

DFTSim: A Simulation Tool for Extended Dynamic Fault Trees

H. Boudali, A.P. Nijmeijer, M.I.A. Stoelinga
University of Twente, CS Department, Enschede, NL.
{hboudali, andre, marielle}@cs.utwente.nl

Keywords: Dependability analysis, dynamic fault trees, simulation tool, reliability benchmark.

Abstract

We present *DFTSim*, a simulation tool for dynamic fault trees (DFT). The simulation is carried out by directly sampling the failure distributions attached to the leaves (called basic events) of the tree and propagating the failure times upwards in the tree.

Sampling the distributions of the DFT leaves is however not obvious. To sample from the correct distributions, the analytical expression of the failure distributions of all basic events (BE) must be known. These are indeed known for non-spare BEs; but for spare BEs, they become conditional on the failure of other BEs. Hence, the derivation of the analytical expression of the spares' failure distributions and their sampling is not a trivial task.

We evaluate *DFTSim* by applying it on an extensive benchmark comprised of seven case studies. We compare its results to two other DFT-based reliability tools (namely Galileo and Coral) that, rather than giving simulation-based estimates, compute exact measures.

Our simulation-based approach is, in particular for large DFTs, much faster than the existing approaches. In fact, the computation time of the exact solution methods is exponential in the number of DFT leaves, whereas simulation time is linear in the number of leaves. Moreover, *DFTSim* (and simulation in general) allows to simulate a wide range of distributions and evaluate Markovian as well as non-Markovian models.

1 INTRODUCTION

Dynamic fault trees (DFT) [7][9] are a popular, graphical formalism for reliability analysis. DFTs model the failures of a system in terms of the failures of its components. They consist of *basic events*, at the leaves of the tree, modeling basic component failures; and *gates* that indicate how failures combine and result in a system failure. Basic events (or components) can be either used as spares or non-spares. Six different DFT gates allow the reliability engineer to express complex functional and temporal dependencies between system components. Recently, we have extended DFTs [2] and increased their modeling power by allowing spares to be any independent subsystem (as opposed to only basic events) and the FDEP gates to trigger the failure of any gate and not only basic events (c.f. Section 2.1). This paper considers extended DFTs. In the sequel, we call standard DFTs the non-extended version of DFTs.

The standard DFT formalism has been around for nearly two decades and has been implemented in several tools (e.g. Galileo [14], Relex [6], and Coral [1] tools). These tools convert a standard DFT into a (homogeneous or non-homogeneous) continuous-time Markov chain (CTMC), and suffer from the well known state-space explosion problem. Indeed, as the DFT grows larger, there is an exponential increase in the memory-space needed and the

computation-time required to solve the DFT.

A simulation-based approach of standard DFTs has been suggested (but not implemented) in [11] as a possible solution technique. In this paper, we propose *DFTSim*, a tool for simulating extended DFTs, where the input DFT file is compatible with the Galileo and the Coral textual DFT format. Such compatibility is indeed desired because it allows interoperability and integration among these DFT tools.

One of the major problems in DFT simulation-based approaches is to sample from the correct distributions. In order to carry out such sampling the analytical expression of the failure distributions of all basic events must be known. Such failure distributions are indeed known for non-spare basic events; unfortunately, for spare basic events the failure distribution becomes conditional on the primary's (i.e. the component to be replaced) failure time, and the derivation of the analytical expression of the spare failure distribution becomes a non-trivial task (see Section 5 for details). In this paper, we rely on previous work we have conducted in order to derive the correct form of these failure distributions' analytical expressions using continuous-time Bayesian networks [3].

As the DFT grows, simulation becomes considerably more efficient than state-based solution approaches (as implemented in e.g. Galileo and Coral). In fact, as we will see in Section 7, the simulation time grows linearly with the size of the DFT (i.e. $O(NE)$, where N is the number of samples taken and E the number of elements, i.e. gates and basic events) and memory consumption remains low. Another advantage of *DFTSim* is the handling of any type of failure distribution. In addition to *fixed probability*¹ and *exponential* distribution (i.e. constant failure rate), *DFTSim* allows for (1) time-dependent failure rates (resulting in *non-homogeneous* CTMCs) characterizing failure distributions such as the *Weibull* distribution, and (2) failure rates which are individually functions of more than one time variable (i.e. there is not a single global "clock" in the system, but rather more than one "clock" upon which failure rates may depend). In the latter case, the model is *non-Markovian* and the presence of a cold (i.e. can not fail while not in use) spare basic event having a Weibull distribution is a concrete example of such a model. To our knowledge, there is not a software tool that provides a correct analytical/numerical solution to this kind of DFT models.

Organization of the paper. The remainder of the paper is divided as follows: In Section 2, we introduce DFTs, followed by an overview of the simulation methodology in Section 3. In Section 4, we provide details on how to compute activation times, which are necessary for sampling the failure distributions. In Section 5, we present the sampling method per se for spare and non-spare basic events as well as gates. Section 6 gives some details on the *DFTSim* tool implementation and Section 7 presents a *benchmark* of seven

¹Interpreted as a uniform distribution.

case studies evaluated using *DFTSim* and compared to two other reliability tools, namely Galileo and Coral. Section 8 discusses related work. Finally, Section 9 concludes the paper and suggests future research.

2 DFT BACKGROUND

A DFT is a tree (or rather, a directed acyclic graph) in which the leaves are called *basic events* (BEs) and the other elements are *gates*. The (unique) top or root gate represents the overall system failure. BEs model the failure of physical/logical non-repairable components and are depicted by circles. The failure of a BE is governed by a certain distribution. An example of such a distribution is the exponential distribution where the probability that the BE fails within t time units equals $1 - e^{-\lambda t}$ (λ is the *failure rate* of the BE). Note that the failure rate λ can be time dependent, and thus non-constant and resulting in a non-exponential failure distribution. The non-leaf elements are gates, modeling how the component failures induce a system failure. Static fault trees have three type of (static) gates: the AND gate, the OR gate and the K/M (also called KofM or VOTING) gate, depicted in Figure 1(a), (b), and (c) respectively. These gates fail if respectively all, at least one, or at least K out of M of their inputs fail.

Dynamic fault trees [7] extend static FTs with three novel types of gates²: The priority AND gate (PAND); the Spare gate (SPARE), modeling the management and allocation of spare components; and the functional dependency gate (FDEP). These gates (depicted in Figure 1(d), (e), and (f)) are described below.

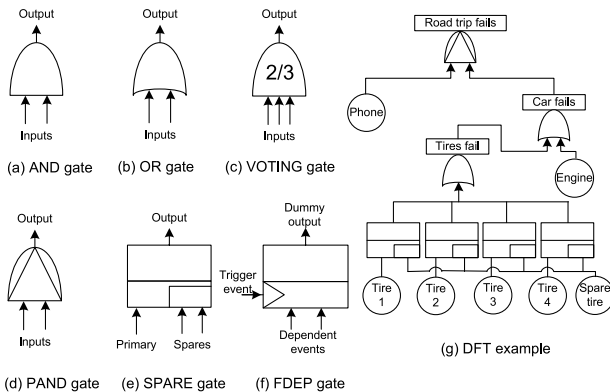


Figure 1. DFT gates and example.

PAND gate. The PAND gate models a failure sequence dependency. It fails if all of its inputs fail from left to right order in the gate’s depiction. If the inputs fail in a different order, the gate does not fail.

Spare gate. The Spare gate has one primary input and zero (which is a degenerated case) or more alternate inputs called *spares*. The primary input of a Spare gate is initially powered on and the alternate inputs are in standby mode. When the primary fails, it is

²A fourth gate called ‘Sequence Enforcing’ (SEQ) gate has also been defined in [7], but it turns out that this gate is expressible in terms of the cold spare gate.

replaced by the first available alternate input (which then switches from the standby mode to the active mode). In turn, when this alternate input fails, it is replaced by the next available alternate input, and so on.

In standby (or dormant) mode, the BE failure rate λ is reduced by a *dormancy factor* $\alpha \in [0, 1]$. Thus, the BE failure rate in standby mode is $\alpha\lambda$. In active mode, the failure rate switches back to λ . Two special cases arise if $\alpha = 0$ or $\alpha = 1$. If $\alpha = 0$, the spare is called a *cold spare* and can not, by definition, fail before the primary. When $\alpha = 1$, the spare is called a *hot spare* and its failure rate is the same whether in standby or in active mode. If $0 < \alpha < 1$, the spare is called a *warm spare*. The Spare gate fails when the primary and all its spares have failed or are unavailable (i.e. used by other Spare gates).

Multiple Spare gates can share a pool of spares. When the primary unit of any of the Spare gates fails, it is replaced by the first available (i.e. not failed, or unavailable because it is taken by another Spare gate) spare unit; which becomes, in turn, the active unit for that Spare gate.

FDEP gate. The functional dependency gate consists of a trigger event (i.e. a failure) and a set of dependent events (or components). When the trigger event occurs, it causes all the dependent components to become inaccessible or unusable (the dependent components can of course also still fail by themselves). The FDEP gate’s output is a ‘dummy’ output (i.e. it is not taken into account during the calculation of the system failure probability).

2.1 Extended DFT

In a nutshell, the extended DFT formalism allows spares to be any independent subsystem or module (as opposed to only BEs as originally defined in [7]) and the FDEP gates to trigger the failure of any gate and not only BEs. We refer to these as the spare-extension and FDEP-extension respectively. The interested reader may see [2] for details on these extensions. In the remainder of the paper, we consider the extended version of DFTs.

2.2 DFT Example

Figure 1(g) shows a DFT modeling a hypothetical road trip. Looking at the top PAND gate, we see that the road trip fails (i.e. we are stuck on the road) if the car fails after the mobile phone has failed; if the car fails first, then we can call the road services to tow the car and continue our journey. The car fails if either the engine fails or the tire subsystem fails, as modeled by the OR gate labeled ‘Car fails’. The car is equipped with a spare tire, which can be used to replace any of the primary tires. When a second tire fails, the tire subsystem fails, causing in turn a car failure. Thus, we model the tire subsystem by four spare gates, each having a primary tire (BEs ‘Tire 1’, ‘Tire 2’, ‘Tire 3’, and ‘Tire 4’) and all sharing a cold spare tire (BE ‘Spare tire’).

3 SIMULATION METHODOLOGY OVERVIEW

The overall simulation procedure consists of generating n samples. For each sample, we record the overall system (i.e. DFT top gate) failure-time. Each sample is obtained by applying the following two steps:

1. Randomly sample each BE in the tree according to its (conditional) failure distribution, and thus obtaining a sample failure-time for each BE.
2. Propagate the obtained BEs' failure-times through the DFT gates. Once the top gate is reached, record this sample system failure-time.

The simulation ends and returns n samples of system failure-time which represent samples from the system's failure probability distribution. Various measures can now be derived such as the system reliability given a specified mission time, the system Mean-Time-To-Failure (MTTF), etc.

Given a set S_1, S_2, \dots, S_n of system failure-time samples; we can, for instance, compute the system reliability R (given a mission time T) using the following *unbiased estimator* [13]:

$$\hat{R} = \frac{1}{n} \sum_{i=1}^n I_{\{S_i > T\}}, \quad (1)$$

where $I_{\{S_i > T\}}$ is an indicator function; i.e., $I_{\{S_i > T\}} = 1$ if $S_i > T$ and 0 otherwise. The estimator \hat{R} is called the *crude Monte Carlo* estimator. Given a confidence level $\alpha \in [0, 1]$, the approximate $(1-\alpha)$ confidence interval for R is given by [13]:

$$\left(\hat{R} - z_{1-\alpha/2} \frac{\hat{\sigma}}{\sqrt{n}}, \hat{R} + z_{1-\alpha/2} \frac{\hat{\sigma}}{\sqrt{n}} \right), \quad (2)$$

where $\hat{\sigma}$ is the samples standard deviation and $z_{1-\alpha/2}$ is the $(1-\alpha/2)$ quantile of the standard normal distribution.

In the sequel, prior to explaining each of the two simulation steps, we first need to compute the BEs' activation times.

4 ACTIVATION TIME

For each (spare) BE in the tree, we need to determine its activation-time (i.e. the time when it switches from a dormant mode to an active mode) as this affects the shape of its failure distribution. The simplest configuration for a spare BE is when a BE A is a spare-input (i.e. not the primary-input) to a single Spare gate, which we assume to be the top DFT gate. In this case, A is activated when the corresponding primary of the Spare gate fails. However, given the spare-extension that we have defined in [2], the BE activation-time computation becomes more involved. According to the spare-extension, a BE can be part of a whole independent³ module acting as a spare. In this case, the BE activation-time is equal to the activation-time of the module it belongs to.

We use the following notation: given an element (i.e gate or BE) X , we let $par(X)$ denote a parent of X (e.g. in Figure 2(a), Y is parent of W), and $prim(X)$ denotes the primary of X if X is a Spare gate. $AT(X)$ stands for activation-time of X and $FT(X)$ stands for failure-time of X .

4.1 BE Activation

The rule for activating a BE is as follows:

1. If the BE A directly inputs (as a spare) to a single Spare gate X (and possibly to other non-Spare gates), then

$$AT(A) = \max\{FT(prim(X)), AT(X)\}.$$

2. If the BE A directly inputs (as a spare) to 2 (or more) Spare gates X_1, X_2 (and possibly to other non-Spare gates), then

$$AT(A) = \min\{\max\{FT(prim(X_1)), AT(X_1)\}, \max\{FT(prim(X_2)), AT(X_2)\}\}.$$

This is the case where A is shared between X_1 and X_2 . The minimum simply indicates that A is activated whenever the first Spare gate activates it.

3. If the BE A inputs only to non-Spare gates (or acts as a primary of a Spare gate), then

$$AT(A) = AT(par(A))$$

Note that any parent gate, excluding FDEP gates for which the BE inputs as a dependent event (a design choice made in [2]), can be picked as all parents must have the same activation-time since we restrict ourselves to independent spare modules.

Note that in the first and second cases, the primary can be a BE or a whole subtree (i.e. any gate except an FDEP). Furthermore, if A is the n^{th} spare of the Spare gate X , then the primary's failure-time is defined as the maximum of the failure-times of the primary and all $n-1$ preceding spares, i.e. $FT(prim(X))$ is replaced by $\max\{FT(prim(X)), FT(spare 1), \dots, FT(spare n-1)\}$.

4.2 Gate Activation

We have defined the BE activation-time in a recursive fashion, using the activation-time of a gate. The activation-time of a gate is derived using the same rules defined for the BE in Section 4.1. The FDEP gate, whose output is a dummy output, is however an exception. When the FDEP gate A inputs (its input position being irrelevant) to a Spare gate X , its activation-time does not depend on the primary's failure-time; and cases 1 and 2 become respectively,

$$AT(A) = AT(X)$$

$$AT(A) = \min\{AT(X_1), AT(X_2)\}.$$

The activation time of the top DFT gate is $t = 0$. It is important to mention that the gates' activation-times are only computed to determine the BEs' activation-times and are not subsequently used to sample the gates' failure distributions.

4.3 Activation Example

Applying the above rules on the DFT in Figure 2(a), we have:

- $AT(T) = AT(X) = AT(Z) = AT(A) = AT(F) = 0$.
- $AT(Y) = \min\{\max\{FT(A), AT(X)\}, \max\{FT(F), AT(Z)\}\} = \min\{\max\{FT(A), 0\}, \max\{FT(F), 0\}\} = \min\{FT(A), FT(F)\}$.
- $AT(W) = AT(Y) = \min\{FT(A), FT(F)\}$.
- $AT(B) = AT(C) = AT(W) = \min\{FT(A), FT(F)\}$.
- $AT(D) = \max\{FT(W), AT(Y)\} = \max\{FT(W), \min\{FT(A), FT(F)\}\}$.
- $AT(V) = AT(Y) = \min\{FT(A), FT(F)\}$.
- $AT(E) = AT(V) = \min\{FT(A), FT(F)\}$.

³See [2] for details.

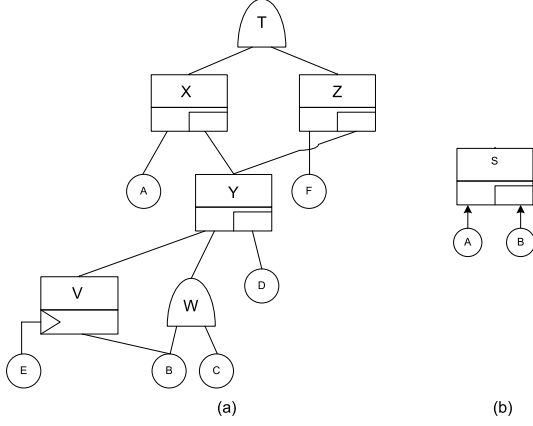


Figure 2. (a) Example activation-times, (b) A simple DFT.

5 FAILURE-TIME SAMPLING

Each element in the fault tree is seen as a random variable (RV) having a certain cumulative distribution function (CDF) F . Thus, a RV X having a CDF F represents the failure-time of element X , and the probability for element X to fail within time t is $F(t) = P[X \leq t]$. The analytical expression of the CDF of each dynamic fault tree element was derived in [3]. For any DFT gate X with two⁴ inputs A and B , the CDF is given as a conditional CDF $F(X|A, B)$, where A and B are the failure-times of the gate's inputs. $F(X|A, B)$ is however deterministic; i.e. if A and B are known, then the failure-time of X has a unique value. For a BE, we distinguish between two cases: (1) when the BE is not used as a spare, and (2) when the BE is used as a spare (directly inputs to a Spare gate or belongs to a spare module). In the first case, the RV X representing the BE has an unconditional CDF $F(t)$. For instance if BE X has an exponential failure distribution with a failure rate λ , then $F(t) = 1 - e^{-\lambda t}$. Figure 3 shows a typical CDF $F(t)$ of a non-spare BE. In the second case,

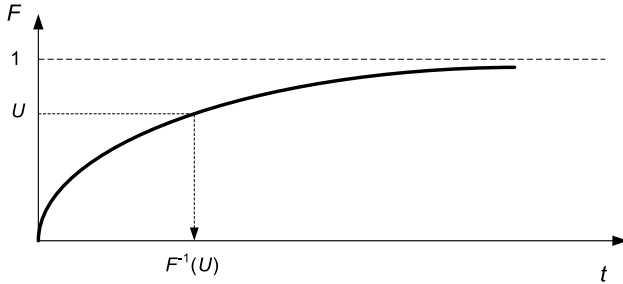


Figure 3. Inverse transform method for a non-spare BE.

the BE is used as a spare and its failure distribution depends on its activation-time. In fact, the BE has a conditional CDF $F(t|a)$, where a is its activation-time. Recall, from Section 4, that the activation-time is a function of other elements' failure-times. We define the in-isolation CDF of a BE X as the unconditional CDF $F_i(t)$ (and the corresponding in-isolation failure (or hazard) rate $\lambda_i(t)$) if X were not used as a spare. In the DFT formalism, the dormancy factor α of a BE is a well-defined notion and means that the BE's hazard rate

⁴The argumentation is the same for any number of inputs.

is reduced by a factor α when dormant and remains the same when active, i.e.:

$$\begin{aligned}\lambda_d(t) &= \alpha\lambda_i(t) \quad \text{when dormant, and} \\ \lambda_a(t) &= \lambda_i(t) \quad \text{when active.}\end{aligned}\quad (3)$$

For the exponential distribution $\lambda_i(t) = \lambda$ (i.e. it is time-independent). However, this is not true in the general case (e.g. for the Weibull distribution $\lambda(t) = k\beta^{-1}(t/\beta)^{k-1}$, where k and β are the shape and scale distribution parameters). Given the in-isolation CDF $F_i(t)$ and its corresponding probability density function (PDF) $f_i(t)$, the activation-time a , and the dormancy factor α ; then, the BE conditional PDF is [4]:

$$\begin{aligned}f(t|a) &= u(a-t)\alpha f_i(t)[1 - F_i(t)]^{\alpha-1} + \\ &u(t-a)f_i(t)[1 - F_i(a)]^{\alpha-1}\end{aligned}\quad (4)$$

where, $u(x)$ is the unit step function⁵ and $\alpha > 0$. The first term (i.e. $u(a-t)\alpha f_i(t)[1 - F_i(t)]^{\alpha-1}$) describes the BE failure while dormant and thus represents the *dormant failure distribution*, while the second term (i.e. $u(t-a)f_i(t)[1 - F_i(a)]^{\alpha-1}$) describes the BE failure while active and thus represents the *active failure distribution*. For $\alpha = 0$ (i.e. cold spare), $f(t|a) = u(t-a)f_i(t-a)$ [3].

We can rewrite Equation 4 as $f(t|a) = u(a-t)f_d(t) + u(t-a)f_a(t)$, where $f_d(t)$ is the PDF during the dormant mode and $f_a(t)$ is the PDF during the active mode. Thus, we have⁶:

$$\begin{aligned}F(t|a) &= \int_0^t f(x|a)dx = u(a-t)F_d(t) \\ &+ u(t-a)\{F_d(a) + F_a(t) - F_a(a)\}\end{aligned}\quad (5)$$

Note that, for a non-spare BE, with an in-isolation CDF F_i , its CDF $F(t|a) = F(t) = F_i(t)$.

Figure 4 shows a typical conditional CDF $F(t|a)$ of a spare BE. The above showed how to derive the appropriate CDF to be sampled

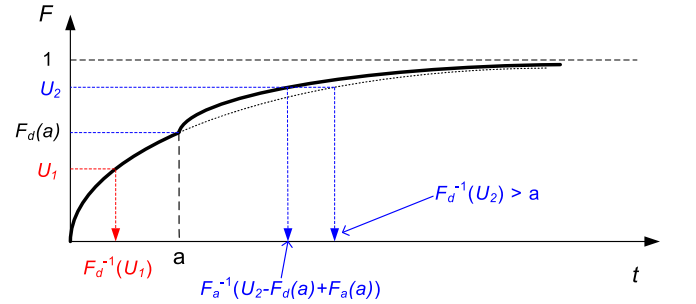


Figure 4. Inverse transform method for a spare BE.

for a spare BE given its in-isolation CDF, its activation time, and its dormancy factor. The obtained CDF is in accordance with the DFT formalism interpretation of the dormancy factor (i.e. Equation 3).

As an illustrative example, let's consider the simple DFT shown in Figure 2(b). The DFT consists of a spare top gate S , a primary BE A , and spare BE B . S has a conditional PDF $f_S(s|a, b)$, A has

⁵ $u(x) = 1$ for $x > 0$, $u(x) = \frac{1}{2}$ for $x = 0$, and $u(x) = 0$ elsewhere [3].

⁶ $F_d(t) = \int_0^t f_d(x)dx = 1 - \{1 - F_i(t)\}^\alpha$ and $F_a(t) = \int_0^t f_a(x)dx = \{1 - F_i(a)\}^{\alpha-1} F_i(t)$.

an unconditional PDF $f_A(a)$, and B has a conditional PDF $f_B(b|a)$ (where the activation-time a is equal to A 's failure-time). Applying the product rule for probability and given the dependencies between the three RVs, the whole DFT possesses the following joint CDF:

$$F(a, b, s) = F_S(s|a, b)F_B(b|a)F_A(a) \quad (6)$$

During simulation, we sample the joint distribution by sequentially sampling $F_A(a)$, then $F_B(b|a)$, and finally $F_S(s|a, b)$. Consequently, we obtain one sample (a, b, s) representing A , B , and S failure-times. Moreover, in our example, the system failure-time sample is the value of s . In reality, we only need to sample the CDFs of BEs. In fact, the CDF of any DFT gate is deterministic and the value of the gate's failure-time is known (with probability 1) given the failure-times of its inputs. For instance, in the case of the spare gate S , its failure-time $s = \max(a, b)$. Therefore, in our simulation framework, we first sample all the BEs' CDFs for which we get a set of failure-times (Sections 5.1 and 5.2). We then simply propagate these failure-times through the tree using propagation rules for each gate (Section 5.3), which is equivalent to sampling the gates' CDFs.

5.1 Non-spare BE Failure-time Sampling

We use the standard *inverse transform method* [13] to generate samples. The inverse transform allows one to sample any distribution by generating samples from a uniform (over $[0, 1]$) distribution. The method works as follows: A uniform random number U is generated using an available (such as in MATLAB) random number generator. Given the CDF F of a BE, we set $F(t) = U$ and find the corresponding t that satisfies $t = F^{-1}(U)$, where F^{-1} is the inverse function of F ⁷. This t corresponds to a sample failure-time of the given BE. For example, if the BE has an exponential distribution with failure rate λ , then $F(t) = 1 - e^{-\lambda t}$ and $t = \frac{\ln(1-U)}{-\lambda}$. Figure 3 illustrates pictorially the inverse transform method in the case of a non-spare BE.

5.2 Spare BE Failure-time Sampling

When the BE is used as a spare, the inverse transform method is slightly more involved. We need to sample a CDF of the form given in Equation 5. In fact, this is done in two stages following these steps:

1. Generate a uniformly-distributed random number U .
2. Sample the dormant distribution F_d using U .
3. If $F_d^{-1}(U) \leq a$, then the BE fails while dormant and we take $F^{-1}(U) = F_d^{-1}(U)$ as the valid failure-time sample.
4. If $F_d^{-1}(U) > a$, then the BE fails while active and consequently we need to adjust the sampling⁸: When $F_d^{-1}(U) > a$, we have $U = F(t|a) = F_d(a) + F_a(t) - F_a(a)$. We set $Y = U - F_d(a) + F_a(a) = F_a(t)$ and finally, we take $F_a^{-1}(Y)$ as the valid failure-time sample.

Figure 4 illustrates the sampling of a spare BE. The case where $F_d^{-1}(U) \leq a$ is shown using U_1 and the case where $F_d^{-1}(U) > a$ is shown using U_2 .

⁷Note that the closed-form of the inverse does not always exist. In this case, we need to resort to some numerical methods to compute $F^{-1}(U)$.

⁸Knowing that a CDF is a non-decreasing function.

It is evident that in order to sample a spare BE failure distribution, its activation time must be known. Therefore, one must first sample all necessary distributions and propagate the needed failure-times to compute the BE's activation-time (c.f. Section 4).

5.3 Failure-times Propagation

Sampling the CDF of a gate boils down to propagating its inputs' failure-times through the gate. Each gate has a propagation rule, that is, given its inputs' failure-times we compute the gate's failure-time. In the following, we show the rules for each of the DFT gates. Generalizing to any number of inputs is straightforward; however, due to lack of space, we do not show this here.

- AND gate: $AND(FT(A), FT(B)) = \max(FT(A), FT(B))$.

- OR gate: $OR(FT(A), FT(B)) = \min(FT(A), FT(B))$.

- KofM gate: $KofM(k, \mathbf{X}) = \text{sort}(\mathbf{X})(k)$.

The KofM gate fails if at least k inputs out of m fail. \mathbf{X} is a vector of the inputs' failure-times. $\text{sort}(\mathbf{X})$ returns \mathbf{X} sorted in increasing order. $\mathbf{Y}(k)$ denotes the k -th element of vector \mathbf{Y} .

- PAND gate:

$$PAND(FT(A), FT(B)) = \begin{cases} FT(B) & \text{if } FT(A) \leq FT(B) \\ \infty & \text{if } FT(A) > FT(B) \end{cases}$$

- Spare gate X with spare S :

$$Spare(FT(\text{prim}(X)), (FT(S), \text{Taken})) = AND(FT(\text{prim}(X)), FT(S) \times \text{Taken}),$$

where Taken is a boolean value equals to 0 if the spare S has been taken by another Spare gate (i.e. occurs when the spare is shared), and 1 otherwise. 'Taken' is easily determined from the spare S activation-time, the failure-time of the primary, and the activation-time of X ⁹. Intuitively, the Spare gate acts as the AND of its primary and its spare if the latter is taken (used) by the Spare gate. If the spare is taken by another Spare gate, then the Spare gate fails when its primary fails; i.e. $AND(FT(\text{prim}(X)), 0) = FT(\text{prim}(X))$.

- FDEP gate: This gate does not have a propagation rule as this gate has a 'dummy' output. However, if a BE X is dependent upon a trigger T its failure-time becomes $OR(FT(X), FT(T))$ as defined in [1].

6 TOOL IMPLEMENTATION

In this section we describe the *DFTSim* tool, whose scheme is shown in Figure 5. We use Matlab [10] to carry out the actual simulation. The DFT is specified using the input language used in the Galileo textual DFT format¹⁰. We use ANTLR [15] and Java to generate a parser for the DFT input language. In order to achieve this, we wrote a DFT grammar (*DFTSim.g*) from which *DFTSim-Lexer* and *DFTSimParser* are automatically generated. We also implemented various Java classes (*DFTElements*) which, for each DFT element (the BE and the six types of gates), specifies which Matlab command needs to be written (i.e. setting the activation-time and the failure-time). *DFTSim* then uses *DFTSimLexer*, *DFTSimParser*, and *DFTElements* to read an input DFT file (*file.dft*) and compile the corresponding Matlab simulation file (*file.m*). In the

⁹The spare is taken by the Spare gate if $AT(S) = FT(\text{prim}(X))$ or $AT(S) = AT(X)$. Note that we run into non-determinism if simultaneous activations or failures (e.g. through an FDEP gate) arise.

¹⁰The language has actually been augmented to support the extended DFT formalism.

Matlab file, we basically write the Matlab commands for computing the activation-time and failure-time (according to the rules described in Sections 4 and 5) of each element in the DFT. As activation/failure time depends on other activation/failure times, the order in which these commands are written is important¹¹. These commands also use some predefined Matlab functions, located in DFTElements Matlab library, for sampling a BE and propagating failure-times. All the samples (i.e. system failure-times) are collected in a vector, and various measures are then output by Matlab, such as the reliability (Equation 1), the confidence interval (Equation 2), etc. At this stage, *DFTSim* does not support two features found in the Galileo tool, namely imperfect coverage¹² and phased-mission systems. However, *DFTSim* supports (like Coral) the spare-extension and FDEP-extension, mentioned in Section 2.1, which are not present in Galileo.

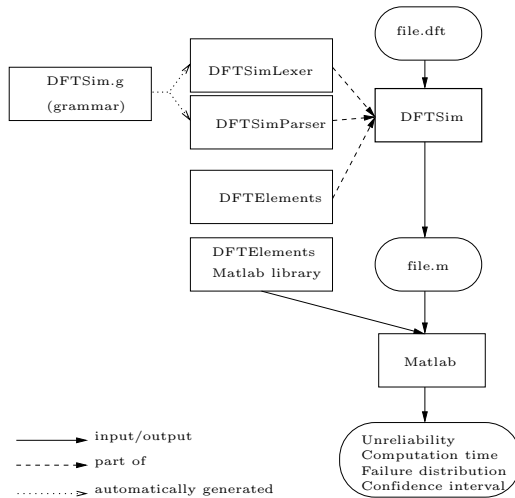


Figure 5. Tool scheme for *DFTSim*.

7 CASE STUDIES

We have assessed *DFTSim* on a benchmark consisting of seven case studies and compared the results with the Galileo and Coral tools results. The seven cases studies are: The cascaded PAND system (CPS), the cardiac assist system (CAS), the multi-processor distributed computer system (MDCS), three versions (standard, large, and Weibull) of the fault tolerant parallel processors (FTPP), and finally a modified version of the FTTP¹³.

The results are shown in Table 3. We ran all experiments (including Galileo and Coral) on a Pentium 4 processor running at 3.2 GHz with 2 GB of memory. The simulations were run with 10,000 and 100,000 samples (column three) and the relative error (i.e. the relative half width of the 95% confidence interval) is shown in column six. Since Galileo and Coral are state-based analytical tools, we report, in columns four and five, their largest (in terms of number of states and transitions) state-space model obtained for each experiment. Finally, we report the system unreliability and the computation time in columns seven and eight respectively.

¹¹For this reason we do not allow cycles in the DFT which can be caused by FDEP gates.

¹²This feature is however easy to implement.

¹³The DFT files are available on <http://fmt.cs.utwente.nl/projects/MOQS/DFTSim/benchmarks/>

It is clear, from the result in Table 3, that the simulation time is roughly linear with the number of samples N and the number of elements E in the DFT (i.e. simulation time is $O(NE)$). As for the memory consumption, given that we store all of the overall system failure-time samples (i.e. the system failure distribution), it is also linear in N . In general, simulation provides a quick way to compute the order of magnitude of the measure of interest. Furthermore, simulation becomes the only feasible (in terms of computation time and memory space) solution when E exceeds a certain value. Next, we provide details on the various case studies.

7.1 The Cascaded PAND System

This is a simple hypothetical example, taken from [2] and shown on Figure 6(a). All BEs have a constant failure rate equals to 1. Note that in general the Coral tool leads to a smaller state-space given its efficient compositional-aggregation technique for generating the state-space [1].

7.2 The Cardiac Assist System

This system, taken from [1] and shown on Figure 6(b), consists of three separate modules (i.e. CPU, motors, and pump units). Table 1 shows the failure rates of the various components. In addition, B is a warm spare with a dormancy factor $\alpha = 0.5$, and MB and PS are cold spares (i.e. $\alpha = 0$).

During analysis, the Galileo tool modularizes the DFT into three independent modules (namely CPU, motors, and pump units) and generates a separate CTMC for each one of them. Given the relatively small size of these CTMCs, Galileo computation time for this particular DFT is very short. This is not the case for Coral which does not use modularization.

7.3 The Multi-processor Distributed Computer System

This case study is also taken from [1] and shown on Figure 6(c). Table 2 shows the failure rates of its components. In addition, D12, D22, and M3 have a dormancy factor $\alpha = 0.5$.

Component	CS	SS	P	B	MA	MB	MS	PA	PB	PS
Rate	0.2	0.2	0.5	0.5	1	1	0.01	1	1	1

Table 1. Failure rates for CAS.

Component	N	P1,P2	D11,D12,D21,D22	M1,M2,M3
Rate	0.00002	0.005	0.8	0.0003

Table 2. Failure rates for MDCS.

7.4 The Standard Fault Tolerant Parallel Processors

This system, taken from [1], consists of 16 processors divided into 4 logical groups. In each group, a processor is used as a shared cold spare. A network element (NE) physically connects 1 processor in each group (thus there are 4 NEs) to the rest of the system. The failure of an NE makes the 4 processors connected to it unavailable (i.e. essentially failed). The requirement is to have at least two processors operational in each group. The DFT is shown on Figure 6(d), where the processors are denoted with T and the network elements with NE . All NEs have a failure rate equal to 0.017, and all processors have a failure rate equal to 0.11. The four spare processors are cold spares.

Case study	Tool	# of samples	Max # of states	Max # of transitions	Relative error (95%)	Unreliability (time = 1)	Time (sec)
CPS	Galileo		4113	24608		0.00135	490
	Coral		133	465		0.00135	67
	<i>DFTSim</i>	10^4			54%	0.00130	4
	<i>DFTSim</i>	10^5			16%	0.00142	40
CAS	Galileo		8	10		0.65790	1
	Coral		36	119		0.65790	94
	<i>DFTSim</i>	10^4			1%	0.65640	4
	<i>DFTSim</i>	10^5			<1%	0.65651	43
MDCS	Galileo		253	1383		0.06664	1
	Coral		190	723		0.06664	82
	<i>DFTSim</i>	10^4			7%	0.06490	4
	<i>DFTSim</i>	10^5			2%	0.06737	39
FTPP standard	Galileo		32757	426826		0.01922	13111
	Coral		1325	14153		0.01922	193
	<i>DFTSim</i>	10^4			14%	0.01920	10
	<i>DFTSim</i>	10^5			4%	0.01981	98
FTPP large	Galileo		-	-		-	> 32400
	Coral		43105	654905		0.00306	329
	<i>DFTSim</i>	10^4			30%	0.00420	12
	<i>DFTSim</i>	10^5			12%	0.00268	121
FTPP Weibull	Galileo		32757	426826		0.01287	10833
	Coral						
	<i>DFTSim</i>	10^4			18%	0.01190	10
	<i>DFTSim</i>	10^5			5%	0.01292	97
FTPP complex	Galileo						
	Coral		1795448	34773150		0.02136	644719
	<i>DFTSim</i>	10^4			13%	0.02390	24
	<i>DFTSim</i>	10^5			4%	0.021012	234

Table 3. Results of the case studies.

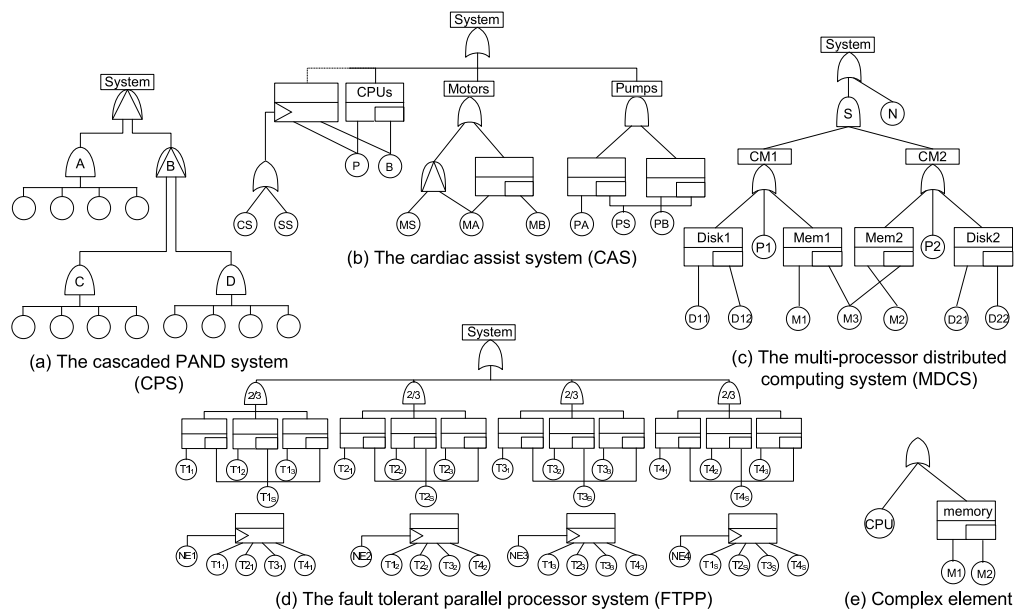


Figure 6. The DFTs of the case studies.

7.5 The Large Fault Tolerant Parallel Processors

To illustrate even further the state-space explosion problem, we took the FTTP system and made it slightly larger by considering 5 processors in each group (where 4 primaries share one spare, and a group's failure is described by a 3/4 voting gate) and 5 network elements instead. The failure rates and dormancy factors remain the same. Unfortunately, there are no results available from Galileo as it ran 'out of memory' after 9 hours of computation.

7.6 The Weibull Fault Tolerant Parallel Processors

This system is the same as the standard FTTP except that the 4 NEs' exponential distributions have been replaced by Weibull distributions with scale parameter equals $\frac{1}{0.017}$ and shape parameter equals 2. In this example, no results are available from Coral as this tool currently supports only exponential distributions.

7.7 The Complex Fault Tolerant Parallel Processors

This case study is an extension to the FTTP system where each of the processors (i.e. Ti_j) is replaced by a complex element shown on Figure 6(e), where CPU, M1, and M2 have a failure rate equal to 0.11 and $\alpha = 0$. No results are available from the Galileo tool as this kind of extended DFT can not be handled by Galileo.

8 RELATED WORK AND DISCUSSION

As prior work on DFT simulation, we mention the work in [5]. The difference with our work is: in [5], the underlying Markov chain of the DFT model is simulated; whereas, in this paper we directly simulate the DFT (see Section 3 on how our simulation methodology works); furthermore, in [5], only the standard DFT formalism is used as opposed to the extended DFT formalism used in our approach.

In [8], the authors present an FPGA-based (i.e. hardware-based as opposed to our software-based simulation) simulation methodology of a tree model called Time-To-Failure tree. The authors also show that any static or standard dynamic fault tree can be translated into a Time-To-Failure tree. However, in this work, sampling a spare BE failure distribution remains problematic. In fact, in [8], and in some simulation tools such as BlockSim [12]¹⁴ where a spare component can be specified, the DFT notion of a dormancy factor is missing and one specifies two distinct and independent CDFs for the component's dormant and active modes. In the DFT formalism, this is of course not adequate as these two distributions are related and depend on the dormancy factor α . In fact, as seen in Equations 4 and 5, the dormant and active CDFs are derived given the in-isolation CDF and the dormancy factor.

9 CONCLUSION AND FUTURE WORK

We have presented *DFTSim*, a tool for extended DFT simulation. As illustrated by the case studies, simulation of DFTs is a viable and fast solution technique when only an estimate of the measures of interest are required as opposed to exact values. Moreover, simulation becomes the only practical technique, in terms of computation time and memory consumption, when dealing with large DFTs.

However, one drawback of simulation is the increased number of samples to be run in order to get meaningful results when dealing

with rare events (i.e. events that occur with a very low probability). Fortunately, there are a number of techniques in order to alleviate this problem, most notably the *importance sampling* technique [13][5]. One direction for future research is to investigate the application of such a technique in the DFT context.

Another avenue of future work would be, given the standard DFT format used in our tool, to integrate the *DFTSim* tool into other DFT analysis tools such as Galileo or Coral and provide an alternative simulation-based technique to be used in conjunction with analytical techniques.

Acknowledgment. This research has been partially funded by the Netherlands Organization for Scientific Research (NWO) under FOCUS/BRICKS grant number 642.000.505 (MOQS); by the EU under grants numbers IST-004527 (ARTIST2) and FP7-ICT-2007-1 (QUASIMODO); and by the DFG/NWO bilateral cooperation programme under project number DN 62-600 (VOSS2).

REFERENCES

- [1] H. Boudali, P. Crouzen, and M.I.A. Stoelinga. A compositional semantics for Dynamic Fault Trees in terms of Interactive Markov Chains. In *Proc. of the 5th International Symposium on Automated Technology for Verification and Analysis*, pages 441–456. LNCS, 2007.
- [2] H. Boudali, P. Crouzen, and M.I.A. Stoelinga. Dynamic fault tree analysis using input/output interactive Markov chains. In *Proc. of the 37th Annual IEEE/IFIP International Conference on DSN*, pages 708–717. IEEE, 2007.
- [3] H. Boudali and J. B. Dugan. A continuous-time Bayesian network reliability modeling and analysis framework. *IEEE Trans. on Reliability*, 55(1):86–97, 2006.
- [4] H. Boudali and J. B. Dugan. Corrections on 'a continuous-time bayesian network reliability modeling and analysis framework'. *IEEE Trans. on Reliability*, 57(3):532–533, 2008.
- [5] M.A. Boyd and S.J. Bavuso. Simulation modeling for long duration spacecraft control systems. In *Proceedings of the annual Reliability and Maintainability Symposium*, Jan 1993.
- [6] Relex Software Corporation. <http://www.relex.com>.
- [7] J. B. Dugan, S. J. Bavuso, and M. A. Boyd. Dynamic fault-tree models for fault-tolerant computer systems. *IEEE Transactions on Reliability*, 41(3):363–377, September 1992.
- [8] A. Ejlali and S. G. Miremadi. FPGA-based Monte Carlo simulation for fault tree analysis. *Microelectronics Reliability*, 44(6):1017–1028, 2004.
- [9] J. B. Dugan et al. DIFTree: a software package for the analysis of dynamic fault tree models. In *Proceedings of Reliability and Maintainability Symposium*, pages 64–70, 1997.
- [10] The MathWorks Inc. <http://www.mathworks.com/>.
- [11] R. Manian, J. B. Dugan, D. Coppit, and K.J. Sullivan. Combining various solution techniques for dynamic fault tree analysis of computer systems. In *Proceedings of the 3rd IEEE International High-Assurance Systems Engineering Symposium*, pages 21–28, Nov 1998.
- [12] ReliaSoft. <http://www.reliasoft.com/blocksim/>.
- [13] R. Y. Rubinstein and D. P. Kroese. *Simulation and the Monte Carlo Method*. Wiley Series in Probability and Statistics. John Wiley and Sons, 2nd edition, 2008.
- [14] K. J. Sullivan, J. B. Dugan, and D. Coppit. The Galileo fault tree analysis tool. In *Proceedings of the 29th Annual Inter. Symp. on Fault-Tolerant Computing*, pages 232–235. IEEE Computer Society, 1999.
- [15] ANTLR v3. <http://www.antlr.org/>.

¹⁴Although BlockSim does not simulate DFTs, but rather reliability block diagrams and fault trees having some dynamic features such as spares.