DEPENDABILITY ANALYSIS OF NETWORKED AUTOMATION SYSTEMS BY PROBABILISTIC DELAY TIME ANALYSIS

Jürgen Greifeneder and Georg Frey

University of Kaiserslautern, FB EIT, JPA²
Erwin-Schrödinger-Str. 12, D-67653 Kaiserslautern, Germany
{greifeneder | frey}@eit.uni-kl.de

Abstract: The dependability of an automation system is a quality that depends on all the systems components. The introduction of non-deterministic networks to the automation field leads to new questions in the analysis of the resulting systems. For the analysis of systems containing non-deterministic components, probabilistic model checking (PMC) is a promising formal approach. This paper investigates the application of existing techniques and tools from PMC to the analysis of delay times in networked automation systems under the consideration of component failures. A case study shows how properties, relevant for the reliable functioning of an automation system, can be checked.

Keywords: Markov decision process, manufacturing, control, reliability analysis, networks, model checking.

1. INTRODUCTION

Dependability is defined by IFIP WG-10.4 as "the trustworthiness of a computing system which allows reliance to be justifiably placed on the service it delivers" [http://www.dependability.org]. Dependability is often regarded as a set of properties such as reliability, availability, safety, fault tolerance, robustness, and security. For the area of automation systems to assess dependability a system of hardware and software (of possibly several layers) has to be taken into account. With the advent of networked systems, this problem becomes even more complicated, since now the influence of the network and its components on the dependability of the overall system needs attention. The network not only adds (possibly non-deterministic) delays but also components that may fail (connections, switches, routers …).

Modeling and analysis of automation systems therefore requires not only detailed knowledge about their functional aspects, but also about their real time behavior and the possible component failures. Control algorithms need to communicate with the process hardware (i.e., sensors and actuators often abbreviated I/O for inputs and outputs) within bounded time intervals as the control algorithm will fail, otherwise. For a dependable automation system a specification of properties like *"A reaction to a change in a sensor value will be issued within 200 ms."* might thus arise.

In classical structures using a single controller and directly connected I/O the answer to questions like this depends mainly on the cycle time of the controller. Considering distributed systems with controllers communicating with the I/O over networks the problem is much harder. Here the network delay (typically not given by a constant value but by a distribution) has to be taken into account. To avoid worst-case analysis that often leads to infeasible demands on the control systems hardware the properties could be relaxed by introducing probabilistic bounds. This leads to properties like: *"With a probability of at least 99.9% a reaction to a change in a sensor value will be issued within 200ms."*

To check properties like this on a complex system, simulation is infeasible since a probabilistic solution will need a very long simulation time. The problem becomes even worse if the analysis is extended from a simple performance check to a detailed reliability analysis where the possibilities of failures in the components are considered during the analysis. In this case the system under consideration contains

Greifeneder, J.; Frey, G.: *Dependability analysis of networked automation systems by probabilistic delay time analysis.* Proceedings of the 12th IFAC Symposium on Information Control Problems in Manufacturing (incom 2006), St. Etienne, France, Vol. 1, pp. 269-274, May 2006.

very short cycle times of a controller together with very long mean times between failures (MTBFs) of the components.

The formal description of systems like this leads to models containing time, stochastic distributions and probabilistic choice. A formal technique providing the means for the description and analysis of systems and properties like the ones described above is Probabilistic Model Checking (PMC). PMC uses an extension of CTL (PCTL – Probabilistic Computation Tree Logic, (Hansson and Jonsson, 1999) to specify properties over systems described by Markov models. Model checking algorithms and tools are available (Kwiatkowska et al., 2002; PRISM-Website; Kwiatkowska et al., 2004). In the presented work PMC is applied to delay time analysis in Networked Automation Systems under consideration of component failures. A key idea is that the nature of signals in automation systems is taken into account (Greifeneder and Frey, 2005). The properties to be checked are not directly related to system failures but to the loss or delay of process relevant information.

The paper is structured as follows. The next Section explains the PMC approach and the used models in some detail. The approach is illustrated at a case study in Section 3. Section 4 concludes the paper and gives an outlook on further work.

2. PMC APPROACH

2.1. General approach

In model checking (Bérard et al., 2001), a model of the system is build using some formal description. In addition, the properties to be checked are formalized using some kind of formal logic. These two descriptions are input to a model checking algorithm that checks whether the properties hold on the system (cf. Fig. 1). System and properties are handled separately. A change in the system affects only the system model, a change in the properties only the formal properties.

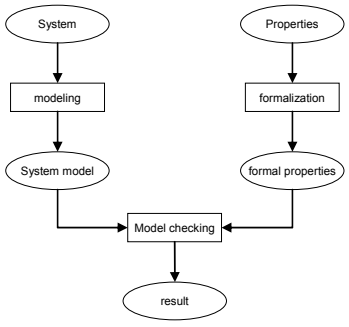

Fig. 1. Workflow of model checking.

The main advantage of model checking is the complete coverage of all possible evolutions of a system (instead of only a subset in simulation and testing). The main drawback is that the state space to be covered tends to increase very fast with the complexity of the model (state space explosion problem). This can be avoided to some extend by the application of proven modeling rules. That is why a strictly hierarchical programming and the use of a broad amount of synchronization techniques are recommended for the description of distributed systems.

In PMC, the nonrecurring termination condition must be considered: In a cycling problem, the probability of an event taking place repeatedly over time can be expressed mathematical by a geometrical progression which indeed leads to a probability being one in an infinite time. That means, the probability of any event in a cycling problem will be one or zero – true or false. Therefore, the first condition to a probabilistic model must be that it will itself check on the occurrence of the event supervised and terminate afterwards. Furthermore, each possible arrangement must be depicted in that very first cycle as otherwise only part of the system's behavior is considered. This changes the design process fundamentally: the separation between model and properties must be given up resulting in a new formal design process (cf. Fig. 2). While the formalization process of the properties stays the same as in Fig. 1 (shown in the middle of Fig. 2), the construction of the model is changed. The first possible way to do so is shown in dashed lines on the left side of Fig. 2: The generic system model gets convolved with the specific formal properties, which amounts to a reduction of the generic model and the inclusion of termination conditions. The second possible way, given in dotted lines on the right hand side of Fig. 2, is to include the properties in the modeling task already. This leads to much finer models, but requires the engineer to rebuild his models for each possible case. In this work, the second (dotted) approach is used, as there must be a deep understanding of what is an optimal model before a suitable reduction algorithm could be proposed.

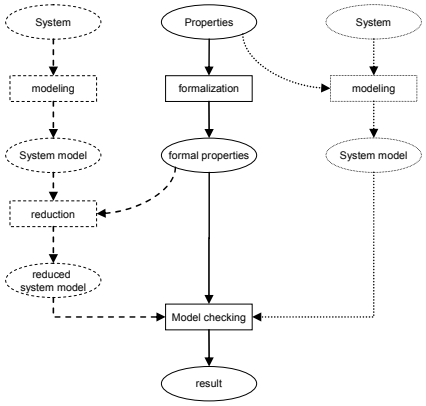

Fig. 2. Workflow of probabilistic model checking.

2.2. System model

The system model is built using finite automata with extensions for timed and probabilistic behavior. In this subsection the model is introduced over several steps. A finite automaton as shown in Fig. 3 consists of states linked by conditional transitions.

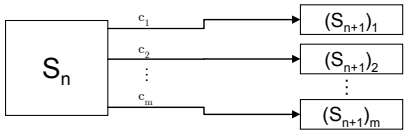

Fig. 3. Finite automaton.

Greifeneder, J.; Frey, G.: *Dependability analysis of networked automation systems by probabilistic delay time analysis.* Proceedings of the 12th IFAC Symposium on Information Control Problems in Manufacturing (incom 2006), St. Etienne, France, Vol. 1, pp. 269-274, May 2006.

For a state S_n with m possible post-states connected via transitions t_1 to t_m with transition conditions c_1 to c_m respectivey the following must hold:

1. There are no two conditions active at the same time:

$$\bigwedge_{i,j\in[1..m];i\neq j} c_i \wedge c_j = false \quad (1)$$

2. The disjunction of all conditions must cover the total state space:

$$\bigvee_{\forall i\in[1..m]} c_i = true \quad (2)$$

Together this means that there is always one and only one active transition in a module.

For reasons of time scaling, a transition should only become active, when a well defined time has passed. That can be done, by using timed automata (Fig. 4a).

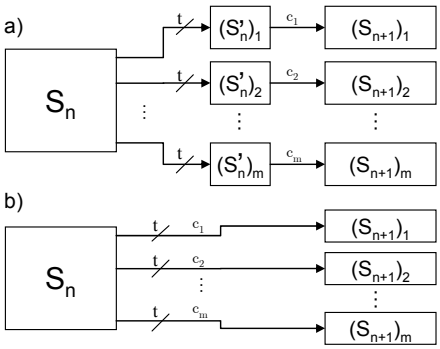

Fig. 4. Timed Automaton.

When the time t has passed, all transitions labeled with the sync-signal *t* will be activated immediately. In this work two restrictions are made: First, there is only one clock (i.e. time) to be synchronized on (instead of several different sync-times possible in a general timed automaton) and second the condition of the following transition will be evaluated in the same moment as the sync-transition becomes activated. Using these assumptions, it makes sense, to simply omit the intermediate states S'$_n$ and use the graphical representation shown in Fig. 4b.

The next step of extension leads directly to the probabilistic timed automata (cf. Fig. 5a). In this probabilistic case, the transitions are assigned a probability $p_i \in [0..1]$, with:

$$\sum_{\forall i\in[1..k]} p_i = 1 \quad (3)$$

In other words: the deterministic automaton becomes enlarged by a non-deterministic choice weighted by probabilities p_i. As the intermediate states S"$_n$ are of transient nature, the graphical representation shown in Fig. 5b is used. To further simplify the representation probabilities equal to one, and conditions, which are true for all times are not written. Note: The PTA definition above is compliant to the one given in (Alur et al., 1999) for the case of dense time steps. However, it differs from definitions used by e.g. (Bérard et al., 2001) as here the transitions are time triggered. The presented approach differs from others constructing a finite-state quotient representation by the fact that discretisation is already done in the modeling step while others (e.g. Henzinger et al., 1992) use dense time in the model formulation and discretize afterwards.

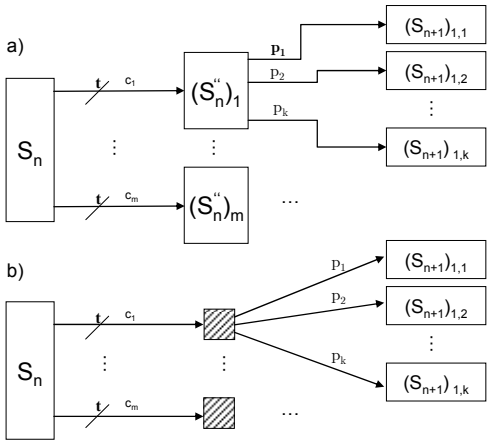

Fig. 5. Probabilistic Timed Automaton.

2.3. Property formulation

Dealing with failures in networked automation systems, the probability that information is lost at all must be considered. That means: determining the probability of information not arriving within a given time frame. In an automated control system, there are three kinds of signals and related questions to discuss (Greifeneder and Frey, 2005):

Type 1: A signal that after it occurs will not be reset until a corresponding action takes place. Question: When will the signal be discovered?

Type 2: A signal that is only present for a limited time. Question: Will the signal be discovered?

Type 3: The combination of 1 and 2: A signal that changes its value with a probability p_{ch} and keeps its value until the next change (at least for τ_{min}). Questions: (a) Will a signals change be discovered before another change occurs and (b) if yes, then when?

2.4. Model checking and formulation in PRISM

As mentioned in Section 1 PCTL is used to specify properties in PRISM. Typically these properties are composed of atomic propositions or predicates over the variables in the model. Normally, PCTL formulas evaluate to a Boolean value. However, it is often useful to know the actual probability rather than just check that it is above or below a given bound. PRISM allows properties of the following form:

P=? [*expr1* U *expr2*]

P=? represents the probability to be determined, *expr1* and *expr2* are propositional formulations which evaluate to Boolean expressions. This command has to be read as follows: Determine the probability, that *expr1* is true (at least as long) until *expr2* becomes true, and *expr2* becomes true. Note: There is no need for *expr2* to stay true, it might become false and true again, which can not be determined directly using this operator but e.g. by nesting probabilistic operators. The easiest way to use this operator is

to replace *expr1* by true and only work on the second predicate, *expr2*:

P=? [true U expr2]

Starting from a given system model and a desired set of properties, the PRISM code must be generated. Doing so, it is important, that the algorithm terminates as soon as possible. There are two principle cases to distinguish:

a) If the interest is to check, how long anything lasts – e.g. the delay time – it is necessary to wait until this process is finished, or (for practical purposes) until a maximum time bound has been reached. This can be done using the following operator:

P=? [true U Runtime=Lf]

Where *Lf* is a parameter – PRISM has to check for each value of *Lf* in a given range – and *Runtime* is a variable which gets assigned by the model. When the desired property occurred, the value of the runtime counter will be assigned to that variable. It is possible to check on the counter variable directly, however, in that case the result would be the integral of the distribution function from *Lf* to *infinity*.

b) If the interest is to check the probability whether the desired property becomes true at all, the calculation should not be terminated by the incidence of the property itself. This task can be achieved only by checking for a significant time period, namely the one within the event can take place once and will not take place for a second time, as the probability check can only determine whether something occurred or not. As elucidated above, it is possible, to test on the 2^{nd}, 3^{rd} … occurrence or implement a counter inside the model; but if testing on the occurrence of an event at all, there is no distinguishing whether it occurred once ore several times. To solve this problem it is important to cover all possible initial states in the first cycle (cf. subsection 2.5) and terminate the automaton after finishing this one cycle.

The PRISM code of the system model itself is build as follows:

[Pr] conditions → assignments;

[Pr] conditions → p1:assignments + p2:assignments + … ;

[Pr] is the synchronization signal mentioned before. If it is omitted, the determinism of the sequence is destroyed. Conditions are a predicate formulation composed of one or more transition conditions c_i (or their negated partners) coupled by binary operators. Assignments are value assignments to one or more variables, which can be functions of the variables values valid just before the transition got active. Finally, p_1, p_2 … are the probabilities used in the PTA.

2.5. Initial State

One of the most important differences between simulation and model checking is that model checking will for sure reach all possible states, while in simulation even after a long period of time this might not be the case. In probabilistic model checking the properties are weighted by a value indicating the probability of their incidence. While simulation might converge to the right numbers for really long simulation periods this should be avoided in PMC (for reasons of state space on the one and the attitude of geometric progressions on the other hand). Therefore, it is important not to distort the results by traversing some paths more often than others, i.e. each possible path must be depicted once and only once. It follows, that the initial states must be chosen to definitely cover all possible paths.

The right assignment of the initial states is difficult, especially as they might occur with different probabilities. If there are at least two processes (modules), which do not cycle using the same cycle time, there will be a large number of possible drift times, especially, if they are not started in parallel (synchronized). In the best case, there are only two drifting modules and all possible drift times will occur in with the same probability. In this case, the initial condition can be found easily by pre-adapting an "equally randomized initial state" cycle just in front of the first time step in one of the two modules. This is shown in Fig. 6. In PRISM this can be coded as follows:

[P] !pre&var<varMax → 1/(varMax-var):(pre=true) + (var-varMax-1)/(var-varMax):(var'=var+1);

[P] pre&(!pre2|!pre3|…|!pren) → true;

In this code, [P] is the sync-operator, *pre* is the binary variable, to detect whether the initial process has been finished or not. !*pre* is the negation of *pre*. *pre2*… *pren* are the detection variables of synchronized modules, as the automaton would deadlock, without this command (cf. equation (3)).

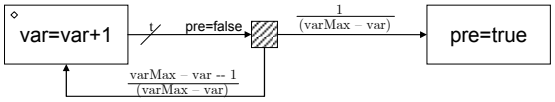

Fig. 6. Equally randomized initial state.

Note: This could be done in a single step also. However the code has to be rewritten then for every value of *varMax* while using the automaton above *varMax* can be used as free parameter.

3. CASE STUDY

To illustrate the approach a case study is presented. The structure of the considered system is given in Fig. 7. The model comprises a PLC, which represents the controller and has a cycle time of 65 time steps (ts). A cycle is made of the following steps: read in all inputs, execute the PLC-code and write out the outputs. While the outputs – in this example – are written out directly, the inputs are read from a TCP/IP based IO-card. This IO-card checks the digital In- and Output module every 40 time steps. This inquiry is passed over the network, represented by the Switch (Sw). From there it is passed to the IO-Module (IO), which reads the current data from the sensor and passes the information back through the network. The property to be investigated is the delay between an input signal change and a corresponding output (signal of Type 1 according to Section 2.3).

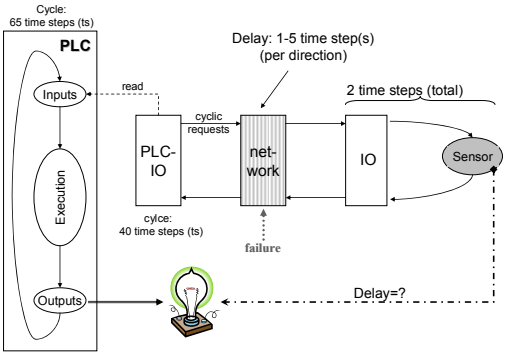

Fig. 7. Setup of the case study.

Each packet needs some time to pass through the network. In this case study a uniformly distributed time is assumed, ranging from one to *SwMax* time steps. To read a sensor value two time steps are required. While this seems to be a deterministic problem, its character is of probabilistic nature as the two cycles (PLC and PLC-IO) are not synchronized. The resulting displacement is assumed to be uniformly distributed and therefore initialized using the algorithm introduced in Fig. 6.

3.1. Detailed models

Fig. 8 shows the automaton of the sensor module in more detail. In the first time step, the value of the sensor changes from 0 to 1. Beginning from this, the delay-timer runs, what means, that the variable *Scounter* becomes incremented, until either the desired output got activated (END=true, cf. Fig. 10) or the maximal variables value (*MaxScount*) is reached. For reasons of readability the END, !END, MSt and !MSt conditions as well as the corresponding termination state are not drawn in the automata shown in Fig. 9 to Fig. 12.

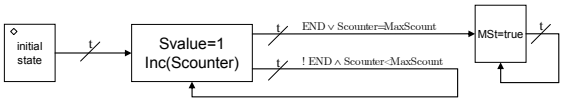

Fig. 8. Automaton for the sensor.

The automaton for the sensor-IO is shown in Fig. 9. The automaton stays in the initial state until a request from the network is received. In this case (and if *Svalue=1*) the two time steps of "sensor reading" are triggered. Then the value *IOa* is set to one for one time step and the automaton returns to its first state.

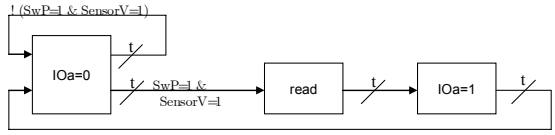

Fig. 9. Automaton for the sensor-IO.

The PLC model consists of two modules: The PLCtimer and the PLC-action scheme. Note: In more complicated examples there is a third module called successor needed to handle a memory function. The PLC timer – as well as the PLC-IO – contains an equal distribution part (as discussed in Fig. 6) and a ring counter. The ring counter is used to produce a sync-signal all *CountMax* time steps. The PLC-action-timer scheme is shown in Fig. 10, where the ring counter is included (*Pct*). At Pct=0 (the end of the cycle) it writes out its outputs: if the signal got read, the automaton transits to its termination state (top mid of the figure), otherwise, the cycle is started again. At *Pct*=1 (the start of the cycle) the inputs are read. If *SwP=1* (see network, Fig. 12), the variable *Success* becomes 1, otherwise it remains 0. In each case, *Pct* becomes incremented. Now, the automaton counts up until *Pct = MPct-1* and goes back to the first state (*Pct=0*).

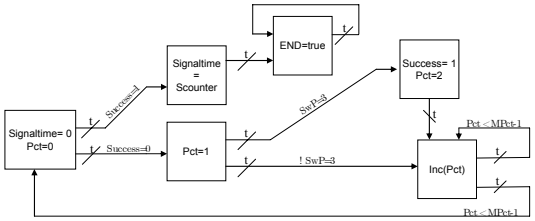

Fig. 10. Automaton for the PLC action-timer.

Finally, there is an automaton for the network. It consists of two modules: the network failure (Fig. 11) and the network transport (Fig. 12). In this example the assumption is made, that the packages from up and down-link will never arrive at the same time – what indeed is not possible, as the IO needs two time steps to answer a request which is sent every 40 time steps only and it is assumed that there is no other traffic on the network.

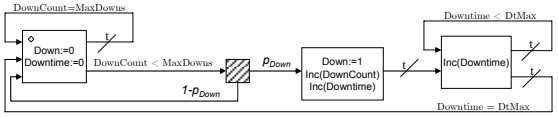

Fig. 11. Automaton for the network failure.

In the case of a network failure (*Down=1*) the network loses all inside information (state at the bottom left of Fig. 12). It stays idle (top left) until another failure occurs or a package arrives (*count=0* → PLC-IO sends package, *IOa=1* → IO sends package).

The two states in the middle of Fig. 12 represent the case of a package being in the network (the network needs 1 to *SwMax* time steps to carry a package). The probability p_{NW} is given as:

$$p_{NW} = \frac{SwMax - NT - 1}{SwMax - NT} \qquad (4)$$

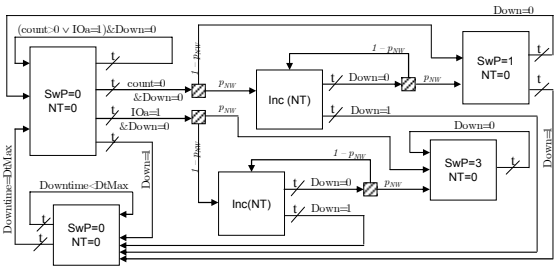

Fig. 12. Automaton for the network.

After a package got delivered to the I/O (*SwP=1*), the network returns to its idle mode. If it got delivered to the PLC-IO (*SwP=3*), the network ends up in a self loop, as the terminal condition will be reached within some time steps for sure.

Greifeneder, J.; Frey, G.: *Dependability analysis of networked automation systems by probabilistic delay time analysis.* Proceedings of the 12th IFAC Symposium on Information Control Problems in Manufacturing (incom 2006), St. Etienne, France, Vol. 1, pp. 269-274, May 2006.

3.2. Results

The results of the experiment (done using PRISM) and the measurements at a real plant are compared in Fig. 13. These measurements were done by the LURPA team at the ENS, Cachan (Poulard, 2003). The bars show the measured values (relative frequency), the black line the probabilistic model checking results (probabilities). The comparison justifies the use of the assumption of an initially uniformly distributed state space as well as the model itself.

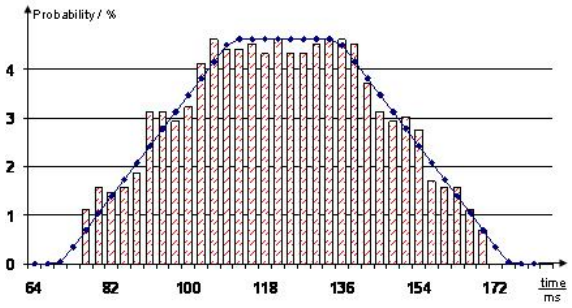

Fig. 13. Time between occurrence and reaction.

In the second experiment, *SwMax* is set to 5 and therefore a packet can take 1, 2, 3, 4 or 5 ts to pass the network. This is shown in Fig. 14. The dashed lines show the case of *SwMax=1*, the solid lines *SwMax=5*. The delay increases as expected. Note: The sampling rate used in Fig. 14 was 1 ts, while in Fig. 13 an interval of 3 ts was used and therefore the values got 3 times larger than in Fig. 14. The probability *p=1,538%* at time 125 means that the probability of a signal to be delayed by 125 time steps is *1,538%.* By building the sum, one can state that the probability of a signal to be delayed more than 160 time steps is *3.00%* in the case of *SwMax=1* and *5.54%* in the case of *SwMax=5*.

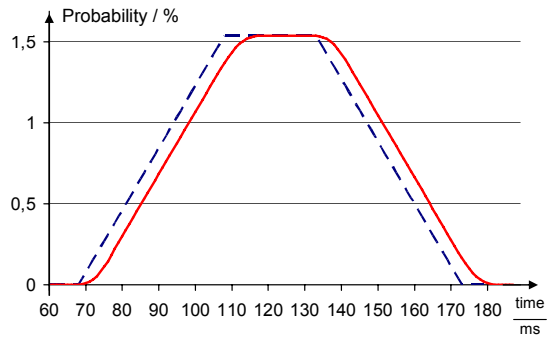

Fig. 14. Delay time without network failure.

In the third experiment, the network is assumed to fail. For this an expected MTBF of 1000 time steps is assumed. After a failure, the network needs 51 time steps to recover. The controller holds a measured value unchanged unless it receives another value. Therefore, a failure in the network or the sensor has no consequences if the measured value does not change during the respective downtime. Fig. 15 shows the consequence (logarithmic scale!): dotted lines without network failures, solid lines including network failures. In the first 165 steps, there is not much of a difference: The values do vary about 0.37‰. However, there are several delay times which are higher than the maximum of 172 time steps observed before. Note: For reasons of calculation time the maximum number of network failures was set to four. In this setting, the probability of a signal to be delayed more than 160 time steps is *4.84%* (*SwMax=1*).

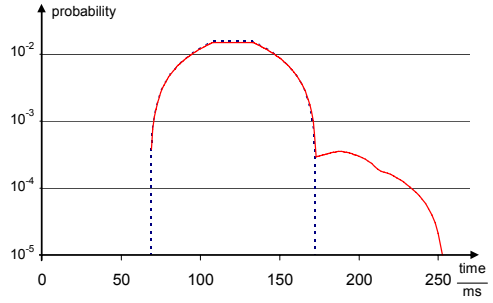

Fig. 15. Delay time with network failures.

4. CONCLUSIONS AND OUTLOOK

Analyzing the consequences of component failures in networked automation systems it is important to take the nature of the exchanged signals into account. Furthermore, using probabilistic model checking, the properties to be checked directly influence the model. A strict separation leads to prohibitive state spaces and verification times. Taking this into account the probability of a critical failure in a systems operation can be determined in an exact way using the presented approach. The next aim for the presented work is to work modeling guidelines with the goal of an automated (computer aided) design process.

REFERENCES

Alur, R., C. Courcoubetis and D. Dill (1999) Model-checking for probabilistic real-time systems, in ICALP'91, LNCS, vol 510, pp. 1 – 100.

Bérard, B., M. Bidiot, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci and Ph. Schnoebelen (2001). *Systems and Software Verification, Model-Checking Techniques and Tools*. Springer.

Greifeneder, J. and G. Frey (2005). Probabilistic Delay Time Analysis in Networked Automation Systems. In: Proc. of the 10th IEEE ETFA 2005, Catania, Italy, Vol. 1, pp. 1065-1068.

Hansson, H. and B. Jonsson (1999). A logic for reasoning about time and reliability. In: Formal Aspects of Computing, 6, no. 4, pp. 512-535.

Henzinger, T., X. Nicollin, J. Sifakis, and S. Yovine (1992). What good are digital clocks? In: Proc. ICALP'92, LNCS, vol. 623, pp. 545–558

Kwiatkowska M., G. Norman and D. Parker (2002). PRISM: Probabilistic symbolic model checker. In: Proc. TOOLS'02, LNCS 2324, pp. 200–204.

Kwiatkowska M., G. Norman, and D. Parker (2004). Modelling and Verification of Probabilistic Systems. In: Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems. CRM Monograph Series, vol 23. AMS, pp. 94 – 215.

Poulard, G. (2003). Modeling and simulation of a control architecture over Ethernet with TCP/IP protocol, Mémoire de DEA, ENS Cachan (F).

PRISM-Website: http://www.cs.bham.ac.uk/~dxp/prism

Greifeneder, J.; Frey, G.: *Dependability analysis of networked automation systems by probabilistic delay time analysis.* Proceedings of the 12th IFAC Symposium on Information Control Problems in Manufacturing (incom 2006), St. Etienne, France, Vol. 1, pp. 269-274, May 2006.