Reliability Improvement Achievable in CAN-based Systems by Means of the ReCANcentrate Replicated Star Topology

Manuel Barranco, Julián Proenza
Dpt. Matemàtiques i Informàtica
Universitat de les Illes Balears, Spain
manuel.barranco@uib.es, julian.proenza@uib.es

Luís Almeida
DEC-FEUP
Universidade do Porto, Portugal
lda@fe.up.pt

Abstract

Despite there is a growing interest in using star-based networks for distributed control systems, the reliability benefits of this topology have not been appropriately quantified. Thus, in previous work we modelled, by means of Stochastic Activity Networks (SANs), the system reliability that can be achieved with CAN and a simplex star we proposed for it called CANcentrate. These models quantitatively compared the system reliability of CAN and CANcentrate when permanent hardware faults occur. However, the strategy that underlies these models was not suitable, in terms of computation time, for ReCANcentrate, a replicated star we more recently proposed for CAN. This paper thoroughly describes a new model that overcomes this drawback and, then, demonstrates that ReCANcentrate can boost the system reliability. Although this paper addresses CAN-based networks, the results herein presented can be extrapolated to other field-bus technologies.

1 Introduction

Controller Area Network [1] (CAN) is a field bus communication protocol widely used in distributed control systems, mainly due to its electrical robustness, low cost and good real-time properties. Nevertheless, CAN presents some shortcomings related to dependability [2] that may discourage its use in highly-dependable applications, e.g. x-by-wire control systems. One of its major limitations is that it relies on a non-redundant bus topology with scarce error-containment and fault-tolerance mechanisms. To overcome this limitation, we developed two CAN-compliant star topologies called CANcentrate [3] and ReCANcentrate [4]. The first one is a simplex star aimed at improving error containment. Its active hub includes novel mechanisms to contain errors at their ports of origin. The second one, ReCANcentrate, is a replicated star that includes two hubs, similar to the one of CANcentrate, to further provide fault tolerance.

Certainly, when compared with buses, stars are inherently more resilient to spatial-proximity and common-mode failures, can yield better error containment and, in particular, replicated stars can even provide some fault tolerance [5]. Thus, the interest in using them instead of buses has also been growing in other technologies, e.g. in in-vehicle systems, such as with TTP/C and FlexRay [6]. However, it is not a priori evident that stars are more dependable than buses, since stars include more hardware components, thereby increasing the probability that faults and errors occur. In fact, as far as we know, no other author has appropriately quantified the dependability benefits stars achieve when compared with buses. Previous mathematical analyses abstract away many important details such as the different hardware components' failure modes [7]. Moreover, fault injection tests that quantitatively demonstrate the better stars' error-containment, e.g. [8], do not clarify if this better error containment actually yields a dependability improvement.

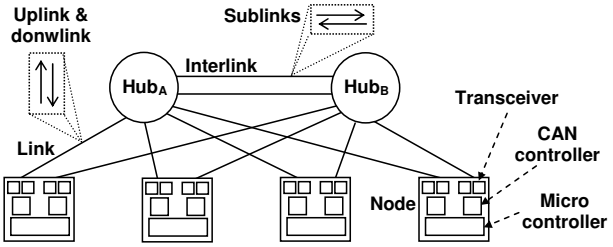

Figure 1. ReCANcentrate architecture

For these reasons, the general framework within which this work is included aims at modelling different bus and star networks to quantify the effect that the above-mentioned advantages of star topologies, e.g. better error-containment, have on the system dependability. Specifically, among all the attributes of dependability we are interested in reliability, which can be defined as the probability with which a system continuously delivers its intended service throughout a given interval of time [9].

Note that in a distributed control system faults prevent nodes from operating or communicating, thereby jeopardizing the service they are intended to provide. A star can play a key role in improving reliability of these systems, since its error-containment capabilities reduce the number of nodes that are affected by a specific fault. However, the actual benefits that a star yields in terms of reliability strongly depend on the system's ability to correctly deliver its service when only a subset of nodes can still operate and communicate among them. Thus, in order to assess the reliability benefits of stars, we differentiate between what we call *non-fault-tolerant/accepting* (NFTA) and *fault-tolerant/accepting* (FTA) systems. NFTA systems can only deliver their services as long as all their nodes are not faulty and can communicate with each other. The reliability of

 doi:10.1109/WFCS.2010.5548633

these systems, i.e. the *non-fault-tolerant/accepting system reliability* (NFTAR), can thus be understood as the *probability with which all nodes of a system can correctly operate and communicate with each other throughout a given interval of time*. In contrast, FTA systems are those that can correctly operate while accepting or tolerating the failure or the disconnection of up to k of N nodes. Examples of these systems could be the intra-building communication system of a hotel, in which the main objective is to provide service to the maximum number of rooms thereby *accepting that a certain number of nodes could be not capable to compute or communicate*, as well as highly reliable distributed control systems that *tolerate* faulty or disconnected nodes by replicating them. We refer to the reliability of a FTA system as the *fault-tolerant/accepting system reliability* (FTAR). More specifically, we can formally define this reliability in terms of k as the FTAR_k, which stands for the *probability with which at least $N-k$ of the N nodes of a system can correctly operate and communicate among them throughout a given interval of time*.

In previous work, [10] and [7], we modelled using SANs (Stochastic Activity Networks, a generalization of Stochastic Petri Nets [11]) the reliability of a system relying on a CAN bus and a CANcentrate star when permanent hardware faults can occur. We showed that the bigger likelihood of faults in a simplex star topology reduces the NFTAR. Fortunately, it also quantitatively corroborated that the error-containment capabilities of a simplex star, such as CANcentrate, can yield benefits in terms of FTAR_k when $k = 1$ (formerly referred to as PNS). Moreover, in [12] we presented a preliminary work whose first results indicate that a replicated star topology like ReCANcentrate can improve not only the FTAR_1 but also the NFTAR.

It is noteworthy that in this latter work we proposed SANs models that follow a different strategy than the ones presented in [10] and [7], as the original modelling approaches were extremely inefficient in terms of computation time for ReCANcentrate. However, [12] merely outlines these new models and a more detailed description is still needed. This paper provides such a description, focusing on the case of ReCANcentrate. We believe that the model presented here is not only relevant in order to completely understand our work, but also to help other researchers to evaluate the reliability that can be achieved with other communication topologies. Furthermore, this paper updates the results of [12], which still were not definitive since some models' features therein presented were preliminary. Also note that this paper's intention is not to calculate absolute figures of reliability. Instead, it is devoted to fairly comparing the reliability that can be achieved with CAN, CANcentrate and ReCANcentrate (when permanent hardware faults occur), by means of models that include parameters for all the relevant aspects of a system relying on these infrastructures. Moreover, these models' parameters allow to both refine the results as more system's details are known and carry out sensitivity analyses with respect to each one of these parameters.

2 ReCANcentrate basics

CAN relies on two fundamental properties: the *dominant/recessive transmission* and the *in-bit response* [1]. The former means that the medium implements a wired-AND function of all nodes' contributions, so that a dominant bit '0' prevails over a recessive bit '1'. The second one guarantees that nodes quasi-simultaneously observe every single bit on the channel.

These properties are kept in ReCANcentrate [4], whose architecture is sketched in Figure 1. It includes two hubs and each node is connected to each of them by a dedicated link containing an uplink and a downlink. Additionally, both hubs are interconnected by at least two interlinks each of which contains two independent sublinks, one for each direction. Each hub receives the contribution to each bit value of each node directly attached to it through the corresponding uplink. It couples all non-faulty node contributions with a logical AND function to calculate what we call the *hub contribution*. Each hub sends to the other one its own contribution, so that the resulting signal that each hub broadcasts to its own nodes is the one that results from coupling its own contribution with the contribution received from the other hub. Hubs perform this coupling within a fraction of the bit time, thereby creating a single logical broadcast domain since both hubs behave like one, transmitting the same value bit by bit in their downlinks.

Each node is constituted by commercial-off-the-shelf components only: two CAN controllers, four transceivers and a microcontroller (Figure 1). Each CAN controller is connected to only one hub using a transceiver for the uplink and another one for the downlink. The single broadcast domain allows nodes to easily manage the replicated traffic [13]. Basically, each node transmits through only one hub, while receiving from both hubs simultaneously.

Regarding the fault model, each hub is able to detect faults at nodes, links, interlinks or at the other hub that manifest as stuck-at-recessive, stuck-at-dominant or bit-flipping streams [3]. Each hub disables any permanently faulty contribution, thus isolating it at its port of origin. Moreover, each node can diagnose when a fault, e.g. in a link or a hub, prevents it from communicating through a given star. For that it basically uses the *Transmit Error Counter* (TEC) and the *Receive Error Counter* (REC) [1] of its CAN controllers. Whenever any of these counters reaches a given threshold, the node stops using the corresponding CAN controller for communicating.

3 Modelling assumptions

The assumptions we made for the models of CAN and CANcentrate were thoroughly explained in [7]. However, it is still needed to explain some additional assumptions that are exclusively related to specific ReCANcentrate aspects. Most assumptions are reflected as model parameters, thereby allowing to perform sensitivity analyses with respect to them. Anyway, so far all assumptions are made favoring whenever possible the CAN bus and always guaranteeing that results are not biased toward stars.

Concerning the networks' layout, we assume the length of each link (and interlink) to be half the total CAN bus length. This is pessimistic for the stars, since a star could use much less cable to cover the area occupied by the nodes the bus interconnects. Additionally, we choose a daisy chain configuration as the way to attach nodes to the bus. This is the most optimistic case for CAN, since it needs no stubs and the least number of connectors [7].

Regarding what are the components that constitute the system, we consider the following ones: microcontrollers, CAN controllers, transceivers, memory ICs, oscillators, PCBs, segments of cable, connectors, network terminations, and ASICs (in the case of the hubs).

Concerning component failures, we suppose that they are independent. The *Time To Failure* (TTF) distribution, F(t), of each component is considered to be exponential with mean $1/\lambda$, where λ is the failure rate expressed in number of failures per hour. We assume that F(t) is Non-Defective [14]. If the failure time is X and F(t) is Non-Defective, then the probability with which the component fails at or before time t, F(t) = Prob($X \leq t$), is 0 when $t = 0$, $1 - e^{-\lambda \cdot t}$ when $0 < t < \infty$, and 1 when $t = \infty$. To calculate the components' failure rates, we use a software based on the MIL-HDBK-217 prediction standard [15].

We assume that a component failure can basically manifest, *from the channel point of view*, in the following modes: stuck-at-recessive, stuck-at-dominant or bit-flipping [3]. Since there is not a real consensus on the components' failure mode proportions, we consider them as equiprobable, except for two node components. On the one hand, we assume that a microcontroller can only cause a stuck-at-recessive, since it cannot manipulate the CAN controller in a way that leads that controller to permanently deliver dominant or bit-flipping values. On the other hand, a CAN controller has a complex internal structure that includes some internal modules whose failure cannot lead the CAN controller to deliver stuck-at-dominant/bit-flipping streams. After analyzing its structure, we came to the conclusion that the proportions with which a CAN controller exhibits a stuck-at-recessive and a stuck-at-dominant/bit-flipping failure are around 66.6% and 16.6% respectively. Due to space limitations, we cannot explain this analysis in detail. Anyway, these proportions' values are only relatively important for the presented results, since they are defined as model's parameters and we plan to carry out sensitivity analyses with respect to them.

Additionally, we consider that a component can also provoke a failure mode we call an *out-of-fault-model* (*ofm*) failure. This mode gathers all faults that are beyond our fault model and that, thus, cannot be treated, e.g. a CAN controller that fails in a *babbling-idiot* manner by continuously sending a message stored in its transmission buffer. In principle, we consider that components exhibit a 0% of ofm failures. Otherwise, an ofm greater than 0% would prevent us from analyzing the reliability achievable by the CAN bus and our stars, since their contribution to the reliability would be masked by the effect of faults they do not address. In fact, in order to fully benefit from our stars, the system should include mechanisms that deal with ofm faults, since it is impossible to increase the system reliability by improving only one of its parts. In this sense, a ofm proportion of 0% is equivalent to assuming that these mechanisms are 100% effective. Also note that some of these mechanisms could be included in our stars. However, our hubs do not present them, since their design is application-independent and the knowledge necessary to address most of these faults, e.g. babbling idiot ones, is strongly related to the application. Anyway, subsequent sensitivity analyses with respect to the ofm proportion of a specific component will allow to analyze the importance of including the corresponding additional mechanisms.

Finally, note that the system reliability is extremely sensitive to the coverage of its fault-tolerance mechanisms. We distinguish between the fault-tolerance mechanisms of the system itself and the fault-tolerance mechanisms of the communication subsystem (CAN, CANcentrate or ReCANcentrate) it relies on. The first set of fault-tolerance mechanisms refers to the ability of the system to actually accept or tolerate the failure or the disconnection of a node, provided that the system is able to accept or tolerate such a situation. We refer to the coverage related to these mechanisms as the *sysFauTolCov*. NFTA systems present a *sysFauTolCov* of 0%, whereas for FTA systems we assume a *sysFauTolCov* of 100%. We believe that a 100% is the most representative value for FTA systems. On the one hand, it is the coverage of FTA systems that intrinsically accept the failure or disconnection of up to k nodes. On the other hand, the effectiveness of the fault-tolerance mechanisms of ultra-reliable FTA systems (in which we are specially interested) must be virtually of 100%, e.g. [16].

Concerning the coverages of the communication subsystem's fault-tolerance mechanisms, next we summarize the most important ones. On the one hand, we consider different *error-containment coverages*, where such a coverage can be defined as the probability of detecting and isolating a fault included in our fault model, given that this fault occurs. Broadly speaking, we define error-containment coverages for the capacity of the CAN controller to isolate faults happening at the media, itself or the rest of its node and that compel it to deliver errors, as well as for the ability of a hub to isolate faults at its uplink and interlink ports. Basically, we assume a coverage of 100% and 95% for stuck-at and bit-flipping faults respectively [7]. On the other hand, only for ReCANcentrate, we model the coverage of the fault-tolerance mechanisms the nodes are provided with. We define two coverages called *connCov* and *decConnCov* for characterizing the probability with which a node can communicate using only one star when a fault prevents it from communicating through the other star. The former specifies this probability when the hubs are coupled and its default value is 95%, as the corresponding node's fault-tolerance mechanisms are quite simple [13]. The second one applies when the hubs are decoupled and its default value is the 0%, since we have not

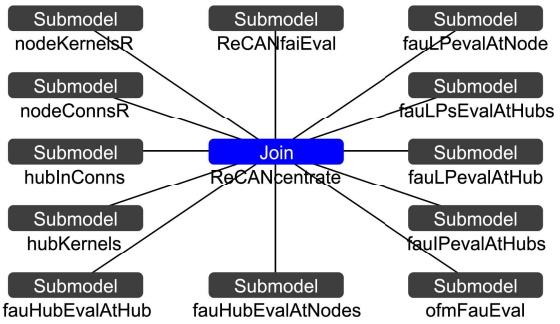

Figure 2. ReCANcentrate model

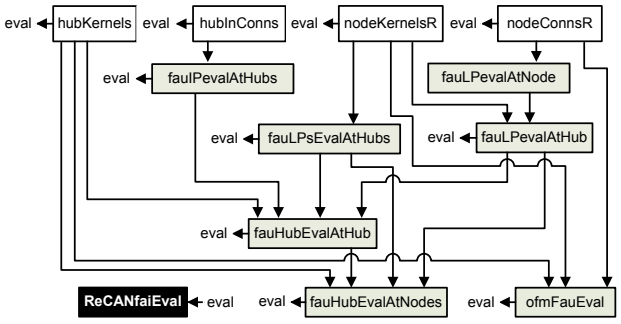

Figure 3. Coverage process

proposed any mechanism that allows a node to tolerate its disconnection from one hub in such a situation. The last nodes' fault-tolerance coverage quantifies their ability to communicate with each other using two independent stars when the hubs become decoupled. We assume a 0% for it because we have not yet thoroughly proposed any mechanism to tolerate hub decouplings.

4 Modelling rationale

In order to keep a reasonable model complexity, we do not model the failure of single components, but of groups of them we call *entities*. We differentiate the following types of entities: (1) the *Node Core*, which basically includes the node microcontroller, memory ICs, sockets and a PCB area; (2) the *Controller*, which represents a CAN controller, its socket and PCB area; (3) the *Node IO* and *Hub IO*, which include the components needed to interface a node and a hub port, respectively, with the medium, i.e. one CAN transceiver and its corresponding socket and PCB area; (4) the *Attachment*, which includes a CAN cable and a pair of straight connectors (it represents the uplink/downlink/sublink in the stars, whereas in the CAN bus it constitutes a given bus section connecting two adjacent nodes); (5) the *Termination*, which is merely a resistor (a pair of Terminations are used to prevent signal reflections in the bus line, as well as at each uplink, downlink and sublink); (6) and the *Hub Core*, which comprises the IC that implements the hub itself (except its transceivers), its socket, an oscillator and the necessary PCB area. This abstraction allowed us to calculate basic entities' dependability parameters, such as failure rates, failure modes and different error-containment coverages, in a simple way.

Regarding the basic structure of the models of CAN, CANcentrate and ReCANcentrate, we built each one of them as a composition of different SANs submodels that share specific places by means of the so called *join* primitive [11]. Figure 2 depicts the whole model of ReCANcentrate. More specifically, we distinguish among what we call *regions* submodels, *coverage* submodels and the *evaluator* submodel. A regions submodel is devoted to modelling faults happening at a particular type of *region*. We define a *region* as an ensemble of entities placed at a specific error-containment region of the system. In this way, if we consider that each Hub Core is an error-containment region, then there is one dedicated regions submodel that models faults happening in any Hub Core.

In principle, we consider the same types of regions for CAN, CANcentrate and ReCANcentrate. However, some regions do not appear in a bus-based or, conversely, in a star-based system. Thus, some types of regions are not modelled in the case of a bus or a star. For instance, a system that relies on a CAN bus does not include any Hub Core and, hence, its model does not have the corresponding regions submodel. Next we specify the types of regions we have considered but, for the sake of succinctness, we exclude those that do not appear in a ReCANcentrate-based system (it is easy to see the correspondence between each region type and its regions submodel in Figure 2): (1) the *Node Kernel*, which only includes a Node Core; (2) the *Node Connection*, which comprises all entities a node needs to be connected to a given hub, i.e. one Controller, two Node IOs, two Attachments, two Hub IOs and four Terminations); (3) the *Hub Interconnection*, which includes all the entities that constitute a given sublink (one Attachment, two Hub IOs and two Termination entities); and (4) the *Hub Kernel*, which includes a Hub Core.

Note that in our previous models presented in [10] and [7], we used a dedicated SANs submodel for modelling faults happening at just one entity or a group of entities of the same type. In contrast, the models presented here reduce the number of necessary SANs submodels, since they include only one submodel per region type. The way in which each regions submodel still models the failure of every individual entity of each one of the regions it represents will be explained later on. At this point, simply note that when a given entity fails, the corresponding regions submodel decides the failure mode with which the fault manifests and initiates an external sequential process that models how the errors generated by that fault propagate, how they are contained and, if appropriate, how the fault is tolerated. We refer to this process as the *coverage process*. This process is basically carried out by different coverage submodels, each of which represents, more or less, a specific error-containment or fault-tolerance mechanism.

Figure 3 depicts a diagram of the coverage process of ReCANcentrate. The regions and coverage submodels are represented by clear and shaded boxes respectively. Depending on whether or not a given error-containment or fault-tolerance mechanism succeeds, the corresponding coverage submodel finishes the process (this is indicated

by an arrow connected to the word *eval*) or compels another coverage submodel to proceed with the evaluation. The process continues until the errors are contained (or the fault is tolerated), or until all error-containment and fault-tolerance mechanisms are exhausted.

During the coverage process and depending on whether or not the fault is successfully isolated and/or tolerated, coverage submodels update a set of places that make it possible to calculate how many nodes can communicate among them. These places are then used by the evaluator submodel, e.g. *ReCANfaiEval* in the case of ReCANcentrate (represented by a black box in Figure 3), to decide if the entire system fails. As will be explained, this decision depends on whether the measure of reliability is the NFTAR or the FTAR_k. When the evaluator submodel considers that the system is faulty, it writes a token at a place called *genFai*. This place is shared among all submodels, which stop evolving when they observe the referred token, thereby significantly reducing the size of the state space of the underlying stochastic process.

Finally, in order to quantify the reliability, we associate a rate reward variable [11] with the marking of the place *genFai*. We call this variable *probNonGen* and its expressions is $1.0 - genFai \rightarrow Mark()$. Where $genFai \rightarrow Mark()$ represents the marking of the place *genFai*. Since the marking of *genFai* can be only 0 or 1, the value of the variable is 1 as long as the system is not faulty. Thus, once the model is analytically solved, the value of this variable at a particular point in time indicates the system reliability (NFTAR or FTAR_k) at that instant of time.

5 ReCANcentrate model

Due to space limitations, we cannot explain our models in detail. Instead, we describe the structure and the role of some representative submodels of ReCANcentrate: *ReCANfaiEval*, *nodeConnsR* and *fauLPevalAtNode*, which constitute relevant submodels of one of the possible execution paths of the coverage process in Figure 3. However, before continuing with their description, let us point out some remarks. First of all, for the sake of clarity, let us refer to the two hubs of ReCANcentrate as *hub A* and *hub B* from now on.

Second, we will call each hub's port to which a node is connected a *branch*. Note that we consider that a branch is faulty when the node connected to that port cannot communicate through it. This implies that a branch fails when: (1) the Node Kernel of the node that uses the branch is faulty; (2) the Node Connection corresponding to the branch fails; or (3) the hub to which the branch belongs is faulty.

Third, we consider that the hubs are decoupled when both of them are not faulty, but there are not enough Hub Interconnections to carry the contribution of the hub A to the hub B or viceversa, i.e. when each hub represents an independent communication domain. Submodels model the hubs' coupling/decoupling by means of a shared place called *decHubs*. The hubs are decoupled if there is a token in this place, whereas they are coupled otherwise.

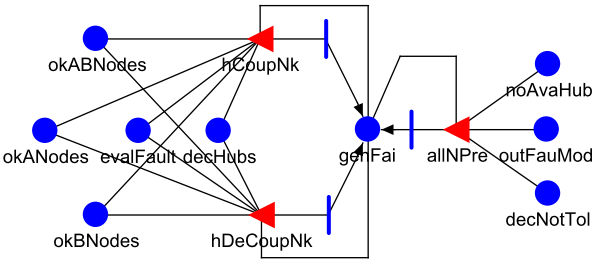

Figure 4. ReCANfaiEval submodel

Finally, it is noteworthy that many submodels base their own decisions on the capacity of each node to communicate through the hub A and on its capacity to do it through the hub B. This is because the impact of each fault on the overall system depends on these capabilities. In order to characterize the communication capacities of each node, submodels share a set of places called: *okABNodes*, *okANodes*, *okBNodes*, *stopANodes* and *stopBNodes*. The marking of these places respectively represent the number of nodes of the following categories. (1) *okAB* nodes: they are nodes whose Node Kernel is not faulty (the nodes are operative) and that can communicate through hub A and hub B. (2) *okA* and (3) *okB* nodes: they are operative nodes that tolerated a fault that prevented them from communicating through one hub, so that they only communicate through hub A and hub B respectively. (4) *stopA* and (5) *stopB* nodes: they are operative nodes that could still communicate through hub A and hub B respectively, but that stop communicating because they did not tolerate the fault that prevented them from communicating through the other hub. Note that this classification leaves aside non-operative nodes (nodes whose Node Kernel is faulty), as well as operative nodes affected by faults that prevent them from communicating through both hubs. This is because we do not need to differentiate between these two types of nodes, and the quantity of them can be calculated by subtracting the number of nodes of the mentioned categories from the total number of nodes .

5.1 *ReCANfaiEval* submodel

ReCANfaiEval is the evaluator submodel of the ReCANcentrate's model. For the sake of clarity, Figure 4 depicts a simplified version of this submodel, which does not show how it takes into account the value of *sysFauTolCov* (see Section 3). As can be seen in this figure, *ReCANfaiEval* includes three input gates. Each one of them detects a different type of situation that implies an overall failure. When one of these situations happens, the appropriate input gate becomes enabled and its corresponding instantaneous activity sets a token in the place *genFai*.

Gates *hCoupNk* and *hDeCoupNk* aim at detecting situations where nodes communicate, but in which the number of them that can do so is not enough for the system to deliver its service. The former gate detects this general failure when the hubs are coupled, whereas the other one acts when the hubs are decoupled. For instance, the expression of *hCoupNk* is:

$genFai \rightarrow Mark() == 0$ *and*
$decHubs \rightarrow Mark() == 0$ *and*
$(okABNodes \rightarrow Mark() + okANodes \rightarrow Mark() +$
$okBNodes \rightarrow Mark()) < (numNodes - kSevere)$ *and*
$evalFault \rightarrow Mark() == 0$

The first term ensures that the input gate is not enabled if an overall failure has already occurred. The second one guarantees that *hCoupNk* remains disabled if the hubs are decoupled. The third term compares the number of nodes that operate and communicate among them, i.e. the quantity of *intercommunicating nodes*, with the minimum number of intercommunicating nodes the system needs in order to deliver its service. Note that when both hubs are coupled, an operative node can communicate as long as it can transmit and receive through at least one hub, no matter which. Thus, the number of intercommunicating nodes can be calculated by simply adding up the marking of the places *okABNodes*, *okANodes* and *okBNodes*. Once this number is obtained, *hCoupNk* compares it with the value of *numNodes* − *kSevere*. The parameter *numNodes* specifies the total number of nodes, whereas *kSevere* is the maximum number of non-intercommunicating nodes a system can accept or tolerate. Note that the parameter *kSevere* allows to configure the model to measure the NFTAR and different degrees of FTAR_k. For instance, if one is interested in measuring the NFTAR, the minimum number of intercommunicating nodes is the total number of nodes and, thus, *kSevere* must be specified as 0. Finally, the fourth term refers to the marking of the place *evalFault*. As will be explained, a token is set in this place to indicate that the coverage process is ongoing. Gate *hCoupNk* remains disabled as long as *evalFault* contains a token, since the marking of places like *okABNodes*, *okANodes* and *okBNodes* are incoherent until the coverage process ends.

Until this point we have explained the role of the gates *hCoupNk* and *hDeCoupNk*, focusing on the former. Regarding gate *allNPre*, it is devoted to detecting situations in which all nodes are prevented from communicating. Gate *allNPre* becomes aware about any of these situations when it receives a token in *noAvaHub*, *outFauMod* or *decNotTol* (see Figure 4). Specifically, *ReCANfaiEval* receives a token in *noAvaHub* when there is not any hub available for communicating, i.e. when both hubs are faulty. Similarly, it receives a token in *outFauMod* when an ofm fault occurs and its errors pollute all the system. Finally, a token is received in *decNotTol* when the hubs become decoupled and nodes were not able to tolerate this situation, i.e. when nodes cannot communicate using both hubs independently.

5.2 *nodeConnsR* submodel

NodeConnsR is the regions submodel responsible for modelling faults happening at any Node Connection region of ReCANcentrate. The other regions submodels model faults following the same strategies we propose in this section. The structure of *nodeConnsR* is depicted in Figure 5.

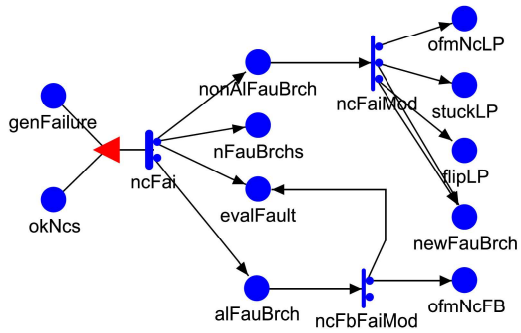

Figure 5. nodeConnsR submodel

The marking of *okNcs* is the number of Node Connections that are not faulty. Its initial value is two times the number of nodes ($2 \cdot numNodes$) since each node initially has two non-faulty Node Connection regions (one per hub). The activity *ncFai* models the Time To Failure (TTF) distribution of all the surviving Node Connections as a whole. Note that this distribution, we call $F_{NodeConns}(t)$, represents the time that elapses until a fault occurs in any entity of any surviving Node Connection, i.e. it represents the TTF of every individual entity of every non-faulty Node Connection region. In order to obtain the expression of $F_{NodeConns}(t)$, note that all surviving Node Connections have exactly the same number and types of entities and that each one of these entities independently fails following an exponentially distributed TTF. Thus, if we suppose that a Node Connection is composed of E entities and that the failure rate of each one of them is denoted by λ_i, with $i \in [1, E]$, then $F_{NodeConns}(t)$ can be written as:

$$F_{NodeConns}(t) = 1 - e^{-\lambda_{NodeConns} \cdot t}$$

where $\lambda_{NodeConns}$ is the rate with which faults occur in any surviving Node Connection. More specifically, if *okNcs*→*Mark()* is the marking of *okNcs*, i.e. the number of surviving Node Connections, then:

$$\lambda_{NodeConns} = okNcs \rightarrow Mark() \cdot \sum_{i=1}^{E} \lambda_i$$

Once activity *ncFai* fires, it selects one of its two cases, which are represented as circles. The first case (the upper one) represents a fault that affects a Node Connection located in a non-faulty branch; whereas the second one models a fault happening in a Node Connection placed at a branch that was already faulty, i.e. a Node Connection attached to an isolated hub port. The proportion with which *ncFai* chooses each case reflects the probability that the Node Connection is placed (or not) at an already faulty branch. For instance, the first case's proportion is:

$$\frac{nBrchs - nFauBrchs \rightarrow Mark()}{okNcs \rightarrow Mark()}$$

Parameter *nBrchs* indicates the number of total branches of ReCANcentrate, whereas *nFauBrchs* is a place that submodels share and whose marking indicates the number of branches that are faulty. Note that the Node Connection of a non-faulty branch is always non-faulty (otherwise the branch would be faulty). Therefore, the above expression

can calculate the probability that a surviving Node Connection is placed in a non-faulty branch by simply dividing the number of surviving branches by the number of Node Connections that were not faulty. Likewise the proportion of the second case, which is the probability that the surviving Node Connection that fails is placed at an already faulty branch, is:

$$\frac{(okNcs \rightarrow Mark() - (nBrchs - nFauBrchs \rightarrow Mark()))}{okNcs \rightarrow Mark()}$$

What activity *ncFai* does after firing depends on the specific case it chooses. The first case adds a token to three places. First, it adds a token to *nFauBrchs* in order to record that a new branch is faulty. Second, it sets a token in *evalFault* and *nonAlFauBrch* in order to initiate the coverage process that will evaluate how the errors generated by the fault are treated. The token set in *evalFault* inhibits *ReCANfaiEval* until the coverage process is finished. The token of *nonAlFauBrch* enables the activity *ncFaiMod*, which instantaneously fires.

This activity decides the way in which the Node Connection manifests at the uplink hub port. The first case corresponds to a Node Connection that exhibits an ofm failure. Although ofm errors cannot be contained, they do not necessarily propagate throughout the system, e.g. it depends on whether or not the hubs are coupled. Thus, *ncFaiMod* sets a token in *ofmNcLP* to compel the *ofmFauEval* submodel to evaluate if the ofm errors pollute the whole system depending on different circumstances. The second and third cases of *ncFaiMod* model that the fault manifests as stuck-at and bit-flipping and they respectively set a token in *stuckLP* and *flipLP*. The coverage submodel *fauLPevalAtHub* will use these two places to know the Node Connection's failure mode. This submodel, which is activated later on (Figure 3), evaluates if the corresponding hub isolates the fault. In this sense, note that *nodeConnsR* only includes one place for both stuck-at-recessive/dominant failures. This is because a hub isolates both types of faults with a perfect coverage and, thus, it is not necessary to differentiate between them. Also note that the second and third cases of *ncFaiMod* set a token in *newFauBrch* to compel *fauLPevalAtNode* to continue before *fauLPevalAtHub*.

As concerns the second case of *ncFai*, it sets a token in *alFauBrch* to indicate that the faulty Node Connection is placed at an already faulty hub port. This token activates *ncFbFaiMod*, which is analogous to the activity *ncFaiMod*. If *ncFbFaiMod* decides that the fault is included in our fault model it selects the case that is left unconnected, as no more actions are needed when a non-ofm fault occurs in an already faulty, and thus isolated, branch. Otherwise, *ncFbFaiMod* sets a token in *ofmNcFB* to activate the *ofmFauEval* submodel. This is because an ofm Node Connection may lead its Node Kernel to fail in an ofm manner and send errors through its other Node Connection.

The proportion with which *ncFaiMod* and *ncFbFaiMod* choose each case reflects the probability with which the Node Connection that fails exhibits the corresponding type of fault/s. Thus, at this point, it is necessary to clarify how we calculate the proportions with which a given region (a Node Connection in this case) manifests each one of its failure modes. First, note that we consider that faults are not near coincident in time, so that the region failure mode is determined assuming that only one of its entities has failed. Second, since a region failure can be provoked by any of its entities, the proportion with which a region manifests a given failure mode reflects the proportion with which each one of its entities exhibits that failure mode. In this sense, let us consider a region constituted by E entities that exhibit M different failure modes. Moreover, imagine that a given failure mode is called fm_j with $j \in [1, M]$, so that the proportion with which an entity i exhibits the failure mode fm_j is denoted by $fmp_{i,j}$ with $fmp_{i,j} \in [0, 1]\ \forall i \in [1, E]$ and $\forall j \in [1, M]$. Then, the proportion with which a region exhibits the failure mode fm_j is a function that depends on the failure rates of its constituent entities, λ_i, as well as on the proportions with which each one of these entities exhibits that failure mode, $fmp_{i,j}$. We call this function Fmp_j and it is denoted by $Fmp_j(\lambda_1, ..., \lambda_E, fmp_{1,j}, ..., fmp_{E,j})$. In order to obtain the detailed expression of this function we can proceed as follows. First, note that the TTF of a region can be expressed as $F_{Region}(t) = (1 - e^{-\lambda_{Region} \cdot t})$ (1); where λ_{Region} is the failure rate of the region. Thus, the probability that a region fails exhibiting the failure mode fm_j is $(1 - e^{-\lambda_{Region} \cdot t}) \cdot Fmp_j$. Moreover, since a region cannot exhibit more than one failure mode simultaneously, we can write $F_{Region}(t) = (1 - e^{-\lambda_{Region} \cdot t}) \cdot \sum_{j=1}^{M} Fmp_j$ (2). Second, note that we can also write λ_{Region} in terms of the entities' failure rates and failure mode proportions as $\lambda_{Region} = \sum_{i=1}^{E} \lambda_i = \sum_{i=1}^{E} \left(\lambda_i \cdot \sum_{j=1}^{M} fmp_{i,j}\right)$ (3). At this point, we can use equation 3 to expand equation 1 in such a way that we obtain an expression from which we can infer the detailed form of Fmp_j in terms of the entities' failure rates and failure mode proportions. Specifically, $F_{Region}(t) = (1 - e^{-\lambda_{Region} \cdot t}) = (1 - e^{-\lambda_{Region} \cdot t}) \cdot \frac{\lambda_{Region}}{\lambda_{Region}} = (1 - e^{-\lambda_{Region} \cdot t}) \cdot \sum_{j=1}^{M} \left(\frac{\sum_{i=1}^{E} \lambda_i \cdot fmp_{i,j}}{\lambda_{Region}}\right)$. If we compare this result with equation (2), we deduce that Fmp_j (4) is:

$$Fmp_j = \frac{\sum_{i=1}^{E} \lambda_i \cdot fmp_{i,j}}{\lambda_{Region}} = \frac{\sum_{i=1}^{E} \lambda_i \cdot fmp_{i,j}}{\sum_{i=1}^{E} \lambda_i} \forall j \in [1, M]$$

In other words, the proportion with which a region exhibits a given failure mode is the weighted arithmetic mean of the proportions with which each one of its entities manifests that failure mode. Specifically, the proportion with which each entity exhibits the failure mode is weighted considering the contribution of its failure rate to the failure rate of the region.

However, when we use Equation 4 for calculating the cases' proportions of *ncFaiMod* in the submodel of a Node Connection, it is necessary to take into account that the CAN controller placed at the Node Connection is able to diagnose faults happening at other entities of this region and, then, to stop operating in order to prevent error prop-

agation. This implies that a given percentage of an entity's failure that lead the Node Connection to manifest as stuck-at-dominant, as well as a given percentage of that entity's failure that lead the Node Connection to manifest as bit-flipping actually will lead the Node Connection to manifest as stuck-at-recessive. Specifically, these percentages are the coverages with which the CAN controller contains stuck-at-dominant and bit-flipping errors generated by that entity respectively. In order to reflect the error-containment capabilities of the CAN controller in the calculus of the Node Connection's failure mode proportions, we basically apply Equation 4 as follows (5):

$$Fmp_{str} = \frac{\sum_{i=1}^{E} \lambda_i \cdot fmp_{i,str}}{\sum_{i=1}^{E} \lambda_i} + \frac{\sum_{i=1}^{E} \lambda_i \cdot fmp_{i,std} \cdot stdCov_i + \lambda_i \cdot fmp_{i,flip} \cdot flipCov_i}{\sum_{i=1}^{E} \lambda_i}$$

$$Fmp_{std} = \frac{\sum_{i=1}^{E} \lambda_i \cdot fmp_{i,std} \cdot (1 - stdCov_i)}{\sum_{i=1}^{E} \lambda_i}$$

$$Fmp_{flip} = \frac{\sum_{i=1}^{E} \lambda_i \cdot fmp_{i,flip} \cdot (1 - flipCov_i)}{\sum_{i=1}^{E} \lambda_i}$$

where now Fmp_{str}, Fmp_{std} and Fmp_{flip} are the proportions with which the Node Connection exhibits a stuck-at-recessive, a stuck-at-dominant and a bit-flipping failure respectively; $fmp_{i,str}$, $fmp_{i,std}$ and $fmp_{i,flip}$ are the proportions with which entity i manifests as stuck-at-recessive, stuck-at-dominant and bit-flipping respectively; and $stdCov_i$ and $flipCov_i$ are the coverages with which the CAN controller detects and isolates the entity i when it manifests as stuck-at-dominant and bit-flipping respectively. Note again that in a star it is not necessary to differentiate between a stuck-at-recessive/dominant failure. Thus, the proportion of the second case of *ncFaiMod* is $Fmp_{str} + Fmp_{std}$, whereas the proportion of the third one is Fmp_{flip}.

5.3 *fauLPevalAtNode* submodel

As said in previous section, when a non-faulty Node Connection fails in a way included in our fault model and causes the failure of a branch that was not already faulty, *nodeConnsR* activates the *fauLPevalAtNode* coverage submodel. This submodel evaluates if the corresponding node tolerates the failure of that connection and, thus, if it is able to continue communicating using its other Node Connection. When *fauLPevalAtNode* finishes, it compels *fauLPevalAtHub* to proceed, in order to assess if the corresponding hub isolates the faulty Node Connection (see Figure 3).

Note that the Node Connection that fails and provoked the execution of *fauLPevalAtNode* was placed in a non-faulty branch (the activity *ncFai* of *nodeConnsR* chose its first case) and, thus, the Node Connection belongs to either an okAB, an okA, an okB, a stopA or a stopB node. Since only okAB nodes have the ability to tolerate the failure of a Node Connection, *fauLPevalAtNode* must clarify which is the type of node the Node Connection that fails belongs to. This is done in two steps. The first one is carried out

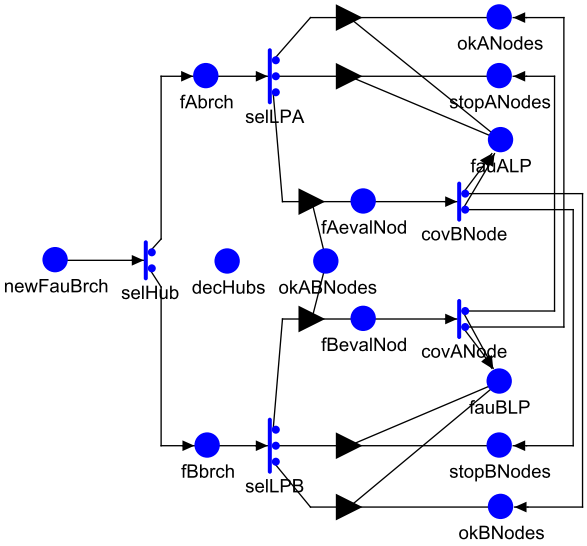

Figure 6. fauLPevalAtNode submodel

by activity *selHub* (see Figure 6), which instantaneously fires to decide whether the Node Connection that fails is placed in a non-faulty branch of the hub A (first case) or in a non-faulty branch of the hub B (second case). In order to calculate these cases' proportions we simply divide the favorable possibilities by the total number of them. For instance, the first case proportion is calculated as:

$$(okABNodes \rightarrow Mark() + okANodes \rightarrow Mark() + stopANodes \rightarrow Mark()) \; / \; (2 \cdot okABNodes \rightarrow Mark() + okANodes \rightarrow Mark() + stopANodes \rightarrow Mark() \; + okBNodes \rightarrow Mark() + stopBNodes \rightarrow Mark())$$

The numerator is the number of Node Connections placed at non-faulty branches of the hub A. All these Node Connections are non-faulty, except the one that has failed. The denominator is the total number of Node Connections placed at non-faulty branches (including the one that has failed). Note that the marking of *okABNodes* is multiplied by 2, since this type of node has two Node Connections: one placed in a non-faulty branch of the hub A and another one located in a non-faulty branch of the hub B.

Once *selHub* decides which is the hub the Node Connection that fails is connected to, *fauLPevalAtNode* performs the second one of the two steps that determine which is the type of node that corresponds to this Node Connection. This is done by activities *selLPA* and *selLPB*. Since they are analogous to each other, let us focus on *selLPA*. It selects its first, second and third cases if the node is an okA, a stopA or an okAB node respectively. The proportions of these cases are calculated following the same strategy just explained: by dividing the number of favorable possibilities by the total number of them. For instance, the proportion of the first case is $okANodes \rightarrow Mark() / (okABNodes \rightarrow Mark() + okANodes \rightarrow Mark() + stopANodes \rightarrow Mark())$. Each case is connected to a dedicated *output gate* [11], which is a SANs primitive that

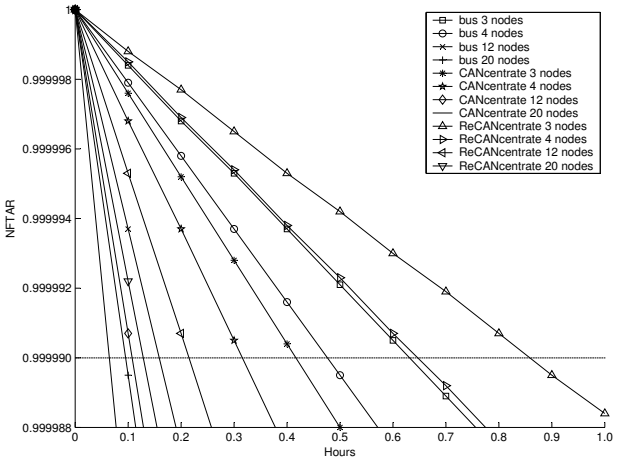

Figure 7. NFTAR vs number of nodes

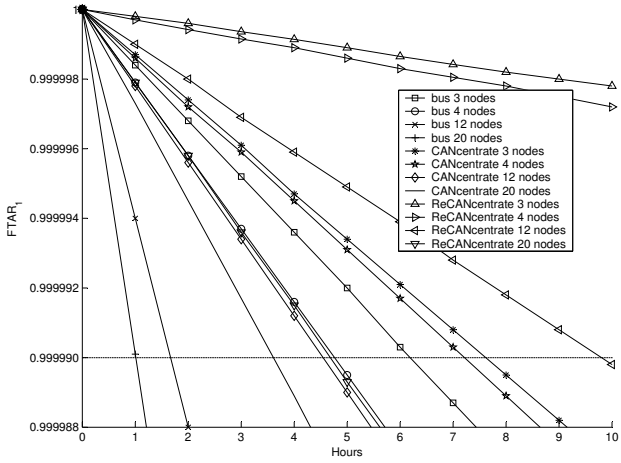

Figure 8. FTAR$_1$ vs number of nodes

allows performing complex marking changes. The output gate of the first case decreases in one unit the marking of *okANodes* to reflect that the node will no longer be an okA node. Likewise, the output gate of the second case decreases the marking of *stopANodes* in one unit. In addition, these gates set a token in place *fauALP*, which compels *fauLPevalAtHub* to continue with the coverage process.

Regarding the output gate of the third case of *selLPA*, it decreases the marking of *okABNodes* in one unit to reflect that the node will no longer be an okAB node. Then, it sets a token in *fAevalNod*, which compels *covBNode* to instantaneously evaluate whether or not the okAB node tolerates the Node Connection failure. The first case of *covBNode* adds a token to the marking of *okBNodes* because it represents the situation in which the okAB node tolerates the fault, so that it is still able to communicate using its connection to the hub B. The second case models the opposite situation and, thus, it increases the marking of *stopBNodes*. The proportions with which *covBNode* selects its first and second cases depend on whether or not the hubs are coupled. The node tolerates the Node Connection failure with a coverage of *connCov* if the hubs are coupled and with a coverage of *decConnCov* otherwise (see Section 3). In order to use the appropriate coverage, *covBNode* consults the marking of *decHubs*, which, as already mentioned, has a token if the hubs are decoupled. Finally, both cases of *covBNode* also set a token in *fauALP* to compel *fauLPevalAtHub* to proceed with the coverage process.

6 Quantitative assessment

In this section we quantitatively assess the system reliability that can be achieved with CAN, CANcentrate and ReCANcentrate. For this purpose, we use the reliability models proposed in this paper. In particular, as already explained in Section 3, we consider a 0% of ofm failures, which allows assessing what would be the reliability benefits of our stars in systems that include the appropriate mechanisms to deal with faults that are beyond the scope of these stars. However, it is noteworthy that result herein presented are likely to be lower bounds to the reliability achievable with CANcentrate and ReCANcentrate. This is because the value of all dependability parameters that characterize them, e.g. coverages, cables' failure rates, etc., have been determined considering assumptions that never favor them when compared with the CAN bus (see Section 3). Some value parameters are: coverages of 95% (almost in all cases); and failure rates (failures/hour) of $3.2531245 \cdot 10^{-6}$ for the *Node Kernel*, $1.9286278 \cdot 10^{-6}$ for the *Node Connection* in the CAN bus, $3.0241227 \cdot 10^{-6}$ for the *Node Connection* in the stars, $1.275596 \cdot 10^{-6}$ for a ReCANcentrate *Hub Kernel* with 3 ports, and $2.1193189 \cdot 10^{-6}$ for a ReCANcentrate *Hub Kernel* with 20 ports.

We compare the system *mission time* [17] achievable with CAN, CANcentrate and ReCANcentrate, i.e. the maximum amount of time during which a system that relies on them exhibits a reliability equal or greater than a certain value. Specifically, we analyze the system NFTAR and FTAR$_1$ as long as they are ≥ 0.99999. This value is just taken as a reference and it corresponds to the reliability required by a throttle-by-wire system [17].

Figure 7 depicts the NFTAR of a system that relies on CAN, CANcentrate and ReCANcentrate for different number of nodes. For any number of them, CAN yields a bigger NFTAR than CANcentrate. This was expected, since a CANcentrate-based system includes more hardware than a CAN-based one and a NFTA system cannot benefit from the error-containment provided by a simplex star topology (see Section 1). However, the reduction of mission time provoked by CANcentrate is not outstanding in absolute terms and is kept almost constant (around the 35%) for any number of nodes, e.g. for 3 nodes CAN and CANcentrate achieve 0.63 and 0.41 hours respectively. This figure also shows that the system NFTAR achievable with ReCANcentrate is higher than the one achieved with both CAN and CANcentrate for any number of nodes. The mission time improvement is relatively important, which demonstrates that the ReCANcentrate fault-tolerance mechanisms amply compensate its extra hardware.

Figure 8 compares the FTAR$_1$. On the one hand, it shows that CANcentrate achieves a higher FTAR$_1$ than CAN for any number of nodes. For instance, the mis-

sion times yielded by CAN and CANcentrate are, respectively, 6.2 and 7.6 hours for 3 nodes, as well as 1.0 and 3.6 hours for 20. This corroborates that reliability of FTA systems can be improved by means of the enhanced error-containment of a simplex star. Moreover, a higher number of nodes implies a bigger difference between the mission time achievable with CAN and CANcentrate. For instance, from the just mentioned mission times it can be inferred that CANcentrate improves the mission time of CAN around the 22% and the 260% for 3 and 20 nodes respectively. On the other hand, Figure 8 shows that ReCANcentrate further improves the reliability of FTA systems for any number of nodes, e.g. it yields mission times of 45.0 and 4.6 hours for 3 and 20 nodes respectively, which compared with CAN represent an improvement of the 626% and of the 460%. This demonstrates that to include fault-tolerance mechanisms in addition to error-containment features, by means of a replicated star like ReCANcentrate, can actually boost the $FTAR_1$.

7 Conclusions and future work

In this paper we proposed SANs' models that measure the reliability (when permanent hardware faults occur) of systems that rely on CAN, CANcentrate and ReCANcentrate. These models are complete as they include parameters for all the relevant aspects that can influence the system reliability. Moreover, these parameters allow to both refine the results as more system details are known and carry out sensitivity analyses with respect to different system's aspects. We used these models for comparing the reliability achievable with each one of the referred communication infrastructures. To fulfill this objective, it is worth noting that the system reliability cannot be increased by improving only one of its parts. Thus, we supposed that the system includes 100% effective mechanisms that deal with the faults that are beyond the error-containment and/or fault-tolerance capacities of CAN, CANcentrate and ReCANcentrate. Additionally, all model parameters that characterize these infrastructures were determined with special care not to favor the stars in the comparison. Thus, results herein presented are likely to be lower bounds to the reliability that can be achieved with the stars. Results quantitatively corroborate that the enhanced error-containment features of CANcentrate can improve reliability of FTA systems, whereas the additional fault-tolerance mechanisms of ReCANcentrate can really boost this reliability. In future work we will carry out sensitivity analyses with respect to different models' parameters such as the error-containment coverage of the hubs and the ofm proportion.

8 Acknowledgement

This work was supported in part by the Spanish Science and Innovation Ministry with grant DPI2008-02195, and in part by FEDER funding.

References

[1] ISO, "ISO11898. Road vehicles - Interchange of digital information - Controller Area Network (CAN) for high-speed communication", 1993.

[2] *Automotive Embedded Systems Handbook*, CRC Press. Edited by Nicolas Navet and Françoise Simonot-Lion, 2009.

[3] M. Barranco, J. Proenza, G. Rodríguez-Navas, and L. Almeida, "An Active Star Topology for Improving Fault Confinement in CAN Networks", *IEEE Transactions on Industrial Informatics, vol. 2, issue 2*, vol. 2, pp. 78–85, May 2006.

[4] M. Barranco, L. Almeida, and J. Proenza, "ReCANcentrate: A replicated star topology for CAN networks", *ETFA 2005. 10^{th} IEEE International Conference on Emerging Technologies and Factory Automation, Catania, Italy*, 2005.

[5] G. Bauer, H. Kopetz, and W. Steiner, "The Central Guardian Approach to Enforce Fault Isolation in the Time-Triggered Architecture", in *Proceedings of the The Sixth International Symposium on Autonomous Decentralized Systems (ISADS'03), Washington, DC, USA*, 2003, p. 37. IEEE Computer Society.

[6] H. Kopetz, "Fault Containment and Error Detection in TTP/C and FlexRay", Research Report 23, Vienna University Of Technology, TU Wien, August 2002.

[7] M. Barranco, J. Proenza, and L. Almeida, "Quantitative comparison of the error-containment capabilities of a bus and a star topology in CAN networks", *IEEE Transactions on Industrial Electronics*, 2010.

[8] A. Ademaj, G. Bauer, H. Sivencrona, and J. Torin, "Evaluation of Fault Handling of the Time-Triggered Architecture with Bus and Star Topology", *IEEE International Conference on Dependable Systems and Networks (DSN 2003), San Francisco*, Jun. 2003.

[9] M. L. Shooman, *Reliability of Computer Systems and Networks*, John Wiley & Sons, Inc., 605 Third Avenue, New York, USA, 2002.

[10] M. Barranco, J. Proenza, and L. Almeida, "First results of the assessment of the improvement of error containment achieved by CANcentrate", in *WFCS'06. IEEE Workshop on Factory Communication Systems, Torino, Italy*, 2006.

[11] W. Sanders and T. B. of Trustees, "Moebius User Manual Version 1.6.0", 2004.

[12] M. Barranco, J. Proenza, and L. Almeida, "First quantitative results of the dependability improvement achieved by ReCANcentrate", in *ETFA 2009. 14^{th} IEEE International Conference on Emerging Technologies and Factory Automation, Palma de Mallorca, Spain*, 2009.

[13] M. Barranco, J. Proenza, and L. Almeida, "Designing and Verifying Media Management in ReCANcentrate", in *WFCS'08. IEEE Workshop on Factory Communication Systems, Dresden, Germany*, 2008.

[14] M. Mahotra and K. S. Trivedi, "Dependability Modeling Using Petri-Nets", *IEEE Transactions on Reliability*, vol. 44, no. 3, September 1995.

[15] DOD, *MIL-HDK-217F-2 Military Handbook, Reliability Prediction Of Electronic Equipment*, Department of Defense Washington DC, 1995.

[16] N. E. Wu, "Reliability Analysis for AFTI-F16 SRFCS Using ASSIST and SURE", in *Proceedings of American Control Conference*, 2002, pp. 4975–4800.

[17] J. Morris and P. Koopman, "Representing Design Tradeoffs in Safety-Critical Systems", *WADS. Workshop on Architecting Dependable Systems, St. Louis, Missouri, USA*, 2005.