Determination of Delay Times in Failure Afflicted Networked Automation Systems using Probabilistic Model Checking

Jürgen Greifeneder and Georg Frey
University of Kaiserslautern
Erwin-Schrödinger-Str. 12
67663 Kaiserslautern, Germany
{greifeneder, frey}@eit.uni-kl.de

Abstract

The determination of delay times in failure afflicted Networked Automation Systems (NAS) is a new challenge for automation engineers. In addition to the new system structures of NAS which share one or more common devices delays resulting from internal network processes have to be taken into account. Furthermore, the considerable influence of data transmission between several asynchronously executed cyclic processes on system performance should not be neglected in a work on delay times.

This paper introduces a modular modeling approach for NAS based on probabilistic timed automata. The generated models allow the determination of delay times by the use of probabilistic model checking (PMC). To illustrate the concept it is applied to a case study determining reliability properties of a NAS.

***Keywords**: Probabilistic Model Checking, Delay Times, Reliability Analysis, Networked Automation Systems.*

1. Introduction

Modeling and analysis of automation systems requires not only detailed knowledge about the aspects concerning their function, but also about their real time behavior and possible failures. It is especially necessary to know about possibly induced delays. Most control algorithms need to communicate with their process hardware (i.e., sensors and actuators often abbreviated I/O for inputs and outputs) within limited time intervals as the control algorithm will fail otherwise. In cases which are not related to safety, but for example to product quality, an accurate analysis of delay time is even more crucial since a small deviance in delay time (e.g. positioning of a drill or a movable processing unit) will have a direct influence on the processing accuracy and thereby on the quality of the end product.

For an automation system to be dependable a specification of properties such as *"A reaction to a change in a sensor value will be issued within 200 ms."* or *"A change in a sensor value which stays active only for a time interval (pulse) of 5 ms is detected by the controller."* might occur.

In classical structures using a single controller and directly connected I/O the answers to questions like this depend mainly on the controller's cycle time. However, even in the simple case of one PLC cyclically processing a signal the calculation of possible delays results in a spectrum rather than a single value. In the easiest case, this spectrum would turn out to be uniformly distributed. If the PLC and the PLC-I/O-card use independent (i.e. not exactly synchronized) cycle times, the overall structure definitely becomes more complex. If there are several processes involved within one system, the complexity of the latter increases; thereby an exact synchronization of the same processes turns out to be highly unlikely. Access conflicts and queuing times lead to additional non-deterministic delays.

Where distributed systems with controllers communicating with I/Os over networks are concerned the problem is even harder. Here, the network delay (typically not given by a constant value but by a distribution) has to be taken into account. As soon as only one single controller with an I/O-module coupled by Ethernet is in use, the delay time of the total system emerges to a distribution whose determination needs formal analysis [1].

In order to avoid infeasible demands on the control system hardware, which are often entailed by worst-case analyses, the observed properties might be relaxed by the introduction of 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."* or *"A change in a sensor value that stays active only for a time interval (pulse) of 5 ms is detected by the controller in 85% of all cases."*

Simulation is infeasible to check properties like this on a complex system, since a probabilistic solution will need a very long simulation time. The problem is even bigger if the analysis is extended from a performance check to a detailed reliability analysis where the possibilities of failures in the components are considered individually during the analysis. Thus, the system under consideration contains very short cycle times of a controller together with very long mean times between failures (MTBFs) of the components.

The formal description of the considered systems leads to models based on 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 [2]). In the presented work PMC is applied to delay time analysis in Networked Automation Systems given different settings and network failures. The studied systems are of a discrete event type. The properties to be checked upon are not directly related to system failures or delays, but to the loss or delay of process relevant information.

The paper is structured as follows: In the next section requirements for and occurring types of signals in discrete event control systems (DECS) are discussed. The third section furnishes an explanation of the modeling approach, and the fourth section contains a case study demonstrating the possible use of PMC for the assessment of the reliability of distributed network based automation systems. In the fifth and final section the conclusion will be followed by an outlook on further work within the framework of the presented approach. In comparison to former works [1], the modeling approach not only got developed further, but also it will be applied in a full case study and is enriched with a new module concept.

2. Requirements and Signals in DECS

2.1. Classification of Requirements

Requirements on classical continuous control systems are transferable to a generic description level (compensation of disturbances, adjustment of the control variables). Therefore, the considered system can be evaluated along the lines of generic quality metrics. This is not, however, possible in the case of discrete event systems control, where it is possible to evaluate the degree of problem specific demand satisfaction. From there on specific quality values can be derived. Then, an abstraction of the problem descriptions to a multitude of templates makes it possible to categorize different requirements.

In general, all requirements for DECS can be reduced to two general properties: value correctness and temporal correctness [3]. These can be further split up into two corresponding questions: *Will the system respond to an input change with the correct output change (value correctness)?* and *Will it do so within the correct time bounds (temporal correctness)?*

Value correctness, in the DECS case, is determined mainly by the control algorithm itself (the correct functioning of sensors and actuators is preliminary). In the context of the presented work, the correctness of the control algorithms is taken as a given. When it comes to temporal correctness, the multitude of different requirements can be mapped down to three cases:

1. Maximum and minimum (delay) times, e.g. the response time passing between the activation of an emergency button and a reaction.
2. Distribution: With which probability is it possible to react to a signal within x seconds?
3. Differences, distances: Time difference between the times of arrival of two consecutive data packets or the probability of these packages arriving in the correct (original) order.

2.2. Types of Signals

When the question of breakdowns in redundant data networks is dealt with, it does not suffice to know about the probability of the occurrence of one specific breakdown. Rather, the time necessary for the determination of the breakdown itself and the additional time needed to regenerate the information lost during this breakdown must be included. Finally, the probability that information is lost entirely has to be considered; the probability of information not arriving within a given time frame must be determined.

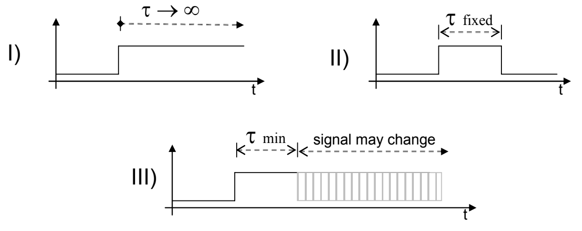


Figure 1. Types of Signals.

In a discrete automated control system, there are three kinds of signals to be discussed (cf. Figure 1):

1. A signal which stays active until it is processed (and necessary activities are consecutively carried out), for example the emergency stop. After it has occurred, it will not be reset until the message has been received and an action has taken place.
   Question: When will the signal be discovered?
2. A signal which is active only for a fixed time period τ_{fixed}. After that period the information is lost if it has not been read out. An example of this type of signal is that given when a metallic object passes an inductive sensor.
   Question: Will the signal be discovered?
3. The combination of the above mentioned signals, for example one which changes its value and keeps this new value until the next change or at least for τ_{min}. Questions:
   a. Will the signal change be discovered before it changes again? and
   b. When will a signal change be discovered?

3. Modeling approach

The networked automation system under consideration consists of one or more controllers connected to the process by a network built of Switches and I/O-modules. The components exchange signals and information using TCP/IP. Some components exist more than once and the model of the system has to be scalable in an easy way.

Therefore, it makes sense to model the different components separately and then connect them to build the system discussed.

For the construction of the individual components probabilistic timed automata (PTA) as well as the method of digital clocks [4] are used. This offers the possibility to eliminate non-deterministic decisions through the reduction of the time axis to a set of discrete time steps. In the course of the experiment, all relevant decisions (transitions) are fired synchronously. Thus, the model no longer records the exact time of occurrence, but only the fact that an event occurred within the last time step. In some systems, though, it is not possible to identify an optimal time increment at all, so that each participating subsystem executes one and only one action. If the time increment is chosen to be smaller than the shortest (time) difference in between two successive system changes, the size of the model tends to increase exponentially.

In the following sections, Probabilistic Model Checking will be introduced and the chosen system model will be presented. Then, the problem of the initial states will be discussed, and the property formulation in PCTL (Probabilistic Computation Tree Logic) will be introduced. Finally, the module concept will be described. In the respective sections the coding in PRISM (a model checker from the University of Birmingham, [5]) will be interlaced.

3.1. Probabilistic Model Checking

If model checking [6] is to be used, a model of the system has to rely on formal description. The properties under observation are also defined in terms of formal logic. These two descriptions are input to a model checking algorithm that checks whether the properties hold on the system (cf. Figure 2). System and properties are dealt with separately from one another; thus, a change in the system affects only the system model, a change in the properties only the formal properties.

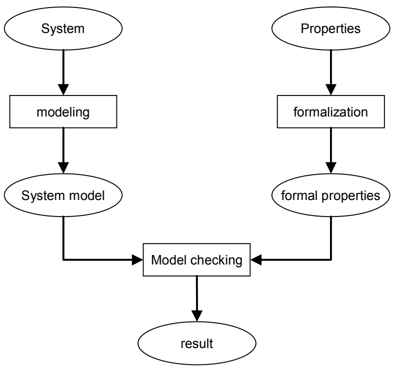


Figure 2. Workflow of model checking.

The main advantage of model checking is the possibility of complete coverage of all possible evolutions of a system (as opposed to a subset only, which is all that can be provided for in the cases of simulation and testing). The main drawback is, however, that the state space to be covered tends to increase very quickly with an increasing complexity of the model (state space explosion problem).

The state space explosion 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 the case of 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 mathematically as 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 either one or zero – true or false. Therefore, the first condition to a probabilistic model must be that it will terminate after it reaches the event supervised. Furthermore, each possible arrangement must be depicted in that very first cycle, and the model must recognize by itself that the event took place. This prerogative changes the design process fundamentally: the separation between model and properties must be abolished and replaced by a new formal design process (cf. Figure 3).

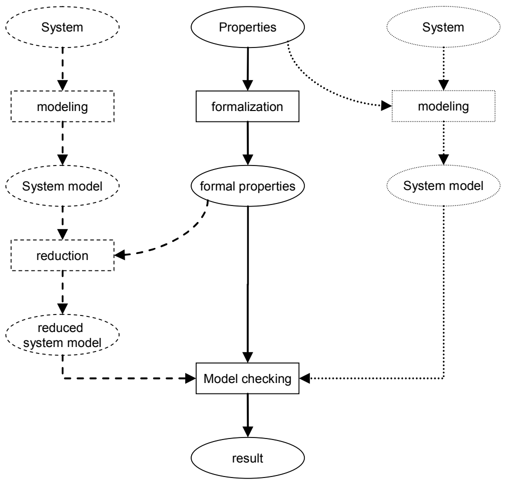


Figure 3. Workflow of probabilistic model checking.

While the formalization process for the properties stays the same as in Figure 2 (shown in the middle of Figure 3), the construction of the model must be changed. The first possible way to do so is shown with dashed lines on the left-hand side of Figure 3: 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 Figure 3, is to include the properties already in the modeling task. This leads to much more refined models, but requires the engineer to rebuild his models for each possible case. In this work, the second (dotted) approach is used. This will provide a deep understanding of what is an optimal model on the ground of which a suitable reduction algorithm can be proposed. In the further course of this project the experience – which will be gathered up to this point in future – in dealing with this kind of modules should lead to a suitable reduction algorithm and therefore towards the dotted (left) branch.

3.2. System model

Finite automata with extensions for timed and probabilistic behavior are used to build the system model. A finite automaton as shown in Figure 4 consists of states linked by conditional transitions.

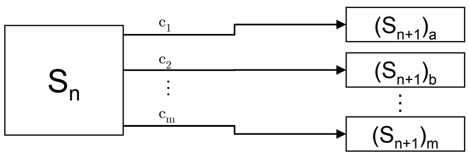


Figure 4. Finite Automaton.

For any state S_n in an automaton with m possible post-states connected via transitions t_1 to t_m with transition conditions c_1 to c_m respectively the following must hold:

1. There are no two conditions active at the same time:
$$\forall_{i,j\in[1..m];i\neq j} \; c_i \wedge c_j = false \qquad (1)$$

2. The disjunction of all conditions must cover the total state space:
$$\bigvee_{\forall i\in[1..m]} c_i = true \qquad (2)$$

The combination of these two preliminaries implies that there is always one and only one active transition within a given module.

For reasons of time scaling, a transition should only become active when a well-defined period of time has passed. The accomplishment of that can be assured by the employment of timed automata (Figure 5a).

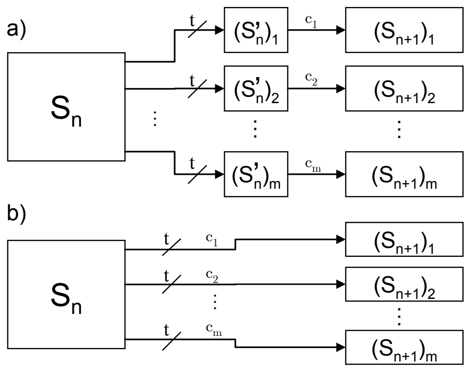


Figure 5. Timed Automaton.

When the time t has passed, all transitions, which are 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) on which all processes are synchronized (instead of several different sync-times which could be used with a general timed automaton) and second, the condition of the transition following will be evaluated in the same moment as the sync-transition becomes activated. Given these assumptions, it makes sense to simply omit the intermediate states S'_n and use the graphical representation shown in Figure 5b.

The next step of extension leads directly to the probabilistic timed automata (PTA [7], cf. Figure 6a). In this probabilistic case, the transitions are assigned a probability $p_i \in [0..1]$, with:

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

In other words, the deterministic automaton is 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 Figure 6b is used.

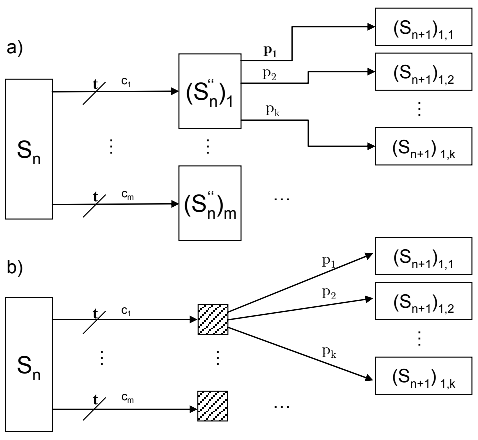


Figure 6. Probabilistic Timed Automaton.

The difference between a non-deterministic choice out of two paths and a stochastic choice out of two paths weighted each with a probability of 50% is that PMC will consider each path of the stochastic choice with a probability of 50% while it will separate the non-deterministic choice in two independent automata, once with the first and once with the second path. In this case, the choice is eliminated and the associated probability will be 100% for the respective transition. Yet, there obviously remains a difference between that probability and the 50% in the case of a stochastic choice.

Note: Probabilities which equal one and conditions which are true for all times are for reasons of graphical simplification not written into the transitions. The PRISM code of the system model itself is constructed as follows:

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

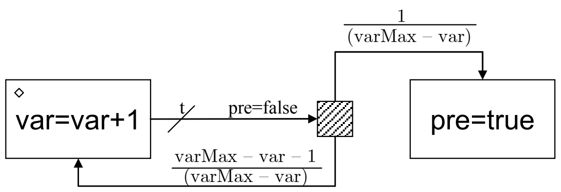
[Pr] is the synchronization signal mentioned before. If it is omitted, then the determinism of the sequence is destroyed. Conditions are a predicate formulation composed of one or more transition conditions c_i (or their negated partners) coupled by binary operators. Assignments are value assignments 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 probabilistic automata.

3.3. Initial state

One of the most important differences between simulation and model checking is that model checking will definitely reach all the possible states at least once, while in simulation even after a long period of time this might not be the case. On the other hand, it would be ridiculous to run model checking more than once, if it is possible to reach every possible state by means as simple as the right definition of the initial conditions. This makes it much easier to achieve the previously reclaimed character of a model checking algorithm to terminate after reaching the state, the user is testing on. The right (and complete) assignment of the initial states is difficult. 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 paralleled (synchronized). In the ideal case there are only two drifting modules and all possible drift times will occur with the same probability. In this case, the initial condition can be found easily by pre-adaption of an "equally randomized initial state" cycle just before the first time step in one of the two modules. This is shown in Figure 7 and can in PRISM best 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;
```

[P] is the sync-operator, *pre* is the binary variable serving as a detector of the completion or non-completion of the initial process. !*pre* is the negation of *pre*. *pre2*… *pren* are the detection variables of synchronized modules, as the automaton would deadlock, without this command (cf. section 3.2).



$\frac{1}{(varMax - var)}$
var=var+1
t
pre=false
pre=true
$\frac{varMax - var - 1}{(varMax - var)}$

Figure 7. Equally randomized initial state.

3.4. Property formulation in PCTL

PMC uses an extension of CTL (PCTL – Probabilistic Computation Tree Logic, [8]) to specify properties of systems described by Markov models. Typically, these properties are composed of atomic propositions or predicates on the variables in the model. Strictly speaking, there is a distinction between state and path constructs, but for the sake of this work it may be neglected. Normally, PCTL formulas evaluate to create a Boolean value. Yet, it is often useful to know the actual probability rather than merely assure a probability above or below a given bound. For that reason several implementations of PMC facilitate this functionality additionally. PRISM allows properties of the following form:

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

P=? represents the probability to be determined, *expr1* and *expr2* are predicates 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 using this operator. 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. It is important that in the course of that action the algorithm terminates as soon as possible. There are two main cases to be distinguished:

a) If the main interest lies on the duration – 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 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 occurs, the value of the time counter will be assigned to that variable. It is possible to check on the counter variable directly; yet, in that case the result would be the integral of the distribution function from *Lf* to *infinity*.

b) If the main interest lies on the probability of an overall occurrence of the desired property, then 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 which 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. To solve this problem it is important to cover all possible initial states in the first cycle (cf. section 3.3) and terminate the automaton after finishing this one cycle.

3.5. Module concept

If the discussed characteristics of the model to be designed are taken into consideration, the code being implemented can be classified towards three categories:

1. Basic functions
2. Architecturally based assignments
3. Signal tracking.

The 'basic functions' category is featured by the fact that those can be implemented independently from the specific problem descriptions or the properties to be observed. An integral part of this category is e.g. the basic function of a sensor to return its signal's value if it is asked for it as well as the cyclic run of a programmable logic controller (PLC). These basic functions must be endorsed problem-specifically. The PLC – for example – must know, which I/O-modules it has to scan and (if this should be modeled explicitly) what has to be done with the signals returned by the I/O-modules. Since these specifications as well as information like which switch a specific I/O-module are connected to have in common their architectural nature, they constitute the second category. The third category is directly linked to the properties observed. All modules which are tracking the completion of properties are therefore part of this third category. They are necessary as PCTL (in the chosen syntax) does not facilitate of specific sequence of states.

Ideally, it is possible to design the modules of the three categories independently from one another. However, this is possible only for few examples and requires a rather high amount of available memory (RAM) capacity. In all other cases a reduction process has to be added after the design using these three categories.

4. Case study

The structure of the case study to be discussed is shown in Figure 8. In this example the investigation concentrates on the superimposition of different cyclic processes.

The system consists of two PLCs which integrate a controller with a cycle time of 3.5ms. Each of the PLCs has a connected I/O-board with an independent cycle of 5ms. These two modules (PLC and corresponding I/O-board) are not synchronized and communicate via a shared memory. While PLC_1 is connected to switch 1, PLC_2 is connected directly to switch 2. The system also contains 8 digital I/O-boards to which one or several sensors and actuators are connected. In this setting the I/O-boards 1 to 4 are connected to the first Switch, while the I/O-boards 5 to 8 are connected to the second Switch. The two switches are connected to one another as well.

The PLC on the left-hand side processes the sensor information of the I/O-boards 1 to 5, while the PLC on the right hand side processes the information of the I/O-boards 4 to 8. All I/O-boards are sampled cyclically.

Note that in the underlying client server protocol the PLC (=Client) sends requests to the I/O-modules (=Server) and the latter "answer" only to these requests, i.e. the I/O-boards do not sent any information by themselves. To dispatch a data packet the following delays occur:

- 0.25ms to pack (send) and unpack a packet.
- 0.06ms per Switch to be passed.

As there is no method implementing a queue module yet (cf. chapter 5) the transference over a switch has been assumed to be waiting time free. For the discussion of a non-constant switching time see [1].

If one of the digital I/O-boards receives a request, 1.5ms pass before the accompanying answer returns over the network – indeed, only if the I/O-board has to process no other request. This can be the case for I/O-boards 4 and 5, as both of them have to process requests from both PLCs. If an I/O-board is processing a request when

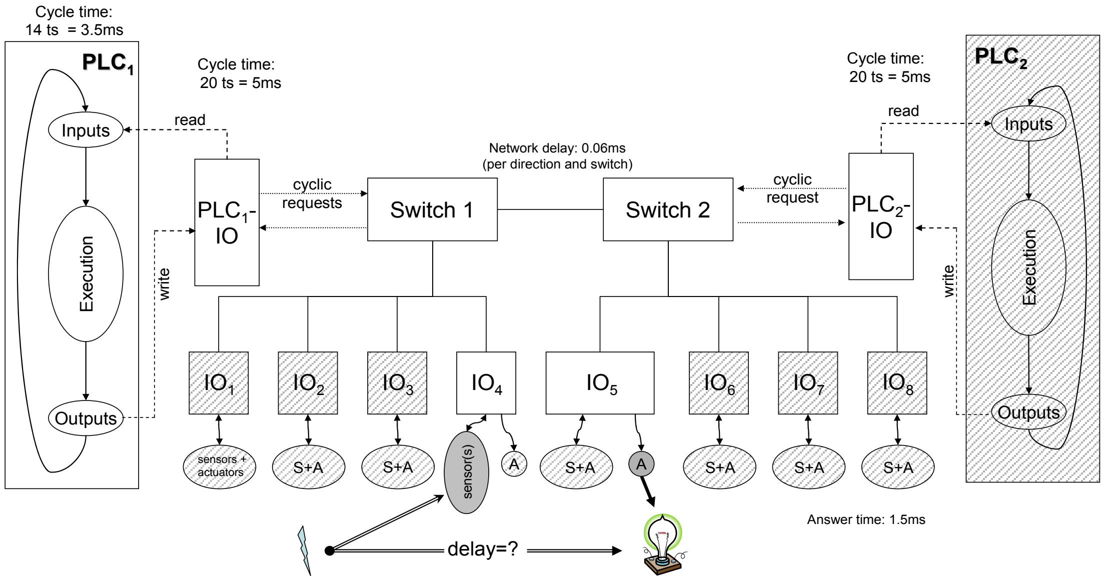


Figure 8. Case study.

another request arrives, the second request must wait until the first request has been treated. These attendance periods create further delays.

It is assumed that PLC_1 activates an actuator located on I/O-board 5 following the occurrence of a signal change at a sensor which should be located on I/O-Board 4. The signal appearing at I/O-board 4 has signal type I characteristics, i.e. that signal will not be reset until the corresponding action took place. The property to be determined is the actual delay time passing between the signal change at I/O-board 4 and the activation of the actuator at I/O-board 5.

In the beginning of their cycle, the PLC-I/O-boards are sending the requests to the I/O-boards they are associated with in numerically ascending order, i.e. PLC-I/O-1 sends first a request to I/O-board1, then to I/O-board2, -3, -4 and -5; PLC-I/O-2 sends a request first to I/O-board4, then -5, -6, -7 and -8. This functional peculiarity justifies the negligence of a detailed network model. By this, the I/O-boards 1, 2, 3, 6, 7 and 8 appear only indirectly in the model, e.g. they can be neglected in the calculations (and therefore in the PRISM code). The same is true for the second PLC located on the top right hand side of Figure 8. As the cycle of this PLC is independent of its I/O-board cycle and this PLC is not part of the later discussion, it will be neglected too.

PLC1, the delay time determination and the basic modules can be implemented as module type 1 (cf. section 3.5). This means that they can be implemented independently of the specific task. Special algorithms are required for the I/O-boards which indicate the linking structures (module type 2). By reason of the relatively simple case study the complete signal tracking (module type 3) can be realized in one single independent module. Also of module type 3 is the termination module. In the following the single modules are briefly introduced.

In addition to these basic settings the incidence of a network failure is implemented. In this case, the network (or part of it) is not available for 4ms. The failure can occur with a probability of 10^{-4}. If the network is down, this can mean, that there is congestion, a rewriting of the IP-table, a reboot of a switch or the specific I/O-board and so on. All these cases are modeled as the same event. It is assumed, that the destination for any IP-packet is not available in the net-down states. Consequently, if an I/O-board tries to send a package to the network in this period, it remains trying to sent this package several (at the maximum of 15) times. The maximum trial time can be calculated to be

$$\sum_{n=1}^{15}(2^n-1)\cdot 2\tau,\ \tau=25{,}6\mu\text{s} \qquad (4)$$

As a resent in the IP-network will occur after (2^n-1) times 51,2 μs (plus 9,6μs, with n the number of trials), the maximum trial time would be 370ms. This however is much more than the net-down time of 4ms (the probability of reaching 370ms is $(10^{-4})^{93} \approx 0$). The I/O-board therefore waits until the network is available again and processes afterwards.

The PLC-I/O-boards are able to receive data at all times. A package's information is passed to the shared memory immediately within the processing time of one time step. This is true even for late-packages.

It is assumed, that the relative time drift in between the modules is less than one time step over the total time period. The given delay times are educated guesses based on lab measurements of similar structures.

4.1. Detailed Modules

The module of the PLC as well as the core modules of the both PLC-I/O-boards are built from ring counters (cf. Figure 9) which count from the initial value up to a maximum value and start again with 0.
As in all following state charts, Figure 9 does not show the reset-transitions which are activated by the termination-module and lead from each state of the chart to a predefined termination-state. These transitions are not shown as they would make the state charts extremely complex without gaining any more important behavioral information.

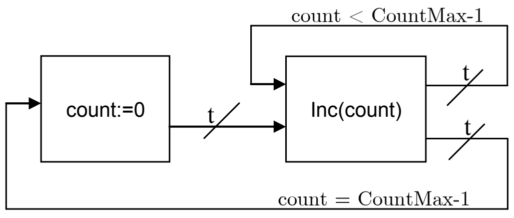


Figure 9. State chart of a ring counter.

For the delay time determination the same module was used, indeed, without reset when the maximum time (CountMax-1) is reached. In this case the counter remains on its maximum position.

For both the PLC-I/O-boards, as well as for the PLC, initial state creation automata were realized as discussed in section 2.3 (cf. Figure 7). It is assumed, that the drift within the different processor time bases is less than one time step (0.25ms) within the calculation time (approximately 25ms).

As the network wasn't modeled explicitly (but only accounted for by a constant time delay) the I/O-boards input modules must provide two functions: First, they must "take up" a new request in dependence of the counter variables of the PLC-I/O-boards. Second they must buffer the request if a second request arrives, before the first one has been processed. Figure 10 shows the accompanying state chart.

Already in this example, it is slightly appreciative why realizing queues in the straight forward variation leads to huge state spaces: If there would be several (possibly in short cycles requesting) PLCs – instead of two PLCs which request within the run time just twice using a relatively large cycle – the state chart (and with it the automaton) would be by far more complex. The accompanying state explosion can then only be shrouded by using another abstraction level in the state charts representation. If queues as they appear, e.g. in switches, were modeled in this visual, direct way, the state explosion would make the models absolutely no more analyzable (under the constraints of the available software tools). Possible alternatives to solve this problem are being discussed actually.

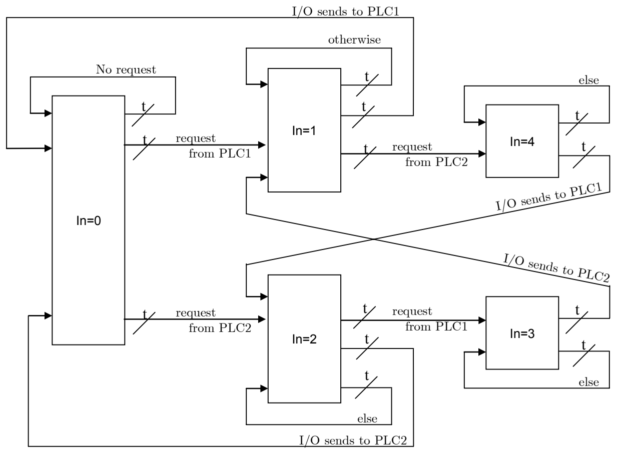


Figure 10. State chart of the I/O-board input queue.

In principle, the two PLC-I/O-boards also need an input buffer. However this was neglected in this implementation, as solely one module for each of the requests was created, which detects whether or not the corresponding signal returned already. Note: In this implementation the modeled data packages have fictive character, i.e. there is no information available on what data they are carrying and if the request really arrived. This information is modeled indirectly by the above discussed modules. The corresponding state chart is shown in Figure 11.

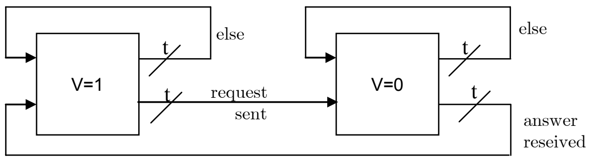


Figure 11. State chart of the PLC-I/O-request-back-module.

While the "request sent" is created indirectly by the value of the corresponding PLC-I/O-counter, the "answer sent" is generated by the I/O-board (cf. Figure 12).

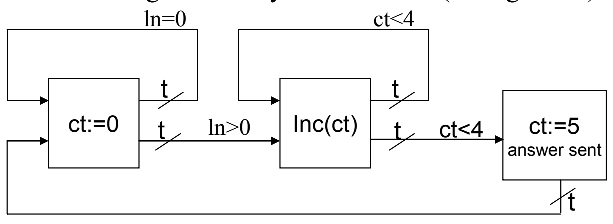


Figure 12. State chart of each of the I/O-boards.

The I/O-board acts as server, i.e. it remains at state 0 until a request (ln>0, see also Figure 10) has arrived. From there it works as a ring counter.

Finally, only the two signal tracking modules are absent. The first one is the already mentioned termination module. This module consists of two states: work and terminate. Initially, the automaton is in the work state and stays there until either the maximal signal delay time has elapsed or the actuator has been activated. The second signal tracking module "observes" the information passing through the system. The first state transition is triggered, if the packet, sent by the sensors I/O-board has arrived at the PLC-I/O-board. The next two state changes occur, as soon as the PLC cycle reaches its "read"-state (first change) and the ensuing "write" state (second change). The consequently following step is the arrival at the I/O-board 5. As the PLC-I/O uses the same request to write out values as to ask for the actual sensor values, writing-out isn't modeled at all, but only tracked by the module. The final (state changing) transition of the tracking automaton is activated when I/O-board 5 successfully processed the request. By passing this transition, the "delay time variable" is set as discussed in section 3.4. This state is used by the termination module to terminate the calculation.

Under the consideration of network failures, the state chart of the PLC-I/O-boards change from a single ring counter (cf. Figure 9) towards the state chart shown in Figure 13. The difference is given by the two blocks in the middle of Figure 13. For all the packages to be send (0<count<6) the I/O-board tries to send a package until it has been sent.

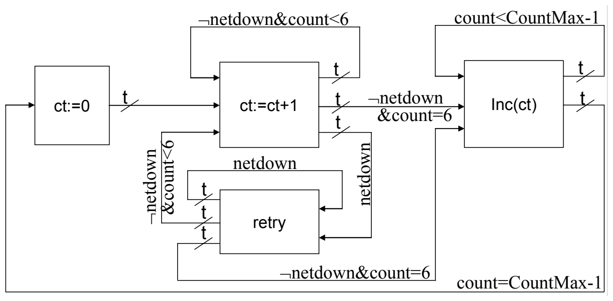


Figure 13. State chart of the I/O-Boards under consideration of network failures.

Obviously, the state charts of the I/O-boards must be expanded also (as the answer can't be send while the network is absent). This is shown in Figure 14.

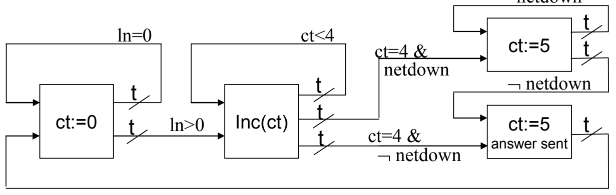


Figure 14. State chart of an I/O-board under the consideration of network failures.

The state chart of the net-down module is shown in Figure 15. While everything is fine (NetDown=0) the network may fail in each time step with a probability $p_{NetDown}=10^{-4}$ or remain in the OK-State with a probability of $(1-p_{NetDown})$. If the transition to the left gets activated, the automaton passes to the state on the right hand side of Figure 15 and stays there until the Net-down-time of 4ms has passed. Afterwards it (normally) returns to the state on the left hand side. From there another failure may occur immediately (with probability $p_{NetDown}$).

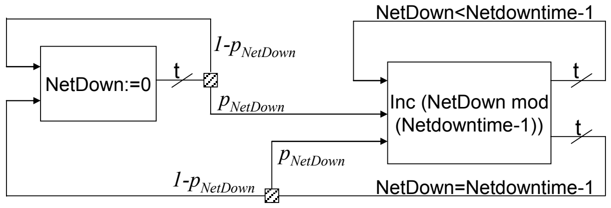


Figure 15. State chart of the net-down module.

4.2. Results

Figure 16 shows the probability of different delay times. The calculated values are marked by little crosses, connected by linear interpolation.

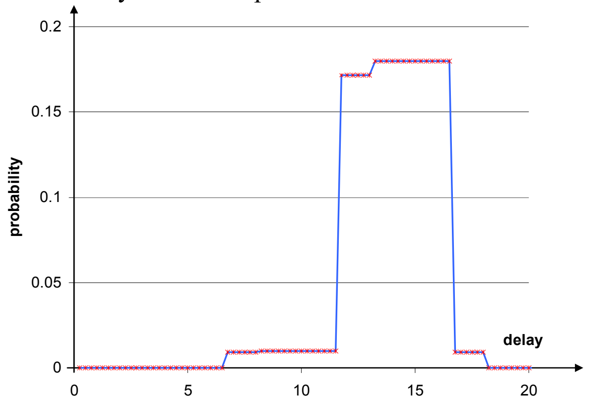


Figure 16. Time between occurrence and reaction.

The value of 18% at 15ms means that the probability for a delay time between 14.75ms and 15ms is 18% multiplied with the synchronization time step width of 0.25 what equals 4.5%.

The shortest delay time (7ms) originates as shown in Figure 17. In this case, the signal change occurs just at the moment when PLC_1-I/O's counter variable equals 4 and there is no waiting queue at I/O-board4. Furthermore, the PLC_1's cycle itself is reading just after the answer from I/O-board4 has arrived.

Starting from 12ms the response times of PLC and PLC-I/O cycle harmonize no more. The length of the plateaus is caused by the answer time of the I/O-board (1.5ms = 6 time steps) and the PLC-I/O-cycle-time (5ms = 20 time steps). The plateau starting at 17ms combines unfavorable waiting periods in the I/O-board with cycles not in harmony of PLC and PLC-I/O. The maximum delay time is 18.25ms. This worst case originates as shown in Figure 18.

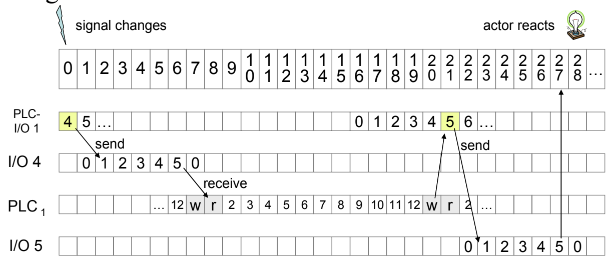


Figure 17. Minimum delay time.

In this case, the signal change occurs exactly when PLC_1-I/O's counter variable equals 5. Therefore, the next request will arrive 19 time steps (4.75ms) later. Unfortunately, there is a waiting queue at I/O-board4, as PLC-I/O-board5 has requested just before. 2.75ms later, the information gets sent back to PLC_1. However, in the corresponding cycle the read state just has passed. Hence, another delay of 3.25ms is added. After being processed by the PLC itself, the information has to wait until PLC_1-I/O's counter variable equals 5 and it can be sent to I/O-board 5. Arriving there, the request from PLC_2 was faster again and therefore another wait-time of 1.5ms is occurring, before the information is being processed by I/O-board 5 and finally handed over to the corresponding actuator. Obviously, this is not the only case resulting in the maximum delay time, as some of the blocks (e.g. PLC_1's cycle) may be shifted in relation to the other cycles by a small time (e.g. 1 time step to the right) without changing the drafted frame.

In conclusion it can be adhered, that 88.5% of the delay times are in the range between 12 and 16.75ms. Furthermore, it can be guaranteed, that in 99% of all cases the delay stays below 17.5ms. Additionally, and for most applications most important, the maximum response time is proven to be 18.25ms. On the other hand – also interesting for safety reasons – there will be no reaction earlier than 7ms after the occurrence of the signal change.

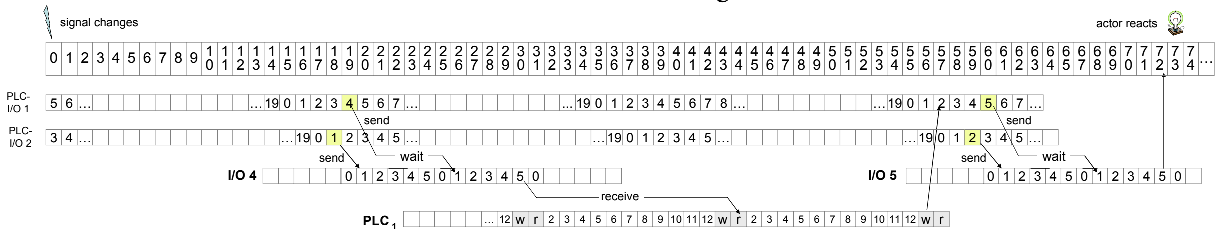


Figure 18. Maximum delay time.

The second experiment to be discussed is the influence of network failures (as defined above) on the delay time. Figure 19 shows the comparison of the graph with and the graph without failures (logarithmic scale). The differences between the two graphs (with failures minus without failures) are given in Figure 20 (linear scale). In relation to the absolute numbers, the two graphs are nearly the same up to 17ms despite some lump differences in the beginning.

Starting from 17.25ms, the number of packages being delayed for the corresponding duration has increased by about 8%. Whereas there are no packages being faster than before, there is quite a good number of late packages now. The probability of no delay times longer than 17.5ms has decreased from 99% in the undisturbed case to 94%. Even worse, the probability of a package being delayed for more than the above mentioned maximum delay of 18.25ms is 5.1% and therefore in a significant range.

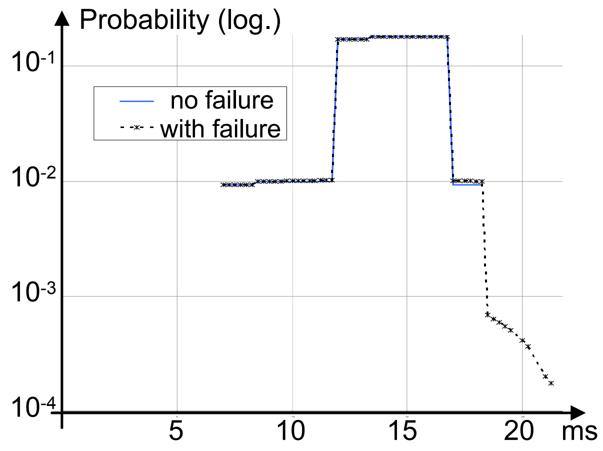


Figure 19. Delay time with network failures.

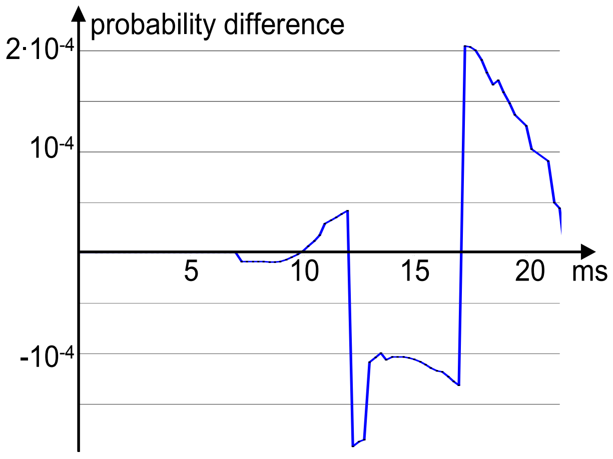


Figure 20. Difference in between delay times (failure minus without failure).

5. Conclusion and Outlook

The reliability of distributed network based automation systems can be discussed by using PMC as shown by a case study. Thereby worst case analyses are successfully avoided.

When PMC is used to analyze the behavior of network-based automation systems, the choice of properties to be proven influences the model to be formulated. Hence, a sharp separation of model and property formulation leads to both a huge state space and a huge verification time. Therefore, a new modular modeling approach considering the properties during the formulation of the system model has been presented.

The next milestones of the introduced work are located in a refinement of the introduced application example as well as in the discussion of possibilities to sensibly implement queues (i.e. without the discussed state explosion). Furthermore, the advantages of different PLC-I/O behaviors and of the use of event based instead of polling mechanisms will be discussed. Last not least the elaboration of experience will lead to a suitable reduction algorithm and the possibility of a computer based design process.

References

[1] J. Greifeneder and G. Frey, "*Probabilistic Delay Time Analysis in Networked Automation Systems*", Proceedings of the 10th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2005, Catania, Italy, Vol. 1, pp. 1065–1068, Sept. 2005.

[2] M. Kwiatkowska, G. Norman, D. Parker, *Modelling and Verification of Probabilistic Systems*, in: Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems. *CRM Monograph Series*, vol 23. American Mathematical Society, 2004.

[3] H. Kopetz, *Time-triggered real-time computing*. Annual Reviews in Control 27 (2003) 3–13, 2003.

[4] T. Henzinger, T., X. Nicollin, J. Sifakis, S. Yovine. *What good are digital clocks?* Proc. ICALP'92, LNCS, vol. 623: 545–558, Springer, 1992.

[5] M. Kwiatkowska, G. Norman, D. Parker. *PRISM: Probabilistic symbolic model checker.* Proc. TOOLS'02, LNCS, vol. 2324: 200–204. Springer, 2002.

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

[7] R. Alur, C. Courcoubetis, D. Dill, *Model-checking for probabilistic real-time systems.* ICALP'91, LNCS, vol 510: 1–100, Springer, 1991.

[8] H. Hansson, B. Jonsson, *A logic for reasoning about time and reliability*, Formal Aspects of Computing, 6, no. 4: 512–535, 1999.