Project Number: FP7-611404

D5.2.1 – System Reliability Estimation Models (preliminary)

Authors[1]

A. Savino (POLITO), S. Di Carlo (POLITO), A. Vallero (POLITO), G. Politano (POLITO)

Version 1.0 – 27/04/2015

Lead contractor: Politecnico di Torino
Contact person: Alessandro Savino Control and Computer Engineering Dep. Politecnico di Torino, C.so Duca degli Abruzzi, 24 I-10129 Torino TO Italy E-mail: alessandro.savino@polito.it
Involved Partners[2]**:** POLITO, UoA, CNRS, UPC, THALES, YOGITECH
Work package: WP5
Affected tasks: T5.3

Nature of deliverable[3]	R	P	D	O
Dissemination level[4]	PU	PP	RE	CO

[1] *Authors listed here only identify persons that contributed to the writing of the document.*

[2] *List of partners that contributed to the activities described in this deliverable.*

[3] *R: Report, P: Prototype, D: Demonstrator, O: Other*

[4] ***PU**: public, **PP:** Restricted to other programme participants (including the commission services), **RE** Restricted to a group specified by the consortium (including the Commission services), **CO** Confidential, only for members of the consortium (Including the Commission services)*

COPYRIGHT

INDEX

Scope of the document

This document is the main outcome of task T5.3 "**System level statistical reliability estimation methodology**", elaborated in the Description of Work (DoW) of the CLERECO project under Work Package 5 (WP5).

Figure 1 depicts graphically the goal of this deliverable, its main results, the inputs it uses and the outputs it provides (including which WPs will use its outputs).

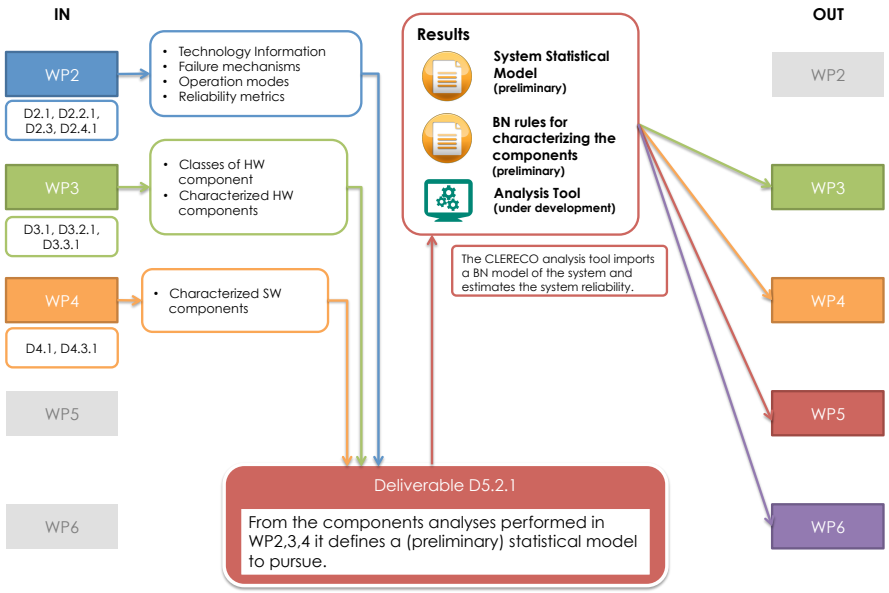


Figure 1: The Inputs and the Outputs of this Deliverable

The main goal and outcome of D5.2.1 is to develop statistical estimation model(s) able to analyze a full system stack from technology up to the application. The model resorts to the characterization effort provided by all technical WPs (i.e., WP2, WP3 and W4), which aims at assessing the reliability at the technological, hardware and software layers of the system. Each WP has already evaluated reliability aspects of *technologies* and *components* and a library of those is going to be built. By describing a full system in terms of the characterized components, the statistical model described in this deliverable enables reliability evaluation at the system level.

The document is organized in the following sections:

- **Introduction**. This section shortly overviews background research on statistical models to estimate reliability of a full system and identifies the target model that will be explored in the CLERECO model
- **Bayesian Networks Basic Concepts**. This section describes all aspects of the statistical model based on the Bayesian Network theory developed in CLERECO.
- **System Modeling with Bayesian Networks**. It includes all modeling details to properly deal with the description and reliability evaluation of a full system.
- **Conclusions**. In this final section, we summarize the work done for the deliverable and we set a roadmap to expand the model and improve actual results.

1. Introduction

System-level reliability estimation is the main and ultimate goal for the CLERECO project.

A system is a collection of components, subsystems and/or assemblies arranged to a specific design in order to achieve desired functions with acceptable performance and reliability. The types of components, their quantities, their qualities and the manner in which they are arranged within the system have a direct effect on the system's reliability. The relationship between a system and its components is often misunderstood or oversimplified leading to wrong conclusions.

Statistical models for reliability evaluation have been already studied in life data analysis and accelerated life testing data analysis [19], as well as other testing activities of several systems where one of the primary objectives is to obtain a life distribution that describes the times-to-failure of a component, subassembly, assembly or system [8]. This analysis is usually based on time-to-failure data of the component, either under use conditions or from accelerated life tests. In system reliability analysis, one constructs a "System" model from these component models. In other words, in system reliability analysis we are concerned with the construction of a model (life distribution) that represents the times-to-failure of the entire system based on the life distributions of the components, subassemblies and/or assemblies ("black boxes") from which it is composed. To accomplish that, the relationships between components are considered and analyzed to infer the overall system reliability, maintainability and/or availability.

Statistical system reliability analysis and modeling of electronic systems is gaining ever-increasing interest given the electronic systems trend to become more complex, which prevents the realization and use in a reasonable time of simulation based models [33][36][37][40]. Complex fault-injection campaigns are rapidly becoming unfeasible on real system posing a real threat for the correct characterization of next generation systems.

However, when it comes to early reliability evaluation of electronic systems as in CLERECO, several modeling problems must be faced. Time-to-failure data of system components, especially for new technologies and architectures are not available and classical RTL or SystemC simulation models even for single components may be missing or already too complex to be analyzed (e.g., in the case of complex microprocessors) in a reasonable time manner.

Within work packages WP2, WP3 and WP4 CLERECO researchers tried to cover this gap developing a set of methods to characterize technologies, hardware and software components early in the design phase even when full models of these components are not available. In this document we accomplish the task of developing a statistical system model able to incorporate all these component level information into a high-level reliability model of the system.

Instruments able to estimate the reliability of a complete system starting from the very early stage of the design cycle are important to give designers an insight into comparative reliability analysis of different components and architectures. They allow a wide range of analyses other than a simple cause-effect analysis of the fault injection campaign, including inferences, causalities analysis, effect correlations, etc. [18][19]. Such analyses have the potential to identify those critical components that may require higher design effort to incorporate error tolerance [3]. Fault-tolerance commonly produces bigger systems (due to hardware redundancy) and slower software (due to error-tolerant software techniques). It must therefore be carefully used and traded-off starting from the very early design phases.

1.1. System models for reliability analysis

System level behavioral models, such as **SystemC** models, have been one of the first attempts to move reliability analysis at a system level (the reader may refer to [7] and its referenced papers for a deeper overview). These models are still simulation models that enable to simulate the activity of the system and to perform fault injection campaigns. However the complexity of the simulation of these systems is still high compared to the complexity of a full designed system. Moreover early and exploratory reliability analysis using these models is unfeasible. Exploring the effect of different design choices on the architecture requires complex changes in the model that, in turn, require a significant design effort. Finally, while these models are effective in modeling the hardware architecture layer of the system, they still lack enough power to tackle with the complexity of the full software stack.

Fault Tree Analysis (FTA) is a very common statistical analysis [28][29][30]. FTA is a top down, deductive failure analysis in which an undesired state of a system is analyzed using Boolean logic to combine a series of lower-level events. Mainly used in the fields of safety engineering and reliability engineering to understand how systems can fail, it is based on modeling the system via Fault Trees Diagrams (FTDs), which are the most popular method for reliability analysis [27]. Figure 2 reports an example of a FTD. It is basically a logic block diagram that displays the state of a system (top event) in terms of the states of its components (basic events). It uses a graphic model of the pathways within a system that can lead to a foreseeable, undesirable loss event (or a failure). The pathways interconnect contributory events and conditions, using standard logic symbols (AND, OR etc.). This type of FTD is called *static* FTD. Two common approaches are used in order to solve a static fault tree: Binary Decision Diagram (BDD) and Cut sets [29][30]. Both approaches are complex and time consuming if a continuous time reliability curve is aimed, particularly for large systems [33]. Thus, hardware-based acceleration techniques are proposed to cope with the problem [32][33].

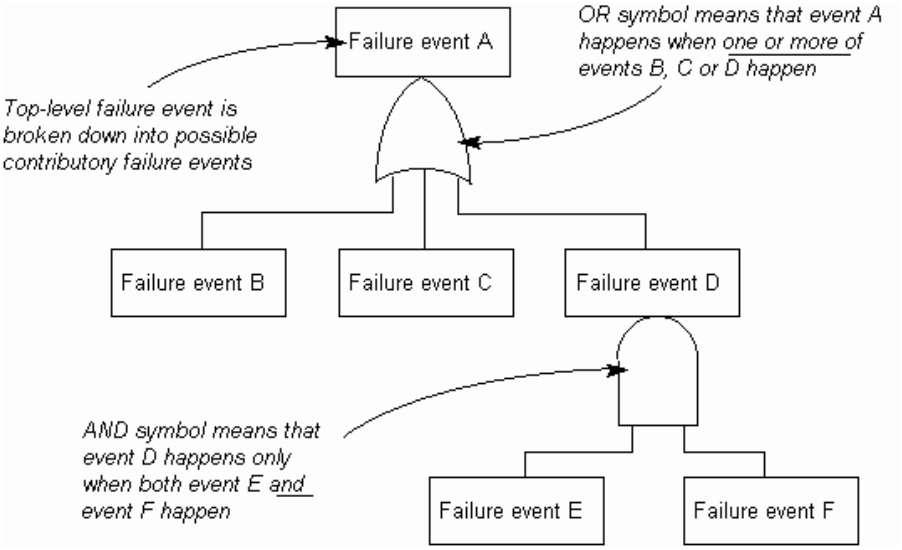


Figure 2: Example of fault-tree model

Another very similar technique that is usually employed to statistically investigate the reliability of a system, are the **Reliability Block Diagrams** (RBDs). Figure 3 reports an example of RBD. The most fundamental difference between FTDs and RBDs is that in an RBD works in the "success space", and thus looks at system successes combinations, while a FTD works in the "failure space" and looks at system failure combinations. Traditionally, fault trees have been used to

access fixed probabilities (i.e., each event that comprises the tree has a fixed probability of occurring) while RBDs may include time-varying distributions for the success (reliability equation) and other properties, such as repair/restoration distributions.

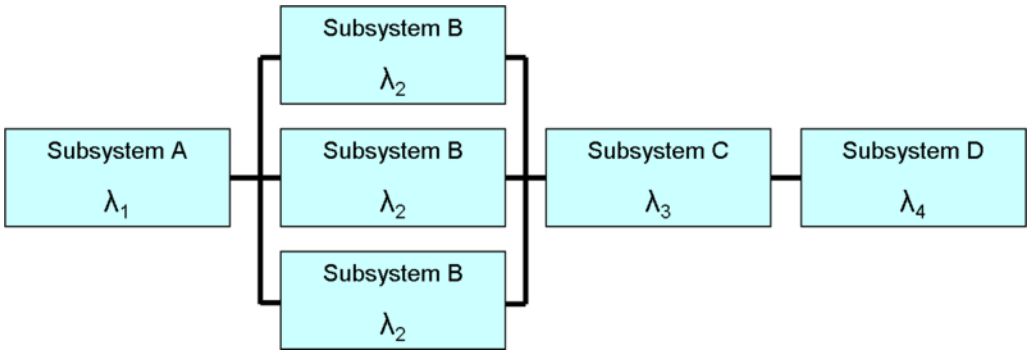


Figure 3: Example of reliability block diagram

Both FTD and RBD do not provide any element or capability to model reliability interactions among components or subsystems, or to represent system reliability configuration changes (dynamics), such as: load-sharing, standby redundancy, interferences, dependencies, common cause failures, and so on [34]. Moreover, they do not allow enhancing the reliability evaluation in terms of the kind of effects related to the reliability issue, such as wrong execution timing, data corruption, etc., or the definition of different metrics but Failures-In-Time (FIT), or Mean Time Between Failures (MTBF) as output of the evaluation.

Markov chains represent a significant alternative to the FTD or RBD analyses [24][25][26]. A Markov chain (see Figure 4) is a random process that undergoes transitions from one state to another on a state space. The probability distribution of the next state only depends on the current state and not on the sequence of events that lead to that state. This is called the Markov property. Markov chains have several modeling issues when applied to reliability analysis. First, the whole system needs to be modeled as a set of states, which have to differ from single components. It means to generate a model that heavily relies on the workload more than on its components. The workload is in fact the major responsible of the transition of the system from one state to the next one. Second, the Markov property (i.e., the system is memory-less) limits the possibility to fully describe the propagation of errors among layers. Moreover, the Markov property prevents the possibility to backward analysis longer than one state. Backward analysis is one of the main instruments in system reliability analysis that enables to locate the sources of failure in a system. Preventing its execution strongly reduces the capability of the model. Finally the model may suffer from state space explosion.

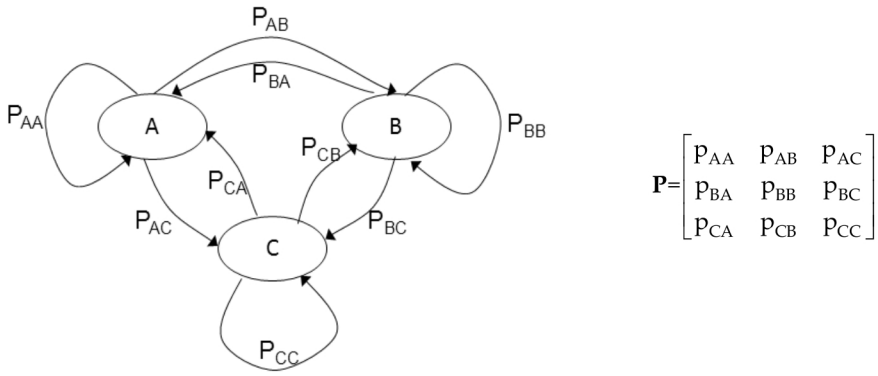


Figure 4: Example of Markov-Chain model

Only few publications consider the application of **Bayesian Networks** (BN) to model system reliability in hardware devices [39], [40]. BNs (see Figure 5) are a statistical model to represent multivariate statistical distribution functions. They can model relationships among random variables and their respective probability density functions by means of conditional probability functions. They have been successfully employed for reliability estimation in some application fields [37]. In the software engineering domain, they have been employed to model software reliability in the distributed domain [38]. They main advantage with respect to the previous techniques is the degree of freedom they have to define input causes of failure: there is no limitation in the number of variables to be defined that carry a manifestation of fault meaning. Moreover, they can be common among other variables. Another very interesting aspect of the BN is that variables can be either discrete (e.g., Boolean or multi-states) or continuous, allowing the reliability analysis to cover different metrics apart from FIT or MTBF. Eventually, BNs require studying the system with locality to populate the network: to connect two variables (in alternative, two nodes of a system) only their relationship must be defined. If a third variable is linked to the network but connected to one of the two variables only, this single relationship will be provided by a conditional probability.

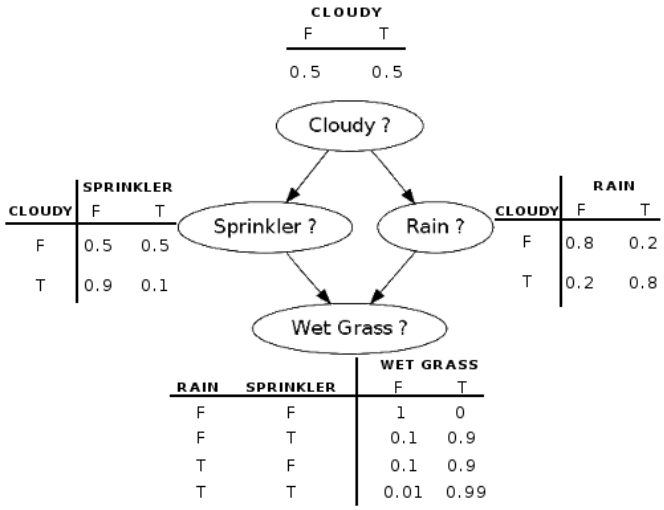


Figure 5: Bayesian model example

Table 1 provides a global comparison of all considered models with respect to main requirements needed for efficient system level early reliability estimation model of complex systems.

Table 1: Comparison Between Statistical Approaches

Characteristics	Statistical Approaches			
	Fault Tree Analysis	Reliability Block Diagrams	Markov Chains	Bayesian Networks
Top Down Analysis	YES	YES	YES	YES
Bottom Up Analysis	NO	NO	LIMITED	YES
Full propagation of events	NO	NO	NO	YES
Multiple Output	NO	NO	NO	YES
Continuous Values	YES	YES	YES	YES
Cycles Definition	NO	YES	YES	YES
Dynamic Modeling	NO	NO	YES	YES
Components as Single element of the model	YES	YES	NO	YES

From this analysis it is clear that among the considered models, BN represent the most attractive solution able to build the CLERCO reliability estimation model.

In this deliverable we propose a first model to address the reliability estimation of a full system stack by means of BN modeling and solving. The document will guide through all concepts needed for a proper description and all the information gathered from the different WPs in the CLERECO project.

2. Bayesian Networks Basic Concepts

2.1. Building a Bayesian Network

Bayesian networks (BNs) are graphical models for reasoning under uncertainty, where the nodes represent variables (discrete or continuous) and arcs represent direct connections among them. These direct connections are often causal connections. More precisely, given a set of random variables, $\mathbf{X} = X_1, \cdots, X_i, \cdots, X_n$, from the domain, the set of directed arcs (or links) connecting pairs of nodes, $X_i \rightarrow X_j$, represents the direct dependencies between variables.

In addition, BNs model the quantitative strength of the connections between variables, allowing probabilistic beliefs about them to be updated automatically as new information becomes available. Assuming discrete variables, the strength of the relationship between variables is quantified by conditional probability distributions associated with each node. The only constraint on the arcs allowed in a BN is that there must not be any directed cycle: you cannot return to a node simply by following directed arcs. Such networks are called **Directed Acyclic Graphs** (DAGs).

There are a number of steps that a knowledge engineer must undertake when building a Bayesian network. At this stage, we will introduce these steps as a sequence; however it is important to remark that in the real world the process may be not so simple.

We will use the following simple hardware diagnosis problem: *A laptop has been suffering from sudden restarts. The technician suspects the main board is going to brake down very*

soon. Nevertheless, he knows for a fact that the same model has been reported to show similar behaviors for an internal fan issue or for hard drive failures. He also knows that other relevant information includes whether the laptop has emitted strange noises (increasing the chances of main board malfunctions or internal fan issue) and the temperature at which the laptop operates. A negative hard drive check (meaning an error detected when checking the hard drive) would indicate either a hard drive failure or a main board malfunction.

2.1.1. Nodes and values

First, the knowledge engineer must identify the variables of interest. This involves answering the question: what are the nodes to represent the system and what values can they take, or what state can they assume? For now we will consider only nodes that take discrete values. The values should be both mutually exclusive and exhaustive, which means that the variables must assume exactly one of these values at a time. Common types of discrete nodes include:

- Boolean/Binary nodes, which represent propositions. They assume binary values, such as true (T) and false (F). In our example, the node *MainBoard* represents the proposition that the laptop has a main board malfunction.
- Ordered values. For example, the node *Temperature* might represent the laptop environment temperature range. It may assume the values {low, high}, where low represents the range 0-26°C and high represents the range 26-40°C.
- Integral values. For example, the node *Age* might represent the laptop age and have possible values from 1 to 10.

Even at this early stage, modeling choices are being made. As instance, an alternative to represent the *Temperature* might be to have it as integral value.

For our example, we will begin with the restricted set of nodes and values shown in Table 2. This choice already limits what can be represented in the network. For instance, there is no representation of other reliability issues, such as a hard drive failure or an internal fan issue. Therefore, the modeled system will not be able to estimate the probability of the laptop suffering one of these failures. Another limitation is lack of differentiation, for example between high or low noise of the motherboard. Note that all nodes have only two values. This keeps the model simple, but in general there is no limit to the number of discrete values.

Table 2: Preliminary choices of nodes and values for the example.

Node Name	Node Short Name	Type	Values
Noise	N	Boolean	{T, F}
Temperature	T	Binary	{L(ow), H(igh)}
MainBoard	M	Boolean	{T, F}
Restarts	R	Boolean	{T, F}
HD_Check	H	Binary	{pos, neg}

2.1.2. Structure

The structure, or topology, of the network should capture qualitative relationships between variables. In particular, two nodes should be connected directly if one affects or causes the other, with the arc indicating the direction of the effect. So, in our example, we might ask what factors affect a laptop's chance of having a main board failure? If the answer is "High temperature and noise", then we should add arcs from Temperature and Noise to MainBoard. Simi-

larly, having a main board failure will affect the changes of laptop's restart and the chances of having a HD check negative result. So we add arcs from MainBoard to Restarts and HD_Check. The resultant structure is shown in Figure 6. It is important to note that this is just one possible structure for the problem.

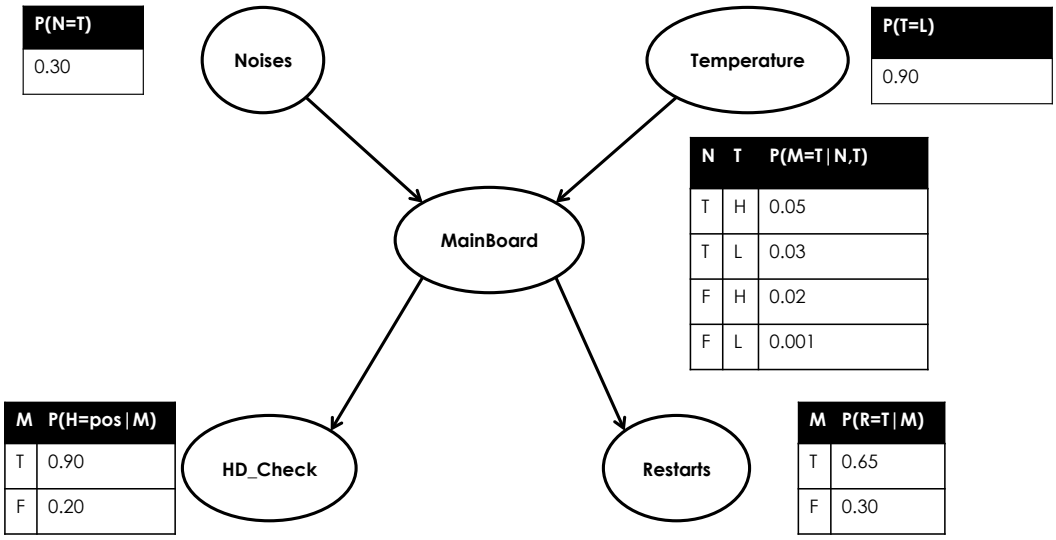


Figure 6: A BN for the Main Board problem.

In talking about network structure it is useful to employ a family metaphor: a node is a parent of a child, if there is an arc from the former to the latter. Extending the metaphor, if there is a directed chain of nodes, one node is an **ancestor** of another if it appears earlier in the chain, whereas a node is a **descendant** of another node if it comes later in the chain. In our example, the MainBoard node has two parents, Temperature and Noises, while Noise is an ancestor of both HD_Check and Restarts. Similarly, HD_Check is a child of MainBoard and descendant of Temperature and Noises. The set of parent nodes of a node X is given by *Parents(X)*.

Another useful terminology commonly used comes from the "tree" analogy (even though Bayesian networks in general are graphs rather than trees): any node without parents is called a **root** node, while any node without children is called a **leaf** node. Any other node (non-leaf and non-root) is called an **intermediate** node. Given a causal understanding of the BN structure, this means that root nodes represent original causes, while leaf nodes represent final effects. In our laptop description example, the causes Temperature and Noise are root nodes, while the effects HD_Check and Restarts are leaf nodes.

By convention, for easier visual examination, networks are usually laid out so that the arcs generally point from top to bottom. This means that the BN "tree" is usually depicted upside down, with roots at the top and leaves at the bottom.

2.1.3. Conditional Probabilities

Once the topology of the BN is specified, the next step is to quantify the relationships between connected nodes: specifying a Conditional Probability Distribution (CPD) for each node does this. As we are only considering discrete variables at this stage, this takes the form of a Conditional Probability Table (CPT).

First, for each node we need to look at all possible combinations of values of its parent nodes. Each such combination is called an **instantiation** of the parent set. For each distinct instantiation of parent node values, we need to specify the probability that the child will take each of its values.

For example, consider the MainBoard node of Figure 6. Its parents are Temperature and Noise and take the possible joint values {< H,T >, < H,F >, < L,T >, < L,F >}. The conditional probability table reported in Figure 6 specifies the probability of MainBoard failure for each of these cases to be: < 0.05, 0.02, 0.03, 0.001 >. Since these are probabilities, and must sum to 1 over all possible states of the MainBoard variable, the probability of no MainBoard malfunction is already implicitly given as one minus the above probabilities in each case; i.e., the probability of no MainBoard malfunction in the four possible parent instantiations is < 0.95, 0.98, 0.97, 0.999 >.

Root nodes also have an associated CPT, although it is degenerate, containing only one row representing its prior probabilities. In our example, the prior for a laptop emitting strange noises is given as 0.3, indicating that 30% of the population that the technician sees emits strange noises, while 90% of the population are exposed to only low temperature in their working environment.

Clearly, if a node has many parents or if the parents can take a large number of values, the CPT can grow very large. The size of the CPT is, in fact, exponential in the number of parents. Thus, for networks wit Boolean states a variable with n parents requires a CPT with 2^{n+1} probabilities. Several solutions are already proposed in literature to cope with the exponential grow of probabilities in CPT. Among them, one seems very promising: the Noisy-MAX approach[42][43][44][45]. The Noisy-Max is a generalization of the interaction of a child node and its direct ancestors in a BN based on three assumptions [45]:

1. The child node and all its ancestors must be variables indicating the degree of presence of an anomaly.

2. Each of the ancestor nodes must represent a cause that can produce the effect (one of the child node state) in absence of other cause.

3. There may be no significant synergies among the causes.

If all assumptions are met, the generalization allows replacing the CPT with a smaller and simpler table, in which each child node state reflects only the single effect of all ancestors' alone. In [44], authors demonstrated the effectiveness of this technique comparing results against a set of known problems described with CPTs and proved how the actual complexity reduction is logarithmic.

2.1.4. The Markov property

In general, modeling with Bayesian networks requires the assumption of the **Markov property**: there are no direct dependencies in the system being modeled that are not already explicitly shown via arcs. In our example, as instance, there is no way for Noise to influence Restarts except by being related to a main board malfunction. Bayesian networks that have the Markov property are also called **Independence-maps** (or, I-maps for short), since all independences suggested by the lack of an arc are real in the system.

Nevertheless, it is important to remember that this is only a construction property that simplifies the construction of the network. As stated before, differently from Markov chains, once the network is built, full back trace analysis is then possible as will be explained later in the document.

2.2. Reasoning with Bayesian Networks

Now that we know how a domain and its uncertainty may be represented in a Bayesian network, we will look at how to use the Bayesian network to reason about the domain. In particular, when we observe the value of some variable, we would like to condition upon the new information. The process of conditioning (also called **probability propagation** or **inference** or **belief updating**) is performed via a "flow of information" through the network. Note that this

information flow is not limited to the directions of the arcs. In our probabilistic system, this becomes the task of computing the posterior probability distribution for a set of **query** nodes, given values for some **evidence** (or **observation**) nodes.

Bayesian networks provide full representations of probability distributions over their variables. That implies that they can be conditioned upon any subset of their variables, supporting any direction of reasoning. For example, one can perform diagnostic reasoning, i.e., reasoning from symptoms to cause, such as when the technician observes the Restarts and then updates his belief about a MainBoard malfunction and whether the laptop emits strange Noise. Note that this reasoning occurs in the opposite direction to the network arcs. Or again, one can perform predictive reasoning, reasoning from new information about causes to new beliefs about effects, following the directions of the network arcs. For example, the laptop's owner may tell his technician that he has listen to some noise; even before any symptoms have been assessed, the technician knows this will increase the chances of the laptop having a main board malfunction. It will also change the technician's expectations that the laptop will exhibit other symptoms, such as restarts or having a negative HD check result.

A further form of reasoning involves reasoning about the mutual causes of a common effect; this has been called inter-causal reasoning. A particular type called explaining away is of some interest. Suppose that there are exactly two possible causes of a particular effect, represented by a v-structure in the BN. This situation occurs in our model of Figure 6 with the causes Temperature and Noise, which have a common effect, MainBoard malfunction. According to the model, these two causes should be independent of each other; that is, a laptop emitting noise (or not) does not change the probability of the laptop to be exposed to some high temperature. Suppose, however, that we learn that a laptop main board is malfunctioning. This will raise our probability for both possible causes of a main board malfunction, increasing the chances both that it emits noises and that he has been exposed to some very high temperature. Suppose then that we discover that its owner has listened to some noise without taking care of them. This new information explains the observed main board malfunction, which in turn lowers the probability that he has been exposed to high temperatures. So, even though the two causes are initially independent, with knowledge of the effect the presence of one explanatory cause renders an alternative cause less likely. In other words, the alternative cause has been explained away.

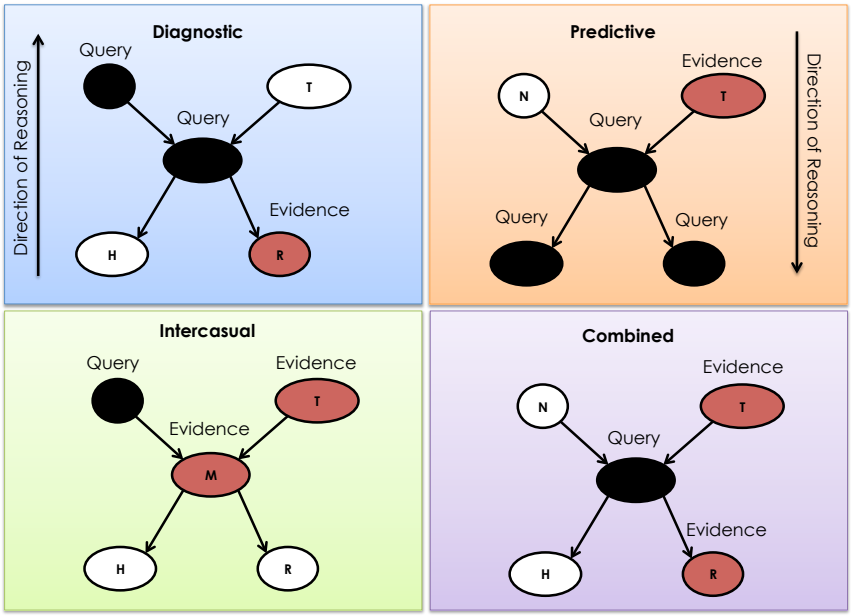


Figure 7: Types of reasoning.

Since any node may be s query node or a evidence node, sometimes the reasoning does not fit neatly into one of the types described above. Indeed, we can combine the above types of reasoning in several ways. Figure 7 shows the different varieties of reasoning using the provided example. Note that the last combination shows the simultaneous use of diagnostic and predictive reasoning.

It should be now clear how Bayesian networks could be used for calculating new beliefs when new information is available. The evidence might be that a node Y has the value y_1 or y_2 (implying that all other values are impossible). Or the evidence might be that Y is not in state y_1 (but may take any of its other values). In fact, the new information might simply be any new probability distribution over Y.

Looking at the example, let us suppose that the technician who has performed the HD check is uncertain. He thinks that the HD check could be right, but is only 80% sure. Such information can be incorporated, and it would correspond to adopting a new posterior distribution for the node in question. In Bayesian networks this is also known as virtual evidence. Since it is handled via likelihood information, it is also known as likelihood evidence.

Eventually, belief updating can be done using a number of exact and approximate inference algorithms. A set of algorithms are widely know to provide it, and choosing among them usually put particular emphasis on how the different algorithms can affect the efficiency of both the knowledge engineering process and the automated reasoning in the deployed system [18][19]. However, since the mathematical modeling is quite complex and its explanation requires a lot of information and since most existing BN software libraries allow the select and use essentially the same algorithms, we do not enter into such details, demanding it to references. Moreover, it also means that, in general, it is quite possible to build and use BNs without knowing the details of the belief updating algorithms.

In the following chapter we focus on resorting to BN towards the full system modeling.

3. System Modeling and Reasoning with Bayesian Networks

System modeling using a BN starts by identifying the nodes that compose the network identifying the main players (components of the system) that influence the system reliability.

In order to work with a realistic use case let us consider the design of an image processing embedded system designed for identification of artifacts in an image. The general characteristics of the system are as follows:

- Hardware architecture: a x86 out-of-order single core microprocessor
- A 1T DRAM main memory
- Both microprocessor and memory are realized in a 14nm FinFET technological process
- The architecture executes the Susan edge algorithm from the MiBench test benches [46] executed on top of the Linux operating system.
- Mass storage memory, and external devices such as I/O devices are not considered to reduce the complexity of the example.

Figure 8 shows a Bayesian network model of the considered system. For each layer of the system stack (i.e., technology layer, HW layer, SW layer), a set of nodes represents players of the reliability problem coming from that layer.

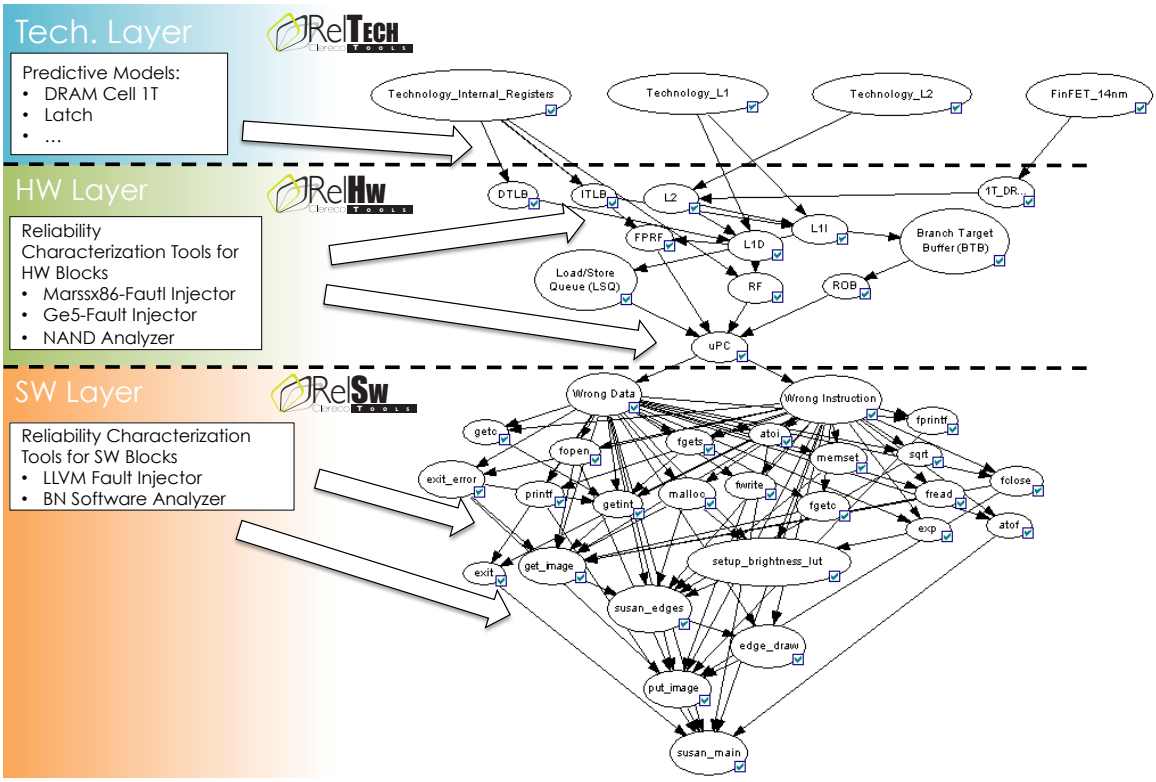


Figure 8: System Level Reliability Evaluation Using a Bayesian Model of the System.

The technology layer plays an active input role. Depending on the technologies employed in the system, the reliability of the system is affected. Nodes at this layer are **root** nodes. They represent technologies for building the hardware structures of the system.

The hardware layer defines the set of hardware resources assembled together to provide hardware functionalities to the system. Since they are built around specific technologies, they are topologically connected to the technology layer nodes. Depending on the internal organization of the hardware subsystem, edges represent interactions among them. The output of the hardware layer is able to follow up to the software layer typically through the system microprocessor.

Eventually, the software layer includes the very last players in the reliability evaluation. While reliability hazards arise into the technological layer, they usually propagate through the hardware components, and, affect the system depending on how the software interacts with the affected hardware components. This interaction is mainly based on how the software manipulates data or on the way the flow of execution follows its proper path. In BN terms, we can identify the functions of the system software as players (the nodes) of the problem. The way they are linked each other depends on the SW functionalities, while the connection to the hardware layer is going to be discussed in the following sections.

Looking at the final BN model, it should be clear why BNs are a very good candidate to model the system stack, where failures at technology level propagate (or mask) at hardware layer or, furthermore, at software layer.

3.1. The Technology Layer

In WP2 several models have been developed to study failure mechanisms of a representative set of current and future technologies. These models enable us to compute failure probability distributions for several types of faults also depending on the environmental conditions.

Figure 9: Technology nodes

Figure 9 shows how technology in our use case is modeled at system level. Models defined in WP2 enable to characterize these nodes with proper failure probabilities depending on the considered failure mechanism. As an example, if we limit the analysis to soft-errors, models developed in WP2 enable us to compute the Raw Error Rate (i.e., the Soft Error Rate) of the technology. It expresses the rate at which the technology is predicted to encounter errors. It is typically expressed as either number of FIT or MTBF. The rate can be easily converted into a probability (or a distribution of probability) [36].

3.2. The Hardware Layer

The hardware layer introduces how errors in the technology propagate and, eventually, can be masked during the systems run. The BN model of the hardware layer is built resorting to a hardware analysis, which should express how data are propagated among HW components of the system, in order to understand where the probability of an error propagate. A feasible hardware layer model for our use case is shown on Figure 10. It represents the model of the internal structure of an x86-out-of-order microprocessor architecture.

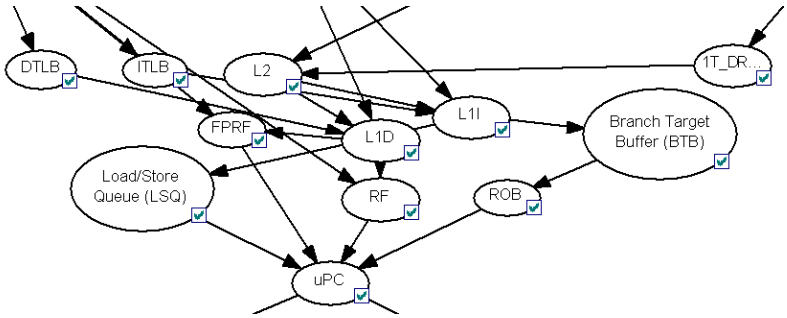


Figure 10: The x86_64 BN model

When defining nodes at this level it is important to consider that, for each node, we must be able to compute the probability of failure in that node given a failure in its direct ancestors.

Looking at the nodes composing the hardware layer we can identify three different cases.

1. Root nodes for the layer

They are nodes with a single direct ancestor, coming from the technology layer. For the sake of simplicity we can consider them as Binary nodes, with two possible *states*:

1. **Error Masked**. An error generated in the technology does not propagate through the outputs of the component
2. **Error**. The output of the component differs from the expected one.

As explained in Section 2.1.3, from a BN perspective, these hardware nodes are described by a 2x2 CPT, as reported in Figure 11.

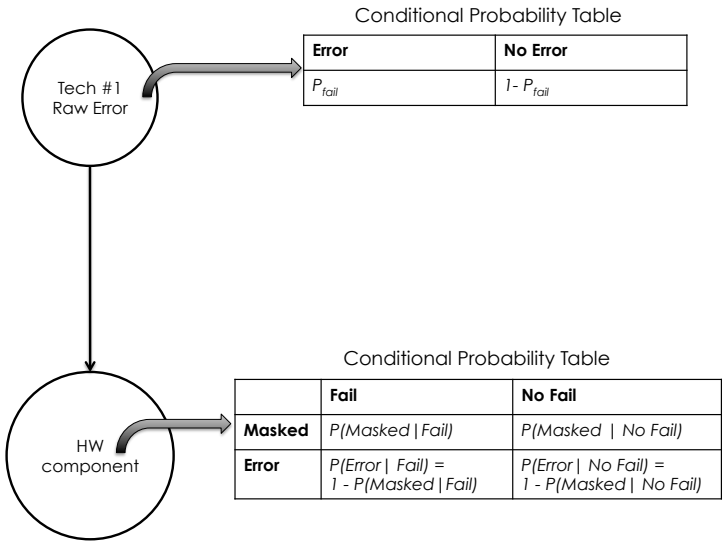


Figure 11: Propagation of Error from a Technology node to an HW node.

2. Intermediate nodes

Intermediate nodes model components directly connected to other components. The correctness of the output of these nodes depends on the correctness of their direct ancestors, as well as on their inner capability of masking incoming errors.

In the example of Figure 12, we show a very small structure, where a first HW component feeds a second one (e.g., cache L2 provides input for the cache L1 of a microprocessor). Since intermediate components have several direct ancestors, their CPTs require additional rows and columns to represent probabilities for all instantiations of the states of the direct ancestor nodes.

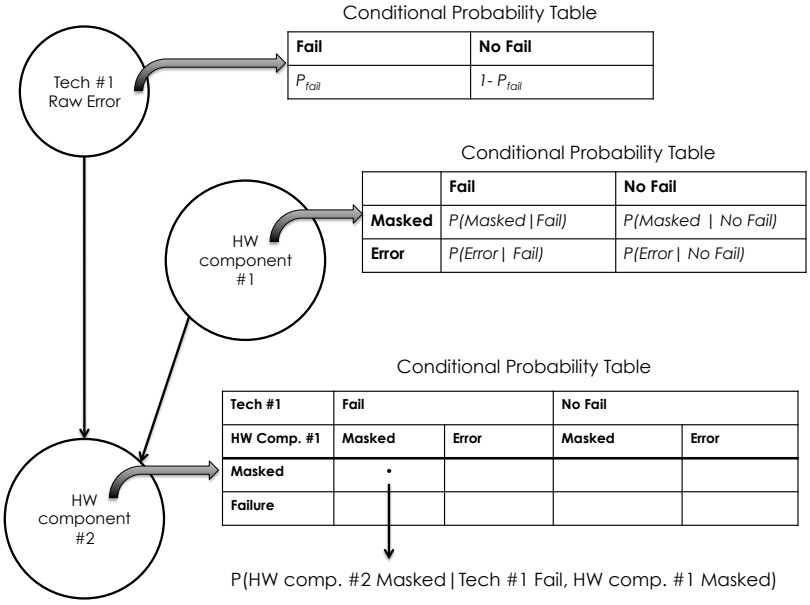


Figure 12: HW node with more than one direct ancestor.

As already stated before, with the growth of the number of ancestors the number of probabilities to compute increases. This issue is a very frequent problem when employing BN models and solutions are already proposed. The application of the Noisy-Max generalization described earlier, is currently under investigation. Figure 13 shows how the previous CPT table of Figure 12 would be simplified when resorting to the Noisy-MAX approach.

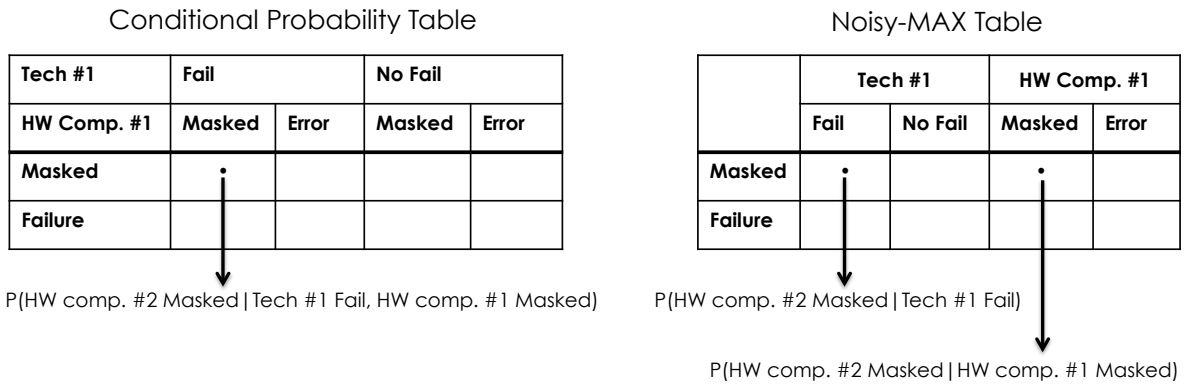


Figure 13: CPT vs. Noisy-MAX

3. Nodes connected to the software layer

These are leaves nodes of the hardware layer. As explained in previous deliverables, to enable analysis of software and hardware in isolation it is mandatory to strongly decouple the hardware and the software layers.

We propose to define a meta-node that projects the propagation of error from the hardware to the software layer. Figure 14 shows an example of this meta-node for the selected use case. The µPC node represents the only node connected to the part of the BN modeling the Software Layer (which will be addressed by Section 3.3), and, therefore, needs to be characterized in a proper way.

First of all, its direct ancestors must be all HW components that manifest their output at instruction level. In fact, the way HW and SW are linked is through the Instruction Set Architecture (ISA) of the microprocessor in the system. It means that if their state directly affects the outputs, the inputs, or the computation of an instruction of the microprocessor, they will be direct ancestors of the µPC node. Moreover, the states of the µPC node can be the same of any other HW component: error *Masked* or not. It represents the final propagation of errors in the hardware layer.

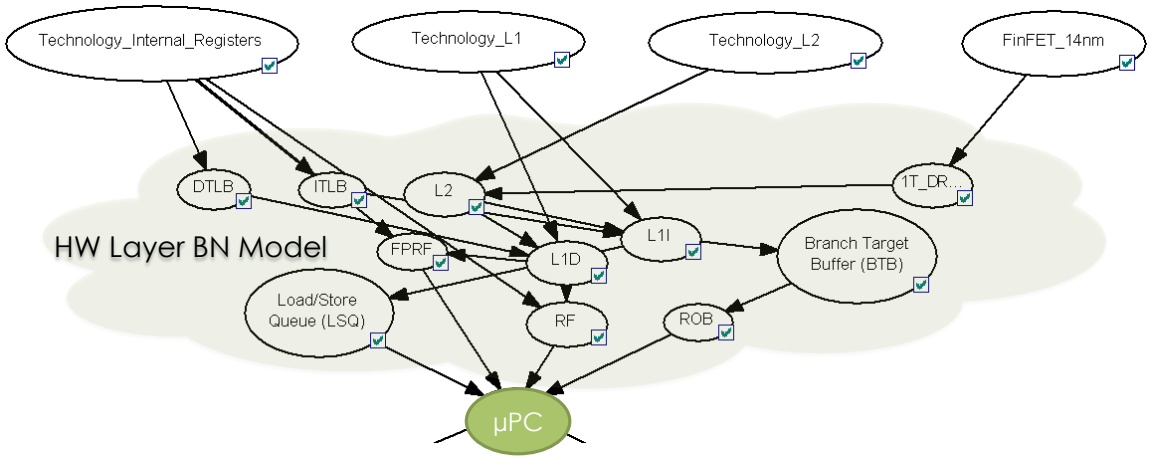


Figure 14: The µPC meta-node

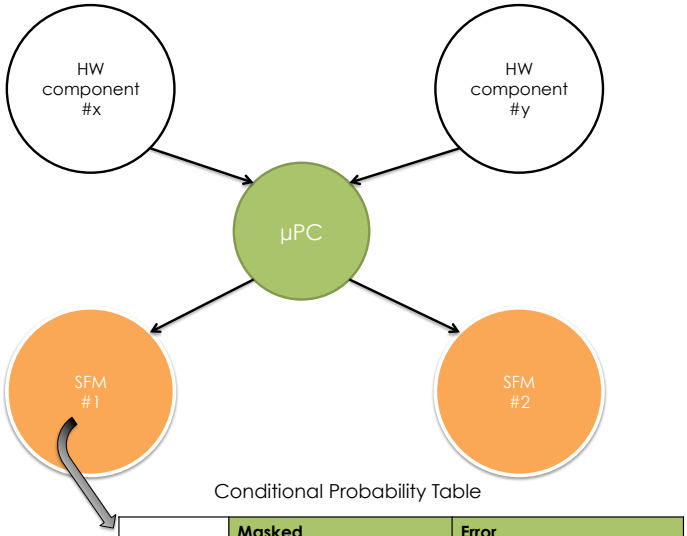
WP3 of CLERECO developed several tools mainly based on architectural fault injectors that enable to compute the probabilities that characterize hardware nodes. Fault injection campaign implemented with these tools can be either performed considering the specific software workload of the system to obtain very precise results or with generic workloads reused in multiple designs to enable fast early estimation with reduced accuracy.

It is important to point out that fault injection campaigns performed using CLERECO tools do not employ complex RTL level models but high-level architectural models that strongly reduce the computation time. Moreover, since the hardware and the software layer are decoupled, simulations do not require executing the full workload but can stop as soon as an error is observed at the ISA level. This further reduces the complexity of the analysis.

3.3. The Software Layer

In order to be able to characterize every SW without knowing the actual HW platform, Deliverable D4.1 defines a set of Software Fault Models (SFMs), which model how hardware errors propagate to software. They mainly rely on alterations that have an impact on the Instruction Set Architecture (ISA) of the microprocessor executing the code, e.g., a change in the content of an operand or a switch in the op-code bits leading to a different instruction, etc.

Since these data represent what the µPC node is expected to deliver, we introduce a SFMs layer of nodes to further connect the HW layer to the SW one. From the HW perspective, these nodes represent the probability of the ISA level manifestation of each SFM when an error from the HW is reported.



	Masked	Error
Masked	P(Masked | Masked)	P(Masked | Error)
Manifested	P(Manifested | Masked)	P(Manifested | Error)

Figure 15: SFM Nodes

In order to populate the BN correctly, we need to define one node for each SFM the designer is going to investigate and to define the related CPT expressing the probability of that fault model given the possible states of the µPC node.

Again these probabilities can be computed by the tools developed in WP3, since as soon as the error is propagated (and noticed) to the ISA level, it is possible to enhance it with the SFMs manifested. Figure 15 shows the SFM nodes (in orange) and the CPT table to be built.

This extra layer feeds the nodes modeling the software employed in the system. The idea is to generate one node for each function of the software and then relate them to build a DAG modeling the propagation of errors as already described for HW. The task requires tracing the software execution to extract its function call graph (FCG). The FCG extractor usually performs the operations described in Figure 16. In this case, for the sake of simplicity, a very simple software evaluating if two consecutive numbers (1 and 2) are even or odd (and printing a label about that classification on screen) is considered. When the software is executed and its traces analyzed, the *main* function is split into several chunks, which identify different parts of the main function and then each *printf* function call is treated as a different call. The obtained FCG is an acyclic graph where each *main* chunk represents a node, as well as each different *printf* instantiation instantiate a node.

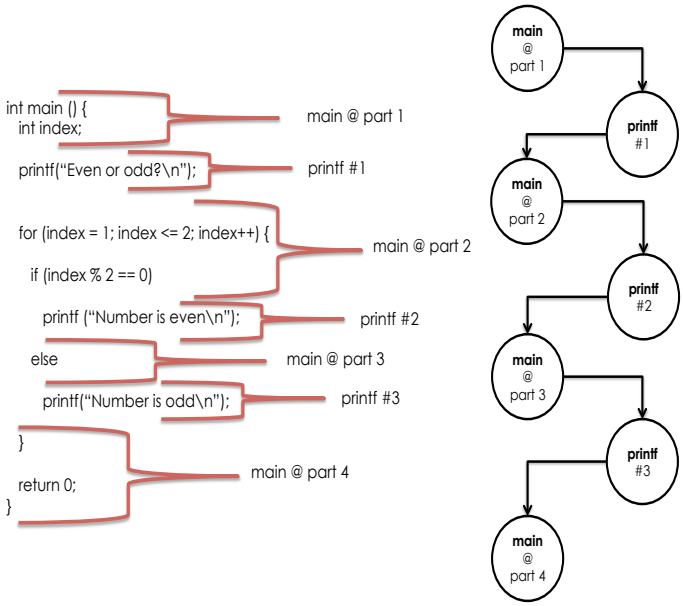


Figure 16: FCG extraction

Each node must be completed with a related CPT table accounting for the events of an error in the ancestor nodes. This way a complete propagation of the error through the software is modeled.

Similarly to the hardware layer, tools developed in WP4 that enable to model software routines on top of the LLVM virtual environment, are also able to perform very fast high-level fault injection campaigns that can be used to fill the software level CPTs.

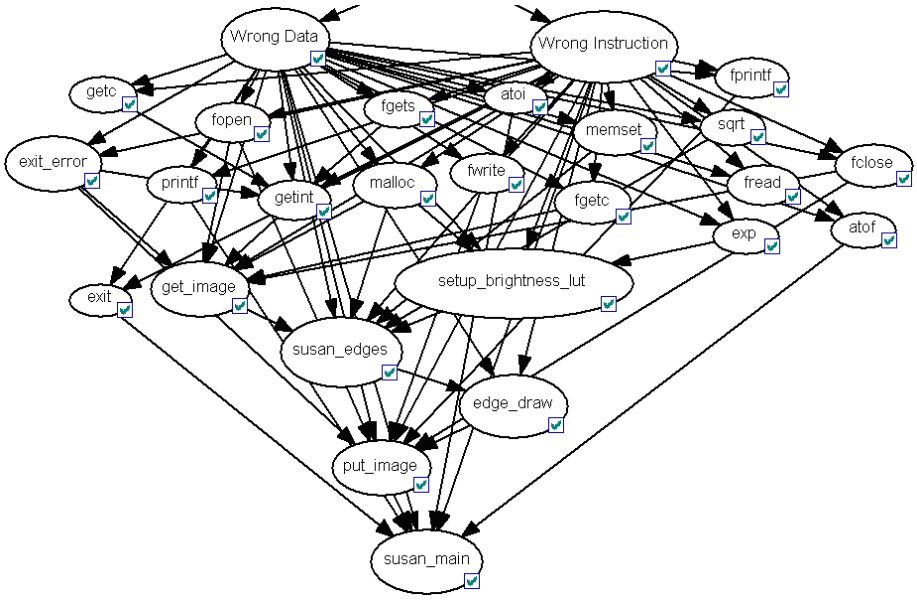


Figure 17: The Susan algorithm BN model

3.4. Reasoning

Once the full model of the system is completely designed and all CPTs filled with related conditional probabilities, BN reasoning approaches as the ones introduced in Section 2.2 can be applied in order to perform system reliability analysis on the modeled system.

4. Conclusion

This deliverable overviews the preliminary version of the CLERECO system level reliability model. We focused on modeling of the structure of the system and on the integration of related information produced by the other work-packages. The next version of this deliverable, that will be released at M36, will include a detailed description of how this model links to all other tools developed in CLERECO, as well as a detailed description of the algorithms to analyze the systems and produce early reliability estimations.

A preliminary version of the modeling and analysis software is already under development and a very preliminary demo prototype is already running. A validation activity has also started to compare results provided by this model with low-level fault injection campaigns on small systems. This is important to assess the accuracy of the developed model.

5. Bibliography

[1] D. Brooks et al. Power, thermal, and reliability modeling in nanometer-scale microprocessors. IEEE Micro, 27:49-62, 2007.
[2] R. A. Sahner et al. Performance and reliability analysis of computer systems: an example-based approach using the SHARPE software package. Kluwer Academic Publishers, Norwell, MA, USA, 1996.
[3] J. B. Bernstein et al. Electronic circuit reliability modeling. Microelectronics and Reliability, 46(12):1957-1979, 2006.

[4] N. Miskov-Zivanov and D. Marculescu. Circuit reliability analysis using symbolic techniques. IEEE Trans. on CAD, 25(12):2638-2649, Dec. 2006.
[5] J. Srinivasan et al. The case for lifetime reliability-aware microprocessors. In Proc. of the 31st Int. Symposium on Computer Arch., ISCA '04, pages 276-, Washington, DC, USA, 2004.
[6] A. Israr and S. A. Huss. Specification and design considerations for reliable embedded systems. In Proc. DATE'08, pages 1111-1116, NY, USA, 2008.
[7] Rishad A. Shafik, Bashir M. Al-Hashimi, Jimson Mathew, Dhiraj Pradhan, Saraju P. Mohanty, "RAEF: A Power Normalized System-Level Reliability Analysis and Estimation Framework," 2014 IEEE Computer Society Annual Symposium on VLSI, pp. 189-194, 2012 IEEE Computer Society Annual Symposium on VLSI, 2012
[8] R. Baumann, "Soft errors in advanced computer systems," Design & Test of Computers, IEEE, vol. 22, no. 3, pp. 258 266, 2005.
[9] S. Borkar, T. Karnik, and V. De, "Design and reliability challenges in nanometer technologies," in Proceedings of the 41st annual Design Automation Conference. ACM, 2004, pp. 75–75.
[10] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," in Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture. IEEE Computer Society, 2003, p. 29.
[11] D. Ernst, S. Das, S. Lee, D. Blaauw, T. Austin, T. Mudge, N. S. Kim, and K. Flautner, "Razor: circuit-level correction of timing errors for low-power operation," IEEE Micro, vol. 24, no. 6, pp. 10–20, 2004.
[12] R. Vadlamani, J. Zhao, W. Burleson, and R. Tessier, "Multicore soft error rate stabilization using adaptive dual modular redundancy," in Design, Automation & Test in Europe Conference & Exhibition (DATE), 2010. IEEE, 2010, pp. 27–32.
[13] M. Dimitrov and H. Zhou, "Unified architectural support for soft-error protection or software bug detection," in Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques. IEEE Computer Society, 2007, pp. 73–82.
[14] N. Nakka, G. P. Saggese, Z. Kalbarczyk, and R. K. Iyer, "An architectural framework for detecting process hangs/crashes," in Dependable Computing-EDCC 5. Springer, 2005, pp. 103–121.
[15] A. Benso, S. Di Carlo, G. Di Natale, P. Prinetto, L. Tagliaferri, and C. Tibaldi, "Promon: a profile monitor of software applications," in 8th IEEE International Workshop on Design and Diagnostics of Electronic Circuits and Systems 2005. DDECS 2005. IEEE, 13-16 April 2005, pp. 81–86.
[16] L. Rashid, K. Pattabiraman, and S. Gopalakrishnan, "Towards under- standing the effects of intermittent hardware faults on programs," in Dependable Systems and Networks Workshops (DSN-W), 2010 International Conference on, June 2010, pp. 101–106.
[17] M.-L. Li, P. Ramachandran, S. K. Sahoo, S. V. Adve, V. S. Adve, and Y. Zhou, "Understanding the propagation of hard errors to software and implications for resilient system design," SIGOPS Oper. Syst. Rev., vol. 42, no. 2, pp. 265–276, Mar. 2008.
[18] F. V. Jensen. "Bayesian Networks and Decision Diagrams". Springer. 2001.
[19] D. Edwards. "Introduction to Graphical Modelling", 2nd ed. Springer-Verlag. 2000.
[20] S. K. S. Hari, S. V. Adve, H. Naeimi, and P. Ramachandran, "Relyzer: Exploiting application-level fault equivalence to analyze application resiliency to transient faults," SIGPLAN Not., vol. 47, no. 4, pp. 123– 134, Mar. 2012.
[21] M.-L. Li, P. Ramachandran, U. Karpuzcu, S. K. S. Hari, and S. Adve, "Accurate microarchitecture-level fault modeling for studying hardware faults," in High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on, Feb 2009, pp. 105–116.
[22] V. Sridharan and D. Kaeli, "Eliminating microarchitectural dependency from architectural vulnerability," in High Performance Computer Ar- chitecture, 2009. HPCA 2009. IEEE 15th International Symposium on, Feb 2009, pp. 117–128.
[23] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: building customized program analysis tools with dynamic instrumentation," ACM Sigplan Notices, vol. 40, no. 6, pp. 190–200, 2005.
[24] M. G. Thomson J. A. Wittaker "Rare Failure State in a Markov Chain Model for Software Reliability" IEEE Trans. Reliab. 48(2) pp. 107-115 1996.
[25] J. G. Konery J. L. Snell "Finite Markov Chains" Springer-Verlang NY 1996.
[26] Ciufudean, C.; Satco, B.; Filote, C., "Reliability Markov Chains for Security Data Transmitter Analysis," Availability, Reliability and Security, 2007. ARES 2007. The Second International Conference on , vol., no., pp.886,894, 10-13 April 2007 doi: 10.1109/ARES.2007.122
[27] R. Gulati and J. B. Dugan "A modular approach for analyzing static and dynamic fault trees " in Proc. Annu. Reliability And Maintainability Symposium (RAMS) Philadelphia Pennsylvania USA January 1997 pp. 57-63.
[28] J. B. Dugan S.J. Bavuso and M. A. Boyd "Dynamic fault-tree models for fault-tolerant computer systems" IEEE Trans. Reliability Vol. 41 No. 3 pp. 363-377 October 1992.
[29] 3. R. Sinnamon and J. D. Andrews "Fault tree analysis and binary decision diagrams" in Proc. Annu. Reliability And Maintainability Symposium (RAMS) Las Vegas Nevada USA January 1996 pp. 215-222.
[30] 4. I. Koren and C. M. Krishna Fault tolerant systems Morgan Kaufmann 500 Sansome Street Suite 400 San Francisco CA 94111 2007 pp. 13-41.
[31] 5. H. Boudali and J. B. Dugan "A new bayesian network approach to solve dynamic fault trees" in Proc. Annu. Reliability And Maintainability Symposium (RAMS) 2005 pp. 451-456.
[32] Kara-Zaitri, C.; Ever, E., "A Hardware Accelerated Semi Analytic Approach for Fault Trees with Repairable Components," Computer Modelling and Simulation, 2009. UKSIM '09. 11th International Conference on, vol., no., pp.146,151, 25-27 March 2009, doi: 10.1109/UKSIM.2009.83

[33] Rajabzadeh, A.; Jahangiry, M.S., "Hardware-based Reliability Tree (HRT) for fault tree analysis," *Computer Architecture and Digital Systems (CADS), 2010 15th CSI International Symposium on* , vol., no., pp.171,172, 23-24 Sept. 2010 doi: 10.1109/CADS.2010.5623587
[34] Distefano, S.; Puliafito, A., "Dynamic Reliability Block Diagrams VS Dynamic Fault Trees," Reliability and Maintainability Symposium, 2007. RAMS '07. Annual , vol., no., pp.71,76, 22-25 Jan. 2007, doi: 10.1109/RAMS.2007.328095
[35] C. Weaver, J. Emer, S. Mukherjee, and S. Reinhardt, "Reducing the soft-error rate of a high-performance microprocessor," Micro, IEEE, vol. 24, no. 6, pp. 30–37, Nov 2004.
[36] A. Savino, S. Carlo, G. Politano, A. Benso, A. Bosio, and G. Di Natale, "Statistical reliability estimation of microprocessor-based systems," Computers, IEEE Transactions on, vol. 61, no. 11, pp. 1521–1534, Nov 2012.
[37] H. Langseth and L. Portinale, "Bayesian networks in reliability," Reliability Engineering & System Safety, vol. 92, no. 1, pp. 92–108, 2007.
[38] L. Yuan-Shun Dai and K. Trivedi, "Performance and Reliability of Tree- Structured Grid Services Considering Data Dependence and Failure Correlation," Computers, IEEE Transactions on, vol. 56, no. 7, pp. 925– 936, 2007.
[39] S. Zhai and S. Z. Lin, "Bayesian networks application in multi-state system reliability analysis," Applied Mechanics and Materials, vol. 347, pp. 2590–2595, 2013.
[40] A. Bobbio, L. Portinale, M. Minichino, and E. Ciancamerla, "Improving the analysis of dependable systems by mapping fault trees into bayesian networks." Rel. Eng. & Sys. Safety, vol. 71, no. 3, pp. 249–260, 2001.
[41] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence," in Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09., April 2009, pp. 502–506.
[42] Max Henrion, Practical Issues in Constructing a Bayes' Belief Network, Eprint 1304.2725, Url: http://arxiv.org/abs/1304.2725
[43] Francisco Javier Diez, Parameter Adjustment in Bayes Networks. The generalized noisy OR-gate, CoRR, 2013, Volume abs/1303.1465, Url: http://arxiv.org/abs/1303.1465
[44] Zagorecki, Adam and Druzdzel, Marek J. (2013) Knowledge engineering for Bayesian networks: How common are noisy-MAX distributions in practice? IEEE Transactions on Systems, Man, and Cybernetics: Systems, 43 (1). pp. 186-195.
[45] F.J. Díez, J. Mira, E. Iturralde, S. Zubillaga, DIAVAL, a Bayesian expert system for echocardiography, Artificial Intelligence in Medicine, Volume 10, Issue 1, May 1997, Pages 59-73, ISSN 0933-3657, http://dx.doi.org/10.1016/S0933-3657(97)00384-9.
[46] University of Michigan at Ann Arbor. Mibench version 1.0. [Online]. Available: http://www.eecs.umich.edu/mibench/
[47] N. Foutris, M. Kaliorakis, S. Tselonis, and D. Gizopoulos, "Versatile architecture-level fault injection framework for reliability evaluation: A first report," in On-Line Testing Symposium (IOLTS), 2014 IEEE 20th International. IEEE, 2014, pp. 140–145.