This is the author's version of the work. The definitive work was published in *Proceedings of the 27th International Conference on Computer Safety, Reliability and Security (SAFECOMP 2008)*, pp. 139-152, 2008. The work is supported in part by the German Science Foundation (DFG), SFB 694 and SPP 1148.

Symbolic Reliability Analysis of Self-healing Networked Embedded Systems

Michael Glaß, Martin Lukasiewycz, Felix Reimann,
Christian Haubelt, and Jürgen Teich

Hardware/Software Co-Design, Department of Computer Science
University of Erlangen-Nuremberg, Germany
{glass, martin.lukasiewycz, felix.reimann, haubelt,
teich}@cs.fau.de

Abstract. In recent years, several network online algorithms have been studied that exhibit self-x properties such as self-healing or self-adaption. These properties are used to improve systems characteristics like, e.g., fault-tolerance, reliability, or load-balancing.
In this paper, a symbolic reliability analysis of self-healing networked embedded systems that rely on self-reconfiguration and self-routing is presented. The proposed analysis technique respects resource constraints such as the maximum computational load or the maximum memory size, and calculates the achievable reliability of a given system. This analytical approach considers the topology of the system, the properties of the resources, and the executed applications. Moreover, it is independent of the used online algorithms that implement the self-healing properties, but determines the achievable upper bound for the systems reliability. Since this analysis is not tailored to a specific online algorithm, it allows a reasonable decision making on the used algorithm by enabling a rating of different self-healing strategies. Experimental results show the effectiveness of the introduced technique even for large networked embedded systems.

1 Introduction

Systems like, e.g., *automotive* or *avionics electronic control unit (ECU) networks*, networks from the area of industrial control automation, *body-area networks*, or *sensor networks* combine the aspects of both embedded systems and networks. Due to constraints in area consumption, monetary costs, and energy consumption, the used resources exhibit limiting properties in the field of, e.g., computational power or memory size which is typical for embedded systems. On the other hand, the resources are distributed within the systems and, thus, they resemble networks. This distribution is crucial to allow controlling, monitoring, and analysis of the system under the aspect of limited and remote installation spaces. Moreover, these systems have to be optimized with respect to different criteria, ranging from monetary costs, area and power consumption, or throughput to flexibility, reliability and fault-tolerance. Commonly, this system category is referred to as *networked embedded systems*.

For the reliability analysis proposed in this work, several aspects of networked embedded systems are of great importance. Commonly, networked embedded systems are

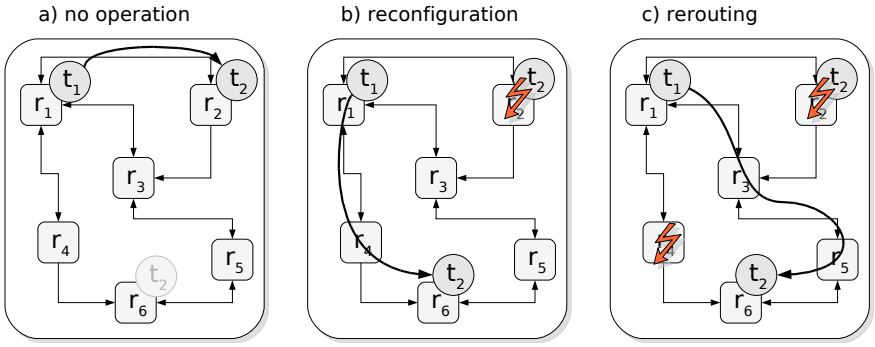

Fig. 1. A self-healing networked embedded system with a data transfer from task t_1 to task t_2. In a), no defect resources are present. In case of resource failures, a reconfiguration activates redundant task instances, cf. b), or reestablishes the communication using a dynamic rerouting, cf. b).

deployed in unattended areas and, thus, administrative tasks and maintenance are expensive and should be avoided or are sometimes even impossible to accomplish. As a matter of fact, resources of the networked embedded systems move from dedicated and protected mounting spaces to installation spaces with destructive agents. These spaces can be found near sensors or within, e.g., an engine or moving parts of vehicles. This trend especially increases the amount of destructive influences on the used hardware, such that permanent failures occur more frequently.

In recent years, systems have been proposed that exhibit so called *self-x* properties. These systems monitor themselves and are able to react autonomously to unwanted system states. Popular properties for self-x systems are *self-adaption* that allows the system to react to different environment conditions or new applications and *self-healing* that allows the system to react to failures and, thus, increases the reliability and fault tolerance of the system, cf. [1]. In this work, we will focus on self-healing networked embedded systems that are based on self-reconfiguration and self-routing. An example of such a system is given in Fig. 1. On the systems architecture, a given set of communicating applications is executed while the communication is implemented via static routes. Hence, there are hardly any dedicated routing resources but the resources perform both the computation of the tasks and the routing using their point-to-point interconnections. In case of a resource failure, the failure has to be detected, cf. [2, 3]. After the detection, the redundant instances of the tasks executed on the defect resource that are available in the system are activated using a reconfiguration of the other resources, cf. [4]. For all newly activated task instances and for all communication routes that use the defect resource, a rerouting is performed. Besides the fact that such a system can act autonomously, self-reconfiguration and self-routing allow a resource sharing of active tasks and redundant, inactive tasks. Thus, highly increased costs introduced by static structural redundancy can be avoided.

In this paper, we present a symbolic reliability analysis of such self-healing networked embedded systems. This analysis aims to determine the achievable reliability of the system, independent of the used algorithms to implement the self-healing property. The knowledge of the achievable reliability is important since it allows to quantify the quality of the available online algorithms. Moreover, in the design phase of the system, the achievable reliability allows a coarse grained selection from different system layouts and can be embedded into a design space exploration, since the calculation is much faster than an evaluation of different online algorithms. The presented reliability analysis is based on *Binary Decision Diagrams* (BDDs) [5] and considers both reconfiguration and routing using a symbolic fixed-point iteration. Moreover, even constraints that depend on the dynamic activation of tasks are encoded in the BDD, allowing to respect constraints like, e.g., maximum computational load of a resource.

The remainder of the paper is as follows: Section 2 discusses prior work. In Sec. 3, a formulation of the problem we target in this work is given. Section 4 introduces our symbolic analysis approach for self-healing networked embedded systems. Sec. 5 shows the results of the introduced technique applied to examples where several implementations of self-healing algorithms are available as well as an networked embedded system that corresponds to systems currently used as automotive ECU networks. The paper is concluded in Sec. 6.

2 Related Work

The reliability of self-healing networks has been widely studied, cf. [6–9]. Hence, all these approaches focus on the network as a communication platform itself while the aspects of nodes of the networked embedded system as both communication and computational resource are neglected. Thus, the approaches are more related to the reliability analysis of classical networks [10] and restricted to given self-healing strategies.

Other approaches can be found in the area of *self-repairing embryonic cells* [11] and *wireless sensor networks* [12, 13]. Embryonic cells have a very special architecture that comes with a high degree of spatial redundancy to implement the self-repairing property and, thus, are not appropriate as a networked embedded system model. Wireless sensor networks on the other hand are highly meshed networks and have changing communication possibilities due to their wireless communication medium. Hence, since these sensor networks often include aspects of maintenance as well, simulative approaches are used to quantify their reliability.

The typical networked embedded system is hard wired with maintenance being hardly possible. Thus, the reliability analysis proposed in this work is inspired by the reliability analysis of embedded systems [14] and networked embedded systems without self-healing properties [15]. We will extend these approaches by considering the reconfiguration and dynamic routing that is used by the online algorithm. Moreover, constraints that depend on the online activation of tasks are included in the analysis as well.

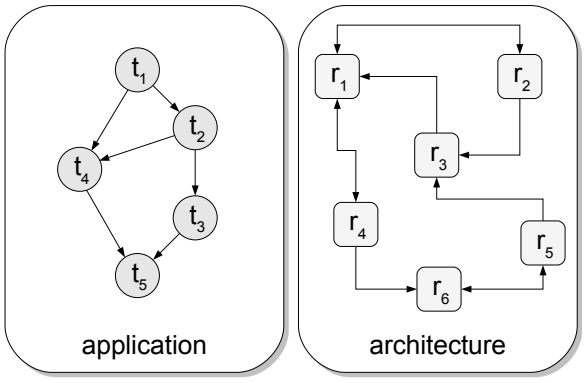

Fig. 2. A system specification with an application and the available resource architecture.

3 Problem Description

In this paper, we target the problem of determining the reliability of a self-reconfigurable and self-routing networked embedded system. Hereby, the tasks that are executed on the resources as well as their data dependencies are given. Moreover, the system topology and the reliability attributes of each resource are known. The assumed failure model is permanent failure due to resource failures.

Our formal specification of a *system* consists of the *application*, the system layout or *architecture* and the relation between these two views:

- The application is modeled by a task graph $g_{\mathsf{t}}(V_{\mathsf{t}}, E_{\mathsf{t}})$ that describes the behavior of the system. The vertices $t_1, ..., t_{|V_{\mathsf{t}}|} \in V_{\mathsf{t}}$ denote tasks whereas the directed edges E_{t} are data dependencies. Attributes like, e.g., memory usage, computational load are assigned to tasks.
- The architecture is modeled by a graph $g_{\mathsf{a}}(V_{\mathsf{a}}, E_{\mathsf{a}})$ and represents possible interconnected hardware resources. The vertices $r_1, ..., r_{|V_{\mathsf{a}}|} \in V_{\mathsf{a}}$ represent resources that can be both processing units or communication units like buses or gateways. The edges E_{a} model available communication links between the resources. Attributes like, e.g., the memory size, maximum computational capacity or reliability are assigned to the resources.

Each resource has the ability to route information to all resources along the directed communication connections, thus, from a routing point of view, all resources can be seen as *network nodes* with point-to-point connections. An example of an application and a given architecture is shown in Fig. 2.

In this model, the execution of a given task is limited to selected resources. This is due to the fact that not each device is generally present on every resource in a heterogeneous system. Therefore, a relation between application and architecture called *mapping* is introduced in the system model:

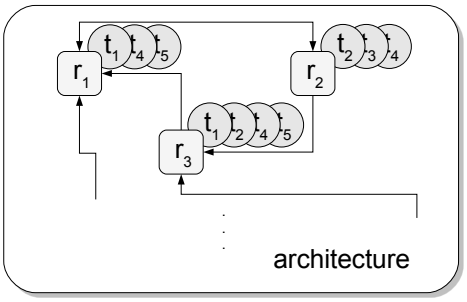

Fig. 3. A part of the application shown in Fig. 2. Depicted are the tasks that are mapped to the resources.

- The mapping $M : V_\mathsf{t} \rightarrow 2^{V_\mathsf{a}}$ assigns to each task t a set of possible resources for its execution and $M : V_\mathsf{a} \rightarrow 2^{V_\mathsf{t}}$ assigns each resource r a set of tasks that can be executed on r, respectively.
- An *instance* $i = (t, r)$ of a task t corresponds to this task being executed on the resource $r \in M(t)$. The set of all available instances of all tasks is defined as $I = \{(t, r) \mid t \in V_\mathsf{t}, r \in M(t)\}$.

In the online phase of the system, at least one instance of each task has to be *activated*, i.e., has to be executed. An example of a part of the application and its relation to the given architecture is shown in Fig. 3.

The task of the used online algorithms is to keep the networked embedded system *feasible*.

Definition 1. *A system is called* feasible *if the execution of each task of the system's applications and their data dependencies can be correctly carried out by proper operating system resources.*

A task execution can be successfully carried out on a resource if the resource has enough capacity to execute the task and is operating properly, i.e., it is not defect. Data-dependencies can be implemented if there exists a set of properly operating resources that allows to pass the data correctly from the sending to the receiving resource. In this definition, failures that happen at task level like, e.g., soft errors or errors in the task itself, are assumed to be handled at task level using, e.g., task re-execution, cf. [16].

4 Reliability Analysis

In this section, the symbolic reliability analysis is presented. The calculation and representation of the so called *structure function* φ is explained in three steps: First, the requirements for a feasible system are introduced. Afterwards, the representation of the dynamic routing within the structure function is presented. In a final step, the given constraints are integrated directly into φ. Moreover, the evaluation of φ to quantify the systems reliability is explained.

4.1 The Structure Function φ

To model the systems behavior under the influence of failures, the structure function $\varphi : \{0,1\}^{|V_\mathsf{a}|} \rightarrow \{0,1\}$ with the Boolean vector $\boldsymbol{V_\mathsf{a}} = (\boldsymbol{r_1}, \ldots, \boldsymbol{r_{|V_\mathsf{a}|}})$ is calculated, cf. [14]. At this, for each allocated resource $r \in V_\mathsf{a}$, a binary variable $\boldsymbol{r}$ is introduced with $\boldsymbol{r} = 1$ indicating a proper operation and $\boldsymbol{r} = 0$ a resource failure, respectively. This Boolean function indicates a proper operating system, i.e., a feasible system by evaluating to $\varphi = 1$ and a system failure by evaluating to $\varphi = 0$, respectively. For a given system specification, this function can be calculated as follows:

$$\varphi(\boldsymbol{V_\mathsf{a}}) = \exists \boldsymbol{I} : \psi(\boldsymbol{V_\mathsf{a}}, \boldsymbol{I}) \tag{1}$$

Whether a system is feasible is highly dependent on which instance of each task is activated. Thus, the *extended structure function* ψ that includes both the resources and the available task instances, is calculated first. At this, $\boldsymbol{I} = (\boldsymbol{i_1}, \ldots, \boldsymbol{i_{|I|}})$ is a vector of Boolean variables encoding a task instance being activated. Applying the *exists-operator* $\exists$ to ψ allows to eliminate the I variables by asking if there exists at least one set of task instances that ensures a feasible system.

The requirements for a feasible system are encoded in ψ:

$$\psi(\boldsymbol{V_\mathsf{a}}, \boldsymbol{I}) = \bigwedge_{t \in V_\mathsf{t}} \left[\bigvee_{i=(t,r) \in I} \boldsymbol{i} \right] \wedge \tag{2a}$$

$$\bigwedge_{i=(t,r) \in I} \boldsymbol{i} \rightarrow \boldsymbol{r} \wedge \tag{2b}$$

$$\bigwedge_{(t,\widetilde{t}) \in E_\mathsf{t}} \bigwedge_{\substack{i=(t,r),\\ \widetilde{i}=(\widetilde{t},\widetilde{r}) \in I}} \boldsymbol{i} \wedge \widetilde{\boldsymbol{i}} \rightarrow R_{r,\widetilde{r}}(\boldsymbol{V_\mathsf{a}}) \wedge \tag{2c}$$

$$\bigwedge_{r \in V_\mathsf{a}} C_r(\boldsymbol{I}) \tag{2d}$$

At least one active instance of each task $t \in V_\mathsf{t}$ is needed to allow each application to work properly. This is ensured by Term (2a). Term (2b) states that an activated task instance implies a proper operating resource. Furthermore, if two instances of data dependent tasks are activated, they must be able to communicate and a correct routing has to be possible, cf. Term (2c). At this, the function $R_{r,\widetilde{r}}(\boldsymbol{V_\mathsf{a}})$ encodes possible routings and, thus, enables to decide whether two data dependent task instances are able to communicate. The calculation of this function is presented in Sec. 4.2. The given constraints that are imposed on the resources are realized by Term (2d) using the function $C_r(\boldsymbol{I})$. The calculation of this function is presented in Sec. 4.3.

4.2 Encoding the Routing

The function $R_{r_s,r_d} : \{0,1\}^{|V_\mathsf{a}|} \rightarrow \{0,1\}$ evaluates to 1 if there exists a *route*, i.e., a loop free path, that implements a communication between the data dependent task instances being executed on resource r_s and r_d by passing data over currently proper

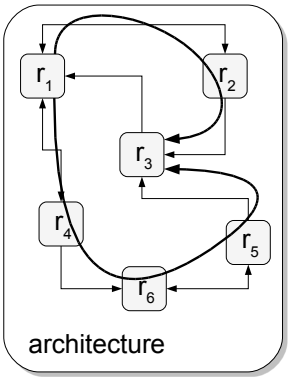

Fig. 4. All possible routes from resource r_1 to r_3 from the example shown in Fig. 2.

operating resources only. Thus, the function evaluates to 0 if there is no route that is able to implement the data dependency. In the following, a fixed-point iteration approach formally introduces the calculation of $R_{r_s,r_d}(\boldsymbol{V}_\mathsf{a})$. Additionally, a symbolic version of this fixed-point iteration is presented that enables an efficient determination of the desired Boolean function.

Fixed-Point Iteration A route from a sender resource r_s to a destination resource r_d is carried out by passing the data from one resource to another using the point-to-point connections between the resources.[1] Passing data from one resource to another is called taking a *hop*, thus, one route consists of a ordered sequence of hops starting from the sender and ending at the destination without visiting a resource more than once. Formally, taking a hop between two resources r and $\widetilde{r}$ is possible if

$$\exists e = (r, \widetilde{r}) \in E_\mathsf{a} \tag{3}$$

That means, a hop can only be taken if the resources are able to communicate using a point-to-point connection. The possible routes from the example depicted in Fig. 2 are shown in Fig. 4.

The determination of $R_{r_s,r_d}(\boldsymbol{V}_\mathsf{a})$ has to take all possible routes into account. In the following, the determination of this Boolean function is done by a fixed-point iteration.

The single state of the fixed-point iteration are in the set $\mathcal{S}$ with

$$\mathcal{S} = V_\mathsf{a} \times 2^{V_\mathsf{a}}. \tag{4}$$

One state $(r, R) \in \mathcal{S}$ consists of a reached resource $r \in V_\mathsf{a}$ and the set of resources $R \subseteq V_\mathsf{a}$ in form of predecessor resources that have been passed starting from the sender

[1] Buses are modeled as a single resource with many point-to-point connections to nodes that are attached to the bus.

resource to reach the current resource r. The function $\delta : \mathcal{S} \rightarrow 2^{\mathcal{S}}$ determines for a given state (r, R) the set of reachable states:

$$\delta((r, R)) = \{(r', R \cup \{r'\}) \mid \text{with } (r, r') \in E_{\mathsf{a}}\} \tag{5}$$

By using δ, for a given set of states $S \subseteq \mathcal{S}$ the successor states are calculated by the successor function $SUCC : 2^{\mathcal{S}} \rightarrow 2^{\mathcal{S}}$:

$$SUCC(S) = \{(r', R') \mid \exists (r, R) \in S : (r', R') \in \delta((r, R))\} \tag{6}$$

Thus, the following function defines a fixed-point iteration that searches all reachable resources with the given set of resources that are required to ensure a route:

$$S_{j+1} = S_j \cup SUCC(S_j) \tag{7}$$

The iteration stops in the iteration k if

$$S_{k+1} = S_k \tag{8}$$

and the fixed-point is reached.

For a given sender resource r_s and destination resource r_d the initial state of the iteration is

$$S_0 = \{(r_s, \{r_s\})\} \tag{9}$$

and the desired states for the fixed-point S_k are those where the current resource equals the destination:

$$\widetilde{S_k} = \{(r_d, R) | (r_d, R) \in S_k\} \tag{10}$$

With the calculated set $\widetilde{S_k}$ the Boolean function that indicates whether a communication between r_s and r_d is possible is as follows:

$$R_{r_s, r_d}(\boldsymbol{V_{\mathsf{a}}}) = \bigvee_{(r,R) \in \widetilde{S_k}} \bigwedge_{\widetilde{r} \in R} \widetilde{\boldsymbol{r}} \tag{11}$$

However, the complexity of this iteration equals the enumeration of all *simple paths* that is known to be #P-complete [17].

Symbolic Approach The basis of the determination of $R_{r_s, r_d}(\boldsymbol{V_{\mathsf{a}}})$ is #P-complete and, thus, of a high computational complexity. In the following, the set-based fixed-point iteration is done by a symbolic approach using *Binary Decision Diagrams* (BDDs). From the experiences of *Model Checking* a symbolic encoding [18] speeds up a fixed-point iteration by some orders of magnitude.

Preliminary, for each $r \in V_{\mathsf{a}}$ a distinct Boolean function b_r is defined as

$$b_r : X \rightarrow \{0, 1\} \text{ with } X = \{0, 1\}^{\lceil ld\ |V_{\mathsf{a}}| \rceil}, \tag{12}$$

with $x \in X$ being of the form

$$x = \{\boldsymbol{x_0}, \ldots, \boldsymbol{x}_{\lceil \boldsymbol{ld}\ |\boldsymbol{V_{\mathsf{a}}}| \rceil}\}. \tag{13}$$

Hereby, for two resources $r, \widetilde{r} \in V_\mathsf{a}$ it holds

$$b_r(x) \neq 0 \tag{14a}$$

$$b_r(x) \wedge b_{\widetilde{r}}(x) = 0 \tag{14b}$$

Thus, the function $b_r(x)$ maps a resource r to a specific binary representation by evaluating to 1 if x is the binary representation of r and evaluating to 0 if x is not the binary representation of r, respectively.

Correspondingly to Eq. (4) a single state or a set of states can be defined in the binary representation and, thus, as a BDD:

$$S : X \times \{0,1\}^{V_\mathsf{a}} \rightarrow \{0,1\} \tag{15}$$

Correspondingly to Eq. (5) the function $\delta : X \times X \times \{0,1\}^{|V_\mathsf{a}|} \rightarrow \{0,1\}$ encodes whether taking a hop is possible represented as a BDD:

$$\delta(x, x', \boldsymbol{V_\mathsf{a}}) = \bigvee_{e=(r,\widetilde{r}) \in E_\mathsf{a}} b_r(x) \wedge b_{\widetilde{r}}(x') \wedge \widetilde{\boldsymbol{r}} \tag{16}$$

This function evaluates to 1 if the requirements stated in Eq. (3) are fulfilled. Otherwise, δ evaluates to 0, respectively. The required paths in the form of predecessor resources stated in Eq. (4) is incorporated through the $\boldsymbol{V_\mathsf{a}}$ variables. This is important since these variables encode the path that is needed for the fixed-point iteration and allow defect resources[2] to falsify the possibility of taking hops at the same time.

Correspondingly to Eq. (6) the successor function is defined as

$$SUCC(S(x, \boldsymbol{V_\mathsf{a}})) = \exists x' : S(x', \boldsymbol{V_\mathsf{a}}) \wedge \delta(x', x, \boldsymbol{V_\mathsf{a}}). \tag{17}$$

Thus, the fixed-point iteration from Eq. (7) is carried out by

$$S_{j+1}(x, \boldsymbol{V_\mathsf{a}}) = S_j(x, \boldsymbol{V_\mathsf{a}}) \vee SUCC(S_j(x, \boldsymbol{V_\mathsf{a}})). \tag{18}$$

The iteration stops and the fixed-point is reached if the BDDs for two subsequent iterations are equal, cf. Eq. (8).

For the sender resource r_s and the destination resource r_d the initial state is defined as a BDD correspondingly to Eq. (9):

$$S_0(x, \boldsymbol{V_\mathsf{a}}) = b_{r_s}(x) \wedge \boldsymbol{r_s} \tag{19}$$

For the fixed-point $S_k(x, \boldsymbol{V_\mathsf{a}})$ the restricted states to the destination resource are determined correspondingly to Eq. (10) as follows:

$$\widetilde{S_k}(x, \boldsymbol{V_\mathsf{a}}) = S_k(x, \boldsymbol{V_\mathsf{a}}) \wedge b_{r_d}(x) \tag{20}$$

Thus, correspondingly to Eq. (11) the desired Boolean function or BDD, respectively, is determined as follows:

$$R_{r_s,r_d}(\boldsymbol{V_\mathsf{a}}) = \exists x : \widetilde{S_k}(x, \boldsymbol{V_\mathsf{a}}) \tag{21}$$

The resulting BDD for the example in Fig. 4 is shown in Fig. 5.

[2] Link failures can be seamlessly introduced by adding binary Variables $\boldsymbol{E}$ that encode a proper operation of the communication links E_a.

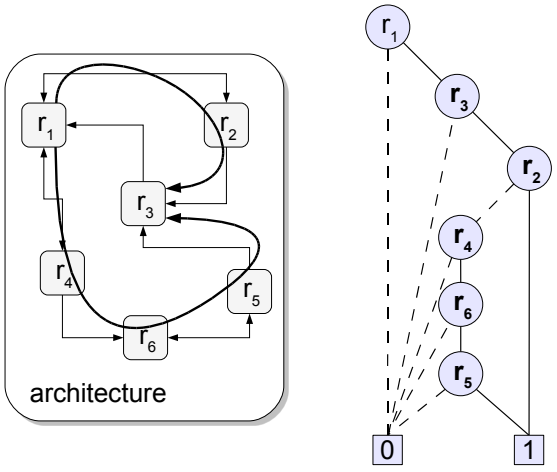

Fig. 5. A BDD encoding the routing possibilities from resource r_1 to r_3 from the example shown in Fig. 2. Edges represent the corresponding variable to be 1 while the dashed edges depict the variable to be 0, respectively.

4.3 Incorporating Constraints

Typical constraints for resources in self-x networked embedded systems are the maximum computational load or the maximum memory capacity of a specific resource. Since the calculation of these objectives can be approximated using linear functions, they can be expressed as linear constraints of the form

$$a^T x \circ b \tag{22}$$

with $a \in \mathbb{Z}^n$, $b \in \mathbb{Z}$ and $\circ \in \{<, \leq, =, \geq, >\}$. A typical constraint for, e.g., the maximum computational load of a resource r has the following form:

$$\sum_{i=(t,r)\in I} l_i \cdot \boldsymbol{i} \leq L_r \tag{23}$$

At this, l_i denotes the computational load arising from activating task t on resource r while the maximum computational load of resource r is denoted as L_r. By incorporating these constraints into ψ, system states that violate a constraint are excluded from the set of feasible system states.

An encoding algorithm for linear constraints as Binary Decision Diagrams has been presented in [19]. Using this algorithm, the function

$$C_r : \{0,1\}^{|I|} \rightarrow \{0,1\} \tag{24}$$

can be realized. The function C_r evaluates to 1 if the task instances that are executed on r do not violate the given constraints and evaluates to 0 if at least one of the encoded constraints is violated, respectively. As an example, resource r_3 from Fig. 2 with an

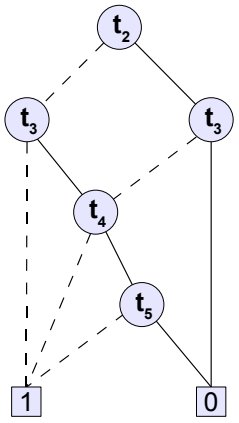

Fig. 6. A BDD encoding the computational load constraint of resource r_3 shown in Fig. 2. Edges represent the corresponding variable to be 1 while the dashed edges depict the variable to be 0, respectively.

L_{r_3} of 5 is to encode. The computational demand of the tasks that are bound to r_3 are $l_{t_2} = l_{t_3} = 3$ and $l_{t_4} = l_{t_5} = 2$. Thus, the constraint can be written as

$$3\boldsymbol{i_{r_2}} + 3\boldsymbol{i_{r_3}} + 2\boldsymbol{i_{r_4}} + 2\boldsymbol{i_{r_5}} \leq 5$$

The resulting BDD for C_{r_3} that is constructed using the Algorithm presented in [19] is shown in Fig. 6.

4.4 Evaluating φ

In the following, we describe how to quantify the reliability of the self-x networked embedded system based on the determined structure function φ. Since the reliability R_r of each resource $r \in V_\mathsf{a}$ is given in the system model, e.g., by distribution functions like an *exponential distribution* or a *Weibull distribution*, the reliability of the system at time t can can be calculated through using a modified *Shannon-decomposition* [20] on the BDD representing φ:

$$R(t) = \varphi = R_r(t) \cdot \varphi|_{\boldsymbol{r}=1} \; + (1 - R_r(t)) \cdot \varphi|_{\boldsymbol{r}=0} \tag{25}$$

If a *Mission Time* (MT) of the system is given, this decomposition directly quantifies the reliability $R(MT)$ of the system. In our experimental results, the *Mean Time To Failure* (MTTF) $= \int_0^\infty R(t)dt$ is used as the measure of reliability and is determined by a numerical integration of Equation (25). MTTF denotes the expected value for the failure-free time of the system.

Table 1. Comparison of theoretical upper bound for the MTTF with the ReCoNets self-healing system.

testcases	symbolic analysis	ReCoNets LB			ReCoNets BCC		
	MTTF_{max}	MTTF	deviation	e	MTTF	deviation	e
small	54.92	50.34	40.04	0.917	51.21	39.44	0.932
medium	66.91	57.20	46.28	0.854	60.25	43.21	0.900
large	176.12	137.18	127.71	0.779	144.64	138.24	0.821

5 Experimental Results

In this section, the results of applying our proposed analysis approach to two different examples are presented. First, examples of an available self-healing technique for networked embedded systems known as *ReCoNets* [4] are analyzed. Afterwards, a networked embedded system with a specification inspired by state-of-the-art ECU networks from the automotive area are used to show the applicability of the proposed approach.

ReCoNets The self-healing technique called ReCoNets implements the self-healing property using a one replication strategy. At this, for each active task, a replica task is created by copying the byte code of the tasks to another resource. Of course, a replica can only be placed at resources that are allowed by the given mappings. There are two strategies available: The *load balancing* (LB) strategy places replicas under load balancing aspects. The strategy that is more focused on lifetime maximization called *BCC* determines *bi-connected components* that can, in case of a failure, lead to a partitioning of the networked embedded system. With this knowledge, the BCC strategy tries to place replicas such that a partitioning does not prevent data dependent tasks from a correct communication. Hence, strategies based on using only one replica can, in general, not achieve the maximum reliability, but reduce the amount of memory needed in every resource since task replicas are created dynamically. With this example, the possibility of quantifying the effectiveness of a self-healing technique using our proposed approach is shown. For this reason, a simple measure for the effectiveness e is used:

$$e = \frac{\text{MTTF}}{\text{MTTF}_{\text{max}}} \tag{26}$$

Table 1 shows the results of the proposed reliability analysis for networked embedded systems where a simulation of the ReCoNets techniques is available. In these testcases, an exponential distribution function was used to model the resource reliability. The size of the networked embedded systems was varied between 10 and 30 resources, each having the same number of tasks. Due to the high vertex degree, these examples can be considered to be complex examples for the analysis. For the small examples, both ReCoNets techniques nearly reach the upper bound of the achievable MTTF. However, the high standard deviation shows that both techniques can also make suboptimal decisions for the replica placement, leading to very early system failures. This is often the result of placing task and corresponding replica in a small subnet that can be isolated

Table 2. Time consumption of a single analysis run for different ECU networks.

testcases	specification			time consumption
	#ECUs	#Tasks	#Buses	[s]
small	30	30	2	1.16
medium	50	50	3	3.63
large	70	70	4	6.69

from the network by a single link or resource failure. For larger networks, the effectiveness of both ReCoNets techniques decreases, but can still be considered as good. In all testruns carried out, the time consumption of the proposed analysis algorithm and the ReCoNets simulation were nearly equal. Morover, the time consumption of the proposed algorithm is small enough to be applied in design space exploration approaches.

Especially interesting is the relatively small difference in the effectiveness of both ReCoNets methodologies with regards to the known upper bound. In [4], the relative difference between the LB and BCC approach seemed significant. With regards to the upper bound, the designer may choose the load balancing approach as well, since this approach is less than 5% worse than the BCC technique and offers a better load balancing of the resources.

ECU Network In this section, our proposed methodology is applied to artificial examples inspired by typical *Electronic Control Unit* (ECU) networks from the automotive domain to show its time consumption. In these examples, the resources have a relatively low vertex degree since they are connected via buses. These buses are typically arranged in a star topology. The large example with 70 ECUs corresponds to recent real world automotive networks in premium class automobiles. The experiments were carried out on an Intel Pentium 4 3.20GHz machine with 1GB RAM.

Table 2 shows the results for three different ECU networks. The time consumption per analysis of 1.16 to 6.69 seconds per analysis run shows that the proposed approach is applicable for these kind of networks. Moreover, the time consumption is small enough to be applied in design space exploration approaches where many different network layouts have to be analyzed in order to find the optimum. However, at a certain complexity, the memory consumption of the calculated BDD exceeds the computers main memory and, thus, makes an analysis impossible. In our testcases, this problem arises at networks with about 90 ECUs and 90 tasks.

6 Conclusion

In this paper, a reliability analysis for self-healing networked embedded systems has been proposed. This technique allows to determine an upper bound for the MTTF that can be achieved by self-healing techniques that rely on self-reconfiguration and self-routing. The technique uses a fast and memory-aware symbolic representation and respects given constraints of the system like, e.g., the maximum computational load. Given the proposed technique, the effectiveness of different self-healing techniques for networked embedded systems can be quantified. Moreover, an effective dimensioning

of the system in the design phase is enabled. In the future, the proposed technique will be extended to respect a possible maintenance of resources or, more accurately, of subnets of the networked embedded system.

References

1. Dai, Y.S.: Autonomic computing and reliability improvement. In: Proceedings of ISORC '05. (2005) 204–206
2. Koch, D., Streichert, T., Dittrich, S., Strengert, C., Haubelt, C., Teich, J.: An operating system infrastructure for fault-tolerant reconfigurable networks. In: Proceedings of ARCS '06. (2006) 202–216
3. Garlan, D., Schmerl, B.: Model-based adaptation for self-healing systems. In: Proceedings of WOSS '02. (2002) 27–32
4. Streichert, T., Glaß, M., Wanka, R., Haubelt, C., Teich, J.: Topology-aware replica placement in fault-tolerant embedded networks. In: Proceedings of ARCS '08. (2008) 23–37
5. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Trans. on Comp. **35**(8) (1986) 677–691
6. Cankay, H.C., Nair, V.S.S.: Reliability and availability evaluation of self-healing sonet mesh networks. In: Proceedings of GLOBECOMM '97. (1997) 252–256
7. Cankay, H.C., Nair, V.S.S.: Accelerated reliability analysis for self-healing sonet networks. SIGCOMM Comput. Commun. Rev. **28**(4) (1998) 268–277
8. Kawamura, R., Sato, K., Tokizawa, I.: Self-healing atm networks based on virtual path concept. IEEE Journal on Selected Areas in Communications **12**(1) (1994) 120–127
9. Lee, J.: Reliability models of a class of self-healing rings. Microelectronics and Reliability **37**(8) (1997) 1179–1183
10. Politof, T., Satyanarayana, A.: Efficient algorithms for reliability analysis of planar networks - a survey. IEEE Trans. on Reliability **35**(3) (1986) 252–259
11. Ortega, C., Tyrrell, A.: Reliability analysis in self-repairing embryonic systems. In: Proceedings of EH '99. (1999) 120–128
12. Dressler, F., Dietrich, I.: Lifetime analysis in heterogenous sensor networks. In: Proceedings of DSD '06. (2006) 606–616
13. Elliot, C., Heile, B.: Self-organizing, self-healing wireless networks. In: Proceedings of Aerospace Conference '00. (2000) 149–156
14. Glaß, M., Lukasiewycz, M., Streichert, T., Haubelt, C., Teich, J.: Reliability-Aware System Synthesis. In: Proceedings of DATE '07. (2007) 409–414
15. Streichert, T., Glaß, M., Haubelt, C., Teich, J.: Design space exploration of reliable networked embedded systems. Journ. on Systems Architecture **53**(10) (2007) 751–763
16. Izosimov, V., Pop, P., Eles, P., Peng, Z.: Synthesis of fault-tolerant schedules with transparency/performance trade-offs for distributed embedded systems. In: Proceedings of DAC '04. (2004) 550–555
17. Valiant, L.G.: The complexity of enumeration and reliability problems. SIAM Journal on Computing **8** (1979) 410–421
18. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang, L.J.: Symbolic model checking: 1020 states and beyond. Inf. Comput. **98**(2) (1992) 142–170
19. Eén, N., Sörensson, N.: Translating Pseudo-Boolean Constraints into SAT. Journal on Satisfiability, Boolean Moelding and Computation **2** (2006) 1–25
20. Rauzy, A.: New Algorithms for Fault Tree Analysis. Reliability Eng. and System Safety **40** (1993) 202–211