Quantifying the Resiliency of Fail-Operational Real-Time Networked Control Systems

Arpan Gujarati, Mitra Nasri, and Björn B. Brandenburg
Max Planck Institute for Software Systems (MPI-SWS)

***Abstract*—In time-sensitive, safety-critical systems that must be fail-operational, active replication is commonly used to mitigate transient faults that arise due to electromagnetic interference. However, designing an effective and well-performing active replication scheme is challenging since replication conflicts with the size, weight, power, and cost constraints of embedded applications. Quantifying the resiliency of a given active replication scheme is thus the first step towards addressing this challenge. To this end, we propose an analysis to quantify the resiliency of fail-operational, CAN-based networked control systems. Since many control systems are inherently robust to a few failed iterations, *e.g.*, one missed actuation does not crash an inverted pendulum, traditional solutions based on hard real-time assumptions are often too pessimistic. Our analysis reduces this pessimism by modeling control robustness with (m, k)-firm specifications.**

I. Introduction

Networked control systems (NCSs)—where sensors, controllers, and actuators belonging to different control loops are connected through a *shared* network such as a CAN bus—are widely deployed in contemporary cyber-physical systems as they offer many practical advantages over dedicated wiring solutions, not the least of which are cost and weight savings [17].

Like other embedded systems, NCSs are susceptible to both internal and external sources of *electromagnetic interference* (EMI), *e.g.*, spark plugs, collision-avoidance radar, TV towers, *etc.* [30]. In fact, the likelihood of soft errors due to EMI across a fleet of devices should not be underestimated. For example, Mancuso [27] observed that, assuming one soft error per bit in a 1 MB SRAM every 10^{12} hours of operation, and a worldwide population of 0.5 billion cars with an average daily operation time of 5%, about 5,000 vehicles per day are affected by a soft error. Since NCSs are time-sensitive, even a few instances of failures due to these soft errors, such as hangs, crashes, incorrect outputs, or message corruptions on the network, can have potentially catastrophic consequences.

EMI-induced component failures must thus be anticipated in the design of safety-critical systems, and are commonly mitigated by means of either active or passive replication. In the context of high-frequency control applications specifically, passive replication, *i.e.*, the use of hot/cold standbys, is insufficient if the failure detection and view-change latencies exceed the control frequency. System engineers thus devise active replication (or static redundancy) schemes to ensure that safety-critical NCSs are *fail-operational* (*e.g.*, see [8, 14, 20]).

However, coming up with a good active replication scheme is no easy task. Engineers face many questions, such as which components, if made more or less resilient (*e.g.*, by adding an extra replica, or shielding), will most impact the overall reliability? Alternatively, which components could be replaced with cheaper consumer-grade parts with the least effect on system reliability? Would dual modular redundancy suffice if the control logic is robust to, say, 10% message loss or would triple modular redundancy be needed? In general, none of these questions (and many more like it) has an obvious answer, and particularly not if size, weight, and power (SWaP) as well as cost constraints must be taken into account, too.

The challenge is further exacerbated by the fact that most well-designed control systems are inherently robust to a few failed iterations, *e.g.*, one missed actuation does not crash an inverted pendulum. That is, requiring that *all* control loop iterations must be correct and timely results in excessively pessimistic answers with regard to the "true" needs of the workload, and consequently in under-utilized, cost-inefficient systems. This rules out the use of classical hard-real-time analyses (see §VI)—to appropriately dimension a fail-operational NCS, a robustness-aware reliability analysis is required.

This paper. We present a sound reliability analysis that quantifies the resiliency of a given configuration of a CAN-based, actively replicated NCS to EMI-induced transient component failures. The objective is to provide system engineers with a method to evaluate the reliability of an active replication scheme (*i.e.*, for a given number of replicas for each task in the NCS) assuming the peak failure rates are known from empirical measurements and/or environmental modeling. Unlike prior approaches, our analysis leverages the robustness of well-designed control systems: since robust control loops tolerate a limited number of transient failures (which result in degraded control performance, but not an unrecoverable plant state), we characterize control loops with (m, k)-firm specifications, where out of every k consecutive control loop iterations, at least m must be "correct and timely" [18].

In a nutshell, the proposed analysis consists of three steps. First, transient failures due to EMI are classified as crash failures (resulting in message omissions), commission failures (resulting in undetected message corruptions), and transmission failures (resulting in retransmissions and hence message delays). Each of these failure types is modeled probabilistically (§II). Second, an intermediate analysis relates the probability of individual message failure events to that of a *failed* control loop iteration, *i.e.*, where the controlled plant is not actuated as expected in a failure-free iteration (§IV). Third, a reliability analysis bounds the *failures in time* (FIT) of the NCS, *i.e.*, the expected number of *control loop failures* in one billion operating hours, where a control loop failure is defined as a violation of its (m, k)-firm specification (§IV).

II. Fault Model and Assumptions

To lay the foundation for our analysis, we give below a precise fault model and introduce key assumptions.

A. Failures-In-Time (FIT)

Any safety-critical NCS consists of multiple hardware components, *e.g.*, motors, cables, connectors, embedded computers, physical sensors, power sources, *etc.*, as well as multiple software components, *e.g.*, applications, OS services, middleware services, clock synchronization protocols, *etc.* Each of these can potentially cause the entire system to fail, unless appropriate mechanisms are in place to tolerate their respective failures. Thus, a reliability analysis of the whole system must take all possible failure sources into account. To make such an analysis manageable, a common approach is to derive individual component FIT rates, which are then added up to bound the overall FIT rate of the entire system.

The objective of this paper is to safely bound the FIT rate of a CAN-based NCS, with a focus on the message exchanges among the distributed hosts attached to the CAN bus. Thus, the analysis evaluates EMI-induced transient faults manifested as different message failure types, and assumes that other system components are reliable, even though the CAN-based NCS subsystem being analyzed may directly depend on them, *e.g.*, the CAN bus wires or the physical sensors. This assumption does not imply that the proposed analysis is not useful if a dependent component fails, rather it provides a FIT rate for one subsystem, which can then be composed with the FITs of other dependent, dependee, or unrelated subsystems.

B. EMI-Induced Transient Faults

Since safety-critical real-time systems are susceptible to EMI, this work focusses on transient faults (also known as *soft errors* or *single-event upsets*) as the primary considered source of failures. We first model the raw EMI-induced transient faults, and then consider the resulting program-visible effects.

Given a set of hosts $H = \{H_1, H_2, \ldots\}$, let τ and λ_i denote the peak rate of raw transient faults affecting the CAN bus and each host $H_i \in H$, respectively. In practice, these rates are empirically determined with measurements or derived from environmental modeling assuming worst-possible operating conditions, and typically include safety margins as deemed appropriate by reliability engineers or domain experts.

We model the raw transient faults as random events following a Poisson distribution (other possible fault models are discussed in §VI). Let $\mathcal{P}(x, \delta, \lambda)$ denote the *probability mass function* of the Poisson distribution, *i.e.*, the probability that x independent events occur in an interval of length δ when the arrival rate is λ. Thus, given the peak rates τ and λ_i, the probability that x raw transient faults affect the CAN bus (respectively, host H_i) in any interval of length δ is bounded by $\mathcal{P}(x, \delta, \tau)$ (respectively, $\mathcal{P}(x, \delta, \lambda_i)$).

C. Program-Visible Message Failures

Raw-transient faults, *i.e.*, bit-flips on the network and in host memory, may manifest as program-visible *transmission*, *crash*, and *commission* failures, as described below.

As a result of the error detection and correction in CAN, raw transient faults may result in automatic retransmissions [10], which we refer to as transmission failures. In the worst case, if the number of retransmissions exceed the number accounted for during schedulability analysis of the CAN bus, transmission failures can delay messages beyond their deadlines.

As in Broster et al.'s analysis [4], we assume that every transient fault on the CAN bus causes a retransmission. Thus, the CAN bus retransmission-rate is also bounded by τ, and the probability that x retransmissions occur in any interval of length δ is bounded by $\mathcal{P}(x, \delta, \tau)$. This is a simplifying (but safe) overestimation because a transient fault may occur when the bus is idle, because multiple transient faults may result in a single retransmission, and because the number of retransmissions in practice is limited, whereas a Poisson distribution has a non-zero probability for any number of events.

We classify raw transient faults affecting hosts into crash failures and commission failures. Crash failures occur if the system suffers an EMI-induced transient corruption that causes an exception to be raised and the system to be rebooted. Although bit flips can cause unbounded hangs in principle (*e.g.*, loops that never terminate due to a bit flip in the termination condition), we assume that each host is equipped with a watchdog timer that reboots the system in case of a hang, *e.g.*, see [29]. A crashed system remains unavailable for some time while it reboots and thus causes an interval in which messages are continuously omitted. R_i denotes an upper bound on this recovery interval for host H_i, and includes any delays due to state re-synchronization.

Prior studies have shown that a large fraction of transient faults have no negative effects [39]. We thus assume a *derating factor* that accounts for masked transient faults, which can be determined empirically [28]. Let f_i denote the derating factor for crash failures on host H_i; the peak rate of crash failures on host H_i is then given by $\rho_i = f_i \lambda_i$. Like raw transient faults, we model crash failures as random events following a Poisson distribution. Thus, the probability that x crash failures occur in any interval of length δ is bounded by $\mathcal{P}(x, \delta, \rho_i)$.

A commission failure occurs if a message is corrupted during preparation before the CAN controller computes the payload checksum included in the CAN header. For example, bit flips in registers or memory of the CAN controller may manifest as commission failures. Like crash failures, we assume a host-specific *derating factor* f'_i for commission failures. The average rate of commission failures on host H_i is then given by $\kappa_i = f'_i \lambda_i$ and the probability that x commission failures occur in any interval of length δ is given by $\mathcal{P}(x, \delta, \kappa_i)$. *Note that our notion of commission failures does not refer to software bugs or Byzantine agents.*

We refer to the interval during which a message is at risk of corruption as its *exposure interval*. For *stateful* tasks such as a PID controller, the message computation relies on both the current input and the application state, and the latter could even be affected by *latent faults* (*i.e.*, state corruptions that have not yet been detected). Thus, the exposure interval depends on the mechanisms in place to tolerate (or avoid) latent faults.

If the hardware platform uses *Error-Correcting Code* (ECC) memory and processors with *lockstep* execution (common in safety-critical systems), then the built-in protections suppress latent faults, and it suffices to consider the *scheduling window* of a message (*i.e.*, the duration from the message's creation to its deadline) as its exposure interval. If no such architectural support is available, then any relevant state can be protected with a *data integrity checker* task that periodically verifies the checksums of all relevant data structures (and that reboots the system in the case of any mismatch). The exposure interval of a message then includes its scheduling window and (in the worst case) an entire period of the data integrity checker.

Independence assumption. Based on the stochastic nature of physical EMI processes, we consider EMI-induced transient faults, and hence basic message failures, to be independent. We do however account explicitly for correlated failures that arise from the system model, *e.g.*, deterministic replicas will produce the same wrong output if given the same wrong input.

III. System Model and Assumptions

We consider a single-input single-output (SISO) control loop connected to a CAN bus that is shared with other traffic, *e.g.*, other control loops, clock synchronization protocol, *etc.*

A. SISO and FT-SISO Control Loops

Let L denote a control loop consisting of plant P and a SISO controller C. The sensor output is generated periodically and broadcasted over CAN by the sensor task, which is denoted by S. The CAN message stream carrying the sensor values is denoted M_s. The controller task C, upon periodic activation, reads the latest sensor message received over CAN, computes a new control command for the plant, updates its local state (*e.g.*, in a PID controller, the integrator), and broadcasts the control command over CAN. C is assigned an appropriate offset to ensure that the sensor message is available before its activation. The CAN message stream carrying the control commands is denoted M_c. The actuator task, denoted A, is directly connected to the plant. Upon periodic activation, it reads the latest control command received over CAN and actuates the plant accordingly. Like C, task A is also assigned an appropriate offset to ensure that the control command is received before its activation. The control loop (*i.e.*, all sensor, control, and actuator tasks) has a period of T time units.

To remain fail-operational despite EMI-induced transient faults, critical tasks in the SISO control loop may be actively replicated on independent hosts. We consider an FT-SISO control loop with multiple functionally identical replicas of the sensor task and the controller task, denoted by sets $\overline{S} = \{S^1, S^2, \ldots\}$ and $\overline{C} = \{C^1, C^2, \ldots\}$, respectively (as a convention, we let superscripts denote replica IDs). We do not assume replicated actuator tasks for now because it requires special hardware for the plant actuator to handle redundant control commands [20]. The issue of replicated actuator tasks is revisited in §VII. The i^{th} runtime activation or job of any sensor task replica S^x, controller task replica C^y, and actuator task A is denoted S^x_i, C^y_i, and A_i, respectively. The sets of the i^{th} jobs of all sensor task replicas in $\overline{S}$ and all controller task replicas in $\overline{C}$ are denoted $\overline{S}_i$ and $\overline{C}_i$, respectively. All tasks in $\overline{S}$ and $\overline{C}$, as well as task A, are deployed on hosts in H.

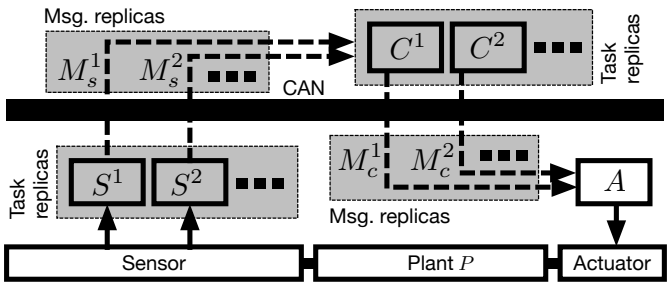


Fig. 1: An FT-SISO control loop. Solid boxes denote hosts. Each dashed grey box denotes a task replica set or a set of message streams transmitted by a task replica set. Dashed arrows denote message streams transmitted over CAN. Message streams directed to a task replica set are received by all tasks in that set, *e.g.*, M_s^1 and M_s^2 are received by all tasks in $\overline{C}$.

Active replication causes additional message streams to be transmitted over CAN. We let $\overline{M}_s = \{M_s^1, M_s^2, \ldots\}$ and $\overline{M}_c = \{M_c^1, M_c^2, \ldots\}$ denote the set of CAN message streams carrying the sensor values and control commands sent by task replicas in $\overline{S}$ and $\overline{C}$, respectively. Each message stream $M_i^x \in \overline{M}_s \cup \overline{M}_c$ is characterized by its period T_i, relative deadline D_i, maximum release jitter J_i^x, offset ϕ_i^x, and priority $prio_i^x$. The k^{th} message in any message stream M_i^x is denoted $M_{i,k}^x$. The sets of the k^{th} messages of the message streams in $\overline{M}_s$ and $\overline{M}_c$ are denoted $\overline{M}_{s,k}$ and $\overline{M}_{c,k}$, respectively

Given all the message streams to be transmitted over CAN, and an upper bound on the CAN bus retransmission rate τ, the probabilities that messages $M_{s,i}^x$ and $M_{c,i}^y$ miss their deadlines is denoted $\mathcal{B}_s^x$ and $\mathcal{B}_c^x$, respectively. These can be derived, for example, using Broster et al.'s analysis [5]. With respect to commission failures, as discussed in §II-C, the exposure interval for each message in a message stream M_i^x is assumed to be upper-bounded by E_i^x. The exposure interval on the actuator task A's host is similarly defined and denoted E_a. A block diagram of a sample FT-SISO control loop using the notation introduced above is illustrated in Fig. 1.

B. Key Assumptions

We assume that hosts are synchronized using a clock synchronization protocol, and that task and message offsets have been chosen to account for the maximum clock synchronization error. Without this assumption, it is much more challenging to ensure replica determinism (*e.g.*, simply assigning appropriate offsets to tasks and messages is insufficient) [31]; the fully asynchronous case remains future work.

We also assume that the CAN protocol guarantees atomic broadcast (*i.e.*, messages are received by either all hosts, or none)—while there exists a theoretical corner case in the CAN protocol that violates this assumption [26], it is of such low likelihood that it is best modeled as a separate, additive failure source and accounted for using its own FIT rate (recall §II-A). We require the controller and the sensor task logic to be deterministic. Thus, given identical inputs and identical states, any

Algorithm 1 A simple voting procedure at task activation.

```
1: procedure PeriodicTaskComputation
2:     Latest_i ← ∅                                   ▷ start voting protocol
3:     for all M_i^k ∈ M̄_i do
4:         if M_i^k not received by its deadline then
5:             continue                               ▷ also accounts for omissions
6:         Latest_i ← Latest_i ∪ latest_i^k
7:     if Latest_i = ∅ then return                    ▷ omit output
8:     result_i ← SimpleMajority(Latest_i)            ▷ deterministic
9:     ...                                            ▷ main logic of the task starts
```

two functionally identical replicas produce identical outputs, except when an output is corrupted due to commission failures.

Algorithm 1 summarizes the voting protocol used by tasks at the start of every iteration to resolve the redundant inputs. Since all inputs are available before the task is activated in a failure-free scenario, message streams that are delayed or omitted due to transmission or crash failures (respectively) are ignored during voting (Line 5). In the worst case, if no input is available to the voter on time due to failures, the task iteration is skipped, *i.e.*, its output for that iteration is omitted (Line 7). Ties are broken deterministically *e.g.*, by using message IDs.

In our analysis, we (pessimistically) assume that corrupted message replicas are identical because it is a worst-case scenario w.r.t. the voting protocol, even though it is highly unlikely in practice. In particular, if the number of corrupted messages exceeds the number of correct messages, then assuming identical corrupted messages implies that the voting outcome is corrupted, while in the case of non-identical corrupted messages there is a high likelihood that correct messages still form the largest quorum. Whether or not corrupted messages are likely to be identical is highly application-specific.

C. Correctness of an FT-SISO Control Loop

Commercially-used controllers are typically safeguarded against disturbances and noise in their inputs using appropriate limiting or clamping mechanisms to ensure that the error term is within a reasonable limit, mechanisms to filter noise, *etc.* Besides these safety mechanisms, well-designed controllers can also withstand a few failed iterations, *i.e.*, the plant remains functional even if a few of the past control iterations were incorrect due to, say, corrupted sensor values or because the plant was actuated with a different value than what the controller had commanded (or not actuated at all).

Prior works have characterized control system performance in the presence of failures using an *asymptotic* requirement, which mandates that, as the number of control loop iterations approaches infinity, the failure rate should not exceed a given threshold, (*e.g.*, see [34]). Alternatively, a stronger robustness requirement mandates that, out of any k consecutive control loop iterations, a given fraction $\frac{m}{k}$ must not fail (*e.g.*, see [7]).

We similarly characterize control system robustness using an (m, k)-firm specification [18]: for the plant to remain functional, out of any k consecutive control loop iterations, at least m must be correct. We consider a control loop iteration to be *correct* if the plant actuation in that iteration is not skipped and the control command used for the actuation was not corrupted or computed using corrupted sensor values.

IV. Probabilistic Analysis

We first analyze the probability that the n^{th} iteration of the control loop fails, for any n. We then argue that the derived probability of a given control loop iteration failing is, in fact, identically and independently distributed (IID) w.r.t. n. In the end, we describe in brief a procedure to obtain from this probability an upper bound on the FIT rate of the control loop.

Let $V_{c,n}^{y}$ denote the voter instance that resolves the redundant inputs for controller job C_n^y, and let $\overline{V}_{c,n}$ denote the set of all such voter instances of controller jobs in $\overline{C}_n$. Similarly, let $V_{a,n}$ denote the voter instance that resolves the redundant inputs for the actuator job A_n, and let U_n denote A_n's output, *i.e.*, the final command issued to the actuator.

Recall from §III that appropriate offsets and deadlines are assigned to tasks and messages such that, in a failure-free scenario, messages in $\overline{M}_{s,n}$ are transmitted before the corresponding consumers in $\overline{C}_n$ are activated, and so on.

As mentioned in §III, due to clock synchronization and the atomic broadcast assumption, message replicas are identical in a failure-free scenario, *i.e.*, the messages in $\overline{M}_{s,n}$ carry identical sensor values and the messages in $\overline{M}_{c,n}$ carry identical control commands. However, due to commission failures, one or more messages in $\overline{M}_{s,n}$ may be corrupted. If the voters $\overline{V}_{c,n}$ choose a corrupted sensor value, then all messages $\overline{M}_{c,n}$ carrying the control commands are also corrupted. Messages in $\overline{M}_{s,n}$ could also be delayed or omitted due to transmission and crash failures, in which case the voters $\overline{V}_{c,n}$ work with fewer inputs. But if all the messages in $\overline{M}_{s,n}$ are either delayed or omitted, the controller jobs $\overline{C}_n$ have no inputs to work with, so the messages $\overline{M}_{c,n}$ are not prepared. Similarly, the controller to actuator information flow may also be affected by failures, resulting in A_n's output U_n being corrupted or omitted.

Next, we bound in small steps of a few lemmas each the probability that the final output U_n is corrupted or omitted.

Step 1: Analyzing Algorithm 1. What is the probability that the simple majority voter in Algorithm 1 chooses an incorrect value? The voting algorithm plays an important part in a system with active replication and is independent of the actual control logic; therefore, we evaluate it first.

Lemma 1. *Given n_c correct inputs and n_i incorrect inputs, an upper bound on the probability that the simple majority procedure in Algorithm 1 selects an incorrect input is:*

$$P\left(\substack{\textit{Algo. 1 output}\\ \textit{incorrect}} \middle| n_c, n_i\right) = \begin{cases} 1 & n_i > n_c \vee n_i = n_c \neq 0, \\ 0 & n_i < n_c \vee n_i = n_c = 0. \end{cases} \quad (1)$$

Proof. A simple majority algorithm chooses the largest-sized quorum as its output. From §III, recall that correct message replicas are identical, and that incorrect message replicas are also identical. If $n_i > n_c$, the largest-sized quorum belongs to incorrect messages, and Algorithm 1's output is incorrect with probability 1. If $n_i = n_c \neq 0$, then there are two largest-sized quorums. Since Algorithm 1 breaks ties deterministically, in the worst case, it may choose the incorrect quorum. If $n_i < n_c$, the largest-sized quorum belongs to correct messages, and Al-

gorithm 1's output is correct, *i.e.*, incorrect with probability 0. If $n_i = n_c = 0$, the voter has received no inputs, so it does not choose any output (Algorithm 1, Line 7). The probability of choosing an incorrect output is thus 0. □

Step 2: Analyzing message failure probabilities. In the following, we evaluate the probability that a sensor message replica $M^x_{s,n}$ or a controller message replica $M^x_{c,n}$ is omitted, delayed beyond its deadline, or corrupted before transmission, and then we evaluate similar probabilities for a set of messages. Since the analysis is identical for both sensor and controller message replicas, we let $M^x_{i,n}$ denote any sensor or controller message replica, *i.e.*, either $M^x_{s,n}$ or $M^x_{c,n}$.

The probability that message $M^x_{i,n}$ is delayed beyond its deadline is bounded by $\mathcal{B}^x_i$, which was defined in §III. Regarding message omission, suppose that message $M^x_{i,n}$ is expected to arrive at time t, and that its sender task is deployed on host H_h. Since R_h is the maximum time to recover from a crash failure on host H_h, if there is at least one crash failure during the interval $[t - R_h,\ t)$, $M^x_{i,n}$'s arrival may be skipped. Thus, the probability of there being at least one crash in the interval $[t - R_h,\ t)$ upper-bounds the probability that $M^x_{s,n}$ is omitted.

$$P(M^x_{i,n}\ \textit{omitted}) = 1 - \mathcal{P}(0,\ R_h,\ \rho_h) \tag{2}$$

Regarding message corruptions due to commission failures before transmission, recall from §III that the exposure interval for any message $M^x_{i,n}$ is upper-bounded by E^x_i. If there are zero commission failures in this interval, then $M^x_{i,n}$ cannot be corrupted due to commission failures, but if there is at least one commission failure in this interval, $M^x_{i,n}$ may be corrupted. Thus, the probability that $M^x_{i,n}$ is corrupted due to commission failures is upper-bounded by the probability that there is at least one commission failure in an interval of length E^x_i.

$$P(M^x_{i,n}\ \textit{corrupted}) = 1 - \mathcal{P}(0,\ E^x_i,\ \kappa_h) \tag{3}$$

Eqs. 2 and 3 denote failure probabilities for an individual message; in the following, we analyze joint failure probabilities of sets of messages. As stated in §II-C, message failures are assumed to be independent. Let $\mathcal{S} = \overline{M}_{i,n}$ for brevity.

The probability that messages in $O_i \subseteq \mathcal{S}$ are omitted whereas the messages in $\mathcal{S} \setminus O_i$ are not omitted is simply the product of the individual omission probabilities given by Eq. 2 (for messages in O_i) and its complement (for $\mathcal{S} \setminus O_i$).

$$P(O_i\ \textit{omitted}) = \left(\prod_{M^x_{i,n} \in O_i} P\big(M^x_{i,n}\ \textit{omitted}\big)\right)\left(\prod_{M^x_{i,n} \in \mathcal{S} \setminus O_i} 1 - P\big(M^x_{i,n}\ \textit{omitted}\big)\right) \tag{4}$$

Let $P(D_i\ \textit{delayed} \mid O_i\ \textit{omitted})$ denote the probability that messages in $D_i \subseteq \mathcal{S}$ are delayed, given that messages in $O_i \subseteq \mathcal{S}$ were omitted. Since an omitted message cannot be delayed because it is not transmitted in the first place, if $D_i \cap O_i \neq \emptyset$, the probability that messages in D_i are delayed is zero. If $D_i \cap O_i = \emptyset$, the probability that messages in set D_i are delayed whereas those in $\mathcal{S} \setminus O_i \setminus D_i$ are not delayed is uppper-bounded by the product of the individual message delay probabilities.

$$P(D_i \text{ delayed} \mid O_i \text{ omitted}) = \begin{cases} 0 & D_i \cap O_i \neq \emptyset \\ \left(\prod_{M^x_{i,n} \in D_i} \mathcal{B}^x_i\right)\left(\prod_{M^x_{i,n} \in \mathcal{S} \setminus O_i \setminus D_i} 1 - \mathcal{B}^x_i\right) & \text{otherwise} \end{cases} \tag{5}$$

The probability $P(I_i \text{ corrupted} \mid O_i \text{ omitted})$ that messages in $I_i \subseteq \mathcal{S}$ are corrupted, given that messages in $O_i \subseteq \mathcal{S}$ were omitted, is derived in a similar way. Since an omitted message cannot be corrupted due to commission failures because it is not prepared in the first place, if $I_i \cap O_i \neq \emptyset$, the probability that messages in I_i are corrupted is zero. If $I_i \cap O_i = \emptyset$, the probability that messages in I_i are corrupted whereas those in $\mathcal{S} \setminus O_i \setminus I_i$ are not corrupted is given by the product of the individual message corruption probabilities from Eq. 3. Thus,

$$P(I_i \text{ corrupted} \mid O_i \text{ omitted}) = \begin{cases} 0 & O_i \cap I_i \neq \emptyset \\ \left(\prod_{M^x_{i,n} \in I_i} P(M^x_{i,n}\ \textit{corrupted})\right) \times & \\ \left(\prod_{M^x_{i,n} \in \mathcal{S} \setminus O_i \setminus I_i} 1 - P(M^x_{i,n} \text{ corrupted})\right) & \text{otherwise.} \end{cases} \tag{6}$$

Using Eqs. 4–6, we next derive the joint probability that messages in any $O_i \subseteq \mathcal{S}$ are omitted, messages in any $D_i \subseteq \mathcal{S}$ are delayed, and messages in any $I_i \subseteq \mathcal{S}$ are corrupted.

Lemma 2. *Given $\mathcal{S} = \overline{M}_{i,n}$ and its subsets $O_i, D_i, I_i \subseteq \mathcal{S}$, the probability that messages in O_i are omitted whereas messages in $\mathcal{S} \setminus O_s$ are not omitted, that messages in D_i are delayed whereas the messages in $\mathcal{S} \setminus O_i \setminus D_i$ are transmitted in time, and that messages in I_i are corrupted whereas the messages in $\mathcal{S} \setminus O_i \setminus I_i$ are not corrupted, is:*

$$P\ (\ O_i\ \textit{omitted},\ D_i\ \textit{delayed},\ I_i\ \textit{corrupted}\ ) = P(O_i\ \textit{omitted}) \times P(D_i\ \textit{delayed} | O_i\ \textit{omitted}) \times P(I_i\ \textit{corrupted} | O_i\ \textit{omitted}). \tag{7}$$

Proof. The conditional probability theorem for three events A, B, and C states that $P(A \cap B \cap C) = P(A)\, P(B|A)\, P(C|B,A)$. Attributing events A to messages in O_i being omitted, B to messages in D_i being delayed, and C to messages in I_i being corrupted, $P(A) = P(O_i\ \textit{omitted})$, $P(B|A) = P(D_i\ \textit{delayed} | \emptyset_i\ \textit{omitted})$, and $P(C|B,A) = P(I_i\ \textit{corrupted} | D_i\ \textit{delayed},\ O_i\ \textit{omitted})$. Furthermore, since message corruptions are independent of delays, $P(C|B,A) = P(I_i\ \textit{corrupted} | D_i\ \textit{delayed},\ O_i\ \textit{omitted}) = P(I_i\ \textit{corrupted} | O_i\ \textit{omitted})$. □

Step 3. Analyzing controller voter instance $V^y_{c,n}$. We evaluate the probability that a controller voter instance $V^y_{c,n}$ chooses a corrupted sensor value because the majority of its inputs is corrupted (Lemma 3), or that it does not choose anything because all its inputs are omitted or delayed (Lemma 4).

Similar to the previous step, let $\mathcal{S} = \overline{M}_{s,n}$ denote the set of all sensor message replicas that are inputs to each controller voter instance $V^y_{c,n}$, O_s denote the set of messages that are omitted, D_s denote the set of messages that are delayed, and I_s denote the set of messages that are corrupted. Lemmas 3 and 4 require evaluating the output of voter instance $V^y_{c,n}$ over all possible sets $O_s \subseteq \mathcal{S}$, $D_s \subseteq \mathcal{S}$, and $I_s \subseteq \mathcal{S}$.

Lemma 3. *Given $\mathcal{S} = \overline{M}_{s,n}$, the probability that any voter instance $V^y_{c,n}$ chooses a corrupted sensor value is:*

$$P(V^y_{c,n} \textit{ output incorrect}) = \sum_{\forall O_s, D_s, I_s \subseteq \mathcal{S}} \left(P \begin{pmatrix} O_s \textit{ omitted}, \\ D_s \textit{ delayed}, \\ I_s \textit{ corrupted} \end{pmatrix} P \left(\begin{matrix} \textit{Algo. 1 output} \\ \textit{incorrect} \end{matrix} \middle| n_c, n_i \right) \right), \quad (8)$$

where $n_c = |\mathcal{S} \backslash O_s \backslash D_s \backslash I_s|$ and $n_i = |\mathcal{S} \backslash O_s \backslash D_s| - |\mathcal{S} \backslash O_s \backslash D_s \backslash I_s|$.

Proof. By considering all possible values for $O_s \subseteq \mathcal{S}$, $D_s \subseteq \mathcal{S}$, and $I_s \subseteq \mathcal{S}$, and using the law of total probability, $P(V^y_{c,n} \text{ output incorrect}) = \sum_{\forall O_s, D_s, I_s \subseteq \mathcal{S}} \phi_{case} \times \phi_{cond}$. In this equation, ϕ_{case} denotes the probability that messages in O_s are omitted, messages in D_s are delayed, and messages in I_s are corrupted, and ϕ_{cond} denotes the conditional probability, given O_s, D_s, and I_s, that voter instances $V^y_{c,n}$'s output is incorrect.

From Lemma 2, $\phi_{case} = P(O_s \textit{ omitted}, D_s \textit{ delayed}, I_s \textit{ corrupted})$. Probability ϕ_{cond} is derived using Lemma 1, as follows. Since messages in O_s and D_s are omitted and delayed, respectively, the voter instance ignores those inputs. Thus, the majority is computed over $n_{total} = |\mathcal{S} \setminus O_s \setminus D_s|$ inputs. Out of these, only $n_c = |\mathcal{S} \setminus O_s \setminus D_s \setminus I_s|$ inputs are correct, whereas the remaining $n_i = n_{total} - n_c$ inputs are corrupted. Thus, from Lemma 1, $\phi_{cond} = P(\textit{Algo. 1 output incorrect} \mid n_c, n_i)$. □

Lemma 4 below deals with the special case in Algorithm 1 (Line 9) where the voter does not choose any output because all its inputs were either delayed or omitted.

Lemma 4. *Given $\mathcal{S} = \overline{M}_{s,n}$, the probability that any voter instance $V^y_{c,n}$ does not choose any value is:*

$$P\left(V^y_{c,n} \textit{ output omitted}\right) = \sum_{\forall O_s, D_s, I_s \subseteq \mathcal{S} \textit{ s.t. } \mathcal{S} \backslash O_s \backslash D_s = \emptyset} P \begin{pmatrix} O_s \textit{ omitted}, D_s \textit{ delayed}, \\ I_s \textit{ corrupted} \end{pmatrix}. \quad (9)$$

Proof. Similar to the proof of Lemma 3, we consider all possible values for $O_s \subseteq \mathcal{S}$, $D_s \subseteq \mathcal{S}$, and $I_s \subseteq \mathcal{S}$, with case probabilities $\phi_{case} = P(O_s \textit{ omitted}, D_s \textit{ delayed}, I_s \textit{ corrupted})$, respectively.

For each case, if $\mathcal{S} \backslash O_s \backslash D_s = \emptyset$, the voter receives no inputs on time and is guaranteed to not choose any output. Thus, the conditional probability that $V^y_{c,n}$'s output is omitted for such cases is 1. For the remaining cases where $\mathcal{S} \backslash O_s \backslash D_s \neq \emptyset$, the voter receives at least one input on time and is guaranteed to choose some output. For such cases, the conditional probability that $V^y_{c,n}$'s output is omitted is 0. These cases are ignored.

Thus, the total probability that $V^y_{c,n}$'s output is omitted is given by $\sum_{\forall O_s, D_s, I_s \subseteq \mathcal{S} \text{ s.t. } \mathcal{S} \backslash O_s \backslash D_s = \emptyset} (\phi_{case} \cdot 1)$. □

Step 4: Analyzing actuator voter instance $V_{a,n}$. We bound the probability that an actuator voter instance $V_{a,n}$ chooses a corrupted sensor value because the majority of its inputs is corrupted (Lemma 5), or that it does not choose anything, because all its inputs are either omitted or delayed (Lemma 6). Let $\mathcal{S} = \overline{M}_{c,n}$ denote the set of all controller message replicas that are inputs to the actuator voter instance $V_{a,n}$.

Since all voter instances $\overline{V}_{c,n}$ operate on the same input values, if a correct voter instance $V^y_{c,n}$ chooses a corrupted sensor value because of wrong inputs, it implies that all correct voter instances in $\overline{V}_{c,n}$ chose a corrupted sensor value. In such a scenario, the actuator voter is guaranteed to get only corrupted inputs, since all of the control messages will be prepared using the corrupted sensor values.

A similar property holds for the voter output omission. Proper deadline and offset assignment guarantees that, in a failure-free scenario, messages in $\overline{M}_{s,n}$ are transmitted before the voter instances in $\overline{V}_{c,n}$ are activated. Thus, each voter instance can decide locally whether a message was received past its deadline (in which case it is discarded, recall Algorithm 1). As a result, if a voter instance $V^y_{c,n}$ does not choose any value because all its inputs are delayed or omitted, all voter instances in $\overline{V}_{c,n}$ do not choose any values, either. Thus, no output is generated by the controller task replicas and the actuator voter omits its output, too, which results in a skipped actuation.

Lemmas 5 and 6 below are thus conditioned on the assumptions that the sensor inputs of the voter instances $\overline{V}_{c,n}$ do not result in a corrupted or omitted output, respectively. In particular, they account for failures due to message corruptions and crash failures on the hosts of the controller tasks, and due to the delay of messages from controller replicas to the actuator. The case that the sensor inputs of the voter instances $\overline{V}_{c,n}$ result in a corrupted or omitted output is accounted for later in Step 5. The proofs of Lemmas 5 and 6 are omitted because they are analogous to those of Lemmas 3 and 4.

Lemma 5. *Given $\mathcal{S} = \overline{M}_{c,n}$, the probability that voter instance $V_{a,n}$ chooses a corrupted control value is:*

$$P(V_{a,n} \textit{ output incorrect}) = \sum_{\forall O_c, D_c, I_c \subseteq \mathcal{S}} \left(P \begin{pmatrix} O_c \textit{ omitted}, \\ D_c \textit{ delayed}, \\ I_c \textit{ corrupted} \end{pmatrix} P \left(\begin{matrix} \textit{Algo. 1 output} \\ \textit{incorrect} \end{matrix} \middle| n_c, n_i \right) \right), \quad (10)$$

where $n_c = |\mathcal{S} \backslash O_c \backslash D_c \backslash I_c|$ and $n_i = |\mathcal{S} \backslash O_c \backslash D_c| - |\mathcal{S} \backslash O_c \backslash D_c \backslash I_c|$.

Lemma 6. *Given $\mathcal{S} = \overline{M}_{c,n}$, the probability that voter instance $V_{a,n}$ omits its output is:*

$$P\left(V_{a,n} \textit{ output omitted}\right) = \sum_{\forall O_c, D_c, I_c \subseteq \mathcal{S} \textit{ s.t. } \mathcal{S} \backslash O_c \backslash D_c = \emptyset} P \begin{pmatrix} O_c \textit{ omitted}, D_c \textit{ delayed}, \\ I_c \textit{ corrupted} \end{pmatrix}. \quad (11)$$

Step 5: Analyzing the final output U_n. We first bound the probability that U_n is corrupted (Lemma 7), followed by the probability that U_n is omitted (Lemma 8), and finally the joint probability of both events (Lemma 9). The proof of Lemma 8 is omitted because it is analogous to that of Lemma 7.

Lemma 7. *U_n is corrupted with a probability of at most*

$$P(U_n \textit{ corrupted}) = \phi_{case1} + \phi_{case2} \times \begin{pmatrix} \phi_{case2a} + \phi_{case2b} \\ - \phi_{case2a}\, \phi_{case2b} \end{pmatrix}, \quad (12)$$

where $\phi_{case1} = P(V^y_{c,n} \textit{ output incorrect})$, $\phi_{case2} = 1 - \phi_{case1}$, $\phi_{case2a} = P(V_{a,n} \textit{ output incorrect})$, $\phi_{case2b} = 1 - \mathcal{P}(0,\ E_a,\ \kappa_a)$, and H_a denotes actuator task A's host.

Proof. We consider two cases based on whether the sensor inputs to voter instances $\overline{V}_{c,n}$ result in corruption of the

controller voter outputs (case 1) or not (case 2).

From Lemma 3, the probability that case 1 occurs is given by $\phi_{case1} = P(V^y_{c,n}$ *output incorrect*$)$ and the probability that case 2 occurs is given by $\phi_{case2} = 1 - P(V^y_{c,n}$ *output incorrect*$)$. For case 1, since the voters in the controller tasks choose an incorrect output, all control commands transmitted were incorrect, so it is guaranteed that U_n is corrupted. Thus, the conditional probability that U_n is corrupted is $\phi_{cond1} = 1$ in this case.

For case 2, the conditional probability that U_n is corrupted depends on two sources: **(a)** voter instance $V_{a,n}$'s output can be incorrect, and **(b)** A's host can be affected by commission failures. The probability for case (a) is $\phi_{case2a} = P(V_{a,n}$ output incorrect$)$, from Lemma 5. The probability for case (b) is $\phi_{case2b} = 1 - \mathcal{P}(0,\ E_a,\ \kappa_a)$, from Eq. 3.

Cases (a) and (b) are independent: (a) occurs because inputs to $V_{n,a}$ were corrupted due to commission failures on the controller tasks' hosts, whereas (b) occurs due to commission failures on the actuator task's host. Thus, using theorem $P(A \cup B) = P(A) + P(B) - P(A)\,P(B)$ for independent events A and B, the conditional probability for case 2 is $\phi_{cond2} = \phi_{case2a} + \phi_{case2b} - \phi_{case2a}\,\phi_{case2b}$.

By law of total probability, the probability that U_n is omitted is given by $\phi_{case1}\,\phi_{cond1} + \phi_{case2}\,\phi_{cond2}$. □

Lemma 8. *U_n is omitted with a probability of at most*

$$P(U_n \text{ omitted}) = \phi_{case1} + \phi_{case2} \times \begin{pmatrix} \phi_{case2a} + \phi_{case2b} \\ - \phi_{case2a}\,\phi_{case2b} \end{pmatrix}, \quad (13)$$

where $\phi_{case1} = P(V^y_{c,n}$ output omitted$)$, $\phi_{case2} = 1 - \phi_{case1}$, $\phi_{case2a} = P(V_{a,n}$ output omitted$)$, $\phi_{case2b} = 1 - \mathcal{P}(0,\ R_a,\ \rho_a)$, and H_a denotes actuator task A's host.

In Lemma 9, we compose the probabilities derived in Lemmas 8 and 7 to derive the probability that the n^{th} control loop iteration fails, *i.e.*, that actuator A's output U_n that is applied to the controlled plant is omitted or corrupted. We do not assume that the probabilities derived in Lemmas 7 and 8 are independent, since it is possible that an omitted control message tilted the majority in favor of the correct quorum, thereby reducing the probability that U_n is corrupted.

Lemma 9. *An upper bound on the probability that the n^{th} control loop iteration fails is:*

$$P\begin{pmatrix} n^{th} \text{ control loop} \\ \text{iteration fails} \end{pmatrix} = P(U_n \text{ omitted}) + P(U_n \text{ corrupted}). \quad (14)$$

Proof. The control loop iteration fails if U_n is either omitted or corrupted. The probability of a control loop failure is thus upper bounded by the sum of the failure probabilities due to these two sources, *i.e.* sum of $P(U_n$ *omitted*$)$ and $P(U_n$ *corrupted*$)$ derived in Lemmas 7 and 8, respectively. □

The IID property. The equation defined in Eq. 14 can be iteratively unfolded until it consist only of terms of the form $\mathcal{P}(x, \delta, \lambda)$, each of which denotes the Poisson probability mass function for one of the three failure types. Thus, it is independent of any parameters that are specific to the n^{th} control loop iteration, *i.e.*, it is identical for any control loop iteration.

In addition, Lemmas 1–9 are all derived under worst-case assumptions w.r.t. interference from other CAN messages. Failure of the n^{th} control loop iteration, defined as a deviation from a failure-free execution of that iteration, is independent of whether past iterations encountered any failures or not. Thus, the probabilities defined using Eq. 14 for any two iterations n_1 and n_2 are mutually independent as well.

In summary, $P(n^{\text{th}}$ control loopiteration fails$)$ satisfies the IID (independent and identical distribution) property w.r.t. n.

Analysis complexity. Lemmas 3–6 require iterating over all partitions of message replica set $\mathcal{S}$ corresponding to all possible values of $O_i \subseteq \mathcal{S}$, $D_i \subseteq \mathcal{S}$, and $I_i \subseteq \mathcal{S}$ (where $i = s$ for Lemmas 3 and 4 and $i = c$ for Lemmas 5 and 6). The analysis for each control loop in the NCS is thus exponential in the number of sensor message streams $|\overline{M}_s|$ and the number of controller message streams $|\overline{M}_c|$. However, since the number of replicas of any task is likely small, *i.e.*, typically under ten, the analysis can be quickly computed.

FIT Analysis. We use the probability of a failed control loop iteration, *i.e.*, the result of Lemma 9, to derive the NCS's FIT rate. First, we derive a lower bound on the mean time to failure (MTTF) of the control loop. Recall from §III-C that a control loop iteration fails if it violates its (m, k) specification. We model this problem in the form of a well-studied *a-within-consecutive-b-out-of-c:F* system model [22], and leverage existing results [36] (which depend on the IID property of the iteration failure probability) on the reliability analysis of this system model to safely lower-bound the MTTF. The full derivation is available online [16]. Given an MTTF lower bound $MTTF_{LB}$ in hours, the FIT rate is computed as $10^9/MTTF_{LB}$ [38].

V. Evaluation

The objective of the evaluation is twofold. First, we demonstrate the ability of our analysis to quantify and compare the reliability of workloads with different (m, k) specifications and failure rates. Second, we illustrate the utility of our analysis by comparing FITs of different replication schemes.

To implement the analysis, we extended the SchedCAT [2] library to support our system model for CAN-based NCSs, and implemented the proposed analysis on top. All computations were carried out at a precision of 200 decimal places using the *mpmath* Python library for arbitrary precision arithmetic [21].

We used an active-suspension workload similar to the one studied by Anta and Tabuada [1]. The system consists of four control loops corresponding to the control of four wheels with magnetic suspension (period $1.5\,ms$), two hard real-time messages that report the current in the power line cable (period $4\,ms$) and the internal temperature of the coils (period $10\,ms$), and a soft real-time message responsible for logging (period $100\,ms$). In addition, the workload consisted of a clock synchronization message with a period of $50\,ms$ [13]. The logging messages carried payloads of eight bytes each, the control loop messages carried payloads of three bytes each, and the remaining messages carried one byte payloads.

Considering a bus rate of $1\,mbit/s$, the workload resulted in a total bus utilization of 46.16%. The clock synchronization message stream was assigned the highest priority, followed by the current and temperature monitoring message streams, the control message streams, and last, the logging message stream.

The recovery time from a crash failure was set to $R_h = 1\,s$ for each host $H_h \in H$, and the data integrity checker period was set to $T_{checker} = 100\,ms$. The failure rates and the (m, k) specifications used in each experiment are mentioned in the corresponding graphs. All failure rates in the following are reported as the number of failures per *ms*. Based on prior work by Ferreira et al. [12] and Rufino et al. [33], peak transmission failure rates range from 10^{-4} in aggressive environments to 10^{-10} in lab conditions, and as per Hazucha and Svensson [19], a 4 Mbit SRAM chip has a fault rate of approximately 10^{-12}. We thus use similar orders of magnitude while varying the respective failure rate parameters. Since the actuator task is not replicated, its host was assumed to be heavily shielded and thus assigned extremely low failure rates.

Exp. 1: FIT for different parameters. We varied the number of replicas of the sensor and the controller task of the wheel 1 control loop. Figs. 2(a)–2(d) illustrate the control loop's FIT as a function of its replication factor for different parameters.

In Fig. 2(a), m and k are varied as follows: $1 \leq m \leq 5$, and $k = 5$ or $k = 2m$; and in Fig. 2(b), m/k is 90%, 95%, 99%, or 100% (while minimizing m and k).

A hard specification, *i.e.*, where $m = k$, yields an extremely high FIT rate compared to all other specifications with $m < k$, even the ones with $m/k \geq 0.9$, which highlights the need for a robustness-aware reliability analysis. For example, in Fig. 2(b), if the desired reliability threshold is 10 FIT, a hard real-time analysis requires the use of three replicas, whereas if a 90% success rate is sufficient, then our analysis indicates that no replication is required. Fig. 2(a) shows that increasing both m and k while keeping m/k constant reduces the FIT rate, which shows that an asymptotic specification that relies only on the ratio m/k can be easily supported by our analysis. Interestingly, different (m, k) specifications can result in very similar FIT rates, *e.g.* $(3, 5)$ and $(2, 4)$.

Next, we varied the transmission failure rate τ across 10^{-4}, 10^{-6}, 10^{-8}, and 10^{-10}. The crash and commission failure rates were set to $\rho_i = \kappa_i = \tau \times 10^{-6}$. The results are illustrated in Fig. 2(c). As expected, the FIT rates decrease as the failure rates are lowered. The FIT rates also decrease with increasing replication, but only up to three replicas, and then start increasing. That is, as previously discussed in [15], active replication becomes counterproductive in real-time systems after some point, because it reduces the slack available for fault-induced retransmissions and results in increased FIT rates. In general, such graphs can help engineers to identify the maximum reliability that they can extract out of a system by increasing its replication factor, or conversely, the minimum replication factor needed to achieve a desired reliability.

To further understand the effects of individual failure types, we computed FIT values for three different scenarios. In each scenario, only one of the three failure types was assigned a significant rate, *i.e.*, $\rho_i = 10^{-12}$, $\kappa_i = 10^{-12}$, and $\tau = 10^{-4}$, respectively, whereas the others were assigned negligible values, *i.e.* 10^{-48}. As apparent in Fig. 2(d), the FIT rates are highest for crash failures. The FIT rates are also high for commission failures. Unlike crash failures, increasing replication helps tolerate commission failures only if the number of replicas is odd. This is expected, *e.g.*, with two replicas, the voting algorithm is unable to distinguish between a correct and an incorrect input, and this scenario occurs with significant probability if the commission failure rate is high. In contrast to the first two cases, the FIT rates are very low for transmission failures, despite a relatively high retransmission rate. This indicates the relative importance of tolerating host failures, at least when hard timeliness is not required.

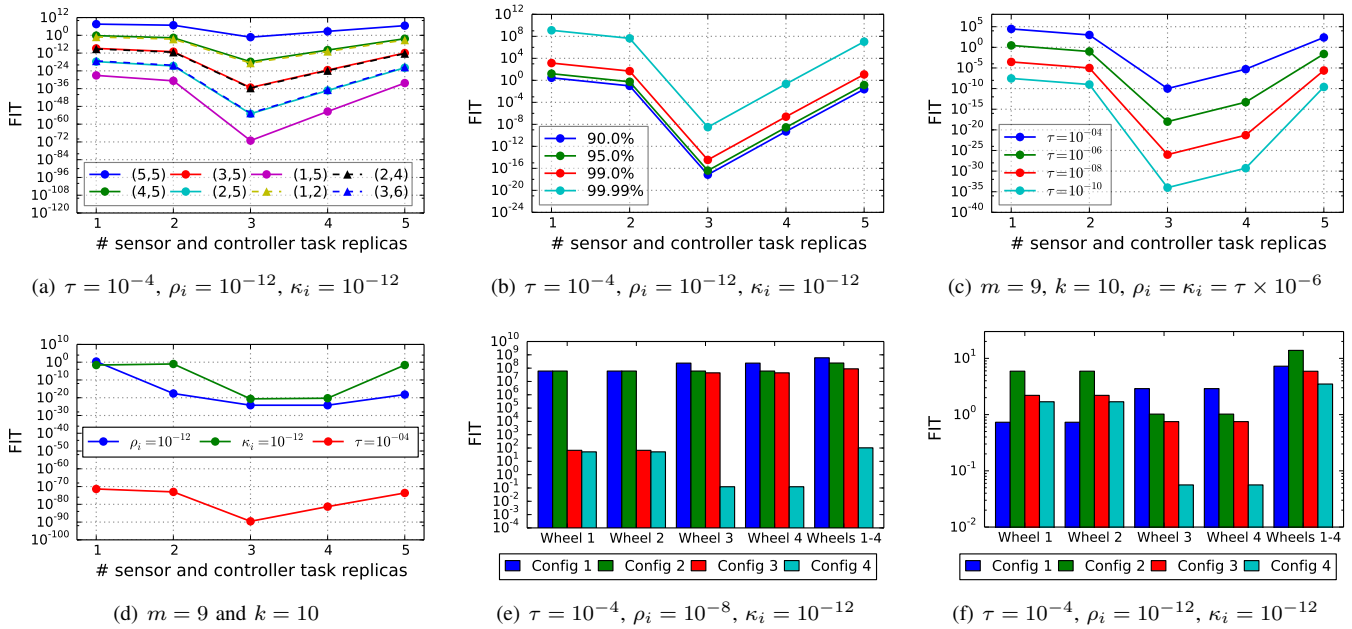
Exp. 2: FIT for different replication schemes. To demonstrate that the analysis helps to identify reliability bottlenecks w.r.t. resource constraints, or to identify opportunities to significantly increase the reliability at modest costs, we analyzed four different replications schemes of the workload for two different sets of failure rates: **(i)** $\tau = 10^{-4}$, $\rho_i = 10^{-8}$, $\kappa_i = 10^{-12}$ and **(ii)** $\tau = 10^{-4}$, $\rho_i = 10^{-12}$, $\kappa_i = 10^{-12}$; scenario (i) has a higher crash failure rate. To model practical design constraints, we assumed that the rear wheels (1 and 2) were close to many electromechanical parts, and assigned the hosts of the respective sensor tasks an order of magnitude higher crash and commission failure rates. The four considered replication configurations are summarized in the table below.

Config. #	Wheel 1	Wheel 2	Wheel 3	Wheel 4	Period
1	$3S$, $1C$	$3S$, $1C$	$1S$, $1C$	$1S$, $1C$	$1.50\,ms$
2	$2S$, $1C$	$2S$, $1C$	$2S$, $1C$	$2S$, $1C$	$1.50\,ms$
3	$2S$, $2C$	$2S$, $2C$	$2S$, $1C$	$2S$, $1C$	$1.75\,ms$
4	$2S$, $2C$	$2S$, $2C$	$2S$, $2C$	$2S$, $2C$	$2.00\,ms$

Parameters xS and yC denote that x and y replicas were provisioned for the sensor and the controller task of the respective wheel-specific control loop. For configurations 3 and 4, the control loop periods were increased to $T = 1.75\,ms$ and $T = 2\,ms$, respectively, at the cost of degrading their *instantaneous quality-of-control* (refer to [1] for details). The benefit of increasing the periods is that, while configurations 1 and 2 allow adding up to four additional task replicas, relaxing T to $1.75\,ms$ and $2\,ms$ allows adding up to six and eight additional task replicas, respectively.

To tolerate the increased chances of crash and commission failures in the rear-wheel sensor tasks, configuration 1 replicates only the sensor tasks of wheels 1 and 2. Configuration 2, in contrast, distributes the replication budget uniformly, *i.e.*, one extra sensor task replica for each loop. In configurations 3 and 4, additional replicas are added for the control tasks (at the cost of slightly degraded control quality). The wheel-specific FITs and the combined FIT of all wheels for each configuration are illustrated in Figs. 2(e) and 2(f) corresponding to the failure rates in (i) and (ii), respectively. The (m, k) specification for each loop is $(9, 10)$.

For the higher crash failure rate in Fig. 2(e), configurations 2 and 3 yield a better overall FIT than configuration 1, but since



(a) $\tau = 10^{-4}$, $\rho_i = 10^{-12}$, $\kappa_i = 10^{-12}$

(b) $\tau = 10^{-4}$, $\rho_i = 10^{-12}$, $\kappa_i = 10^{-12}$

(c) $m = 9$, $k = 10$, $\rho_i = \kappa_i = \tau \times 10^{-6}$

(d) $m = 9$ and $k = 10$

(e) $\tau = 10^{-4}$, $\rho_i = 10^{-8}$, $\kappa_i = 10^{-12}$

(f) $\tau = 10^{-4}$, $\rho_i = 10^{-12}$, $\kappa_i = 10^{-12}$

Fig. 2: **(a, b)** Parameters m and k are varied. **(c, d)** Failure rates τ, κ_i, and ρ_i are varied. **(e, f)** Replication factors of the different control loops are varied.

the FIT rates of the wheel 3 and 4 control loops are still high, these improvements are insignificant overall. Configuration 4, however, reduces the FIT rates of all control loops, which shows that a substantial reduction of the overall FIT rate is possible if the slightly degraded control quality is acceptable.

In scenario (ii), however, the overall FIT rates of the four configurations exhibit a different pattern (Fig. 2(f)). Configuration 2 significantly increases the FIT rates of the of wheel 1 and 2 control loops, thus negatively affecting the total FIT rate. Configurations 3 and 4 reduce the total FIT, but not by much. There is no change in the order of magnitude of the overall FIT, and hence, in this scenario, it is not worthwhile to degrade the control quality in favor of additional replicas.

In general, we observe that increasing the control period to $1.75\,ms$ is not a worthwhile tradeoff in either case, whereas increasing the period to $2\,ms$ shows clear reliability benefits in scenario (i), but not in scenario (ii). This simple case study thus highlights the importance of identifying and strengthening the weakest link of a system (in this case, the wheel 3 and 4 control loops), and that the proposed analysis is an effective aid for doing so. In summary, the presented analysis enables a systematic design-space exploration for the objective of maximizing the overall system reliability.

VI. Related Work

The (m, k)-firm model was first studied in the context of real-time streams [18]. Since then, many analyses and system designs have been proposed for applications with (m, k)-firm specifications, mainly focussing on their temporal aspects (*e.g.*, see [3]). We use the (m, k)-firm specification to model the control system robustness, where the specification is a function of control loop iteration failures, including failures in the time domain as well as failures in the value domain.

With regard to real-time networks, several reliability analysis techniques have been proposed to date, particularly of the CAN bus, under EMI-induced transmission errors. For example, Punnekkat et al. [32] and Broster et al. [4] proposed analyses to bound the response time of CAN messages assuming a sporadic and a Poisson model for EMI, respectively, and recently, Sebastian et al. [35] proposed using hidden Markov models. We reuse Broster et al.'s results [4] in our work.

Related work in the NCS domain has focussed on evaluating the criteria for control stability and performance, *i.e.*, to what extent a control system can deviate from ideal operating conditions without jeopardizing its functionality, in the wake of various network failures (*e.g.*, [6, 25]). We solve an orthogonal problem of determining *when* and *how frequently* such robustness criteria are not met, *i.e.*, how likely it is that an inherently robust control system deviates from ideal operating conditions to such a degree that its correctness is at risk.

In related work targeting overall system reliability, Dugan and Van Buren [11] evaluateds the reliability of fly-by-wire systems with passive replication (hot standbys) using Markov models to evaluate the state transition probabilities when a system component fails, and fault trees to evaluate the individual states. In contrast to our system model, their system is not distributed and the analysis does not target general control applications. Sinha [37] proposed a reliability analysis of a fail-operational brake-by-wire system networked using CAN and FlexRay buses, but, unlike ours, Sinha's analysis is not defined at the message granularity and does not consider general control applications. Both works focus on specific system designs, whereas our analysis is for a generic NCS.

An alternative approach for quantifying the reliability of NCSs faced with transient component failures is the use of probabilistic model checkers such as PRISM [23] and

Storm [9]. For example, Kwiatkowska et al. [24] have analyzed the reliability of a very simple NCS model using PRISM. A systematic comparison with model-checking approaches and a thorough assessment of the respective benefits and limitations is beyond the scope of this paper, and left as future work.

VII. Conclusion and Future Work

We have proposed the first analysis to safely bound the FIT rate of CAN-based SISO NCSs that employ active replication to mitigate transient errors. Our analysis accounts for failures in both the time and value domains, and exposes the inherent robustness of NCSs in the form of (m, k)-firm constraints.

There is plenty of scope for future work, especially on more complex system models. To tolerate failures in the actuator task, it could be replicated like the sensor and controller tasks. Assuming that the physical actuator has some mechanism for redundancy suppression (*e.g.*, a hardware voter), such a system can be analyzed similarly to the presented analysis.

A fault-tolerant multi-input single-output (FT-MISO) system can be analyzed by modifying Step 3 in §IV to account for all replicas of the distinct sensor tasks in the system. In contrast, a fault-tolerant multi-input multi-output system can be analyzed as multiple independent FT-MISO systems, if an (m, k)-firm specification is given for each actuator.

For adaptive systems that allow dynamic reconfiguration of task replication factors based on runtime monitoring of the error rates, our analysis can be used to evaluate the reliability of different system modes. Similarly, in systems using passive replication or subject to permanent failures, our analysis yields a FIT rate for each state in the system's lifetime, *i.e.*, given a set of alive/dead replicas for that state, as in [11].

References

[1] A. Anta and P. Tabuada, "On the benefits of relaxing the periodicity assumption for networked control systems over CAN," in *RTSS 2009*.

[2] B. B. Brandenburg, "The schedulability test collection and toolkit," 2017, available at https://github.com/brandenburg/schedcat.

[3] I. Broster, G. Bernat, and A. Burns, "Weakly hard real-time constraints on controller area network," in *ECRTS 2002*.

[4] I. Broster, A. Burns, and G. Rodríguez-Navas, "Probabilistic analysis of CAN with faults," in *RTSS 2002*.

[5] ——, "Timing analysis of real-time communication under electromagnetic interference," *Real-Time Systems*, vol. 30, no. 1-2, pp. 55–81, 2005.

[6] A. Cetinkaya, H. Ishii, and T. Hayakawa, "Networked control under random and malicious packet losses," *IEEE Transactions on Automatic Control*, vol. 62, no. 5, pp. 2434–2449, 2017.

[7] K.-H. Chen, B. Bönninghoff, J.-J. Chen, and P. Marwedel, "Compensate or ignore? meeting control robustness requirements through adaptive soft-error handling," in *ACM SIGPLAN Notices*, vol. 51, no. 5. ACM, 2016, pp. 82–91.

[8] C. I. Chihaia, "Active fault-tolerance in wireless networked control systems," Ph.D. dissertation, Universität Duisburg-Essen, Fakultät für Ingenieurwissenschaften Elektrotechnik und Informationstechnik Automatisierungstechnik und komplexe Systeme, 2010.

[9] C. Dehnert, S. Junges, J.-P. Katoen, and M. Volk, "A storm is coming: A modern probabilistic model checker," *preprint arXiv:1702.04311*, 2017.

[10] M. Di Natale, H. Zeng, P. Giusto, and A. Ghosal, *Understanding and using the controller area network communication protocol: theory and practice*. Springer Science & Business Media, 2012.

[11] J. B. Dugan and R. Van Buren, "Reliability evaluation of fly-by-wire computer systems," *Journal of Systems and software*, vol. 25, no. 1, pp. 109–120, 1994.

[12] J. Ferreira, A. Oliveira, P. Fonseca, and J. Fonseca, "An experiment to assess bit error rate in CAN," in *RTN*, 2004.

[13] M. Gergeleit and H. Streich, "Implementing a distributed high-resolution real-time clock using the CAN-bus," in *iCC 1994*.

[14] A. Girault, H. Kalla, and Y. Sorel, "An active replication scheme that tolerates failures in distributed embedded real-time systems," in *Design Methods and Applications for Distributed Embedded Systems*. Springer, 2004, pp. 83–92.

[15] A. Gujarati and B. Brandenburg, "When is CAN the weakest link? A bound on failures-in-time in CAN-based real-time systems," in *RTSS 2015*.

[16] A. Gujarati, M. Nasri, and B. Brandenburg, "Lower-bounding the MTTF for systems with (m,k) constraints and IID iteration failure probabilities," 2017, available at https://people.mpi-sws.org/~arpanbg/papers_pdf/gujarati2017mttf.pdf.

[17] R. A. Gupta and M.-Y. Chow, "Overview of networked control systems," in *Networked Control Systems*. Springer, 2008, pp. 1–23.

[18] M. Hamdaoui and P. Ramanathan, "A dynamic priority assignment technique for streams with (m, k)-firm deadlines," *IEEE transactions on Computers*, vol. 44, no. 12, pp. 1443–1451, 1995.

[19] P. Hazucha and C. Svensson, "Impact of CMOS technology scaling on the atmospheric neutron soft error rate," *IEEE Transactions on Nuclear science*, vol. 47, no. 6, pp. 2586–2594, 2000.

[20] R. Isermann, R. Schwarz, and S. Stolzl, "Fault-tolerant drive-by-wire systems," *IEEE Control Systems*, vol. 22, no. 5, pp. 64–81, 2002.

[21] F. Johansson, "mpmath - Python library for arbitrary-precision floating-point arithmetic," 2017, available at http://mpmath.org/.

[22] W. Kuo and M. J. Zuo, *Optimal reliability modeling: principles and applications*. John Wiley & Sons, 2003.

[23] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *CAV 2011*.

[24] ——, "Controller dependability analysis by probabilistic model checking," *Control Engineering Practice*, vol. 15, no. 11, pp. 1427–1434, 2007.

[25] F.-L. Lian, J. Moyne, and D. Tilbury, "Analysis and modeling of networked control systems: MIMO case with multiple time delays," in *ACC 2001*.

[26] G. M. Lima and A. Burns, "A consensus protocol for CAN-based systems," in *RTSS 2003*.

[27] R. Mancuso, "Next-generation safety-critical systems on multi-core COTS platforms," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 2017, available at http://hdl.handle.net/2142/97399.

[28] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," in *MICRO, 2003*.

[29] N. Nakka, G. P. Saggese, Z. Kalbarczyk, and R. K. Iyer, "An architectural framework for detecting process hangs/crashes," in *EDCC 2005*.

[30] J. Noto, G. Fenical, and C. Tong, "Automotive EMI Shielding–Controlling Automotive Electronic Emissions and Susceptibility with Proper EMI Suppression Methods," *Laird Technologies, White Paper*, 2010.

[31] S. Poledna, *Fault-tolerant real-time systems: The problem of replica determinism*. Springer Science & Business Media, 2007, vol. 345.

[32] S. Punnekkat, H. Hansson, and C. Norstrom, "Response time analysis under errors for CAN," in *RTAS 2000*.

[33] J. Rufino, P. Verissimo, G. Arroz, C. Almeida, and L. Rodrigues, "Fault-tolerant broadcasts in CAN," in *FTCS*. IEEE, 1998.

[34] I. Saha, S. Baruah, and R. Majumdar, "Dynamic scheduling for networked control systems," in *HSCC 2015*.

[35] M. Sebastian, P. Axer, and R. Ernst, "Utilizing hidden markov models for formal reliability analysis of real-time communication systems with errors," in *PRDC 2011*.

[36] M. Sfakianakis, S. Kounias, and A. Hillaris, "Reliability of a consecutive k-out-of-r-from-n:F system," *IEEE Transactions on Reliability*, vol. 41, no. 3, pp. 442–447, 1992.

[37] P. Sinha, "Architectural design and reliability analysis of a fail-operational brake-by-wire system from ISO 26262 perspectives," *Reliability Engineering & System Safety*, vol. 96, no. 10, pp. 1349–1359, 2011.

[38] S. Stanley, "MTBF, MTTR, MTTF & FIT explanation of terms," *IMC Networks*, 2011.

[39] N. J. Wang, J. Quek, T. M. Rafacz, and S. J. Patel, "Characterizing the effects of transient faults on a high-performance processor pipeline," in *DSN 2004*.