Lot 4.2

Technologie de modélisation

Probabilités

Parametric and Probabilistic Verification of Nicollin-Sifakis-Yovine's Model of the CSMA/CD Protocol

Description :	Dans cette fourniture, nous vérifions les propriétés du protocole CSMACD modelisé par Nicollin Sifakis et Yovine de manière non probabiliste (en utilisant HyTech) et de manière probabiliste (avec PRISM).
Auteur(s) :	Laurent Fribourg, Stephane Messika, Claudine Picaronny
Référence :	Averroes / Lot 4.2 / Fourniture 3 / V1.0
Date :	14 juin 2004
Statut :	Version a valider
Version :	1.0

Réseau National des Technologies Logicielles
Projet subventionné par le Ministère de la Recherche et des Nouvelles Technologies

CRIL Technology, France Télécom R&D, INRIA-Futurs, LaBRI (Univ. de Bordeaux – CNRS), LIX (École Polytechnique, CNRS) LORIA, LRI (Univ. de Paris Sud – CNRS), LSV (ENS de Cachan – CNRS)

Résumé

Carrier Sense Multiple Access/Collision Detection (CSMA/CD) is the protocol for carrier transmission access in Ethernet networks (international standard IEEE 802.3). This protocol involves the use of probabilities in order to avoid the repetition of collision between messages. In this paper, we consider the classical Nicollin-Sifakis-Yovine's model, where the probabilistic assignments are replaced by non-deterministic ones [14]. In this non-probabilistic framework, they performed an automatic verification of several progress and safety properties using the tool KRONOS. We show here that, using the tools HYTECH and PRISM, it is possible to perform an automatic model-checking of their model, enjoying two main new advantages.

- First, in the non-probabilistic framework, a crucial progress property is automatically proved in a *parametric* way, using the tool HYTECH [7] ;
- Second, some quantitative reachability properties of the *probabilistic* version of the parametric model are automatically proved : A parametric reachability graph is generated (by forward reachability) using HYTECH in order to construct a *finite* Markov decision process, which is checked in turn using the probabilistic tool PRISM [8].

1 Introduction

The Carrier Sense Multiple Access/Collision Detection (CSMA/CD) protocol is a fundamental distributed protocol for networks. Indeed, it is one of the most important part of the widely used IEEE 802.3 international standard (Ethernet Network Communication protocol). In this protocol, several stations share a same channel of communication, and messages can collide. Due to the time of signal propagation, the detection of a collision between two furthest stations takes a time of (at most) σ, where σ is a characteristic feature of the modelled system. In case of collision, the stations do not try to reemit immediately in order to avoid a new collision, but each one selects at random a waiting time before a new attempt.

In [14], Nicollin, Sifakis and Yovine (NSY) gave for the first time a model of the CSMA/CD protocol in the framework of timed automata. In timed automata, the system may evolve either by performing an *action* (or discrete transition), that is a discrete change of location or by a *delay* transition, which lets time pass. Operationally, timed automata are finite state automata enriched with clocks which evolve continuously at the same rate, and can be reset to 0 during the actions of the automata. The probabilistic aspect of the protocol is removed in their model : the random choice of a waiting time is replaced by a non-deterministic assignment. They are thus able to perform automatic verification of several progress and safety properties, using the automatic model checker KRONOS (cf. [17, 4]).

Besides, the removal of the probabilistic apect, the NSY model makes several strong assumptions for the sake of simplicity. In particular, when two messages sent by two stations collide, the detection of such collision is done *simultaneously* by the stations (at *exactly* σ after the sending of the last message). Many other modelling of the CSMA/CD protocol have been proposed since then, which relax this assumption (see, e.g., [5] for a recent modelling, and for discussion of related work). In this paper, we go back to the original Nicollin-Sifakis-Yovine's model : we will show that, with the tools built meanwhile, viz. HYTECH and PRISM, it is possible to perform an automatic model-checking of their model, enjoying two main advantages :

- First, the protocol is modelled in a framework where σ, as well as λ, the time for emitting a message, and c, the maximum value of the non-deterministic assignment of the waiting time are *parameters*, instead of specific values. A crucial progress property is then automatically proved in a generic manner. using the tool HYTECH [7].
- Second, some quantitative reachability properties of the *probabilistic* version of the model are automatically proved : the HYTECH tool is used for generating (by forward reachability) a *finite* Markov decision process where σ and λ are parameters (and c has a specific value), which is checked in turn using the probabilistic tool PRISM [8].

Plan of the paper In Section 2 we recall original Nicollin-Sifakis-Yovine model of CSMA/CD. We, then, give a parametric proof in HyTech of a nonprobabilistic version (Section 3), and a proof of a probabilistic reachability property in PRISM (Section 4). We conclude in Secion 5.

2 Nicollin-Sifakis-Yovine (NSK) model of CSMA/CD

2.1 High-Level Modelling

The CSMA/CD protocol (Carrier Sense Multiple Access with Collision Detection) is a network arbitration protocol which regulates communication between several agents who communicates by a unique channel.

A very abstract and simplified description of the protocol, borrowed from [14], is the following. When a station has a data to send, it first listens to the channel. If it is sensed to free (i.e., no other station is "seen as" transmitting), the station begins sending its message. However, if it detects a busy channel, it waits for a random amount of time and then repeats the operation. When a collision is detected, because several stations transmit (almost) simultaneously, then all of them detect it, abort their transmissions immediately and wait a random time to start all over again. If two messages collide then they are both lost.

The propagation delay of the channel plays an important role in the performance of the CSMA/CD protocol. Let σ be the time for the signal to propagate between the two farthest stations. Suppose that a t a time t a station begins sending a message. Thus within the time interval $[t, t+\sigma)$, it is still possible that some other station transmits if it has data, causing a collision. however, after time $t+\sigma$, the channel will be sensed busy by any station until the current message is delivered. Hence, the maximum time the channel could be sensed idle by any station, after the beginning of transmission, is σ.

In case of collision, each station waits randomly a time between 0 and $c(= b\sigma$ where b is a given integer constant) before trying again.

Assume that only messages of equal length are sent and let λ be the time to send a message. Then, if no collision occurs, a message will be delivered in a time equal to $\lambda + \sigma$.

The CSMA/CD model of NSK supposes that only two stations are connected and that a single error-free bidirectional channel is available for all communications. Only sending messages is modeled and no buffering is allowed.

The protocol is the parallel compositions of two senders and a channel. The actions *ready* and *busy* are used to indicate that the channel is sensed, by the senders, idle or transmitting, respectively. The beginning (resp. end) of a transmission by the i-th sender is indicated by an action beg_i (resp. end_i) synchronized with the channel. A collision is indicated by an action cd synchronized pairwise between the i-th sender and the channel (for all $i = 1, 2$).

The behavior of the channel is as follows. Initially, it is idle and can accept a message from any sender. Suppose that one sender begins transmitting. There is a time interval of length σ within which the channel can accept data from the other sender, causing a collision. In the case of a collision, it takes time σ to the channel to propagate it. If no collision occurs, the channel waits for the terminating signal. When it arrives, the channel waits for the terminating signal. When it arrives, it waits σ and then returns to the initial state, ensuring that the message is completely transmitted in time $\sigma + \lambda$.

Notice that there is a minimum distance of σ between two consecutive transmissions due to the propagation delay of the channel.

2.2 Modelling with probabilistic timed automata

Following [9, 11], we use the framework of *probabilistic timed automata* for modelling the protocol under study. Probabilistic timed automata are extensions of classical timed automata [1] with the ability to express relative likelihoods of actions under the form of probability distributions [12]. The probabilistic aspect appears here under the form of a random assignmment of the form $alea = random[0, ..., c]$, where $alea$ is a discrete variable and c a positive integer. The model of the CSMA/CD protocol consists of three components operating in parallel, namely $Sender_1$, $Sender_2$ (sending stations) and $Chan$ (the channel). In the synchronized product $Sender_1 \| Sender_2 \| Chan$ the set $\mathcal{X}$ of clocks is $\{x_1, x_2, y, t\}$, where x_1 represents the clock of $Sender_1$, x_2 the clock of $Sender_2$, y the clock of $Chan$, and t a clock which is never reset, used for measuring time bounds on reachability properties. The state of the system is of the form $((\ell_1, \ell_2, \ell_3), \overline{v})$ where ℓ_i $(i = 1, 2)$ is the location of $Sender_i$, ℓ_3 the location of $Chan$, and $\overline{v}$ is a valuation of $\mathcal{X} = \{x_1, x_2, y, t\}$. In the following, we use standard conventions for the graphical representation of timed automata : locations are represented as circles with attached invariants of the form $x_i \leq a$, where a is a constant or parameter ; the passing of time is implicit in each location. Actions are represented as oriented edges, labelled with expressions of the form :

`<guard>` | **label**, `<update>`,

where `<guard>` is an (in)equality of the form $x \sim b$ with $\sim \in \{<, =, \leq\}$, $x \in \mathcal{X}$ and b a constant or a parameter, **label** is the label of the synchronized action (or is void if the action is internal), and `<update>` is of the form $x := 0$, $w_i := 0/1$ or $alea_i := random[0, .., c]$.

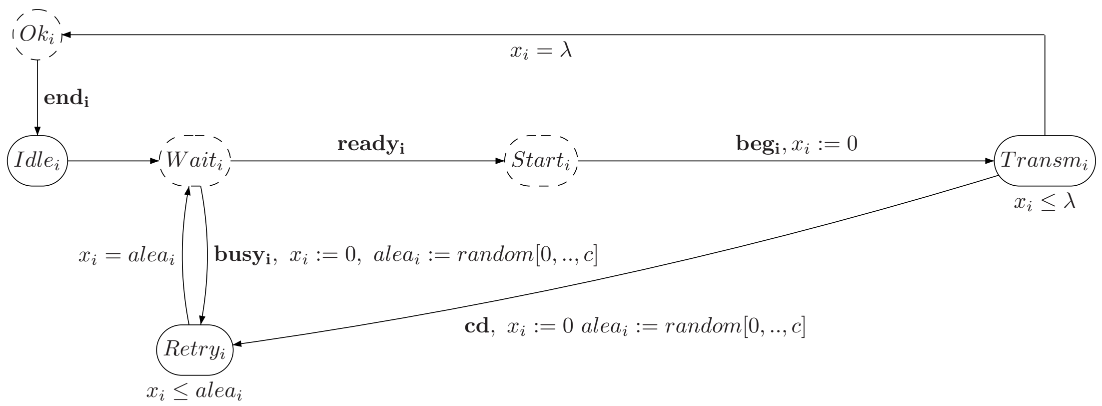


Fig. 1 – Automaton of the i-th sender.

Senders Automata These automata describe the behavior of each sender i. It contains a clock (x_i) and a variable storing the random waiting time ($alea_i$). The automaton of the i-th sender is shown in Figure 1.

There are six locations, among which, locations Ok_i, $Wait_i$ and $Start_i$ (represented as dashed circles) are urgent intermediate locations, where no time can pass. The initial location is $Idle_i$. After spending a certain amount of time (which corresponds to one source of nondeterminism), the sender wants to emit a message, and goes to the transient location $Wait_i$, where the channel is sensed for trying to send the message. If the channel is sensed free (action $\mathbf{ready_i}$), the sender begins to transmit, going to location $Transm_i$ (action $\mathbf{beg_i}$). Otherwise (action $\mathbf{busy_i}$), it goes to location $Retry_i$, where it waits a random time $alea_i$ before a new attempt (going back to $Wait_i$) [1]. In location $Transm_i$, it takes the sender a time λ to complete the transmission, after which it emits a termination signal end_i, and returns to location $Idle_i$ (via transient location Ok_i). Meanwhile, a collision may have occurred (action **cd**), forcing the sender to abandon the transmission : it leaves location $Transm_i$ for $Retry_i$ where it waits a random time $alea_i$ before a new attempt. The maximum value of $alea_i$ is c.

Channel Automaton The probabilistic automaton *Chan*, which represents the channel, is shown in Figure 2. There are seven locations. At the initial location *Idlebus*, the channel is ready to be engaged in the transmission of a message. If one of the senders, say $Sender_1$, starts sending a message ($\mathbf{beg_1}$), the channel resets the clock y, and goes to location $Preactive_1$. While y is less than σ, the channel is still sensed free and the other sender $Sender_2$ may start sending a message itself ($\mathbf{beg_2}$) : a collision then occurs (**cd**). Otherwise, when the value of y equals σ, the channel performs an internal move to location $Active_1$ where it waits for a termination signal ($\mathbf{end_1}$) from $Sender_1$, and responds busy to $Sender_2$. When $\mathtt{end_1}$ arrives, the channel goes to location *End*, where it waits for a time equal to σ before returning to its initial location, thus becoming ready to transmit again.

The originality of this paper is that σ, the worst case propagation delay, and λ the time for achieving the transmission of a message, have not a fixed value (e.g., $\sigma = 26$, $\lambda = 780$, as in [14]), but are *parameters*, with constraint : $\sigma > 0$ and $\lambda \geq 2\sigma$. In the non-probabilistic version, we consider also c, the upper bound of the waiting time $alea_i$, as a parameter (with constraint $c \geq \sigma$). The lower value 2σ is taken for λ, because, for $\sigma < \lambda < 2\sigma$, an anomalous behaviour happens : a collision between two messages may occur (transition of the channel to *Col*), and be followed by the completion of the emission of the first sent message (transition of the station to location *Ok*), prevented the collision to be detected (see Sec. 3.2).

[1] Another modelling consists in waiting a *fixed* (rather than random) amount of time when the channel is sensed to busy. The verification has been also successfully performed along the same lines for such a variant.

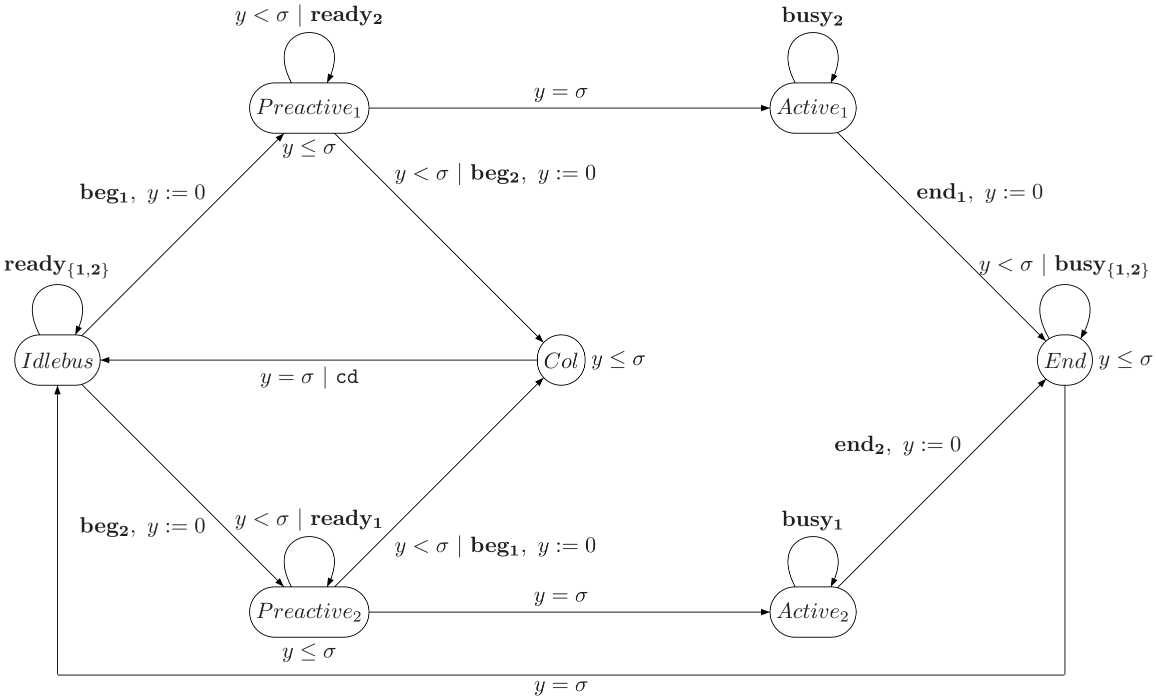


FIG. 2 – Automaton of the channel.

3 Verification of the non-probabilistic model

3.1 Progress property

In the i-th sender, the random assignment $alea_i := random[0, ..., c]$ is replaced by a non-deterministic assigment of $alea_i$ to a number less than or equal to c. We are interested by the following progress property :

Whenever a sender, say $Sender_1$, has a message to transmit, there exists a successful transmission.
This is expressed in the timed remporal logic TCTL [6] as

Prop$_1$: $Init \Rightarrow \forall\Box(\ell_1 = Wait_1 \Rightarrow \exists\Diamond\ \ell_1 = Ok_1)$.
where $Init$ corresponds to the initial state of the system. In the proof, the variables σ, λ and c of the system will be considered as parameters satisfying $0 < 2\sigma \leq \lambda$ and $\sigma \leq c$.

3.2 Verification using HYTECH

The channel and senders automata can be directly implemented into HYTECH [7]. The non-deterministic assignments of $alea_i$ in the sender automata are achieved in HYTECH by means of the instruction $alea_i' \geq 0$, and by attaching the invariant $alea_i \leq c$ to location $Retry_i$. The upper bound c of $alea$ is a parameter initialized with constraint $c \geq \sigma$. Likewise, the parameter λ is initialized with constraint $2\sigma \leq \lambda$, and σ with constraint $\sigma > 0$. The transient location $Wait_i$ (resp.$Start_i$, Ok_i) is implemented by attaching an invariant of the form $z = 0$ to it, where z is a special clock variable, reset to 0 on each edge entering the location. The full code HYTECH is given in Appendix A.

The forward iteration $Post^*(Init)$ requires 21 iterations, where the state $Init$ corresponds to the Hytech code :

ℓ_1=`Idle1` & ℓ_2=`Idle2` & ℓ_3=`Idlebus` & `x1=0` & `x2=0` & `y=0`
& `lambda>=2sigma` & `c>=sigma` & `sigma>0` ;

The backward iteration $Pre^*(Final)$ requires 80 iterations, where the state $Final$ corresponds to the Hytech code :
ℓ_1`=Ok1 & lambda>=2sigma & c>=sigma & sigma>0 & lambda<=k*sigma ;`
where `k` is a constant. The introduction of an upper bound for λ (viz., $k\sigma$) is needed for allowing the termination of Pre^*. For example, in the case $k = 30$[2], the computation of $Pred^*(Final)$ requires 80 iterations.

Property $Prop_1$ is then put under the following equivalent form :
$(Post^*(Init) \cap \{\ell_1 = Wait_1\}) \subseteq Pred^*(\ell_1 = Ok_1)$,
Actually, we prove with HYTECH :

$$(Post^*(Init) \cap \{\ell_1 = Wait_1\} \cap \{\lambda \leq k\sigma\}) \cap \neg(Pred^*(Final) \cap \{\ell_1 = Wait_1\}) = \emptyset.$$

For $k = 30$, the whole process takes 1780 secs using a Pentium IV 2.80GHz with 1 Gb of RAM.

Note that the property does *not* hold when $\lambda < 2\sigma$. A trace using HYTECH can be generated, corresponding to the following anomalous behaviour. First $Sender_2$ emits at some time, say $t = 0$. Then $Sender_1$ moves to location $Wait_1$, and starts to emit at time t', with $\sigma > t' > \lambda - \sigma$. So a collision occurs at time t', making $Chan$ move at location Col while y is reset to 0. The completion of the transmission of the first message is then achieved at time $t'' = \lambda$, with $t'' < t' + \sigma$, making the whole system move to location $(Transm_1, Ok_2, Col)$ with $y = \lambda - t' < \sigma$. Now, the time cannot pass any longer, because action $\mathbf{end_2}$ cannot take place since $Chan$ is not at location $Active_2$, and **cd** cannot take place since $y < \sigma$. This behaviour corresponds to a case of "Zenoness" [**?**]. The location Ok_1 is therefore not reachable from $Wait_1$, when λ is smaller than 2σ. On the other hand, when $\lambda \geq 2\sigma$, it is easy to see, using $Prop_1$, that all the states are non-Zeno (i.e : whatever the system does at the state, it does not prevent time to diverge).

4 Verification of the probabilistic model

4.1 Modifications of the NSY model for the probabilistic verification

Before proving the probabilistic version of CSMA/CD we adopt a specification closer to the standard IEEE [13] : The random drawing of the delay time (*alea*) is only invoked in case of collision ; when the Channel is sensed *busy*, the Sender wait for a fixed delay ($\sigma/2$) in a new state (*Retrybis*) before trying to reemit (see figure 3).

Recall that, after a collision (location $Retry_i$), the maximum value of of the waiting time $alea_i$ before a new attempt is equal to $c = b\sigma$.

In the following, we assume that $alea_i$ is a *multiple* of σ. In other terms, $alea_i$ is uniformly selected at random on the set $\{0, \sigma, 2\sigma, \cdots, b\sigma\}$ (rather than the whole integer interval $\{0, 1, 2, \cdots, b\sigma\}$).

The new Sender automata is represented as follows :

The synchronized product of these automata is itself a probabilistic timed automaton, exhibiting both nondeterministic and probabilistic choices. We are intersted in minimum probability of reachability that requires to find the scheduling (i.e : ordering of the nondeterministic choices) the more unfavorable as possible.

4.2 Probabilistic properties

We are interested in the verification of the following minimum probabilistic reachability properties :
The minimum probability that, after a collision, both senders send correctly their messages within a deadline d i.e. (within LTL logic) :

[2]This value of k corresponds to the typical ratio $\lambda/\sigma = 780/26$ for a 10-Mbps Ethernet, according to the IEEE 802.3 international standard.

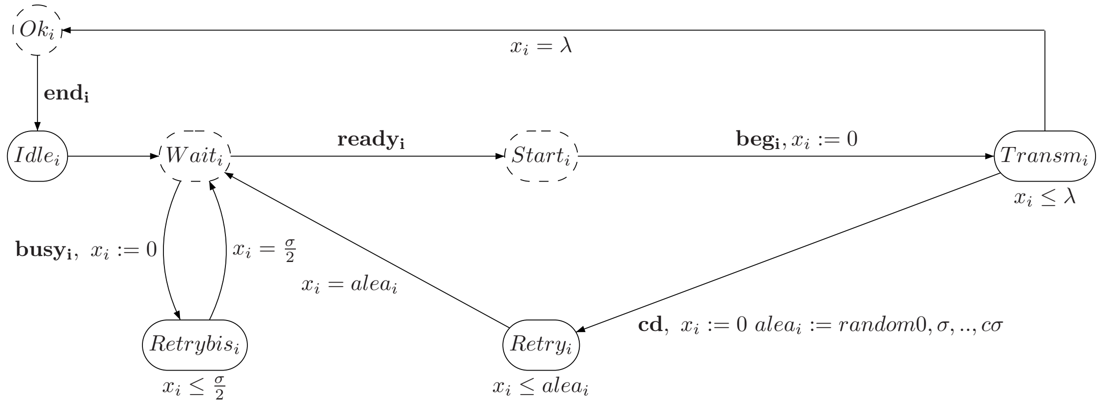


Fig. 3 – New version of the automaton of the i-th sender.

Starting from $Init' = Retry_1.Retry_2.Idlebus$ (A collision has just occurred) :
$P_{min}[trueU\{F\ Ok_1 \wedge F\ Ok_2 \wedge t \leq d\}]$
In order to check this property in the model checker PRISM [8]. we introduce a discrete variable f_i $(i = 1, 2)$ that is set up to one when $Sender_i$ has correctly sent its message. The property writes (within PTCTL logic [12]) :
$P_{min}[trueU\{f_1 = 1 \wedge f_2 = 1 \wedge t \leq d\}]$.

In this case, it is clear that the most unfavorable scheduling corresponds to the behaviour where each *Sender* always wants to send a message. More precisely, it waits a null time in *Idle* before trying to emit. In this situation, there is no longer any source of nondeterminism in the system, which can now be considered as a Markov chain ($P_{min} = P$).

4.3 Forward Reachability using HyTech

Following ([9, 11, 3]), we check the property using (a simplification of) the forward exploration method. The analysis is simplified by the fact that there is no nondeterminism (see observation above). Using HyTech, and the methodology described in [9, 11], we are able to construct the reachability graph (see Figure 4). The state *GOOD* is an additional (absorbing) state where the system goes when $f_1 = f_2 = 1$. More precisely, we compute the relevant symbolic states, together with information concerning transitions made between such symbolic states. Given a number of auxiliary variables HyTech is able to give us the reachability graph as a Markov Chain. Note that since there is no nondeterminism, the probability computation is exact.
An original feature of the graph is that σ and λ are parameters. The graph makes appear, that the probability computation depends only on the ratio (λ/σ).

Comparison with PRISM case study on CSMA/CD

In [16], a similar experiment has been conducted on a NSY version CSMA/CD protocol by PRISM team. This case study is also use to test the feasability or the symbolic state model checking algorithm introduced in [10]. A major difference between the two approaches, is that, in their model a sender can not reemit a message after a good transmission. On the other hand, they consider all the possible schedulings ; furthermore, they benefit from a prototype implementation while our construction of the reachability graph is partially manual.

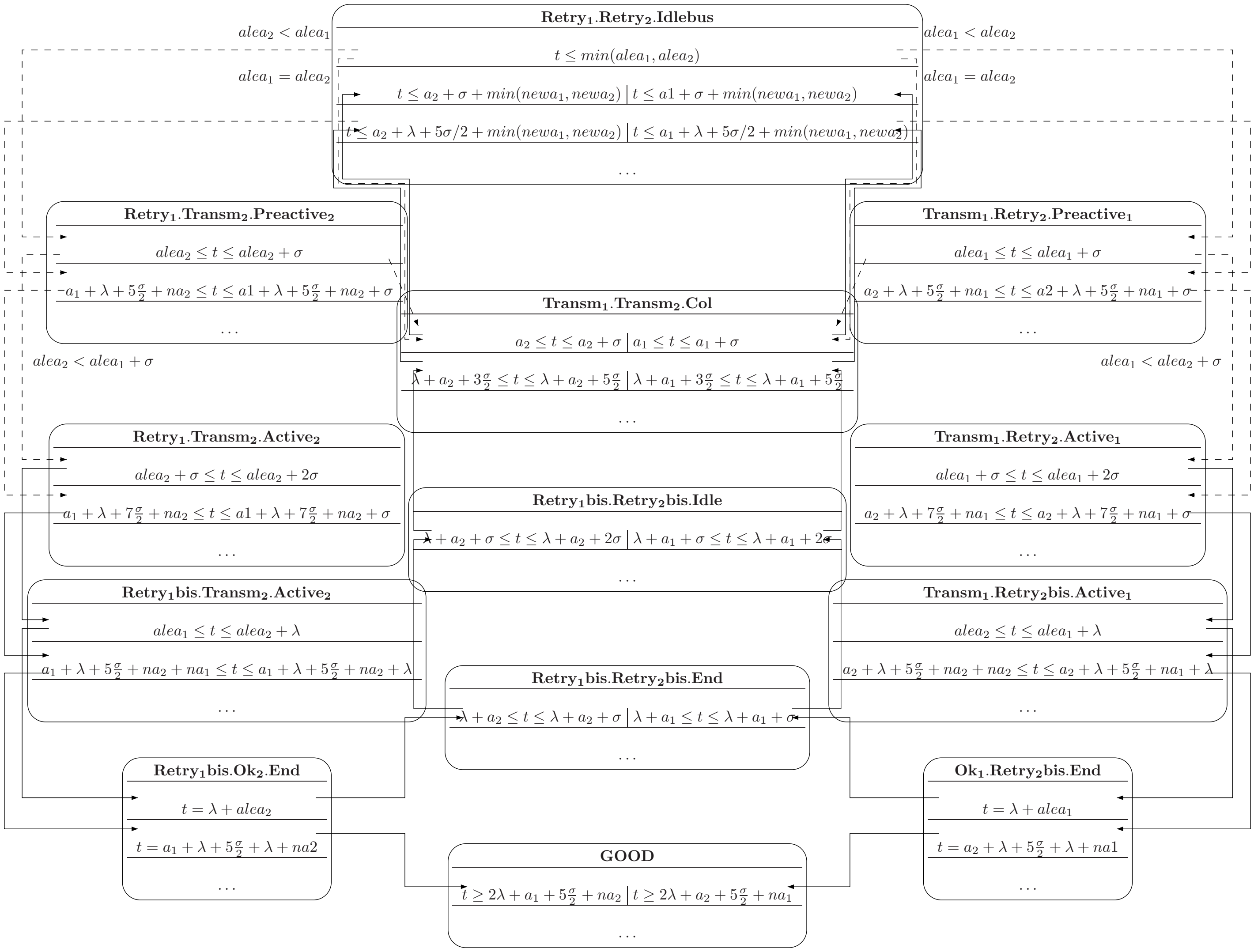
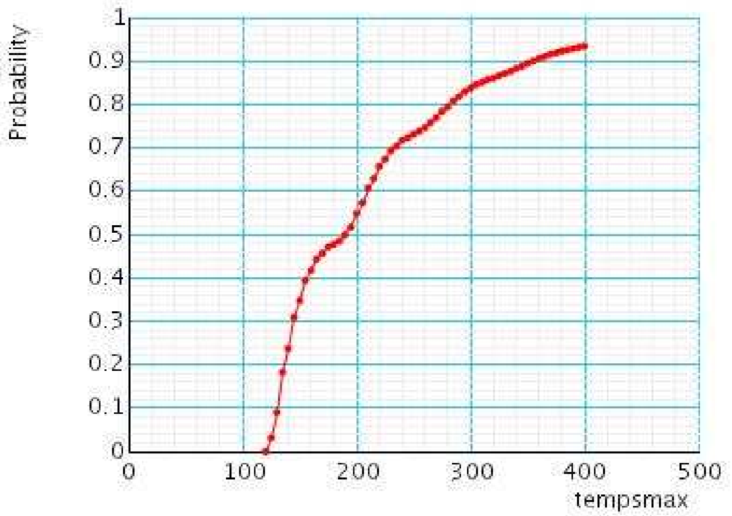


FIG. 4 – Reachability graph.



($\lambda/\sigma = 30$, $alea_{max} = b = 3$)
FIG. 5 – Probability of emission as function of the deadline.

4.4 Results with PRISM

As in [9, 11], we then perform probabilitic model checking on the resulting symbolic state probabilistic syste musing the tool PRISM, and check the following property (in PCTL logic [2]) :

$$P_{=?}[true\ U\ GOOD]$$

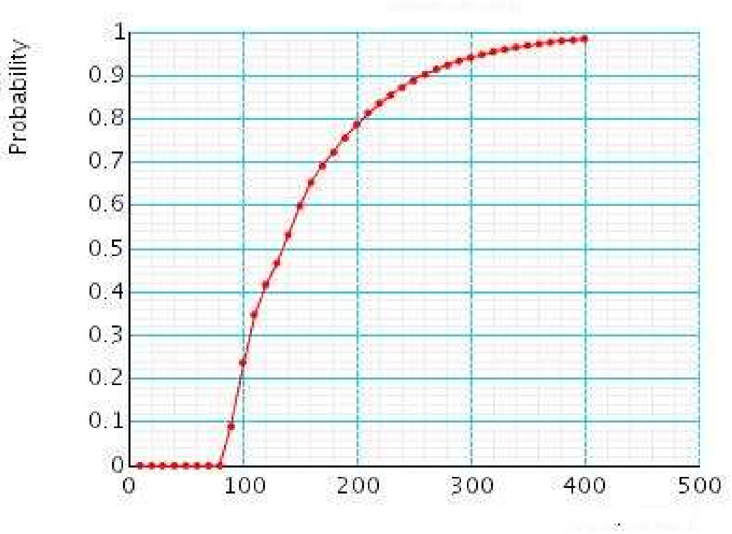
.
Here the graph representing the evolution of this probability within the deadline for two differents ratios $\lambda/\sigma = 30$ and $\lambda/\sigma = 20$.

Figures 5 and 6 analyse the probability that two messages have been successfully transmitted, as function of the time after a preliminary collision. We can see a gap from the zero probability to a non-zero probability of emission. After this gap, the probability of emission increases quickly up to an asymptotic behavior, close to 1. This meets a goal of the protocol, being that the probability a message is transmitted after a short delay is high . This courbes are similar with the corresponding one observed in [5] (see Figures 4 and 7).

5 Final Remarks

To our knowledge, it is the first time that a parametric proof of CSMA/CD protocol has been done, both in the nonprobabilistic and the probabilistic versions.
Note that there is much room for improving our probabilistic version :
First,as in [16] we would like to mechanize the production of the reachability graph, which still relies on a manual interpretation of HyTech results; second, we would like to be able to generate a reachability graph for all scheduling, not only for a fixed one as done here. This is the subject of our on-going work.

Aknowledgment : We are grateful to Jeremy Sproston for stimulating discussion.



($\lambda/\sigma = 20$, $alea_{max} = b = 3$)
FIG. 6 – Probability of emission as function of the deadline.

Références

[1] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126 :2, pp. 183-235, 1994.

[2] A. Bianco and L. de Alfaro. Model Checking of Probabilistic and Nondeterministic Systems. In *Proc. 15th Conference on Foundations of Software Technology and Theoretical Computer Science*, pp 499-513, LNCS :1026, 1995.

[3] C. Daws, M. Kwiatkowska and G. Norman . Automatic Verification of the IEEE-1394 Root Contention Protocol with KRONOS and PRISM *ENTCS vol.66 No 2*, 2002.

[4] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. In *Hybrid Systems III : Verification and Control*, pp 208-219, LNCS :1066, 1996.

[5] M. Duflot, L. Fribourg, T. Herault, R. Lassaigne, F. Magniette, S. Messika, S. Peyronnet, and C. Picaronny. Probabilistic model checking of the CSMA/CD protocol using PRISM and APMC. Research Report LSV-04-9, Lab. Specification and Verification, ENS de Cachan, Cachan, France, April 2004.

[6] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. *I*nformation and Computation, 111(2) :193-244, June 1994.

[7] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech : A Model Checker for Hybrid Systems. *S*oftware Tools for Technology Transfer 1 :110-122, 1997.

[8] M. Kwiatkowska, G. Norman and D. Parker. PRISM : Probabilistic Symbolic Model Checker. In *Proc. 12th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation (TOOLS'02)*, pp 200-204, LNCS :2324, April 2002.

[9] M. Kwiatkowska, G. Norman, and J. Sproston. Probabilistic Model Checking of the IEEE 802.11 Wireless Local Area Network Protocol. Proc. of the 2nd Int. Worshop PAPM-PROBMIV 2002, LNCS no 2399, pp. 169-187, July 2002.

[10] M. Kwiatkowska, G. Norman, and J. Sproston. Symbolic Model Checking for Probabilistic Timed Automata. Technical report CSR-03-10, U.Birmingham, Oct. 2003.

[11] M. Kwiatkowska, G. Norman, and J. Sproston. Probabilistic Model Checking of Deadline Properties in the IEEE 1394 FireWire Root Contention Protocol. *Formal Aspects of Computing* 14 :3, pp. 295-318, 2003.

[12] M. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theoretical Computer Science*, vol. 282, pp. 101–150, 2002.

[13] IEEE Web Page `http ://www.ieee802.org/3`

[14] X. Nicollin, J. Sifakis and S. Yovine. Compiling Real-Time Specifications into Extended Automata. IEEE Transactions on Software Engineering, vol.18, no 9, p. 794-804, Sept. 1992.

[15] A. Pnueli The temporal logic of programs *P*roc. 18th FOCS, Providence, RI, (1977), 46–57.

[16] PRISM Case Studies Web Page `http ://www.cs.bham.ac.uk/ dxp/prism/casestudies/csma.html`

[17] S. Yovine. Kronos : A verification tool for real-time systems. International Journal of Software Tools for Technology Transfer, Vol. 1, Issue 1/2, pages 123-133, oct. 1997, Springer-Verlag.

Appendix A : Code of CSMA/CD in Hytech

```
   - ====================================================================
- ====================================================================
- HyTech source file
- by Laurent Fribourg and Stephane Messika and Claudine Picaronny (LSV)
-
- CSMA/CD protocol
- ====================================================================
- ====================================================================

   var
x1,x2,y,z,t : clock ;
alea1,alea2 : discrete ;
lambda, sigma, c : parameter ;

   - -----------------------------------
automaton Sender1
synclabs : beg1,cd,end1,ready1,busy1 ;

   initially Idle1 ;

   loc Idle1 :
while x1>=0 & x2>=0 & y>=0 & t>=0 wait {}
when True do {z'=0} goto Wait1 ;

   loc Wait1 :
while x1>=0 & x2>=0 & y>=0 & t>=0 & z=0 wait {}
when True sync ready1 do {z'=0} goto Start1 ;
when True sync busy1 do {x1'=0, alea1'>=0} goto Retry1 ;

   loc Start1 :
while x1>=0 & x2>=0 & y>=0 & t>=0 & z=0 wait
when True sync beg1 do {x1'=0} goto Transm1 ;

   loc Retry1 :
while x1>=0 & x2>=0 & y>=0 & t>=0 & x1 <= alea1 & alea1 <= c wait {}
when x1=alea1 do {z'=0} goto Wait1 ;

   loc Transm1 :
while x1>=0 & x2>=0 & y>=0 & t>=0 & x1 <= lambda wait {}
when True sync cd do {x1'=0, alea1'>=0} goto Retry1 ;
when x1=lambda do {z'=0} goto Ok1 ;

   loc Ok1 :
while x1>=0 & x2>=0 & y>=0 & t>=0 & z=0 wait {}
when True sync end1 goto Idle1 ;

   end - Sender1

   - -----------------------------------
automaton Sender2
synclabs : beg2,cd,end2,ready2,busy2 ;

   initially Idle2 ;

   loc Idle2 :
while x1>=0 & x2>=0 & y>=0 & t>=0 wait {}
```

```
when True do {z'=0} goto Wait2 ;

   loc Wait2 :
while x1>=0 & x2>=0 & y>=0 & t>=0 & z=0 wait {}
when True sync ready2 do {z'=0} goto Start2 ;
when True sync busy2 do {x2'=0, alea2'>=0} goto Retry2 ;

   loc Start2 :
while x1>=0 & x2>=0 & y>=0 & t>=0 & z=0 wait
when True sync beg2 do {x2'=0} goto Transm2 ;

   loc Retry2 :
while x1>=0 & x2>=0 & y>=0 & t>=0 & x2 <= alea2 & alea2 <= c wait {}
when x2=alea2 do {z'=0} goto Wait2 ;

   loc Transm2 :
while x1>=0 & x2>=0 & y>=0 & t>=0 & x2 <= lambda wait {}
when True sync cd do {x2'=0, alea2'>=0} goto Retry2 ;
when x2=lambda do {z'=0} goto Ok2 ;

   loc Ok2 :
while x1>=0 & x2>=0 & y>=0 & t>=0 & z=0 wait {}
when True sync end2 goto Idle2 ;

   end - Sender2

   ------------------------------------
automaton Chan
synclabs : beg1,beg2,cd,end1,end2,ready1,ready2,busy1,busy2 ;

   initially Idlebus ;

   loc Idlebus :
while x1>=0 & x2>=0 & y>=0 & t>=0 wait {}
when True sync ready1 goto Idlebus ;
when True sync ready2 goto Idlebus ;
when True sync beg1 do {y'=0} goto Preactive1 ;
when True sync beg2 do {y'=0} goto Preactive2 ;

   loc Preactive1 :
while x1>=0 & x2>=0 & y>=0 & t>=0 & y<= sigma wait {}
when y<sigma sync ready2 goto Preactive1 ;
when y=sigma goto Active1 ;
when y<sigma sync beg2 do {y'=0} goto Col ;

   loc Preactive2 :
while x1>=0 & x2>=0 & y>=0 & t>=0 & y<= sigma wait {}
when y<sigma sync ready1 goto Preactive2 ;
when y=sigma goto Active2 ;
when y<sigma sync beg1 do {y'=0} goto Col ;

   loc Col :
while x1>=0 & x2>=0 & y>=0 & t>=0 & y<= sigma wait {}
when y = sigma sync cd goto Idlebus ;

   loc Active1 :
while x1>=0 & x2>=0 & y>=0 & t>=0 wait {}
when True sync busy2 goto Active1 ;
```

```
when True sync end1 do {y'=0} goto End ;

   loc Active2 :
while x1>=0 & x2>=0 & y>=0 & t>=0 wait {}
when y<sigma sync busy1 goto End ;
when y<sigma sync busy2 goto End ;
when True sync busy1 goto Active2 ;
when True sync end2 do {y'=0} goto End ;

   loc End :
while x1>=0 & x2>=0 & y>=0 & t>=0 & y<= sigma wait {}
when y = sigma goto Idlebus ;

   end - Chan

   - ----------------------------------
- analysis commands

   var init, predit, postit, final : region ;

   init :=
loc[Sender1]=Idle1 & loc[Sender2]=Idle2 & loc[Chan]=Idlebus
& x1=0 & x2=0 & y=0 & t=0
& lambda >= 2*sigma & c>=sigma & sigma > 0 ;

   final :=
loc[Sender1]=Ok1& lambda>= 2*sigma & c>=sigma & sigma>0
& lambda <= 30*sigma ;

   prints "iterated succ" ;
postit := (reach forward from init endreach) & loc[Sender1]=Wait1 ;
print hide t,z in postit endhide ;

   prints "iterated pred" ;
predit := (reach backward from final endreach) & loc[Sender1]=Wait1 ;
print hide t,z in predit endhide ;

   prints "INTERSECTION EMPTY ?" ;
if empty(postit & lambda<= 30*sigma & ¬(predit))
then prints "OK" ;
else
print hide t,z in (postit & ¬(predit)) endhide ;
endif ;
```