Statistical Abstraction and Model-Checking of Large Heterogeneous Systems

Ananda Basu, Saddek Bensalem, Marius Bozga, Benoît Delahaye, Axel Legay

Noname manuscript No.
(will be inserted by the editor)

Statistical Abstraction and Model-Checking of Large Heterogeneous Systems

Ananda Basu · Saddek Bensalem · Marius Bozga · Benoît Delahaye · Axel Legay

the date of receipt and acceptance should be inserted later

Abstract We propose a new simulation-based technique for verifying applications running within a large heterogeneous system. Our technique starts by performing simulations of the system in order to learn the context in which the application is used. Then, it creates a stochastic abstraction for the application, which takes the context information into account. This smaller model can be verified using efficient techniques such as statistical model checking. We have applied our technique to an industrial case study: the cabin communication system of an airplane. We use the BIP toolset to model and simulate the system. We have conducted experiments to verify the clock synchronization protocol i.e., the application used to synchronize the clocks of all computing devices within the system.

1 Introduction

Systems integrating multiple heterogeneous distributed applications communicating over a shared network are typical in various sensitive domains such as aeronautic or automotive embedded systems. Verifying the correctness of a particular application inside such a system is known to be a challenging task, which is often beyond the scope of existing exhaustive validation techniques. The main difficulty comes from network communication which makes all applications interfering and therefore forces exploration of the full state-space of the system.

Statistical Model Checking [16,25,28,29,20,23,15,7] has recently been proposed as an alternative to avoid an exhaustive exploration of the state-space of the model. The core idea of the approach is to conduct some simulations of the system and then use statistical results in order to decide whether the system satisfies the property or not. Statistical model checking techniques can also be used to estimate the probability that a system satisfies a given property [16,15]. Of course, in contrast with an exhaustive approach, a simulation-based solution does not guarantee a correct result. However, it is possible to bound the probability of making an error. Simulation-based methods are known to be far less memory and time intensive than exhaustive ones, and are sometimes the only option [30,18]. Statistical model checking is widely accepted in various research areas such as systems biology [13,19] or software engineering, in particular for industrial applications. There are several reasons for this success. First, it is very simple to implement, understand and use. Second, it does not require extra modeling or specification effort, but simply an operational model of the system, that can be simulated and checked against state-based properties. Third, it allows model-checking of properties [12] that cannot be expressed in classical temporal logics. Nevertheless, statistical-model checking still suffers from the system's complexity. In particular, for the case of heterogeneous systems, the number of components and their interac-

This work has been supported by the Combest EU project. A preliminary version of the paper [6] was published in the International Conference on Formal Techniques for Distributed Systems.

A. Basu, S. Bensalem and M. Bozga
Verimag Laboratory,
Université Joseph Fourier
Grenoble, CNRS

B. Delahaye
Université de Rennes 1/IRISA,
Rennes, France

A. Legay
INRIA/IRISA,
Rennes, France

tions are limiting factors on the number and length of simulations that can be conducted and hence on the accuracy of the statistical estimates.

We propose to exploit the structure of the system in order to increase the efficiency of the verification process. The idea is conceptually simple: instead of performing an analysis of the entire system, we propose to analyze each application separately, but under some particular context/execution environment. This context is a *stochastic abstraction* that represents the interactions with other applications running within the system and sharing the computation and communication resources. We propose to build such a context automatically by simulating the system and learning the probability distributions of key characteristics impacting the functionality of the given application.

The overall contribution of this paper is an application of the above method on an industrial case study, the *heterogeneous communication system* (HCS for short) deployed for cabin communication in a civil airplane. HCS is a heterogeneous system providing entertainment services (e.g., audio/video on passengers demand) as well as administrative services (e.g., cabin illumination, control, audio announcements), which are implemented as distributed applications running in parallel, across various devices within the plane and communicating through a common Ethernet-based network. The HCS system has to guarantee stringent requirements, such as reliable data transmission, fault tolerance, timing and synchronization constraints. An important requirement, which will be studied in this paper, is the *accuracy of clock synchronization* between different devices. This latter property states that the difference between the clocks of any two devices should be bounded by a small constant, which is provided by the system designer and depends on his needs. Hence, one must be capable of computing the smallest bound for which synchronization occurs and compare it with the bound expected by the designer. Unfortunately, due to the large number of heterogeneous components that constitute the system, deriving such a bound manually from the textual specification is an unfeasible task. In this paper, we propose a formal approach that consists in building a formal, operational model of the HCS, then applying simulation-based algorithms to this model in order to deduce the smallest value of the bound for which synchronization occurs. We start with a fixed value of the bound and check whether synchronization occurs. If yes, then we make sure that this is the best one. If no, we restart the experiment with a new value.

At the top of our approach, there should be a tool that is capable of modeling heterogeneous systems as well as simulating their executions and the interactions between components. In this paper, we propose to use the BIP[1] toolset [5] for doing so. BIP supports a methodology for building systems from atomic components encapsulating behavior, that communicate through interactions, and coordinate according to priorities. BIP also offers a powerful engine to simulate the system and can thus be combined with a statistical model checking algorithm in order to verify properties. Our first contribution is to study all the requirements for the HCS to work properly and then derive a model in BIP. Our second contribution is to study the accuracy of clock synchronization between several devices of the HCS. In HCS the clock synchronization is ensured by the *Precision Time Protocol* (PTP for short) [2], and the challenge is to guarantee that PTP maintains the difference between a master clock (running on a designated server within the system) and all the slave clocks (running on other devices) under some bound. Since this bound cannot be pre-computed, we have to verify the system for various values of the bound until we find a suitable one. Unfortunately, the full system is too big to be analyzed with classical exhaustive verification techniques. A solution could be to remove all the information that is not related to the devices under consideration. This is in fact not correct as the behavior of the PTP protocol is influenced by the other applications running in parallel within the heterogeneous system. Our solution to this state-space explosion problem is in two steps (1) we build a stochastic abstraction for a part of the PTP application between the server and a given device; the stochastic part will be used to model the general context in which PTP is used, and (2) we apply statistical model checking on the resulting model.

Thanks to this approach, we have been able to derive precise bounds that guarantee proper synchronization for all the devices of the system. We also computed the probability of satisfying the property for smaller values of the bound, i.e., bounds that do not satisfy the synchronization property with probability 1. Being able to provide such information is of clear importance, especially when the best bound is too high with respect to the designer's requirements. We have observed that the values we obtained strongly depend on the position of the device in the network. We also estimated the average and worst proportion of failures per simulation for bounds that are smaller than the one that guarantees synchronization. Checking this latter property has been made easy because BIP allows us to reason on one execution at a time.

As another contribution, we have also considered the influence of clock drift on the synchronisation re-

[1] BIP states for *Behaviour-Interaction-Priority.*

sults. Drift can be used to model that, due to the influence of the hardware, clocks of the various components may not progress at the same rate. We have observed that small values of the drifts have no influence on the results. Again, we observe that it is easy to handle drift when reasoning on an execution at a time. Finally, we have also studied the influence on synchronization due to scheduling policies applied within the network for different categories of traffic. For doing so, we have compared two different scheduling algorithms: fixed priorities and weighted fair queuing. We have observed that fixed priorities, with highest priority given to PTP packets, guarantees the best precision, but may prevent some packets to be sent. The experiments highlight the generality of our technique, which can be applied to other versions of the HCS as well as to other heterogeneous applications [4].

Structure of the paper. Section 2 introduces the concept of stochastic abstraction that will be used to reduce the complexity of the model under verification. Sections 3 and 4 are dedicated to introductions to the BIP toolset and Statistical Model Checking, respectively. The HCS case study is introduced in Section 5 and the experiments are reported in Section 6. Finally, Section 7 concludes the paper and discusses future and related works.

2 Problem and Approach

Consider a system consisting of a set of distributed applications running on several computers and exchanging messages on a shared network infrastructure. Assume also that network communication is subject to given bandwidth restrictions as well as to routing and scheduling policies applied on network elements. Our method attempts to reduce the complexity of validation of a particular application of such system by decoupling the timing analysis of the network and functional analysis of each application.

We start by constructing a model of the whole system. This model must be executable, i.e., it should be possible to obtain execution traces, annotated with timing information. For a chosen application, we then learn the probability distribution laws of its message delays by simulating the entire system. The method then constructs a reduced stochastic model by combining the application model where the delays are defined according to the laws identified at the previous step. Finally, the method applies statistical model-checking on the resulting stochastic model.

Our models are specified within the BIP framework [5]. BIP is a component-based framework for construction, implementation and analysis of systems composed of heterogeneous components. In particular, the tool fulfills all the requirements of the method suggested above, that are models are operational and can be thoroughly simulated. BIP models can easily integrate timing constraints, which are represented with discrete clocks. Probabilistic behaviour can also be added by using external C functions.

The BIP toolset [8], which includes a rich set of tools for modeling, execution, analysis (both static and on-the-fly) and static transformations of models. It provides a dedicated programming language for describing BIP models. The front-end tools allow editing and parsing of BIP programs, and generating an intermediate model, followed by code generation (in C) for execution and analysis on a dedicated middleware platform. The platform also offers connections to external analysis tools. A more complete description of BIP is given in the next section.

3 An Overview of BIP

The BIP framework, presented in [5], supports a methodology for building systems from *atomic components*. It uses *connectors*, to specify possible interaction patterns between components, and *priorities*, to select amongst possible interactions.

Atomic components are finite-state automata that are extended with variables and ports. Variables are used to store local data. Ports are action names, and may be associated with variables. They are used for interaction with other components. States denote control locations at which the components await for interaction. A transition is a step, labeled by a port, from a control location to another. It has associated a guard and an action, that are respectively, a Boolean condition and a computation defined on local variables. In BIP, data and their transformations are written in C.

For a given valuation of variables, a transition can be executed if the guard evaluates to true and some *interaction* involving the port is enabled. The execution is an atomic sequence of two microsteps: 1) execution of the interaction involving the port, which is a synchronization between several components, with possible exchange of data, followed by 2) execution of internal computation associated with the transition.

We provide in Figure 1 a graphical representation for an atomic component, named *Router*, that models the behavior of a simplified network router. The router receives network packets through an input port and delivers them to the respective output port(s), based on

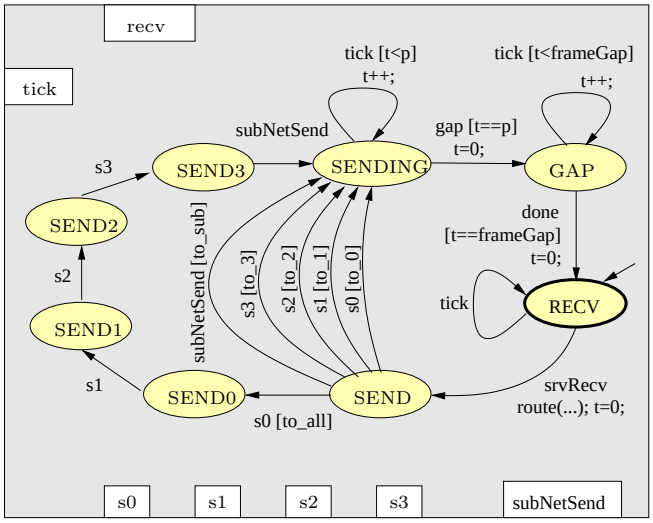


Fig. 1: An atomic component: Router.

the destination address of the packets. The port *recv* acts as an input port, while *s0*, *s1*, *s2*, *s3*, and *subNetSend* act as output ports. The port *tick* is used for modeling discrete time progress: an interaction through this port corresponds to the progress of time by one (tick) unit. The control locations are *RECV*, *SEND*, *SEND0*, *SEND1*, *SEND2*, *SEND3*, *SENDING* and *GAP*, with *RECV* being the initial location. An example transition is from the initial location *RECV* to *SEND*, which is executed when an interaction including the port *recv* takes place (i.e., the guard being *true*). On execution, the internal computation step is the execution of the C routine *route(...)*, followed by the reset of the variable *t*. The complete description of the *Router* component using the BIP language is provided below.

```
atomic type Router
  /* parameters* /
(  int id, // ID of the router
   bool server, // if it is in a server
   bool device, // if it is in a device
   int frameRate, // rate of frame transmission
   int frameGap) // inter frame time-gap
   /* local variables */
data Frame f
data int t = 0 // the clock
data int p = 0 // frame propagation time
data bool to_0 = false
data bool to_1 = false
data bool to_2 = false
data bool to_3 = false
data bool to_Subnet = false
data bool to_All = false
   /* interface ports */
export port FramePort recv(f) = recv
export port FramePort s0(f) = s0
export port FramePort s1(f) = s1
export port FramePort s2(f) = s2
export port FramePort s3(f) = s3
export port FramePort subNetSend(f) = subNetSend
export port TickPort tick = tick
   /* internal ports */
port Port done
port Port gap
   /* places */
place RECV
place SEND, SEND0, SEND1, SEND2, SEND3
place SENDING
place GAP
   /* initialization */
initial to RECV
   /* transitions */
on tick from RECV to RECV
on recv from RECV to SEND
   do { route(f, id, server, device,
        to_0, to_1, to_2, to_3, to_Subnet, to_All);
        t = 0;
        p = f.getSize() / frameRate; }
on s0 from SEND to SENDING provided (to_0)
on s1 from SEND to SENDING provided (to_1)
on s2 from SEND to SENDING provided (to_2)
on s3 from SEND to SENDING provided (to_3)
on subNetSend from SEND to SENDING
   provided (to_Subnet)
on s0 from SEND to SEND0 provided (to_All)
on s1 from SEND0 to SEND1
on s2 from SEND1 to SEND2
on s3 from SEND2 to SEND3
on subNetSend from SEND3 to SENDING
on tick from SENDING to SENDING
   provided (t < p) do  t = t + 1;
on gap from SENDING to GAP
   provided (t == p) do  t = 0;
on tick from GAP to GAP
   provided (t < frameGap) do  t = t + 1;
on done from GAP to RECV
   provided (t == frameGap) do  t = 0;
end
```

Composite components are defined by assembling sub-components (atomic or composite) using *connectors*. Connectors relate ports from different sub-components. They represent sets of interactions, that are, non-empty sets of ports that have to be jointly executed. For every such interaction, the connector provides the guard and the data transfer, that are, respectively, an enabling condition and an exchange of data across the ports involved in the interaction.

Figure 2 shows the graphical representation of a composite component, named *Server*. The server contains atomic components e.g., *Master Clock*, *Audio Generator*, *Smoke Detector*, *Video Surveillance*, and composite components e.g., *Classifier*. The connectors are shown by lines joining the ports of the components. With the exception of the *tick* interaction which involve 5 components, all other interactions are binary. The textual BIP description is provided below.

```
compound type Server
```

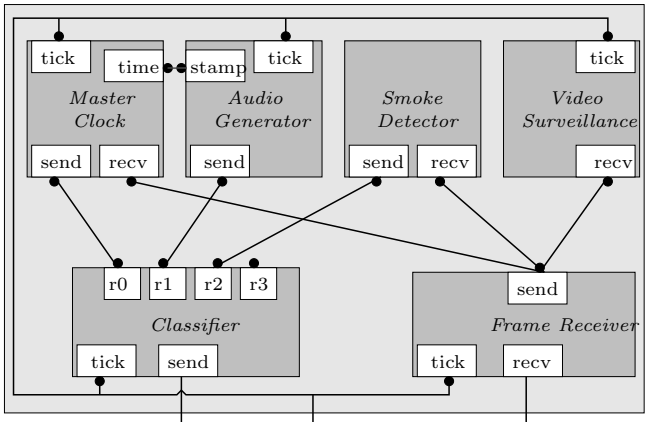


Fig. 2: Composite Component: Server.

```
   /* parameters */
(  int frameRate, int frameGap,
   int audioDelay, int nChunk, int fChunk)
   /* network subcomponents */
component FrameReceiver FRecv(frameRate)
component Classifier3X1 C(frameRate, frameGap)
   /* services sub-components */
component MasterClock Master(2000)
component AudioGenerator
   AudioGen (audioDelay, nChunk, fChunk)
component SmokeDetector SmokeDetect
component VideoSurveillance VideoSurv
   /* connectors */
connector SendMatchingFrame
   FRecv_Master(FRecv.send, Master.recv)
connector SendMatchingFrame
   FRecv_EventDetect (FRecv.send, SmokeDetect.recv)
connector SendMatchingFrame
   FRecv_VideoSurv (FRecv.send, VideoSurv.recv)
connector SendFrame Master_C (Master.send, C.r0)
connector SendFrame AudioGen_C (AudioGen.send, C.r1)
connector SendFrame
   SmokeDetect_C (SmokeDetect.send, C.r2)
connector ReadTime
   Master_AudioGen (Master.time, AudioGen.stamp)
   /* tick connector */
connector Tick5
   Tick (FRecv.tick, Master.tick, AudioGen.tick,
         VideoSurv.tick, C.tick,)
   /* interface ports */
export port FramePort send is C.send
export port FramePort recv is FRecv.recv
export port TickPort tick is Tick
end
```

Finally, *priorities* provide a mean to coordinate the execution of interactions within a BIP system. They are used to specify scheduling or similar arbitration policies between simultaneously enabled interactions. More concretely, priorities are rules, each consisting of an ordered pair of interactions associated with a condition. When the condition holds and both interactions of the corresponding pair are enabled, only the one with higher-priority can be executed.

4 An Overview of Statistical Model Checking

Consider a stochastic system[2] $\mathcal{S}$ and a property ϕ. *Statistical model checking* refers to a series of simulation-based techniques that can be used to answer two questions: (1) **Qualitative:** Is the probability that $\mathcal{S}$ satisfies ϕ greater or equal to a certain threshold? and (2) **Quantitative:** What is the probability that $\mathcal{S}$ satisfies ϕ? Contrary to numerical approaches, the answer is given up to some correctness precision. In the rest of the section, we overview several statistical model checking techniques. Let B_i be a discrete random variable with a Bernoulli distribution of parameter p. Such a variable can only take 2 values 0 and 1 with $Pr[B_i = 1] = p$ and $Pr[B_i = 0] = 1 - p$. In our context, each variable B_i is associated with one simulation of the system. The outcome for B_i, denoted b_i, is 1 if the simulation satisfies ϕ and 0 otherwise.

4.1 Qualitative Answer using Statistical Model Checking

The main approaches [28,25] proposed to answer the qualitative question are based on *hypothesis testing*. Let $p = Pr(\phi)$, to determine whether $p \geq \theta$, we can test $H : p \geq \theta$ against $K : p < \theta$. A test-based solution does not guarantee a correct result but it is possible to bound the probability of making an error. The *strength* (α, β) of a test is determined by two parameters, α and β, such that the probability of accepting K (respectively, H) when H (respectively, K) holds, called a Type-I error (respectively, a Type-II error), is less or equal to α (respectively, β).

A test has *ideal performance* if the probability of the Type-I error (respectively, Type-II error) is exactly α (respectively, β). However, these requirements make it impossible to ensure a low probability for both types of errors simultaneously (see [28] for details). A solution to this problem is to relax the test by working with an *indifference region* (p_1, p_0) with $p_0 \geq p_1$ ($p_0 - p_1$ is the *size of the region*). In this context, we test the hypothesis $H_0 : p \geq p_0$ against $H_1 : p \leq p_1$ instead of H against K. If the value of p is between p_1 and p_0 (the indifference region), then we say that the probability is sufficiently close to θ so that we are indifferent with respect to which of the two hypotheses K or H is accepted. The thresholds p_0 and p_1 are generally defined

[2] A stochastic system is aa process that evolves over time, and whose evolution can be predicted in terms of probabilities.

in terms of the single threshold θ, e.g., $p_1 = \theta - \delta$ and $p_0 = \theta + \delta$. We now need to provide a test procedure that satisfies the requirements above. In the next two subsections, we recall two solutions proposed by Younes in [28,29].

Single Sampling Plan. To test H_0 against H_1, we specify a constant c. If $\sum_{i=1}^{n} b_i$ is larger than c, then H_0 is accepted, else H_1 is accepted. The difficult part in this approach is to find values for the pair (n, c), called a *single sampling plan (SSP in short)*, such that the two error bounds α and β are respected. In practice, one tries to work with the smallest value of n possible so as to minimize the number of simulations performed. Clearly, this number has to be greater if α and β are smaller but also if the size of the indifference region is smaller. This results in an optimization problem, which generally does not have a closed-form solution except for a few special cases [28]. In his thesis [28], Younes proposes a binary search based algorithm that, given p_0, p_1, α, β, computes an approximation of the minimal value for c and n.

Sequential Probability Ratio Test. The sample size for a single sampling plan is fixed in advance and independent of the observations that are made. However, taking those observations into account can increase the performance of the test. As an example, if we use a single plan (n, c) and the $m > c$ first simulations satisfy the property, then we could (depending on the error bounds) accept H_0 without observing the $n - m$ other simulations. To overcome this problem, one can use the *sequential probability ratio test (SPRT in short)* proposed by Wald [27]. The approach is briefly described below.

In SPRT, one has to choose two values A and B $(A > B)$ that ensure that the strength of the test is respected. Let m be the number of observations that have been made so far. The test is based on the following quotient:

$$\frac{p_{1m}}{p_{0m}} = \prod_{i=1}^{m} \frac{Pr(B_i = b_i \mid p = p_1)}{Pr(B_i = b_i \mid p = p_0)} = \frac{p_1^{d_m}(1-p_1)^{m-d_m}}{p_0^{d_m}(1-p_0)^{m-d_m}}, \quad (1)$$

where $d_m = \sum_{i=1}^{m} b_i$. The idea behind the test is to accept H_0 if $\frac{p_{1m}}{p_{0m}} \geq A$, and H_1 if $\frac{p_{1m}}{p_{0m}} \leq B$. The SPRT algorithm computes $\frac{p_{1m}}{p_{0m}}$ for successive values of m until either H_0 or H_1 is satisfied; the algorithm terminates with probability 1[27]. This has the advantage of minimizing the number of simulations. In his thesis [28], Younes proposed a logarithmic based algorithm SPRT that given p_0, p_1, α and β implements the sequential ratio testing procedure.

4.2 Quantitative Answer using Statistical Model Checking

In [16,21] Peyronnet et al. propose an estimation procedure to compute the probability p for $\mathcal{S}$ to satisfy ϕ. Given a *precision* δ, Peyronnet's procedure, which we call PESTIMATION, computes a value for p' such that $|p' - p| \leq \delta$ with *confidence* $1 - \alpha$. The procedure is based on the *Chernoff-Hoeffding bound* [17]. Consider n be a number of experiments et let $p' = (\sum_{i=1}^{n} b_i)/n$. The Chernoff-Hoeffding bound [17] gives $Pr(|p' - p| > \delta) < 2e^{-\frac{n\delta^2}{4}}$. As a consequence, if we take $n \geq \frac{4}{\delta^2} \log(\frac{2}{\alpha})$, then $Pr(|p' - p| \leq \delta) \geq 1 - \alpha$. Observe that if the value p' returned by PESTIMATION is such that $p' \geq \theta - \delta$, then $\mathcal{S} \models Pr_{\geq\theta}$ with confidence $1 - \alpha$.

Peyronnet's method can be used to decide whether $\mathcal{S} \models Pr_{\geq\theta}(\phi)$ in a way similar to the single sampling plan method. In the rest of the document, we will use the name PESTIMATION to refer to an implementation that allows to compute p' based on the above approach. In his work, Younes showed that the single sampling plan method will always be at least as efficient as (i.e., will never require to perform more simulations) PESTIMATION Algorithm.

4.3 Playing with Statistical Model Checking Algorithms

The efficiency of the above algorithms is characterized by the number of simulations needed to obtain an answer. This number may change from executions to executions and can only be estimated (see [28] for an explanation). However, some generalities are known. For the qualitative case, it is known that, except for some situations, SPRT is always faster than SSP. When $\theta = 1$ (resp. $\theta = 0$) SPRT degenerates to SSP; this is not problematic since SSP is known to be optimal for such values. PESTIMATION can also be used to solve the qualitative problem, but it is always slower than SSP [28]. If θ is unknown, then a good strategy is to estimate it using PESTIMATION with a low confidence and then validate the result with SPRT and a strong confidence.

5 Case Study: Heterogeneous Communication System

The case study concerns a distributed *heterogeneous communication system* (HCS) providing an all electronic communication infrastructure to be deployed, typically for cabin communication in airplanes or for building automation. HCS contains various devices such as sensors (video camera, smoke detectors, temperature, pressure,

etc.) and actuators (loudspeakers, light switches, temperature control, signs, etc.) connected through a wired communication network to a central server. The server runs a set of services to monitor the sensors and to control the actuators. The devices are connected to the server using network access controllers (NAC) as shown for an example architecture in Figure 3.

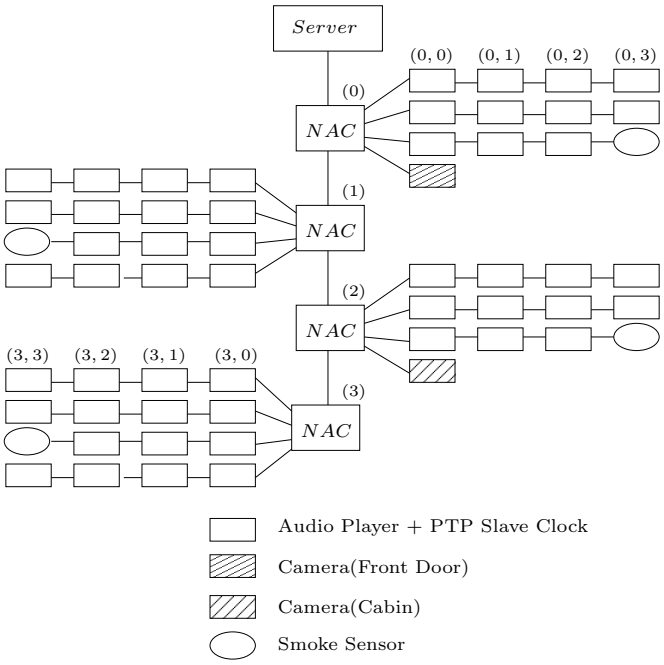


Fig. 3: HCS Example Model.

The architecture and functionalities delivered by HCS are highly heterogeneous. The system includes different hardware components, which run different protocols and software services ensuring functions with different characteristics and degree of criticality e.g, audio streaming, device clock synchronization, sensor monitoring, video surveillance. Moreover, HCS has to guarantee stringent requirements, such as reliable data transmission, fault tolerance, timing and synchronization constraints. For example, the latency for delivering alarm signals from sensors, or for playing audio announcements should be smaller than certain predefined thresholds. Or, the accuracy of clock synchronization between different devices, should be guaranteed under the given physical implementation of the system.

The HCS case study poses challenges that require component-based design techniques, since it involves heterogeneous components and communication mechanisms, e.g. streaming based on the data-flow paradigm as well as event driven computation and interaction. Its modeling needs combination of executable and analytic models especially for performance evaluation and analysis of non-functional properties.

5.1 Overview

We have developed a structural model of HCS using BIP. At top level, the structure of the model follows the natural decomposition into physical elements e.g., the server, the network access controllers and the devices are the top-level components. Moreover, these components are connected and interact according to the wired network connections defined in the original system. Then, one level down, every physical component has a functional decomposition. Subcomponents realize the main functionalities corresponding to network operation (e.g., packet delivery, filtering, routing, scheduling, ...), protocols (e.g., clock synchronization) or services (e.g., audio/video streaming, event handling, etc.).

Let us remark that most of the atomic components are subject to timing constraints (e.g., periodicity constraints, network transport delays, execution delays, ...). They are represented as discrete time components, that are, components using a particular *tick* port to react on progress of time. All *tick* ports are strongly synchronized, therefore, time progress is global and uniformly observed by all components in the system. In our model, every tick interaction corresponds to the progress of time by a fixed amount, which is one microsecond.

We have completely modeled an instance of HCS in BIP. As shown in Figure 3, the system consists of one *Server* connected to a daisy chain of four NACs, addressed $0 \cdots 3$, and several devices. Devices are connected in daisy chains with the NACs, the length of each chain being limited to four in our example. For simplicity, devices are addressed (i, j), where i is the address of the NAC and j is the address of the device. The model contains three types of devices, namely *Audio Player*, *Video Camera* and *Smoke Sensor*. The devices connected to NAC(0) and NAC(2) have similar topology. The first two daisy-chains consist of only *Audio Player* devices. The third daisy-chain ends with a *Smoke Sensor*, and the fourth daisy-chain consists of just one *Video Camera*. The devices connected to NAC(1) and NAC(3) have exactly the same topology, consisting of several *Audio Players* and one *Smoke Sensor* devices.

A description of the top-level components is given in the following paragraphs.

5.2 Server

The server, previously illustrated in Figure 2, runs various protocols and services including: 1) PTP *Master Clock*, that runs the PTP master-clock protocol between the server and the devices in order to keep the device (hardware) clocks synchronized with the master-clock. The protocol exchanges PTP packets of size 512 bits between the server and the devices, and runs once every 2 seconds. 2) *Audio Generator*, that generates audio streams to be play-backed by the *Audio Player* devices. It generates audio streams at 32kHz with 12 bit resolution (audio chunks). We have assumed that 100 audio chunks are sent in a single packet over the network, (that gives the size of an audio packet to be 1344 bits) at the rate of 33 packets per second. 3) *Smoke Detector* service that keeps track of the event packets (size 736 bits) sent from the *Smoke Sensor*, and 4) *Video Surveillance* service for monitoring the *Video Cameras*. In addition, the server needs to handle the scheduling and routing of outgoing packets over the communication backbone.

5.3 Network Access Controller

The NACs perform the packet routing from the server to the subnet devices and vice versa. A NAC consists of a *Router* (see Figure 1), that transmits the packets forward, from server to devices, and a *Classifier* (see Figure 4), that sends the packets backward, from devices to the server. The *Classifier* selects the packets to be sent, based on their types and a scheduling policy. As a result, packets may be served differently, and get delayed on their route to the server. Hence, the scheduling policy in the *Classifier* plays an determinant role in the transmission delay of different types of the packets.

The packets sent on the network are classified in four categories that are (1) *PTP*, (2) *Audio*, (3) *Events* and (4) *Video*. The *PTP* packets are exchanged in the process of the PTP synchronization. They will be further detailed in Section 6.1. *Audio* packets are sent between the server and audio devices. *Events* packets are sent by smoke detectors to the server. Finally, *Video* packets are sent by video camera devices to the server.

We have considered two scheduling policies, amongst the most commonly supported by commercial network routers. The first scheduling policy is based on *static fixed-priorities* of the packets. The second policy, that is called *Weighted Fair Queuing* (WFQ for short), ensures a fair share of the bandwidth of the network to each type of packets, according to some fixed, predefined ratios. We now give more details on these scheduling algorithms.

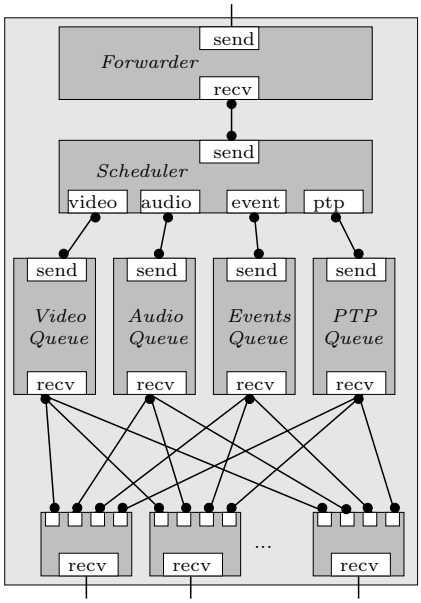


Fig. 4: Composite Component: Classifier

5.3.1 Fixed priorities

It is possible to classify the packets by their order of importance. The highest priority goes to *PTP* packets. Indeed, they need to be transmitted as soon as possible because they are critical for clock synchronization within the system. *Audio* and *Events* packets may be critical in case of a problem during the system operation e.g., if fire is detected, then the information has to be transmitted as soon as possible to the server. On the other hand, system users have to be informed without delay. Finally, the *Video* packets are less critical.

One can use this classification to define the scheduling in the NACs by following the order of importance it defines. This is the principle of fixed priorities: use as many FIFO queues to store the incoming packets as there are levels of priorities. When several queues are ready to send, empty first the one with the highest priority, then the next, etc...

The static priority policy is straightforward to implement in BIP using priorities. In our model, there are four interactions between the queues and the scheduler components namely $ptp \cdot send$, $event \cdot send$, $audio \cdot send$ and $video \cdot send$. The static priority is simply enforced by adding the following priority to the model, that is $video \cdot send \prec audio \cdot send \prec event \cdot send \prec ptp \cdot send$.

Unfortunately, the static policy has an important drawback. If the network is flooded by high-priority packets (e.g., in case of a faulty equipment), then low-priority packets get accumulated within their respective

queues, and either get (rarely) sent with important delays or get dropped, due to queue size limitations. This problem may be solved by using another scheduling algorithm that we now present.

5.3.2 Weighted fair queuing (WFQ)

Weighted fair queueing[22] is a dynamic scheduling policy which attempts to serve different queues by dividing the available network band-width according to predefined ratios.

The scheduling proceeds as follows. All incoming packets are timestamped on a virtual time line. This virtual time line re-constructs a common time reference for all queues which takes into account the available network bandwidth (r) and the allocated service ratios $(r_i)_{i=1,m}$. Notice that, in general, $\sum r_i$ can be different from r. Let us fix a queue i and consider the kth incoming packet. Assume the packet has length L_k and enters the queue at absolute time a_k. Its virtual start time $S_i(k)$, respectively virtual finish time $F_i(k)$, are computed by the following (mutually dependent) equations:

$$S_i(k) = \max(F_i(k-1), \tfrac{a_k \cdot r}{\sum r_i})$$
$$F_i(k) = S_i(k) + \tfrac{L_k}{r_i}$$

where, initially, $F_i(0) = 0$. Using this virtual timestamping, the weighted fair scheduling policy serves packets in increasing order of their virtual start times. For more details, please refer to [22].

This mechanism has been implemented as such in BIP. The *Scheduler* component keeps track of the absolute time and computes the virtual time stamps for packets, as soon as they enter the waiting queues. Then, the packet with the minimal virtual start time is selected and delivered to the *Forwarder* component, and transmitted further on the network.

Clearly, this policy hardly depends on the ratio used for each type of packets. For example, modifying the ratio may have a significant effect on the delay introduced on PTP packets. This will be further studied in Section 6.4.

5.4 Devices

Each device runs one or more services that communicate with their counter-parts in the server. As devices are connected in daisy chains, they also perform a minimal networking functionality i.e., routing and scheduling of packets on the daisy-chain. Services considered in our example are *Audio Player*, PTP *Slave Clock*, *Smoke Sensor* and *Video Camera*. More specifically for the latter, video packets are generated at a rate of 25 packets per second, the size of the video packets being given as a probability distribution. Different distributions are provided for high-resolution camera (with mean packet size of 120 kb) and for the low-resolution camera (with mean packet size of 30kb).

Name	S	V_d	V_t	C	Size	Number
Router	8	7	1	5-120	2^{11}	-
Forwarder	4	1	1	5-120	2^{8}	-
Frame Receiver	2	1	1	5-120	2^{7}	-
Master Clock	3	1	1	0-2000	2^{12}	-
Audio Generator	2	1	1	0-3125	2^{13}	-
Smoke Detector	3	1	1	0-300	2^{10}	-
Video Generator	3	1	1	0-40000	2^{16}	-
NAC	-	-	-	-	2^{34}	4
Server	-	-	-	-	2^{120}	1
Audio Player	-	-	-	-	2^{68}	52
Camera	-	-	-	-	2^{84}	2
Smoke Sensor	-	-	-	-	2^{85}	4
HCS System	-	-	-	-	2^{3122}	1

Table 1: State-space estimation.

5.5 Wrap-up

The system depicted in Figure 3 contains 58 devices in total. The BIP model contains 297 atomic components, 245 clocks (that are, discrete variables used to enforce timing constraints), and its state-space is of order 2^{3000}. The size of the BIP code for describing the system is approximately 2500 lines, which is translated to an executable simulation model of approximately 10000 lines in C++.

Table 1 gives an overview about the number and the complexity of model components defined in BIP. The first half of the table provides information about atomic components. The relevant columns are as follows: S is the number of control locations; V_d is the number of discrete variables (can be Boolean or arbitrary type like an abstract packet (including type, size and destination) or an array of packets); V_t is the number of clocks; C is the clock range; $Size$ is a rough approximated of the size of the state-space. The second half of the table provides information about composite components and their number of occurrences in the HCS system (the $Number$ column).

6 Experiments on the HCS

One of the core applications of the HCS case study is the PTP protocol, which allows the synchronization of the clocks of the various devices with the one of the server. It is important that this synchronization occurs

properly, i.e., that the difference between the clock of the server and the one of any device is bounded by a small constant. Studying this problem is the subject of this section. Since the BIP model for the HCS is extremely large (number of components, size of the state space, ...), there is no hope to analyse it with an exhaustive verification technique. Here, we propose to apply our stochastic abstraction. Given a specific device, we will proceed in two steps. First, we will conduct simulations on the entire system in order to learn the probability distribution on the communication delays between this device and the server. Second, we will use this information to build a stochastic abstraction of the application on which we will apply statistical model checking. We start with the stochastic abstraction for PTP (Section 6.1), then we report on learning distributions (Section 6.3). Finally, we report our results (Section 6.4).

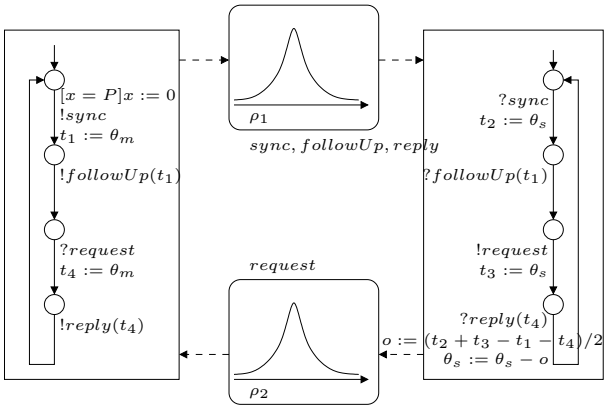


Fig. 5: Abstract stochastic PTP between the server and a device.

6.1 The Precision Time Protocol IEEE 1588

The Precision Time Protocol [2] has been defined to synchronize clocks of several computers interconnected over a network. The protocol relies on multicast communication to distribute a reference time from an accurate clock (*the master*) to all other clocks in the network (*the slaves*) combined with individual offset correction, for each slave, according to its specific round-trip communication delay to the master. The accuracy of synchronization is negatively impacted by the jitter (i.e., the variation) and the asymmetry of the communication delay between the master and the slaves. Obviously, these delay characteristics are highly dependent on the network architecture as well as on the ongoing network traffic.

We present below the abstract stochastic model of the PTP protocol between a device and the server in the HCS case study. The model consists of two (deterministic) application components respectively, the master and the slave clocks, and two probabilistic components, the media, which are abstraction of the communication network between the master and the slave. The former represent the behaviour of the protocol and are described by extended timed automata. The latter represent a random transport delay and are simply described by probability distributions. Recap that randomization is used to represent the context, i.e., behaviors of other devices and influence of these behaviors on those of the master and the device under consideration.

The time of the master process is represented by the clock variable θ_m. This is considered the reference time and is used to synchronize the time of the slave clock, represented by the clock variable θ_s. The synchronization works as follows. Periodically, the master broadcasts a *sync* message and immediately after a *followUp* message containing the time t_1 at which the *sync* message has been sent. Time t_1 is observed on the master clock θ_m. The slave records in t_2 the reception time of the *sync* message. Then, after the reception of the *followUp*, it sends a delay *request* message to the master and records its emission time t_3. Both t_2 and t_3 are observed on the slave clock θ_s. The master records on t_4 the reception time of the *request* message and sends it back to the slave on the *reply* message. Again, t_4 is observed on the master clock θ_m. Finally, upon reception of *reply*, the slave computes the offset between its time and the master time based on $(t_i)_{i=1,4}$ and updates its clock accordingly. In our model, the offset is computed differently in two different situations. In the first situation, which is depicted in Figure 5, the average delays from master to slave and back are supposed to be equal i.e., $\mu(\rho_1) = \mu(\rho_2)$. In the second situation, delays are supposed to be asymmetric, i.e., $\mu(\rho_1) \neq \mu(\rho_2)$. In this case, synchronization is improved by using an extra offset correction which compensates for the difference, more precisely, $o := (t_2 + t_3 - t_1 - t_4)/2 + (\mu(\rho_2) - \mu(\rho_1))/2$. This offset computation is an extension of the PTP specification and has been considered since it ensures better precision when delays are not symmetric (see Section 6).

Encoding the abstract model of timed automata given in Figure 5 in BIP is quite straightforward and can be done with the method presented in [5]. The distribution on the delays is implemented as a new C function in the BIP model. It is worth mentioning that, since the two automata are deterministic, the full system depicted in Figure 5 is purely stochastic.

The accuracy of the synchronization is defined by the absolute value of the difference between the mas-

ter and slave clocks $|\theta_m - \theta_s|$. Our aim is to check the (safety) property of bounded accuracy ϕ_Δ, that is, *always* $|\theta_m - \theta_s| \leq \Delta$ for arbitrary fixed non-negative real Δ.

We introduce hereafter an analytic method to estimate the precision achieved within one round of the PTP protocol, depending on several (abstract) parameters such as the initial difference and the bounds (lower, upper) on the allowed drift of the two clocks, the bounds (lower, upper) of the communication delay between the master and the slave, etc.

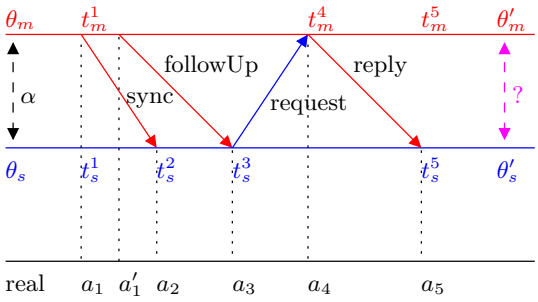


Fig. 6: One round of the PTP protocol.

The difference between the master and the slave clocks after one PTP round can be determined from a system of arithmetic non-linear constraints extracted from the model of the protocol and communication media. Let us consider one complete round of the protocol as depicted in Figure 6. The first two axes correspond to the (inaccurate) clocks of the master and slave respectively. The third axis corresponds to a perfect reference clock. Using the notation defined on the figure we can establish several constraints relating initial and final values of the master and slave clocks $(\theta_m, \theta_s, \theta'_m, \theta'_s)$, timestamps (t_1, t_2, t_3, t_4), offset (o), communication delays (L_1, U_1, L_2, U_2), reference dates $(a_1, a'_1, a_2, a_2, a_4)$ as follows:

- initial constraints and initial clock difference α
  $\theta_m - \theta_s = \alpha$, $\theta_m = t_m^1$, $\theta_s = t_s^1$
- evolution of the master clock is constrained by some maximal drift ϵ_m
  $(1 - \epsilon_m)(a_4 - a_1) \leq t_m^4 - t_m^1 \leq (1 + \epsilon_m)(a_4 - a_1)$
  $(1 - \epsilon_m)(a_5 - a_4) \leq t_m^5 - t_m^4 \leq (1 + \epsilon_m)(a_5 - a_4)$
- evolution of the slave clock is constrained by some maximal drift ϵ_s
  $(1 - \epsilon_s)(a_2 - a_1) \leq t_s^2 - t_s^1 \leq (1 + \epsilon_s)(a_2 - a_1)$
  $(1 - \epsilon_s)(a_3 - a_2) \leq t_s^3 - t_s^2 \leq (1 + \epsilon_s)(a_3 - a_2)$
  $(1 - \epsilon_s)(a_5 - a_3) \leq t_s^5 - t_s^3 \leq (1 + \epsilon_s)(a_5 - a_3)$
- communication delays, forward (L_1, U_1) and backward (L_2, U_2)
  $L_1 \leq a_2 - a_1 \leq U_1$
  $L_1 \leq a_3 - a'_1 \leq U_1$
  $L_2 \leq a_4 - a_3 \leq U_2$
  $L_1 \leq a_5 - a_4 \leq U_1$
- internal master delay (l, u) for sending the *followUp* after *sync*
  $l \leq a'_1 - a_1 \leq u$
- offset computation and final clocks values
  $o = (t_s^2 + t_s^3 - t_m^1 - t_m^4)/2$, $\theta'_m = t_m^5$, $\theta'_s = t_s^5 - o$

This system of constraints encodes precisely the evolution of the two clocks within one round of the protocol. The synchronization achieved corresponds to the difference $\theta'_m - \theta'_s$. We analyze different configurations and we obtain the following results:

1. symmetric delays ($L_1 = L_2 = L$, $U_1 = U_2 = U$), no drift ($\epsilon_m = \epsilon_s = 0$) then $-\frac{U-L}{2} \leq \theta'_m - \theta'_s \leq \frac{U-L}{2}$
2. symmetric delays ($L_1 = L_2 = L$, $U_1 = U_2 = U$), no master drift ($\epsilon_m = 0$) then $-\frac{U-L}{2} - \frac{\epsilon_s(5U-L+u)}{2} \leq \theta'_m - \theta'_s \leq \frac{U-L}{2} + \frac{\epsilon_s(2U+2L+u)}{2}$
3. asymmetric delays, no drift ($\epsilon_m = \epsilon_s = 0$) then $-\frac{U_2-L_1}{2} \leq \theta'_m - \theta'_s \leq \frac{U_1-L_2}{2}$
4. asymmetric delays, no master drift ($\epsilon_m = 0$) then $-\frac{U_2-L_1}{2} - \frac{\epsilon_s(3U_1+2U_2-L_1+u)}{2} \leq \theta'_m - \theta'_s \leq \frac{U_1-L_2}{2} + \frac{\epsilon_s(2U_1+2L_2+u)}{2}$

We remark that, in general, the precision achieved does not depend on the initial difference between the two clocks. Nevertheless, it is strongly impacted by the communication jitter, which is, the difference $U - L$ in the symmetric case and differences $U_2 - L_1, U_1 - L_2$ in the asymmetric case.

Moreover, we remark that in the asymmetric case, the lower and upper bounds are not *symmetric* i.e., the precision interval obtained is not centered around 0. The bounds of the interval suggest us an additional offset correction:

$$\delta_o = \frac{(U_2 - U_1) + (L_2 - L_1)}{4}$$

which will *shift* the interval towards 0. For example, using this additional correction we obtain in the case of asymmetric delays with no drift better precision: $-\frac{(U_2+U_1)-(L_1+L_2)}{4} \leq \theta'_m - \theta'_s \leq \frac{(U_1+U_2)-(L_1+L_2)}{4}$

This analysis shows that it is indeed possible to precisely relate the precision of clock synchronization to the network communication jitter (and the clock drift, if any). That is, a bound on the jitter can be used to derive an upper bound on the precision guaranteed by PTP. Nevertheless, this estimation method appears too pessimistic for concrete application to the HCS case study: the bounds on the jitter being far too big than the expected clock synchronization accuracy. For this

reason, we turn to a stochastic analysis, which can provide finer answers, such as, probabilities for satisfying the synchronization, or the average proportion of failures, etc.

6.2 Model Simulations

In this section, we describe our approach to learn the probability distribution over the delays. Consider the server and a given device. In a first step, we run simulations on the system and measure the end-to-end delays of all PTP messages between the selected device and the server. For example, consider the case of delay *request* messages and assume that we made 33 measures. The result will be a series of delay values and, for each value, the number of times it has been observed. As an example, delay 5 has been observed 3 times, delay 19 has been observed 30 times. The probability distribution is represented with a table of 33 cells. In our case, 3 cells of the table will contains the value 5 and 30 will contain the value 19. The BIP engine will select a value in the table following a uniform probability distribution.

According to our experiments, 2000 delay measurements are enough to obtain an accurate estimation of the probability distribution. However, for confidence reasons, we have conducted 4000 measurements for each device. In Figure 8, we give the distributions that are obtained using 4000, 8000, 16000, 24000 and 32000 measures on Devices (0,3) and (3,3). One can observe that the increment in terms of number of measures does not influence the shape of the distribution.

We have also observed that the value of the distribution clearly depends on the position of the device in the topology. This is shown in Figure 7, where Figure 7a shows the distribution of delays from Device(0,3) to the server and Figure 7b shows the delay from Device(3,3)to the server. It is worth mentioning that running one single simulation allowing 4000 measurements of the delay of PTP frames requires running the PTP protocol with an increased frequency i.e., the default PTP period (2 seconds) being far too big compared with the period for sending audio/video packets (tens of milliseconds). Therefore, we run simulations where PTP is executed once every 2 milliseconds and, we obtain 4000 measurements by simulating approximately 8 seconds of the global system lifetime. Each simulation uses microsecond time granularity and takes around 40 minutes on a Pentium 4 running under a Linux distribution.

The reader could wonder whether the distribution on the delays is not time or state dependent. The reason is that we experimentally observed that the delays are independent from the time when they appear. See Figure 9 for an illustration.

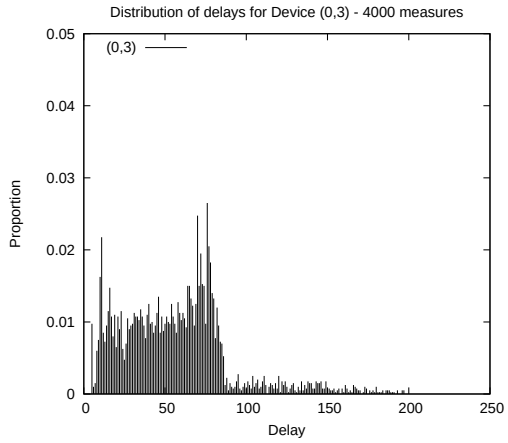
Remark 1 In BIP, simulations of the heterogeneous system are generated by computing on-the-fly part of the composition of the many components that participate in the design. When performing this computation, one has to resolve the non-determinism that arises from the composition of the components. This is done by random choices using uniform distributions among enabled interactions. The key observation, which is relevant to statistics, is that the mixing of those many random effects results in smooth distributions characterizing the random behaviors of the subsystems of interest. Furthermore, the particular form for the random choices performed during the simulation does not really influence the resulting stochastic behavior of the stochastic abstraction — this relies on arguments of convergence toward so-called *stable distributions* [31]. Our approach is thus clearly different from those who would have artificially characterized the stochastic behavior of the subsystems.

6.3 Experiments on Precision Estimation for PTP

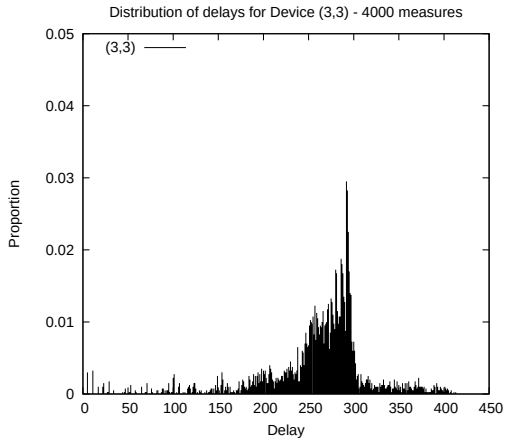
We now report on our experiments. We first assume that packets are scheduled with the fixed-priority mechanism introduced in Section 5.3.1. At the end of the section, we report on the influence of using another scheduling algorithm that is the WFQ scheduling algorithm of Section 5.3.2.

Three sets of experiments are conducted. The first one is concerned with the bounded accuracy property (see Section 6.1). In the second one, we study average failure per execution for a given bound. Finally, we study the influence of drift on the results.

Property 1 : Synchronization. Our objective is to compute the smallest bound Δ under which synchronization occurs properly for any device. We start with an experiment that shows that the value of the bound depends on the place of the device in the topology. For doing so, we use $\Delta = 50\mu s$ as a bound and then compute the probability for synchronization to occur properly for all the devices. In the paper, for the sake of presentation, we will only report on a sampled set of devices :$(0,0)$, $(0,3)$, $(1,0)$, $(1,10)$, $(2,0)$, $(2,3)$, $(3,0)$, $(3,3)$, but our global observations extend to any device. We use PESTIMATION with a confidence of 0.1. The results, which are reported in Figure 10a, show that the place in the topology plays a crucial role. Device $(3,3)$ has the best probability value and Device $(2,0)$ has the worst one. All the results in Figure 10a



(a) Distribution of delays for Device (0,3) with 4000 measures



(b) Distribution of delays for Device (3,3) with 4000 measures

Fig. 7: Delay distribution for Device(0,3) and Device(3,3).

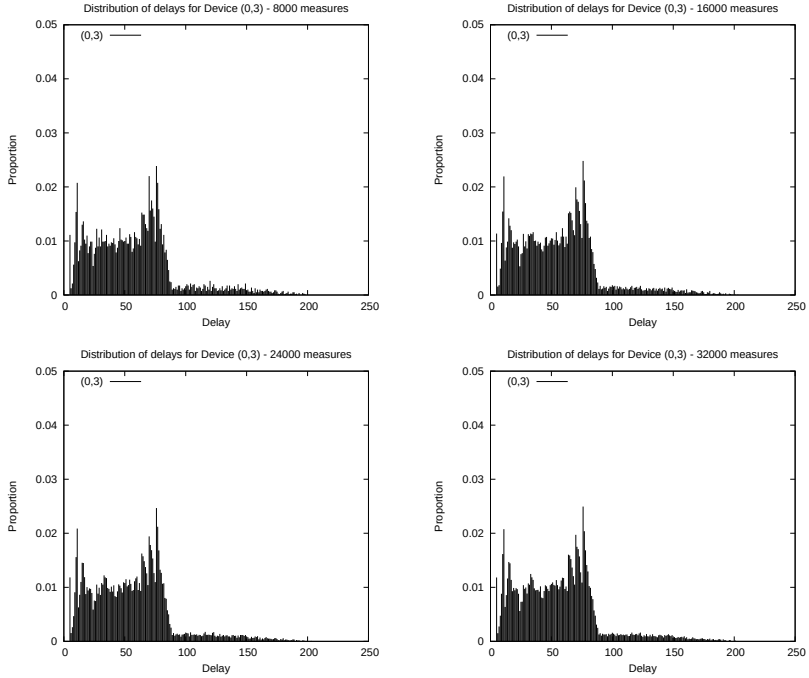
have been conducted on the PTP model with asymmetric delays correction. For the symmetric case, the probability values are much smaller. As an example, for Device $(0,0)$, it decreases from 0.388 to 0.085. The above results have been obtained in less than 4 seconds. As a second experiment, we have used SPRT and SSP to validate the probability value found by PESTIMATION with a higher degree of confidence. The results, which are presented in Table 2 for Device $(0,0)$, show that SPRT is faster than SSP and PESTIMATION.

Our second step was to estimate the best bound. For doing so, for each device we have repeated the previous experiments for values of Δ between $10\mu s$ and $120\mu s$. Figure 11a gives the results of the probability of satisfying the bounded accuracy property as a function of the bound Δ for the asymmetric version of PTP. The figure shows that the smallest bound which ensure synchronization for any device is $105\mu s$ (for Device $(3,0)$). However, devices $(0,3)$ and $(3,3)$ already satisfy the property with probability 1 for $\Delta = 60\mu s$.

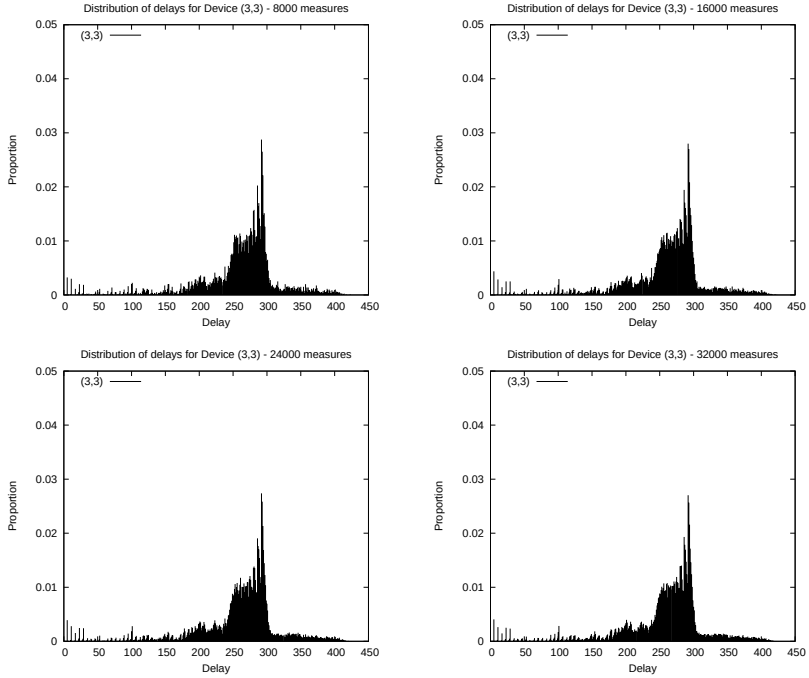
Table 3 shows, for Device (0,0), a comparison of the time and number of simulations required for PESTIMATION and SSP with the same degree of confidence.

The above experiments have been conducted assuming simulations of 1000 BIP interactions and 66 rounds of the PTP protocol. Since each round of the PTP takes two minutes, this also corresponds to 132 minutes of the system's life time. We now check whether the results remain the sames if we lengthen the simulations and hence system's life time. Figure 12 shows, for Devices $(0,0)$ and $(3,0)$, the probability of synchronization for various values of Δ and various length of simulations (1000, 4000, 8000 and 10000 (660 minutes of system's life time) steps). We used PESTIMATION with a precision and a confidence of 0.1. The best bounds do not change. However, the longest the simulations are, the more the probability tends to be either 0 or 1 depending on the bound.

Property 2 : Average failure. In the previous experiment, we have computed the best bound to guarantee the bounded accuracy property. It might be the case that the bound is too high regarding the user's requirements. In such case, using the above results, we can already report on the probability for synchronization to occur properly for smaller values of the bound. We now give a finer answer by quantifying the average and worst number of failures in synchronization that occur *per simulation* when working with smaller bounds, that means, how often the synchronization property gets violated. For a given simulation, the *proportion of failures* is obtained by dividing the number of failures by the number of rounds of PTP. We will now estimate, for a simulation of 1000 steps (66 rounds of the PTP), the average and worst value for this proportion. To this purpose, we have measured (for each device) this proportion on 1199 simulations with a synchronization bound of $\Delta = 50\mu s$. As an example, we obtain average proportions of 0.036 and 0.014 for Device $(0,0)$ using the symmetric and asymmetric versions of PTP respectively. As a comparison, we obtain average proportions of 0.964 and 0.075 for Device $(3,0)$. The average proportion of failures with the bound $\Delta = 50\mu s$ and the asymmetric version of PTP is given in Figure 10b. Figure 13a presents, for the sampled devices, the worst proportion of failures using the asymmetric version of PTP. The worst value is $0,25$, which is obtained for Device $(2,0)$. On the other hand, the worst value is only $0,076$ for



(a) Distributions of delays for Device (0,3)



(b) Distributions of delays for Device (3,3)

Fig. 8: Probability distributions over the delays for devices (0,3) and (3,3) observed with different number of measures.

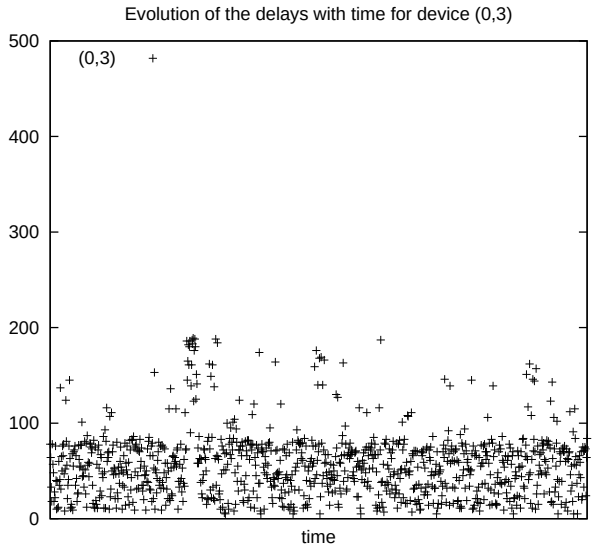
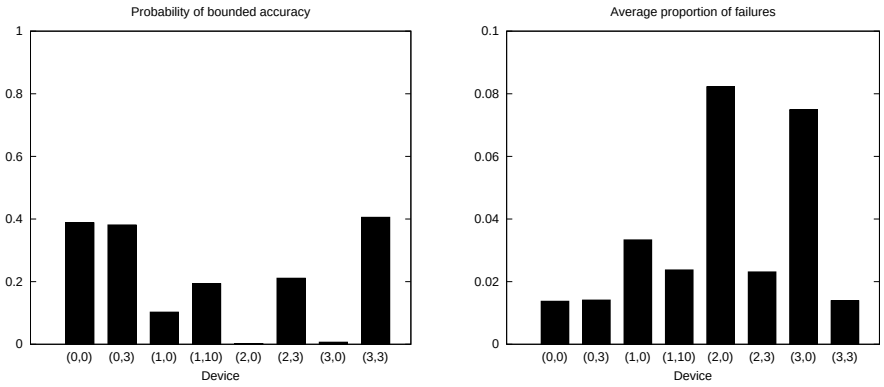


Fig. 9: Evolution of the delays with time for device (0,3)



(a) Probability of satisfying bounded accuracy. (b) Average proportion of failures.

Fig. 10: Probability of satisfying the bounded accuracy property and average proportion of failures for a bound $\Delta = 50\mu s$ and the asymmetric version of PTP.

Device $(0, 0)$. The experiment, which takes about 6 seconds per device, was then generalized to other values of the bound. Figures 11b and 13b give the average and worst proportion of failure as a function of the bound.

The above experiment gives, for several values of Δ and each device, the worst failure proportion with respect to 1199 simulations. We have also used PESTIMATION with confidence of 0.1 and precision of 0.1 to verify that this value remains the same whatever the number of simulations is. The result was then validated using SSP with precision of 10^{-3} and confidence of 10^{-10}. Each experiment took approximately two minutes. Finally, we have conducted experiments to check whether the same results hold for longer simulations. Figure 14a shows that the average proportion does not change and Figure 14b shows that the worst proportion decreases when the length of the simulation increases.

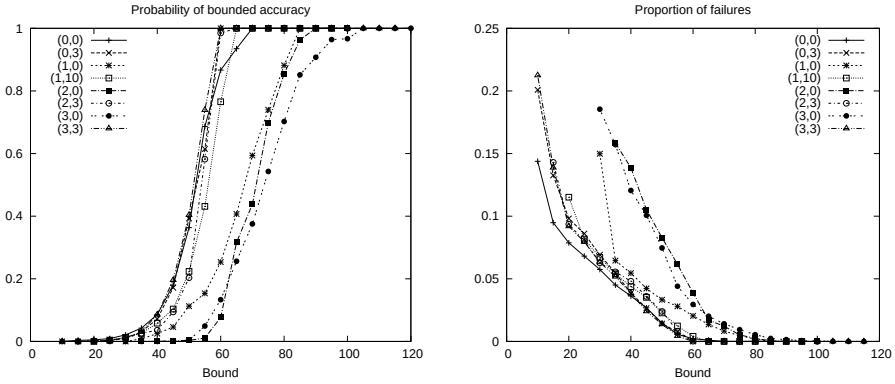
Clock Drift. We have considered a modified version of the stochastic PTP model with drifting clocks. Drift is used to model the fact that, due to the influence of the hardware, clocks of the master and the device may not progress at the same rate. In our model, drift is incorporated as follows: each time the clock of the server is increased by 1 time unit, the clock of the device is increased by $1 + \varepsilon$ time units, with $\varepsilon \in [-10^{-3}, 10^{-3}]$. Using this modified model, we have re-done the experiments of the previous sections and observed that the result remains almost the same. This is not surprising as the value of the drift significantly smaller than the communication jitter, and therefore it has less influ-

Precision	10^{-1}		10^{-2}		10^{-3}	
Confidence	10^{-5}	10^{-10}	10^{-5}	10^{-10}	10^{-5}	10^{-10}
PESTIMATION	4883 $17s$	9488 $34s$	488243 $29m$	948760 $56m$	48824291 $> 3h$	94875993 $> 3h$
SSP	1604 $10s$	3579 $22s$	161986 $13m$	368633 $36m$	16949867 $> 3h$	32792577 $> 3h$
SPRT	316 $2s$	1176 $7s$	12211 $53s$	22870 $1m38s$	148264 $11m$	311368 $31m$

Table 2: Number of simulations / Amount of time required for PESTIMATION, SSP and SPRT.

Precision	10^{-1}		10^{-2}		10^{-3}	
Confidence	10^{-5}	10^{-10}	10^{-5}	10^{-10}	10^{-5}	10^{-10}
SSP / SPRT	110 $1s$	219 $1s$	1146 $6s$	2292 $13s$	11508 $51s$	23015 $1m44s$

Table 3: Number of simulations / Amount of time required for PESTIMATION and SSP.



(a) Probability of satisfying bounded accuracy. (b) Average proportion of failures.

Fig. 11: Probability of satisfying the bounded accuracy property and average proportion of failures as functions of the bound Δ for the asymmetric version of PTP.

ence of the synchronization. A drift of 1 time unit has a much higher impact on the probability. As an example, for Device $(0, 0)$, it goes from a probability of $0,387$ to a probability of $0,007$. It is worth mentioning that exhaustive verification of a model with drifting clocks is not an easy task as it requires to deal with complex differential equations. When reasoning on one execution at a time, this problem is avoided.

Experiments with WFQ. We now consider the influence of the scheduling policy by replacing the fixed priorities mechanism with the WFQ algorithm presented in Section 5.3.2. As we already said, the result of applying this algorithm depend on the pre-defined allocated service ratio for every category of packets. We consider 3 scenarios. Probabilities are estimated and validated using PESTIMATION, SSP, and SPRT.

We start with a scenario that should lead to results that are close to those we obtained for fixed-priorities. This scenario consists in giving a very high ratio to the PTP packets. This is done to ensure that these packets never have to wait before being sent. More precisely, we used the following ratio: PTP packets have a ratio $r_{\mathsf{PTP}} = 5$, Audio packets have a ratio $r_A = 2$, Event packets have a ratio $r_E = 2$ and Video packets have a ratio $r_V = 1$. This configuration of the ratios is addressed as 5:2:2:1. The results of this experiments are given in Figure 15. We observe that the results are not as good as for fixed-priorities. More precisely, the best and worst bounds for satisfying bounded accuracy with probability 1 are $70\mu s$ (obtained for Device (0,0)) and $130\mu s$ (obtained for Device (2,0)), respectively. For fixed-priorities, we obtained $60\mu s$ and $105\mu s$, respectively.

In the second scenario, we decrease the importance of PTP packets in order to observe degradations in the results, if any. PTP packets have a ratio $r_{\mathsf{PTP}} = 4$, Audio packets have a ratio $r_A = 3$, Event packets have a

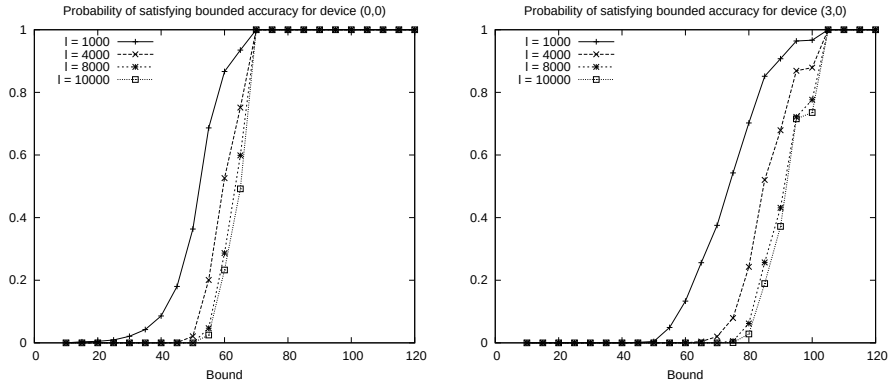
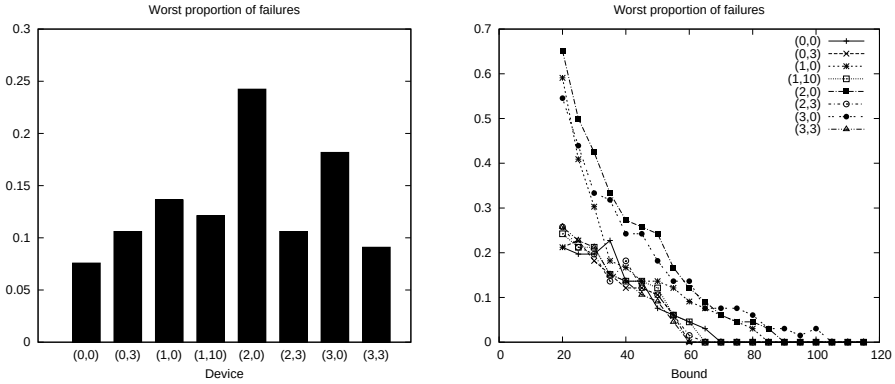


Fig. 12: Evolution of the probability of satisfying the bounded accuracy property with the length of the simulations for the asymmetric version of PTP.



(a) Worst proportion of failures for bound $\Delta = 50\mu s$.

(b) Worst proportion of failures as a function of the bound Δ.

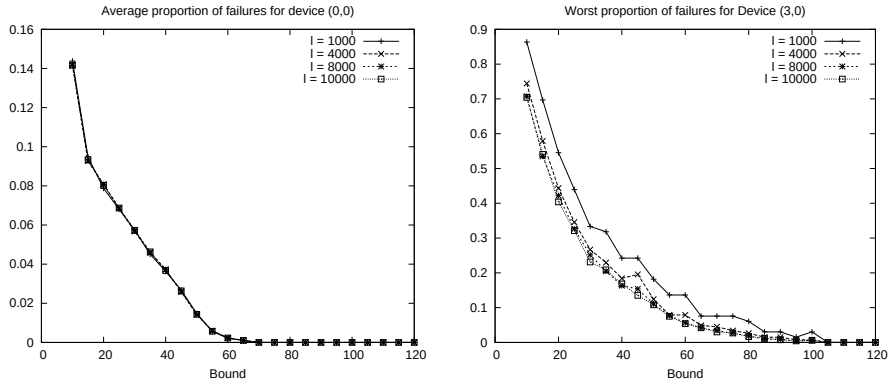
Fig. 13: Worst proportion of failures for the industrial bound $\Delta = 50\mu s$ and as a function of the bound Δ for the asymmetric version of PTP.

ratio $r_E = 2$ and Video packets have a ratio $r_V = 1$. This configuration of the ratios is addressed as 4:3:2:1. Results of this experiment are given in Figure 16. Those results are worse than those obtained for the first scenario. Indeed, the best bound for satisfying bounded accuracy with probability 1 is now $120\mu s$, that is obtained for Device (0,0), and the worst bound is $295\mu s$, that is obtained for Device (2,0).

The last scenario consists in considering ratios that are closer to the reality of the bandwidth needed for each type of packets. PTP packets have a ratio $r_{\mathsf{PTP}} = 2$, Audio packets have a ratio $r_A = 3$, Event packets have a ratio $r_E = 1$ and Video packets have a ratio $r_V = 4$. This configuration of the ratios is addressed as 2:3:1:4. Results of this experiment are given in Figure 17. The results are even worse than those obtained for the second scenario. Indeed, the best bound for satisfying bounded accuracy with probability 1 is $140\mu s$, that is obtained for Device (0,3), and the worst bound is $425\mu s$, that is obtained for Device (2,0).

7 Conclusion and Future Work

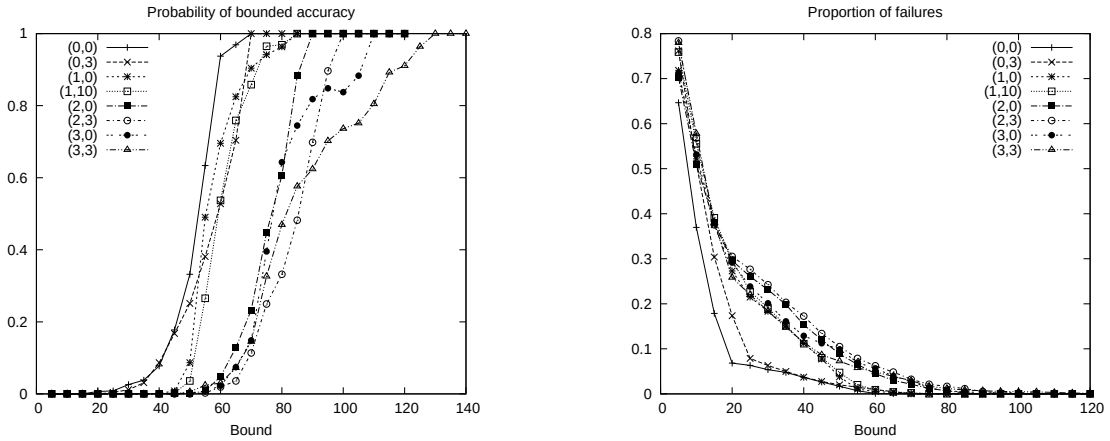
This paper introduces the concept of stochastic abstraction and studies one of its applications in the context of verifying properties of a large heterogeneous case study that cannot be handled by existing formal method techniques. It is worth mentioning that we have also applied the stochastic abstraction principle to verify properties of a *Avionics Full Duplex Switched Ethernet (AFDX)* [1]. For this AFDX case study, we have shown that stochastic abstraction and statistical model checking perform better and are more general than techniques such as network calculus [10,11,24] or timed model checking [3].



(a) Average proportion of failure for Device (0,0).

(b) Worst proportion of failure for Device (3,0).

Fig. 14: Evolution of the average and worst proportion of failures with the length of the simulations for the asymmetric version of PTP.



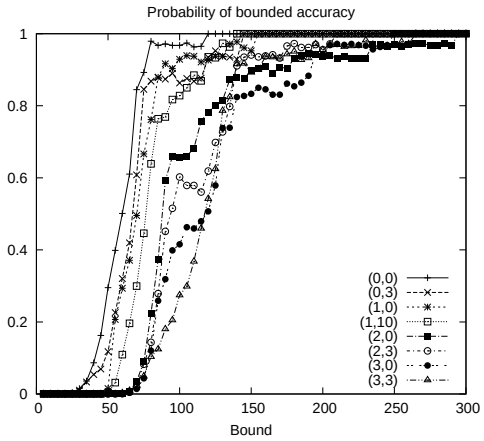
(a) Probability of satisfying bounded accuracy using WFQ with ratio 5:2:2:1

(b) Average proportion of failures using WFQ with ratio 5:2:2:1

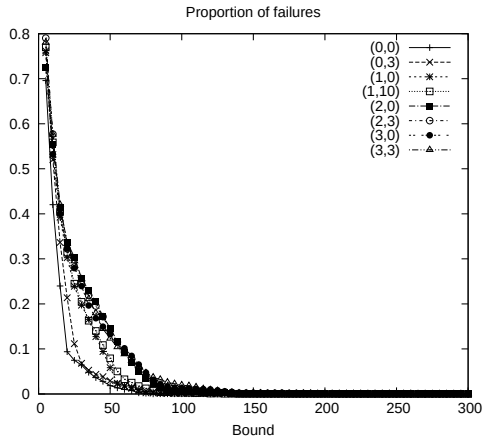
Fig. 15: Probability of satisfying bounded accuracy and average proportion of failures using WFQ with ratio 5:2:2:1

As a future work, one could improve the applicability of existing statistical model checking techniques by considering properties that cannot be verified on finite-time traces [23, 25]. Another interesting direction is to improve the efficiency of statistical model checking. Due to his engineering knowledge about the system, the designer may guess some prior knowledge regarding the probability for the system to violate the property. This information could be used to improve the efficiency of the statistical model checking algorithms by making prior hypothesis on the probability for the system to be correct, which may reduce the number of simulations needed to conclude. Also, as the system is assumed to be "well-designed", one can postulate that the property under verification should rarely be falsified. This means that we are trying to compute probabilities of violation that should be very close to 0. Statistical model checking algorithms should address this issue in an efficient manner. A solution could be to combine the statistical model checking approach with the concept of rare event simulation [9].

As we have seen, the *stochastic abstraction* is obtained by computing simulations of the entire heterogeneous (system level model). The objective is to learn an estimate of the distribution representing the environment where the subsystem under consideration is running. For the HCS case study, the estimation was computed from a high number of simulations, which should guarantee a good accuracy (even though we were not able to characterize it). However, in general, generating simulations of a complex design may take time. We thus suggest to use techniques from the statistical area

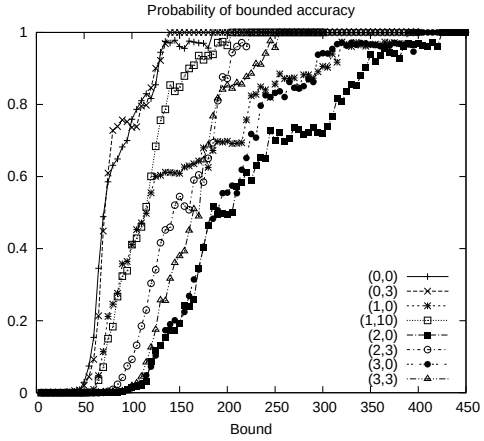


(a) Probability of satisfying bounded accuracy using WFQ with ratio 4:3:2:1

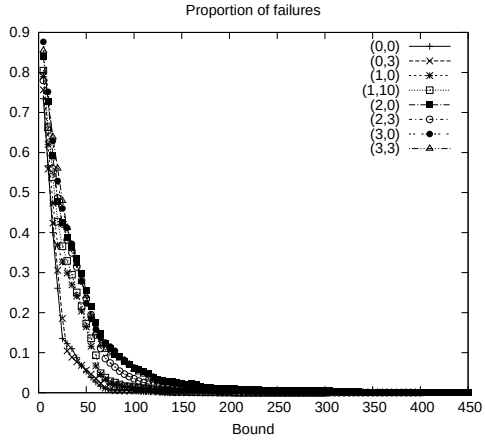


(b) Average proportion of failures using WFQ with ratio 4:3:2:1

Fig. 16: Probability of satisfying bounded accuracy and average proportion of failures using WFQ with ratio 4:3:2:1



(a) Probability of satisfying bounded accuracy using WFQ with ratio 2:3:1:4



(b) Average proportion of failures using WFQ with ratio 2:3:1:4

Fig. 17: Probability of satisfying bounded accuracy and average proportion of failures using WFQ with ratio 2:3:1:4

such as bootstrap [14] to better exploit the simulations in generating an accurate estimate of the distribution. Stochastic abstraction may also be combined with classical abstraction techniques, especially when memory has to be considered in the design.

In this paper, we have observed that the BIP framework allows to describe a faithful model of the HCS, and the observation made on the BIP model should also remain valid on the concrete implementation. However, this is only an observation, not a theoretical guarantee. This means that in order to cope with many other industrial case studies, we will certainly have to integrate our technology in the tool chain of industrials. Such an integration introduces new difficulties. As an example, it requires to be able to jointly simulate models of different parts of the system, possibly expressed using different formalisms. Fortunately, corresponding so-called "hosted and co simulation" technologies (see [26] for an illustration) have been recently developed by tool vendors (such as our industrial partner) to cope with this problem. We will integrate this technology and extend it to a more general context. Another major difficulty will be to provide feedback to the designer in case his requirements are not satisfied.

Finally, we believe it is a very challenging problem to relate the confidence we have on the estimated distribution with the confidence degree of SMC algorithms. Being able to answer this question, which was not considered in this paper, would give a higher confidence on the correctness of the heterogeneous system.

References

1. ARINC 664, Aircraft Data Network, Part 7: Avionics Full Duplex Switched Ethernet (AFDX) Network. (2005)
2. 61588, I.I.: Precision clock synchronization protocol for networked measurement and control systems (2004)
3. Alur, R., Dill, D.: A Theory of Timed Automata. Theoretical Computer Science 126, 183–235 (1994)
4. Basu, A., Bensalem, S., Bozga, M., Delahaye, B., Legay, A., Siffakis, E.: Verification of an afdx infrastructure using simulations and probabilities. In: Proc 1st Conference on Runtime Verification (RV), Malta, 2010. Springer-Verlag (2010)
5. Basu, A., Bozga, M., Sifakis, J.: Modeling Heterogeneous Real-time Systems in BIP. In: SEFM06, Pune, India. pp. 3–12 (September 2006)
6. Basu, A., Bensalem, S., Bozga, M., Caillaud, B., Delahaye, B., Legay, A.: Statistical abstraction and model-checking of large heterogeneous systems. In: FORTE 2010. pp. 32–48. LNCS 6117, Springer-Verlag (2010)
7. Bensalem, S., Delahaye, B., Legay, A.: Statistical model checking: Present and future. In: Proc 1st Conference on Runtime Verification (RV), Malta, 2010. Springer-Verlag (2010)
8. The BIP Toolset, http://www-verimag.imag.fr/∼async/bip.php
9. Bucklew, J.: Introduction to Rare event Simulation. Springer (2004)
10. Charara, H., Fraboul, C.: Modelling and simulation of an avionics full duplex switched ethernet. In: Proceedings of the Advanced Industrial Conference on Telecommunications/ Service Assurance with Partial and Intermittent Resources Conference/E-Learning on Telecommunication Workshop. IEEE (2005)
11. Charara, H., Scharbarg, J.L., Ermont, J., Fraboul, C.: Methods for bounding end-to-end delays on AFDX network. In: ECRTS. IEEE Computer Society (2006)
12. Clarke, E.M., Donzé, A., Legay, A.: Statistical model checking of mixed-analog circuits with an application to a third order delta-sigma modulator. In: HVC. LNCS, vol. 5394, pp. 149–163. Springer (2008), to appear
13. Clarke, E.M., Faeder, J.R., Langmead, C.J., Harris, L.A., Jha, S.K., Legay, A.: Statistical model checking in biolab: Applications to the automated analysis of t-cell receptor signaling pathway. In: CMSB. LNCS, vol. 5307, pp. 231–250. Springer (2008)
14. Efron, B., Tibshirani, R.: An Introduction to the Bootstrap. Hall/CRC Press Monographs on Statistics and Applied Probability (1994)
15. Grosu, R., Smolka, S.A.: Monte carlo model checking. In: TACAS. LNCS, vol. 3440, pp. 271–286. Springer (2005)
16. Hérault, T., Lassaigne, R., Magniette, F., Peyronnet, S.: Approximate probabilistic model checking. In: VMCAI. LNCS, vol. 2937, pp. 73–84. Springer (2004)
17. Hoeffding, W.: Probability inequalities. Journal of the American Statistical Association 58, 13–30 (1963)
18. Jansen, D.N., Katoen, J.P., M.Oldenkamp, Stoelinga, M., Zapreev, I.S.: How fast and fat is your probabilistic model checker? an experimental performance comparison. In: HVC. LNCS, vol. 4899. Springer (2007)
19. Jha, S.K., Clarke, E.M., Langmead, C.J., Legay, A., Platzer, A., Zuliani, P.: A bayesian approach to model checking biological systems. In: CMSB. LNCS, vol. 5688, pp. 218–234. Springer (2009)
20. Katoen, J.P., Zapreev, I.S.: Simulation-based ctmc model checking: An empirical evaluation. In: Proc. of 6th Int. Conference on the Quantitative Evaluation of Systems (QEST). pp. 31–40. IEEE Computer Society (2009)
21. Laplante, S., Lassaigne, R., Magniez, F., Peyronnet, S., de Rougemont, M.: Probabilistic abstraction for model checking: An approach based on property testing. ACM Trans. Comput. Log. 8(4) (2007)
22. Parekh, A.K., Gallagher, R.G.: A generalized processor sharing approach to flow control in integrated services networks: the multiple node case. IEEE/ACM Trans. Netw. 2(2), 137–150 (1994)
23. Rabih, D.E., Pekergin, N.: Statistical model checking using perfect simulation. In: Proc. 7th Int. Conference on Automated Technology for Verification and Analysis (ATVA). Lecture Notes in Computer Science, vol. 5799, pp. 120–134. Springer (2009)
24. Scharbarg, J.L., Fraboul, C.: Simulation for end-to-end delays distribution on a switched ethernet. In: ETFA. IEEE (2007)
25. Sen, K., Viswanathan, M., Agha, G.: Statistical model checking of black-box probabilistic systems. In: CAV. pp. 202–215. LNCS 3114, Springer (2004)
26. Steinkellner, S., Andersson, H., Lind, I., Krus, P.: Hosted simulation for heterogeneous aircraft system development. In: Proc. of 26th Int. Congress of the Aeronautical Sciences (2008)
27. Wald, A.: Sequential tests of statistical hypotheses. Annals of Mathematical Statistics 16(2), 117–186 (1945)
28. Younes, H.L.S.: Verification and Planning for Stochastic Processes with Asynchronous Events. Ph.D. thesis, Carnegie Mellon (2005)
29. Younes, H.L.S.: Error control for probabilistic model checking. In: VMCAI. pp. 142–156. LNCS 3855, springer-verlag (2006)
30. Younes, H.L.S., Kwiatkowska, M.Z., Norman, G., Parker, D.: Numerical vs. statistical probabilistic model checking. STTT 8(3), 216–228 (2006)
31. Zolotarev, V.M.: One-dimensional stable distribution. American Mathematical Society (1986)