Performance Evaluation of Complex Systems Using the SBIP Framework

Ayoub Nouri[1], Marius Bozga[2], Axel Legay[3], and Saddek Bensalem[2]

[1] Univ. Grenoble Alpes, F-38000 Grenoble, France
CEA, LETI, F-38054 Grenoble, France
[2] Univ. Grenoble Alpes, VERIMAG, F-38000 Grenoble, France
CNRS, VERIMAG, F-38000 Grenoble, France
[3] INRIA, Rennes, France

Abstract. In this paper we survey the main experiments performed using the SBIP framework. The latter consists of a stochastic component-based modeling formalism and a probabilistic model checking engine for verification. The modeling formalism is built as an extension of BIP and enables to build complex systems in a compositional way, while the verification engine implements a set of statistical algorithms for the verification of qualitative and quantitative properties. The SBIP framework has been used to model and verify a large set of real life systems including various network protocols and multimedia applications.

1 Introduction

Probabilistic model checking is an automated verification method used for systems with stochastic behavior [2]. Recently, a statistical approach was proposed to overcome scalability issues occuring in numerical methods that are classically used to check such systems. This novel technique, called Statistical Model Checking (SMC) [28, 12], requires, as in classical model checking, to build an operational formal model of the system to verify and to provide a formal specification of the property to check, generally using temporal logic. The idea is then to explore a sample of execution traces produced through discrete event simlation in order to verify if the property holds on the system under consideration. Statistical Model Checking is receiving increasing attention and is being applied for a wide range of verification problems occurring in biology [10], communication protocols [4], multimedia [3], avionics [6], etc.

SBIP provides an extension of the BIP (Behavior, Interaction, Priority) framework [7] that allows stochastic modeling and statistical verification. On one hand, it relies on BIP expressiveness to handle heterogeneous and complex component-based systems. On the other hand, it uses SMC techniques to perform quantitative verification targeting non-functional properties.

The framework implements both *hypothesis testing* [28] and *probability estimation* techniques [12] such as similar existing tools [14, 26, 16, 11, 9]. Some other related tools provide in addition a distributed version of the statistical tests like [1, 14, 29, 13] and moreover implement numerical or hybrid methods [16]. The

main difference between the mentioned tools is the system modeling and the properties specification formalisms.

For instance, Uppaal-smc [11] supports *Priced Timed Automata* (PTAs) for system modeling and *Weighted Metric Temporal Logic* (WMTL) for properties specification, while Prism [16] considers *Discrete/Continuous Time Markov Chains* (DTMCs/CTMCs), *Markov Decision Process* (MDPs) and recently PTAs for the modeling part and *Probabilistic Computation Tree Logic* (PCTL), *Continuous Stochastic Logic* (CSL), *Linear-time Temporal Logic* (LTL), and PCTL* as properties input language. Other tools like Vesta [26] support, in addition to D/CTMC, algebraic specification languages like PMaude [15]. PlasmaLab [14] is a modular and extensible statistical model checker that may be extended with external simulator and checkers. The default configuration accepts discrete-time models specified in the Prism format and requirements expressed in PBLTL [22]. Ymer [29] is one of the first tools to implement sequential hypothesis testing algorithms. It considers GMSPs and CTMCs specified using an extension of the Prism language and accepts both PCTL and CSL for requirements specification.

SBIP relies on the stochastic extension of BIP [22, 21], which enables for describing DTMCs and MDPs in a component-based way, for the system modeling. For properties specification, it uses probabilistic bounded LTL. In addition, the SBIP models used for analysis can be equally used to generate concrete implementation to be deployed on real platforms. Implementations are guaranteed to be correct, i.e. preserve the properties established during analysis [5].

Outline. In section 2, we present the stochastic BIP formalism for system modeling, and the property specification language using temporal logic. Technical details about the implementation and the structure of the SMC engine are provided in section 3. In section 4, we survey the main case studies realized using the SBIP framework. Finally, section 5 concludes the paper.

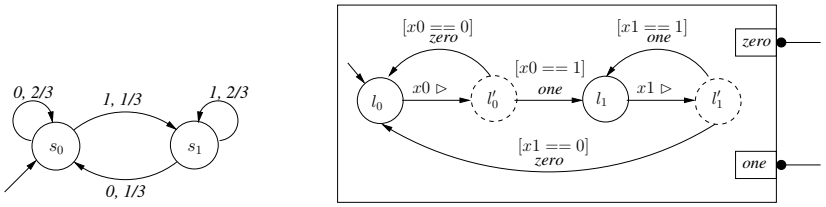
2 Model Specification in SBIP

BIP (Behavior, Interaction, Priority) is a highly expressive component based framework for rigorous system design [7]. It allows the construction of complex, hierarchically structured models from atomic components characterized by their behavior and their interfaces. Such components are transition systems enriched with variables. Transitions are used to move from a source to a destination location. Each time a transition is taken, component variables may be assigned new values, computed by user-defined C/C++ functions.

Component composition in BIP is expressed by layered application of interactions and of priorities. Interactions express synchronization constraints between actions of the composed components while priorities are used to filter amongst possible interactions e.g. to express scheduling policies.

SBIP extends BIP with a new semantics (see [8, 22, 21] for details) that enables modeling stochastic systems. The aforementioned extension is made through a C++ library integrated to the BIP framework [7]. It enables the definition of stochastic components that have probabilistic variables. The latter

could be defined with respect to both empirical and standard probability distributions. SBIP allows to build two types of models: DTMCs and MDPs that are modeled as classical BIP components augmented with probabilistic variables as shown in Figure 1.



(a) A DTMC of a binary values generator (b) Corresponding SBIP model

Fig. 1: Example of a DTMC model in SBIP.

Figure 1a shows a DTMC model for binary values generation. It has two states s_0 and s_1 where it generates 0 and 1 respectively through actions modeled here using transitions. Each transition in the figure has a label (0 or 1) and is associated with a probability to be fired. Figure 1b shows a graphical representation of the equivalent SBIP model. In SBIP, the next state probability distributions of the DTMC are captured by the probabilistic variables $x0$ and $x1$ which take values in $\{0, 1\}$ with respect to the following probability distributions: $x0$ is assigned 0 with probability $\frac{2}{3}$ and 1 with probability $\frac{1}{3}$. Similarly, variables $x1$ takes 0 or 1 with probabilities $\frac{1}{3}$ and $\frac{2}{3}$ respectively. This transformation associates each transition in the original DTMC with two transitions in the SBIP model. The first is a sampling step over the next state distribution ($x0 \triangleright$) and the second is a selection step using guards (the expressions between brackets $[x0 == 0]$) which are Boolean expressions defining transitions enableness.

As an example, the binary generator component illustrated in Figure 1b is described in the SBIP language as shown in Figure 2.

Besides empirical discrete distributions (defined using external text files, e.g. "dist_0.txt"), the SBIP modeling language allows using predefined standard distributions, such as *Uniform*, *Normal*, *Exponential*, etc. For a *Uniform* distribution, the *select()* function (used to sample the probability distributions in Figure 2) could be called without initialization phase (the *init_distribution()* in Figure 2) and by providing it with interval bounds as parameters. For instance, $select(100, 500)$ will uniformly sample values in the interval $[100, 500]$.

In addition to probabilistic helper functions, the library provides tracing capabilities that are required to monitor state variables involved in the property to check. In the previous example, assume that $x0$ is subject to verification, then the following function call should be used in order to monitor it:

```
trace_i(‘‘binary_generator.x0’’, x0);
```

```
/* Declaration of an atomic component */
atomic type binary_generator
  /* Declaration of a probabilistic variable */
  data int x0
  ...
  /* Declaration of a probabilistic distribution */
  data distribution_t dist_0
  ...
  /* Declaration and export of a port */
  export port intPort zero()
  ...
  /* Declaration of control locations */
  place l0, l1, l0', l1'
  initial to l0 do {
    /* Initialize dist. variables from empirical probability dist. */
    dist_0 = init_distribution(``dist_0.txt'');
    ... }
  ...
  /* Transition from l0 to l0' */
  internal from l0 to l0' do {
    /* Update x0 using dist_0 */
    x0 = select(dist_0); }
  /* Transition from l0' to l0 */
  on zero from l0' to l0 provided (x0 == 0)
  /* Transition from l0' to l1 */
  on one from l0' to l1 provided (x0 == 1)
  ...
end
```

Fig. 2: Description of the binary values generator in the SBIP language.

2.1 Properties Specification in SBIP

The properties specification language over stochastic systems in SBIP is a probabilistic variant of bounded Linear-time Temporal Logic (LTL). Using this language, it is possible to formulate two type of queries on a given system:

- Qualitative queries : $\mathbf{P}_{\geq\theta}[\varphi]$, where $\theta \in [0,1]$ is a probability threshold and φ is a bounded LTL formula (also called path formula).
- Quantitative queries : $\mathbf{P}_{=?}[\varphi]$, where φ is a bounded LTL formula.

Note that it is possible through those queries to either ask for the actual probability of a property φ to hold on a system (using the second type of queries) or to determine if the property satisfies some threshold θ (using the first type).

Path formulas, in SBIP, are defined using four bounded temporal operators namely, Next ($\mathsf{N}\psi_1$), Until ($\psi_1 \mathsf{U}^{bound} \psi_2$), Eventually ($\mathsf{F}^{bound}\psi_1$), and Always ($\mathsf{G}^{bound}\psi_1$), where *bound* is an integer value that specifies the length of the considered system trace and ψ_1, ψ_2 are called state formulas, that is Boolean

predicates evaluated on the system states. For example, the PBLTL formula

$$\mathbf{P}_{=?}[\mathsf{G}^{1000}(abs(Master.tm - Slave.ts) \leq 160)]$$

is equivalent to ask "What is the probability that the absolute value of the difference between master variable tm and slave variable ts is always under the bound 160 ?". In this example, the path formula is $\mathsf{G}^{1000}(abs(Master.tm - Slave.ts) \leq 160)$ and the state formula is $abs(Master.tm - Slave.ts) \leq 160$. Note that SBIP gives the possibility to use built-in predefined mathematical functions in state formulas. For the example above, $abs()$ function is called to compute the absolute value of the difference between variables tm and ts.

3 The SMC Engine: BIP$^{\mathcal{SMC}}$

The SMC engine implements several statistical testing algorithms for stochastic systems verification, namely, Single Sampling Plan (*SSP*), Simple Probability Ratio Test (*SPRT*) [27, 28], and Probability Estimation (*PESTIMATION*) [12]. Figure 3 shows the most important modules of the tool and how they interact in order to perform statistical model checking.

The tool takes as inputs a stochastic model description in the stochastic BIP format, a PBLTL property to check, and a set of confidence parameters required by the statistical test.

During an initial phase, the tool performs a syntactic validation of the PBLTL formula through a parser module. Then, it builds an executable model and a monitor for the property under verification. Next, it will iteratively trigger the stochastic BIP engine to generate execution traces which are monitored to produce local verdicts. This procedure is repeated until a global decision can be taken by the SMC core module (that implements the statistical algorithms). As our approach relies on SMC and since it considers bounded LTL properties, we are guaranteed that the procedure will eventually terminate.

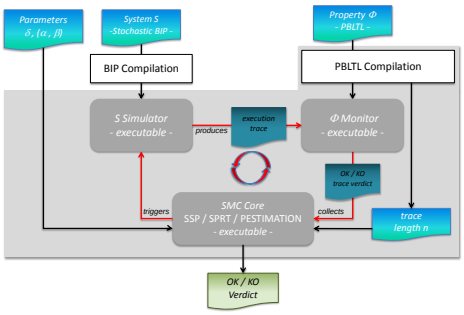


Fig. 3: BIP$^{\mathcal{SMC}}$ architecture.

BIP$^{\mathcal{SMC}}$ is fully developed in the Java programming language. It uses JEP 2.4.1 library[4] (under GPL license) for parsing and evaluating mathematical expression, and ANTLR 3.2[5] for PBLTL properties parsing and monitoring. At this stage, BIP$^{\mathcal{SMC}}$ only runs on GNU/Linux operating systems since it relies on the

[4] `http://www.singularsys.com/jep/index.html`
[5] `http://www.antlr.org/`

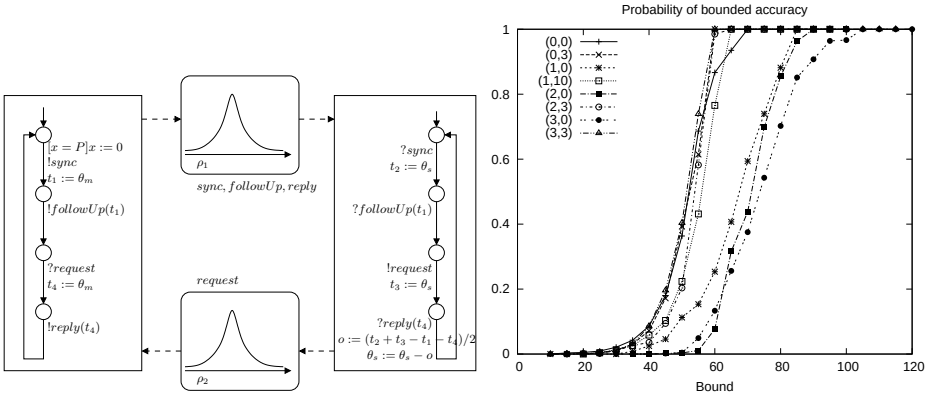
BIP simulation engine. The current release of the tool has been enriched with a graphical user interface for more convenience, in addition to the command line mode. The current version also includes supports of the BIP2 language (the new version of BIP)[6] while still compatible with the previous version. The model checker is available for download from `http://www-verimag.imag.fr/Statistical-Model-Checking.html`, where additional information on how to install it and use it with the BIP framework can be found.

4 Case Studies

While still at the first release, the SBIP framework has been used to evaluate several large scale systems that covers different application domains. The first three studies below consider the modeling and verification of network protocols, while the two remaining present multimedia applications.

4.1 Precision Time Protocol IEEE 1588

In this study, the Precision Time Protocol (PTP) is deployed as part of a distributed heterogeneous communication systems (HCS) [4] in an aircraft. It is used to synchronize the clocks of various devices with the one of a specific server on the network. This synchronization is important to guarantee a correct behavior of the whole system.



(a) The stochastic PTP model. (b) Results of accuracy analysis.

We used SBIP to check the accuracy of clock synchronization which is defined as the absolute value of the difference between the master clock θ_m and a slave clocks θ_s (see Figure 4a). More precisely, we estimate the probability

[6] `http://www-verimag.imag.fr/New-BIP-tools.html`

that the clock deviation always stays under some specific bound Δ for each slave device. This requirement is expressed by the following PBLTL formula: $\mathbf{P}_{=?}[\mathsf{G}^{1000}(abs(\theta_m - \theta_s) \leqslant \Delta)]$. The ultimate goal of the study is to compute the minimal bound Δ that ensures full synchronization, i.e, the synchronization of all the slave clocks in the network with the master clock with probability 1.

The results illustrated in Figure 4b shows the probability evolution of the devices synchronization (in the y-axis) with respect to various times bounds in micro seconds (in the x-axis). We can see different curves corresponding to several devices identified through their addresses in the network. Remark that the synchronization is guaranteed for a specific device whenever its curve reaches probability 1. Thus, we can conclude from these experiments that the minimal bound that ensure full synchronization (synchronization of all the slaves with the server) occurs at 120μs.

4.2 MPEG2 Decoder

In this study, we used SBIP to check QoS properties of an MPEG2 Decoder for a video streaming application [3]. This work is about finding a trade-off, when designing such multimedia systems, between buffer sizes and video quality. In fact, an acceptable amount of quality degradation can be tolerated (less than two consecutive frames within a second [3]) in order to reduce buffers sizes.

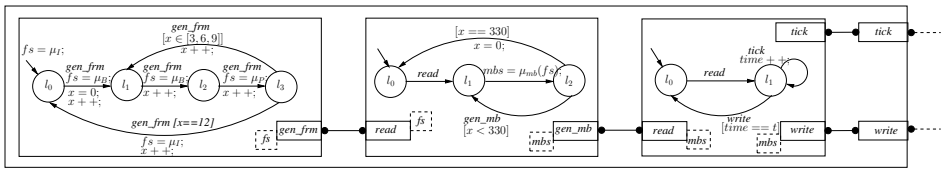


Fig. 4: The *Generator* compound component. μ_I, μ_B, μ_P are probability distributions corresponding to each frame type, μ_{mb} is the probability distribution of the macro-blocks and depends on the generated frame type.

In the study, quality loss is seen as buffer underflow which occurs whenever the display device does not find sufficient macro blocks to read from the playout buffer. The amount of underflow can be controlled using the initial playout delay parameter i.e. the delay after which the video starts to display. Figure 4 shows the *Generator* component of the stochastic BIP model of the decoder system. This is the first element in the video decoding unit, which models the macro-blocks arrival to the input buffer.

In the first component of Figure 4 (from the left), frames are stochastically generated with respect to 3 probability distributions (μ_I, μ_B, and μ_P) that correspond to MPEG2-coded frames types. Next, macro-blocks are generated with

respect to a frame-dependent distribution. Finally, the third component represents macro-blocks arrival time to the input buffer.

Figure 5 illustrates some of the results obtained when analyzing this system. It shows three separate curves representing the probability that the stream loss is less than two consecutive frames within a second for three different videos, namely `cact.m2v`, `mobile.m2v`, and `tennis.m2v`. The considered videos have the same resolution of 352×240 and were obtained from an open source.

Note that for this experiments, BIP$^{\mathcal{SMC}}$ used about 44 to 7145 traces each time and spent around 6 to 8 seconds in average to check the property with an error bound of 10^{-2}.

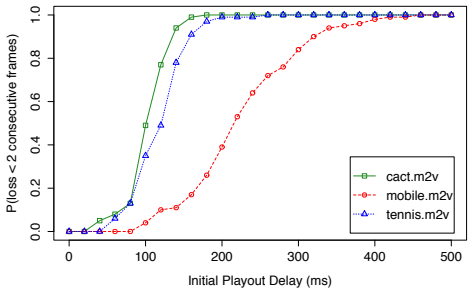


Fig. 5: Results of QoS property analysis.

4.3 Avionics Full-DupleX Switched Ethernet

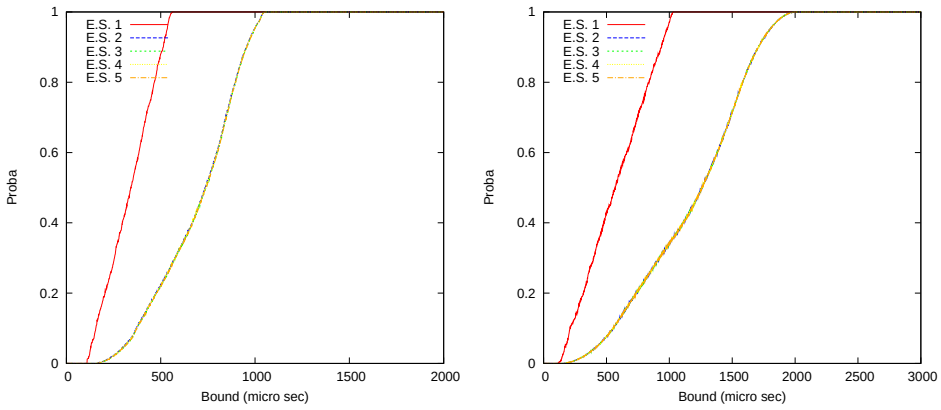
SBIP has been also used for the analysis of QoS properties of the Avionics Full-DupleX switched ethernet (AFDX) network [6]. AFDX protocol was proposed as a solution to resolve problems due to the spectacular increase of the quantity of communication and thus of the number of connections in avionics network. The main idea behind AFDX is to simulate point-to-point connections between all the devices in a network using *Virtual Links* (VL). For such systems, one challenging point is to guarantee bounded delivery time on every VL.

In order to check the latency requirements, two configurations with different number of virtual links were considered: 10 and 20 which have the same characteristics. This experiment consist of using PESTIMATION algorithm with precision 0.01 and a confidence of 0.01 to estimate probabilities for bounds between $0\mu s$ and $2000\mu s$ for $X = 10$ and between $0\mu s$ and $3000\mu s$ for $X = 20$. The results are given in Figures 6a and 6b for respectively $X = 10$ and $X = 20$ links. We also used SPRT and SSP algorithms with a confidence of 10^{-10} and a precision of 10^{-7} to validate the results we obtained with PESTIMATION.

4.4 Wireless Sensor Netwrok

The SBIP framework has been used to verify several networked systems based on different technologies, CAN-based [18], Sensor Network using WiFi [17], and IoT applications [19]. We briefly present the utilization of SBIP for the modeling and analysis of a Wireless Sensor Network (WSN) case study.

This case study concerns audio capturing and reproduction over a WiFi wireless network. The goal of the study is to check the synchronization between the



(a) Probability of having a delay lower than a given bound ($X = 10$).

(b) Probability of having a delay lower than a given bound ($X = 20$).

Fig. 6: Results of latency analysis for the AFDX case study.

different nodes of the network essentially the sender-to-receiver. The synchronization protocol is as follows. The base station broadcasts periodically (period T=5s) a frame containing the hardware clock value (*synchro* process) to all the nodes through the wireless network. Each node applies a Phase Locked Loop (PLL) synchronization technique, to construct a software clock. The PLL system takes the broadcasted clock as input and keeps the local clock synchronized to it. The expected synchronization accuracy, defined as the difference between the input and output clock, is specified as 1μs. The resulting clock is used by the *micro* process to timestamp the audio frames. Subsequently, the base station is able to reproduce the received audio frames through the *speaker* process in the correct chronological order.

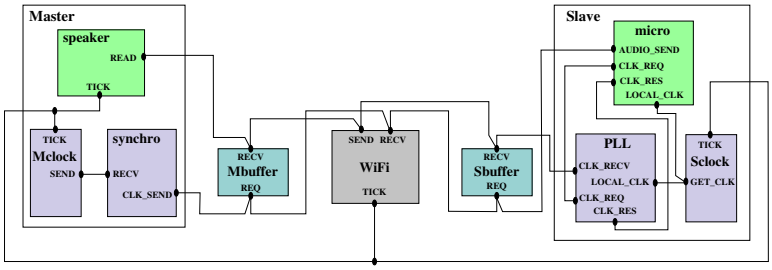


Fig. 7: SBIP model of the Wireless Sensor Network

For the implementation of the WSN application, a wireless sensor network that consists of three nodes was used. Each node is a UDOO platform[7], which consists of a computational core, a WiFi card, and a sound card. The computational core is responsible for the node's processing operations, the WiFi card for the wireless communication of the network and the sound card for capturing or reproducing sound. The wireless network is supported by a Snowball SDK platform[8] used as Access Point (AP).

We conducted two sets of experiments, focusing on equally important requirements in the development of multimedia sensor networks. The first analyzed the utilization of the buffer components concerning only the audio capturing and reproduction in the system. Thus, this experiment focused on a functional requirement, which is influenced by non-functional requirements such as the packet delivery ratio and the end-to-end delays. In the second experiment we focused on the obtained clock synchronization accuracy. Therefore, we observed the difference between the Master clock θ_m and the software clock computed in every Slave θ_s without the impact of the audio capturing and reproduction. These requirements were described as probabilistic temporal properties, using PBLTL. The obtained results are presented hereafter.

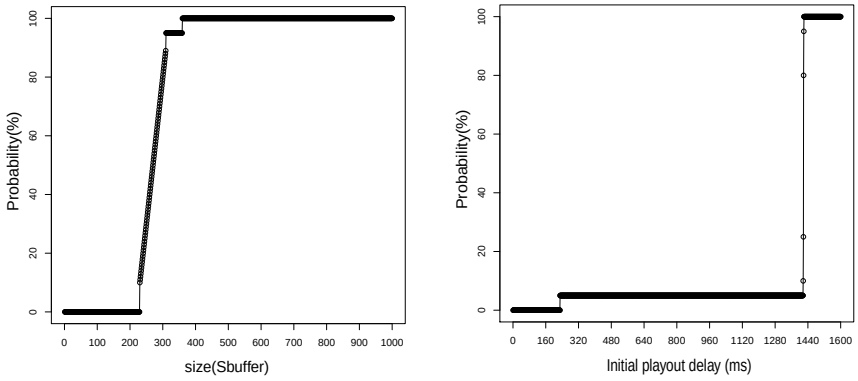


Fig. 8: Probabilities of ϕ_1 (left) as a function of the Sbuffer size and ϕ_2 (right) as a function of the initial playout delay.

We evaluated the property of avoiding overflow or underflow in each buffer component by considering the following properties: $\phi_1 = \mathsf{G}^l(S_{Sbuffer} < MAX)$, as well as $\phi_2 = \mathsf{G}^l(S_{Mbuffer} > 0)$, where $S_{Sbuffer}$ and $S_{Mbuffer}$ indicate the size of the Slave and Master buffer respectively (see Figure 7). The value of MAX is considered as fixed and equal to 400. As illustrated in Figure 8 $P(\phi_1) = 1$ for the considered value of MAX, meaning that the overflow in the SBuffer is avoided.

[7] `http://www.udoo.org/features/`

[8] `http://www.calao-systems.com/articles.php?pg=6186`

Furthermore, the probability of underflow avoidance in the Mbuffer depends on the initial playout delay. Specifically, in Figure 8 we can observe this for delays greater than 1430 ms $P(\phi_2) = 1$, meaning that the Master should start the consumption of audio packets when this time duration has elapsed.

The property of maintaining a bounded synchronization accuracy is defined as: $\phi_3 = \mathsf{G}^l(|(\theta_m - \theta_s) - A| < \Delta)$, where A indicates a fixed offset between the Master and each computed software clock and Δ is a fixed non-negative number, denoting the resulting bound. Initially, we used several probabilistic distributions from the execution results of the application to test if the expected bound $\Delta = 1\mu s$ is achieved. However, the achieved bound by the simulations was always above the defined bound of 1μs for $A = 100\mu$s. We accordingly repeated the previous experiments, in order to estimate the best bound. Therefore, we tried to estimate the smallest bound which ensures synchronization with probability 1, by repeating the previous experiment for a variety of Δ between $10\mu s$ and 80 μs. The simulations have shown that the synchronization bound was 76 μs.

4.5 Image Recognition on Many-cores

In this case study, the SBIP framework is used as part of the design of an embedded system consisting of the HMAX image recognition application deployed on the STHORM many-core architecture [23, 24].

The HMAX models algorithm [20] is a hierarchical computational model of object recognition which attempts to mimic the rapid object recognition of human brain. In the present case study, we only focus on the first layer of the HMAX Models algorithm (see Figure 9) as it is the most computationally intensive.

We are interested on the overall execution time and the time to process single lines of the input image. More precisely, we will compute the probabilities that the overall execution time is always lower than a given bound Δ and that the variability in the processing time of successive lines is always bounded by Ψ. To this end, we specify respectively the above requirements in BLTL as $\phi_1 = \mathsf{G}^l(t < \Delta)$, where t is the monitored overall execution time and $\phi_2 = \mathsf{G}^l(|tl| < \Psi)$, where tl is the difference between the processing time of successive lines.

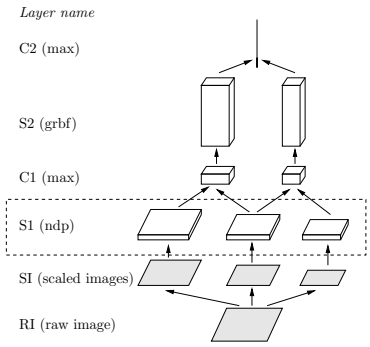


Fig. 9: HMAX overview.

We developed a parametric SBIP model for the S1 layer of HMAX (see Figure 10), where every image is handled by one "processing group" consisting of a *Splitter*, one or more *Worker* processes and a *Joiner*, connected through FIFO channels. The computation of the entire S1 layer is coordinated by a single

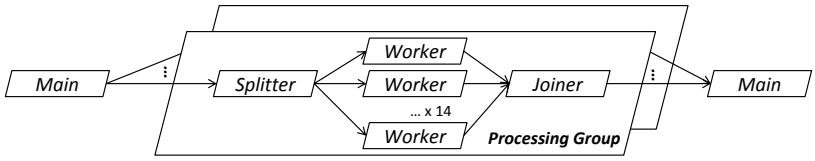


Fig. 10: The SBIP model of HMAX (S1 layer).

main process. In this model, several image scales are handled concurrently and the processing is pipelined using a pipelining rate PR.

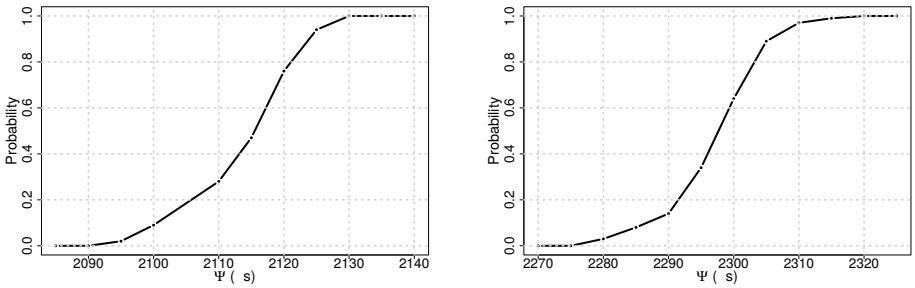
We checked the aforementioned performance requirements, i.e. ϕ_1 and ϕ_2 for different pipelining rate $PR = \{0, 2\}$ and bounds Δ, Ψ. In this experiment, we chose arbitrary FIFOs sizes: *Main-Splitter*= 10 KB, *Splitter-Worker*= 112 B, *Worker-Joiner*= 336 B, and *Joiner-Main*= 30 KB (see Figure 10) to fit the STHORM L1 memory of a single cluster.

Table 1: Probabilities to satisfy ϕ_1 for different Δ ($PR = 0$).

$\Delta(ms)$	572.75	572.8	572.83	572.85	572.89	572.95
$P(\phi_1)$	0	0.28	0.57	0.75	0.98	1
Nbr of required traces	66	1513	1110	488	171	66

Table 1 shows the probabilities to satisfy the first requirement ϕ_1 for different values of Δ, in the case where $PR = 0$. The table also reports, in the last column, a performance metric, i.e. the number of traces that were necessary for the SPRT algorithm to decide each time. For instance, based on these results, one can conclude that the expected overall execution time (for processing one image scale) is bounded by $\Delta = 572.91ms$ with probability ≥ 0.99.

Figure 11 shows the probabilities to satisfy the second requirement ϕ_2 when varying Ψ. Figure 11a is obtained with no pipelining, i.e. ($PR = 0$), whereas Figure 11b is obtained with $PR = 2$. One can note that the two curves show similar evolutions, albeit the curve in Figure 11b is slightly shifted to the right, i.e. the values of Ψ in this case are greater than Figure 11a. This actually means that this configuration induces more processing time variation between successive lines. We recall that when $PR = 0$, all the processes are perfectly synchronized which yields small variation over successive lines processing time. Using $PR > 0$ however leads to greater variation since it somehow alters this synchronization. Concretely, Figure 11 shows that without pipelining, we obtain smaller expected time variation (of processing successive lines). For instance when $PR = 0$, $\Psi = 2128\mu s$ with probability 0.99, whereas for $PR = 2$, $\Psi = 2315\mu s$ with the same probability. Hence, one may choose $PR = 0$ if interested in a higher throughput.



(a) $PR = 0$ (b) $PR = 2$

Fig. 11: Probability to satisfy ϕ_2 for $PR \in \{0, 2\}$.

We finally note that the SMC time was relatively small given the obtained model size: 5 hours in average for each property. It is worth mentioning that we used the SPRT algorithm iteratively (with a binary search and a fixed decimal precision) to compute specific probability values as it normally provides a yes/no answer. The stochastic BIP model has 47 components and about 6000 lines of code. Components have in average 20 control locations and 10 integer variables each, which induces a big state space. For instance, processing a single line by the 14 parallel *Workers* can lead to approximately 5^{14} states, as each *Worker* performs each time 5 steps, namely *read, compute,* and 3 *writes* (3 directions).

5 Conclusion

In this paper, we presented the SBIP framework which consists of a stochastic extension to the BIP formalism, and the BIP$^{\mathcal{SMC}}$ statistical model checker. It is worth mentioning that stochastic BIP models can be analyzed independently using other techniques such as numerical probabilistic model checking.

As shown in Section 4, the SBIP model checker has been used in several case studies. However, several ameliorations are still ahead to enhance its performance. Compared to more mature tools like Prism [16] or Uppaal-smc [11], it still needs various improvements. A major amelioration is at the level of the interface with the BIP simulation engine, which is quite rudimentary for the moment and induces a considerable latency. We are planning to re-implement it more properly for the next release of the tool. Another amelioration will consist to extend the PBLTL input language to support nested operators and to improve the automatic generation of properties monitors.

In the future, we are also planning to extend the graphical user interface with a plotting feature such as in Prism and Uppaal to enable building curves within the tool. Finally, a more long-term extension will consist to implement a parallel version of the statistical model checking algorithms. SMC still actually suffers from scalability issues when confronted with industrial-size system models. A

parallel implementation, together with a model abstraction technique [25], may eventually enable a more efficient model analysis.

References

1. M. AlTurki and J. Meseguer. PVeStA: A parallel statistical model checking and quantitative analysis tool. In *Proceedings of the 4th International Conference on Algebra and Coalgebra in Computer Science, CALCO'11*, August 2011.
2. C. Baier and J.-P. Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.
3. R. Balaji, A. Nouri, D. Gangadharan, M. Bozga, M. M. Ananda Basu, A. Legay, S. Bensalem, and S. Chakraborty. Stochastic modeling and performance analysis of multimedia socs. In *International conference on Systems, Architectures, Modeling and Simulation, SAMOS'13*, pages 145–154, 2013.
4. A. Basu, S. Bensalem, M. Bozga, B. Caillaud, B. Delahaye, and A. Legay. Statistical abstraction and model-checking of large heterogeneous systems. In *Forum for fundamental research on theory, FORTE'10*, volume 6117 of *LNCS*, pages 32–46. Springer, 2010.
5. A. Basu, S. Bensalem, M. Bozga, J. Combaz, M. Jaber, T.-H. Nguyen, and J. Sifakis. Rigorous component-based system design using the bip framework. *IEEE Softw.*, 28(3):41–48, May 2011.
6. A. Basu, S. Bensalem, M. Bozga, B. Delahaye, A. Legay, and E. Siffakis. Verification of an AFDX infrastructure using simulations and probabilities. In *Runtime Verification, RV'10*, volume 6418 of *LNCS*. Springer, 2010.
7. A. Basu, M. Bozga, and J. Sifakis. Modeling heterogeneous real-time components in bip. In *Proceedings of the Fourth IEEE International Conference on Software Engineering and Formal Methods*, SEFM'06, pages 3–12, Washington, DC, USA, 2006. IEEE Computer Society.
8. S. Bensalem, M. Bozga, B. Delahaye, C. Jégourel, A. Legay, and A. Nouri. Statistical Model Checking QoS Properties of Systems with SBIP. In *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation, ISOLA'12*, pages 327–341, 2012.
9. J. Bogdoll, L. M. F. Fioriti, A. Hartmanns, and H. Hermanns. Partial order methods for statistical model checking and simulation. In *Forum for fundamental research on theory, FMOODS/FORTE'11*, pages 59–74, June 2011.
10. A. David, K. G. Larsen, A. Legay, M. Mikucionis, D. B. Poulsen, and S. Sedwards. Statistical model checking for biological systems. *Int. J. Softw. Tools Technol. Transf.*, 17(3):351–367, June 2015.
11. A. David, K. G. Larsen, A. Legay, M. Mikuăionis, and D. B. Poulsen. Uppaal smc tutorial. *Int. J. Softw. Tools Technol. Transf. (STTT)*, 17(4):397–415, August 2015.
12. T. Hérault, R. Lassaigne, F. Magniette, and S. Peyronnet. Approximate Probabilistic Model Checking. In *International Conference on Verification, Model Checking, and Abstract Interpretation, VMCAI'04*, pages 73–84, January 2004.
13. T. Herault, R. Lassaigne, and S. Peyronnet. APMC 3.0: Approximate verification of discrete and continuous time markov chains. In *Proceedings of the 3rd international conference on the Quantitative Evaluation of Systems*, QEST '06, pages 129–130, Washington, DC, USA, 2006. IEEE Computer Society.

14. C. Jegourel, A. Legay, and S. Sedwards. A platform for high performance statistical model checking — plasma. In *Proceedings of the 18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'12, pages 498–503, Berlin, Heidelberg, 2012. Springer-Verlag.
15. N. Kumar, K. Sen, J. Meseguer, and G. Agha. A rewriting based model for probabilistic distributed object systems. In E. Najm, U. Nestmann, and P. Stevens, editors, *FMOODS*, pages 32–46, 2003.
16. M. Kwiatkowska, G. Norman, and D. Parker. Prism 4.0: verification of probabilistic real-time systems. In *Proceedings of the 23rd international conference on Computer aided verification*, CAV'11, pages 585–591, Berlin, Heidelberg, 2011. Springer-Verlag.
17. A. Lekidis, P. Bourgos, S. Djoko-Djoko, M. Bozga, and S. Bensalem. Building distributed sensor network applications using BIP. In *2015 IEEE Sensors Applications Symposium SAS 2015*, 2015 IEEE Sensors Applications Symposium SAS 2015, Zadar, Croatia, April 13-15, 2015, Zadar, Croatia, Apr. 2015. IEEE.
18. A. Lekidis, M. Bozga, D. Mauuary, and S. Bensalem. A model-based design flow for CAN-based systems. In *13th International CAN Conference*, iCC'13, Paris, France, October 2013.
19. A. Lekidis, E. Stachtiari, P. Katsaros, M. Bozga, and C. K. Georgiadis. Using BIP to reinforce correctness of resource-constrained IoT applications. In *10th IEEE International Symposium on Industrial Embedded Systems, SIES 2015*, pages 245–253, Siegen, Germany, June 2015. IEEE.
20. J. Mutch and D. G. Lowe. Object class recognition and localization using sparse features with limited receptive fields. *International Journal of Computer Vision*, 80(1):45–57, 2008.
21. A. Nouri. *Rigorous System-level Modeling and Performance Evaluation for Embedded System Design*. Ph.d. dissertation., Université Grenoble Alpes, April 2015.
22. A. Nouri, S. Bensalem, M. Bozga, B. Delahaye, C. Jegourel, and A. Legay. Statistical model checking QoS properties of systems with SBIP. *Int. J. Softw. Tools Technol. Transf. (STTT)*, 17(2):171–185, April 2015.
23. A. Nouri, M. Bozga, A. Molnos, A. Legay, and S. Bensalem. Building faithful high-level models and performance evaluation of manycore embedded systems. In *Twelfth ACM/IEEE International Conference on Formal Methods and Models for Codesign, MEMOCODE 2014, Lausanne, Switzerland, October 19-21, 2014*, pages 209–218, 2014.
24. A. Nouri, M. Bozga, A. Molnos, A. Legay, and S. Bensalem. Astrolabe: A rigorous approach for system-level performance modeling and analysis. *ACM Trans. Embed. Comput. Syst.*, 15(2):31:1–31:26, Mar. 2016.
25. A. Nouri, B. Raman, M. Bozga, A. Legay, and S. Bensalem. Faster statistical model checking by means of abstraction and learning. In *Proceedings of the 5th International Conference on Runtime Verification, RV'14, Toronto, ON, Canada, September 22-25, 2014*, pages 340–355, 2014.
26. K. Sen, M. Viswanathan, and G. A. Agha. Vesta: A statistical model-checker and analyzer for probabilistic systems. In *International Conference on the Quantitative Evaluation of Systems, QEST'05*, pages 251–252, 2005.
27. A. Wald. Sequential tests of statistical hypotheses. *Annals of Mathematical Statistics*, 16(2):117–186, 1945.
28. H. L. S. Younes. *Verification and Planning for Stochastic Processes with Asynchronous Events*. PhD thesis, Carnegie Mellon, 2005.
29. H. L. S. Younes. Ymer: A statistical model checker. In *COMPUTER AIDED VERIFICATION, CAV'05*, pages 429–433. Springer, 2005.