and K. G. Larsen. *Probabilistic Extensions of Process Algebras*, chapter 11, pages 685–710. North Holland, 2001.
8. K. G. Larsen and A. Skou. Compositional verification of probabilistic processes. In *CONCUR*, LNCS 630, pp. 456–471, 1992.
9. X. Nicollin and J. Sifakis. An overview and synthesis on timed process algebras. In *CAV*, LNCS 575, pp. 376–398, 1991.
10. X. Nicollin and J. Sifakis. The algebra of timed processes ATP: Theory and application. *Information and Computation*, 114(1):131–178, 1994.
11. M. L. Puterman. *Markov Decision Processes*. Wiley, 1994.
12. K. L. Swartz, R. L. Cottrell, and M. Dart. Adventures in the evolution of a high-bandwidth network for central servers. In *LISA*. USENIX, 1994.
13. N. Trčka and S. Georgievska. Branching bisimulation congruence for probabilistic systems. In *QAPL'08*, *Electronic Notes in Theoretical Computer Science*, 220:129–143, 2008.
14. R. J. van Glabbeek and W. P. Weijland. Branching time and abstraction in bisimulation semantics (extended abstract). In *IFIP congress*, pp. 613–618, 1989.