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

Arpan Gujarati, Mitra Nasri, and Björn B. Brandenburg[1]

1 **Max Planck Institute for Software Systems, Kaiserslautern, Germany**
`arpanbg@mpi-sws.org`, `nasri@mpi-sws.org`, and `bbb@mpi-sws.org`

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 (EMI). 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. To enable a systematic and rigorous exploration of the resulting tradeoffs, we present an analysis to quantify the resiliency of fail-operational networked control systems against EMI-induced memory corruption, host crashes, and retransmission delays. Since control systems are typically 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 a control system's inherent robustness as an (m, k)-firm specification. A case study with an active suspension workload indicates that the analytical bounds closely predict the failure rate estimates obtained through simulation, thereby enabling a meaningful design-space exploration, and also demonstrates the utility of the analysis in identifying non-trivial and non-obvious reliability tradeoffs.

1998 ACM Subject Classification Embedded and cyber-physical systems

Keywords and phrases Probabilistic analysis, reliability analysis, networked control systems

Digital Object Identifier 10.4230/LIPIcs...YY

1 Introduction

Networked control systems (NCSs)—where sensors, controllers, and actuators belonging to one or more control loops are connected by a *shared* network—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 [21].

Like other embedded systems, NCSs are susceptible to both internal and external sources of *electromagnetic interference* (EMI), *e.g.*, spark plugs, TV towers, *etc.* [40]. In fact, the likelihood of soft errors due to EMI across a fleet of devices should not be underestimated. For example, Mancuso [35] 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 unmitigated soft errors can result in potentially catastrophic system failures, EMI-induced error scenarios are anticipated in the design of safety-critical systems, and 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 [10, 18, 24]).

Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

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, such questions (and many more like them) do not have obvious answers, 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 commercially-used controllers are typically safeguarded against disturbances and noise using appropriate limiting or clamping mechanisms, and 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—*i.e.*, completely unaffected by soft errors—forces excessively pessimistic answers relative to the "true" needs of the workload, and consequently results in under-utilized, cost-inefficient systems. Thus, to appropriately dimension a fail-operational real-time NCS, a robustness-aware reliability analysis is required.

This paper. We present a sound reliability analysis that evaluates a given configuration of an actively replicated NCS and quantifies its resiliency to EMI-induced transient errors, including message omission errors due to host crashes, incorrect computation errors due to memory corruption, and deadline violations due to retransmission delays. The objective is to provide system engineers with a sound method to evaluate (*i.e.*, safely bound) the reliability of an active replication scheme (*i.e.*, for a given number of replicas for each task in the NCS) assuming peak failure rates are known from empirical measurements and/or environmental modeling. We consider NCSs that are networked using a broadcast medium such as CAN (or Ethernet with a reliable broadcast primitive implemented on top) and evaluate them at the granularity of message exchanges between the distributed components.

Unlike traditional solutions based on hard real-time assumptions, 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" [22]. Blind and Allgöwer [4] have shown that the (m, k)-firm model is strictly stronger than the classical *asymptotic* requirement for control robustness (*e.g.*, as recently studied by Saha *et al.* [45]), which mandates that, as the number of control loop iterations approaches infinity, the failure rate should not exceed a given threshold. We thus use this model to bound the *failures in time* (FIT) of an NCS, *i.e.*, the expected number of *control failures* in one billion operating hours, where control failure denotes a violation of the (m, k)-firm constraint.

Overview. The proposed analysis consists of three steps. Given a model of a fault-tolerant single-input single-output (FT-SISO) control loop with active replication of its critical tasks (§2), the program-visible effects of EMI are first classified as crashes (resulting in message omissions), memory corruption (resulting in incorrect messages), and message retransmissions (resulting in deadline violations), and each of these errors is modeled probabilistically (§3). Second, an intermediate analysis (§4) then relates the probability of individual message errors to that of a *failed iteration* of a control loop, *i.e.*, where the controlled plant is not actuated as expected in an error-free iteration.[1] Finally, a reliability analysis upper-bounds

[1] Note the difference between a *failed iteration* of a control loop and *control failure.* A failed iteration

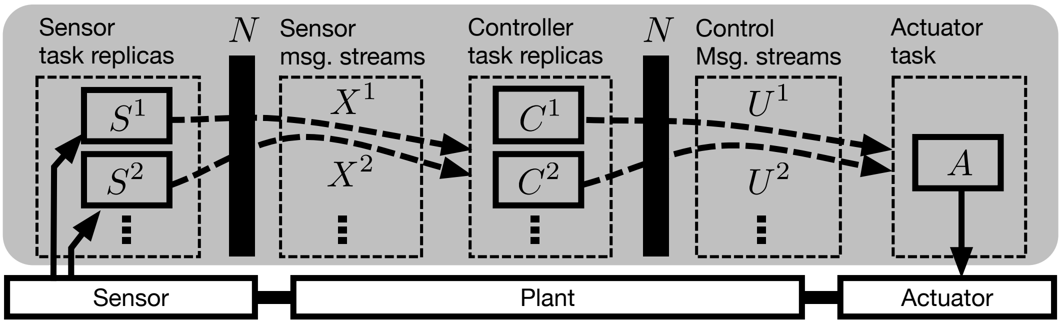


Figure 1 An FT-SISO control loop. Solid boxes denote hosts. Each dashed box denotes a task replica set or a set of message streams transmitted by a task replica set. Dashed arrows denote message streams broadcasted over the shared network N, *e.g.*, X^1 and X^2 are received by all tasks in C.

the FIT of an NCS, which may consist of one or more FT-SISO control loops, as a function of the control loops' respective (m, k)-firm specifications.

We have evaluated the proposed analysis with a case study exploring replication options for a CAN-based active suspension workload (§5). Our results show that analysis and simulation results closely track each other when configuration parameters are varied. We also demonstrate how the analysis can help in identifying non-obvious reliability tradeoffs, and identify the underlying timing analysis of the CAN bus [11] as the single greatest individual source of pessimism in our analysis due to its reliance on a *critical instant* that occurs only rarely.

2 System Model

We consider an FT-SISO networked control loop L deployed on hosts $H = \{H_1, H_2, \ldots\}$ connected by a broadcast medium N, which is shared with other traffic as well, *e.g.*, other control loops, the clock synchronization protocol, *etc.* A block diagram is shown in Fig. 1.

The sensor task replicas $S = \{S^1, S^2, \ldots\}$ periodically generate sensor output and broadcast it over N. As a convention, we let superscripts denote replica IDs. We let X^i denote the message stream carrying the sensor values of the i^{th} replica of the sensor task, and let $X = \{X^1, X^2, \ldots\}$ denote the set of all such message streams.

The controller task replicas $C = \{C^1, C^2, \ldots\}$, upon periodic activation, read the latest received sensor messages, compute a new control command for the plant, update their local states (*e.g.*, in a PID controller, the integrator), and broadcast the control command. They are assigned appropriate offsets to ensure that, in an error-free execution, the sensor messages are available before any controller task replicas are activated. The message streams carrying control commands are denoted $U = \{U^1, U^2, \ldots\}$.

The actuator task A is directly connected to the plant. Upon periodic activation, it reads the latest received control commands and actuates the plant accordingly. Like the controller tasks, A is also assigned an appropriate offset to ensure that, in an error-free execution, all control commands are received before its activation. Unlike the sensor and controller tasks, the actuator task A is not replicated since it requires special hardware in the plant actuator to handle redundant inputs [24]. We revisit this issue in §7.

All tasks and messages in the control loop have a period of T time units. The i^{th} runtime activations or jobs of sensor task replicas in $S = \{S^1, S^2, \ldots\}$ and controller task replicas in

is simply a deviation from an ideal, error-free scenario. Multiple failed iterations may lead to control failure if they violate the control loop's (m, k)-firm specification.

Algorithm 1 Voting procedure before the i^{th} activation of any controller task. Voting procedure for the actuator task is defined similarly by replacing the input set X_i with U_i in the following pseudocode.

```
1: procedure PeriodicControllerTaskActivation
2:     $Latest_i \leftarrow \emptyset$                                  ▷ start voting protocol
3:     for all $X_i^k \in X_i$ do
4:         if $X_i^k$ not received by its deadline then
5:             continue                                       ▷ also accounts for omissions
6:         $Latest_i \leftarrow Latest_i \cup X_i^k$
7:     if $Latest_i = \emptyset$ then return                              ▷ omit output
8:     $result_i \leftarrow SimpleMajority(Latest_i)$         ▷ break ties based on message IDs
9:     . . .                                               ▷ main logic of the task starts
```

$C = \{C^1, C^2, \dots\}$ are denoted $S_i = \{S_i^1, S_i^2, \dots\}$ and $C_i = \{C_i^1, C_i^2, \dots\}$, respectively; and the i^{th} job of actuator task A is denoted A_i. Similarly, the i^{th} messages in sensor message streams $X = \{X^1, X^2, \dots\}$ and controller message streams $U = \{U^1, U^2, \dots\}$ are denoted $X_i = \{X_i^1, X_i^2, \dots\}$ and $U_i = \{U_i^1, U_i^2, \dots\}$, respectively.

Finally, we let $\mathcal{U}_i$ denote the actuator command applied to the physical plant in the i^{th} iteration, *i.e.* output of job A_i, and let $\mathcal{U} = \{\mathcal{U}_1, \mathcal{U}_2, \dots\}$ denote the ordered set of such commands applied to the physical plant across all iterations.

Resolving redundant inputs. We assume that tasks resolve redundant inputs at the start of every iteration through voting (Algorithm 1). We let $V_i = \{V_i^1, V_i^2, \dots\}$ denote the set of voter instances that resolve the redundant inputs for controller jobs $C_i = \{C_i^1, C_i^2, \dots\}$, respectively, and let V_i^A denote the voter instance that resolves the redundant inputs for the actuator job A_i. Since all inputs are available before the task is activated in an error-free scenario, message streams that are delayed or omitted due to transmission or crash errors are ignored during voting (Line 5 of Algorithm 1). In the worst case, if no input is available on time to the voter due to errors, the task's activation is skipped, *i.e.*, the task's output for that iteration is omitted (Line 7). While computing the simple majority (Line 8), any ties in quorum size are broken deterministically using message IDs.

We (pessimistically) assume that corrupted message replicas are identical because it is a worst-case scenario w.r.t. the voting protocol. In particular, if the number of corrupted messages exceeds the number of correct messages, then assuming identically corrupted messages implies that the voting outcome is corrupted, while in the case of non-identically 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.

Assumptions. We assume that NCS hosts are synchronized using a clock synchronization protocol (such as the Precision Time 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) [41]. We also require that all NCS tasks are deterministic. Thus, given identical inputs and identical states, any two sensor (controller) task replicas produce identical sensor (control) messages, unless one is affected by memory corruption.

3 Fault Model

To lay the foundation for our analysis, we first give a precise fault model.

EMI-induced transient faults. We model the *raw transient faults*, *i.e.*, bit-flips on the network and in host memory, as random events following a Poisson distribution. 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. Let τ

and λ_i denote the peak rate of raw transient faults affecting the network and each host $H_i \in H$, respectively. We define the probability that x raw transient faults affect the network (respectively, host H_i) in any interval of length δ as $\mathcal{P}(x, \delta, \tau)$ (respectively, $\mathcal{P}(x, \delta, \lambda_i)$).

In practice, the peak fault 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. As a result, a Poisson process is a good *approximation* of the worst-case scenario, as previously discussed by Broster *et al.* [8]. For instance, in the case of network faults, τ is likely to exceed any transient actual fault rate τ_{actual} experienced in practice, which also varies over time and/or based on a system's current surroundings. Thus, as per the Poisson model, while the *actual* probability that the network experiences at least one transient fault in any interval of length δ is given by $\sum_{x>0} \mathcal{P}(x, \delta, \tau_{actual})$, we upper-bound this probability in our analysis by $\sum_{x>0} \mathcal{P}(x, \delta, \tau)$. That is, if $\tau > \tau_{actual}$, then $\sum_{x>0} \mathcal{P}(x, \delta, \tau) > \sum_{x>0} \mathcal{P}(x, \delta, \tau_{actual})$.[2]

Raw transient faults may manifest as program-visible retransmission, crash, and incorrect computation errors, which are also modeled probabilistically, as described below.

Retransmission errors. Networking protocols incorporate explicit mechanisms to mitigate the effects of transient faults on the wire, *e.g.*, error detection and correction in CAN [13]. Thus, we assume that network message corruptions are always detected, but may result in retransmission errors which may eventually lead to deadline violations. As in [7], we make the simplifying (but safe) assumption that every transient fault on the network causes a retransmission. Thus, we define the retransmission rate as τ, and the probability that x retransmissions occur in any interval of length δ as $\mathcal{P}(x, \delta, \tau)$. Given this, an upper bound on the probability that a message misses its deadline can be derived using prior work [8, 49].

In this work, we assume that an upper bound on the worst-case deadline-miss probability of any message instance belonging to any sensor message stream X^x or any control message stream U_x is known and denote this bound as $B(X^x)$ or $B(U^x)$, respectively.

Crash errors. These occur if the system suffers an EMI-induced corruption that causes an exception to be raised and the system to be rebooted, or that induces an unbounded hang that causes the system's watchdog timer to trigger a reboot, *e.g.*, see [37]. A crashed system remains unavailable for some time while it reboots and thus causes an interval in which messages are continuously omitted. We assume that the recovery interval on each host H_i is upper-bounded by R_i, which we assume also includes any delays that arise due to the need to resynchronize any application state after a crash.

Prior studies have shown that a large fraction of transient faults have no negative effects [53]. We thus assume a *derating factor* that accounts for masked transient faults, which can be determined empirically [36]. Let f_i denote the derating factor for crash errors on host H_i; the peak rate of crash errors on host H_i is then given by $\rho_i = f_i \lambda_i$. Using the peak crash error rate, we model crash errors like raw transient faults as random events following a Poisson distribution. Thus, we define the probability that x crash errors occur on host H_i in any interval of length δ as $\mathcal{P}(x, \delta, \rho_i)$.

Incorrect computation errors. These may occur if a message is corrupted before transmission (during preparation), before the network controller computes a checksum for subsequent error detection. Like crash errors, assuming a host-specific *derating factor* f'_i for incorrect computation errors, the average error rate on host H_i is given by $\kappa_i = f'_i \lambda_i$ and the probability

[2] This basic fact can be proved by representing the *cumulative density function* of the Poisson distribution in the form of an *upper incomplete gamma function* [3].

that x errors occur in any interval of length δ is given by $\mathcal{P}(x, \delta, \kappa_i)$.[3] Our notion of incorrect computation errors does not refer to software bugs or Byzantine errors.

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 be affected by *latent faults* (*i.e.*, state corruptions that have not yet been detected). Thus, the exposure interval of a message 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.

We assume that the worst-case exposure interval for each message in X^x, U^y, and $\mathcal{U}$ is known in advance and denote it using $E(X^x)$, $E(U^y)$, and $E(\mathcal{U})$, respectively.

Assumptions. Based on the stochastic nature of physical EMI processes, we consider EMI-induced transient faults, and hence basic message errors, to be independent. We do however account explicitly for correlated errors that arise from the system architecture, *e.g.*, deterministic replicas will produce the same wrong output if given the same wrong input. We also implicitly account for correlated surges in error rates across all components (*e.g.*, a UAV flying through a strong radar beam) since we analyze peak rates for all components.

While evaluating the EMI-induced errors discussed above, we assume that other system components are reliable, even though the NCS subsystem being analyzed may directly depend on them, *e.g.*, the power sources, the physical sensors and the actuators, the controlled physical plant, or the clock synchronization mechanism. 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, *e.g.*, using a fault tree analysis. This is a common way of decomposing the reliability analysis of the whole system into manageable components.

We also assume that the network protocol guarantees atomic broadcast, *i.e.*, messages are received consistently by either all hosts, or none. While Byzantine error scenarios violate this assumption, *e.g.*, [33], they occur with such low likelihood that they are best modeled as a separate, additive failure source and accounted for using a separate FIT analysis.

4 Probabilistic Analysis

We analyze the probability that the n^{th} iteration of the control loop fails, for any n.

As mentioned in §2, due to clock synchronization and the atomic broadcast assumption, message replicas are identical in an error-free scenario, *i.e.*, the messages in X_n carry identical sensor values and the messages in U_n carry identical control commands. However, due to incorrect computation errors, one or more messages in X_n may be corrupted. If the voters

[3] The choice of Poisson distribution for modeling both crash and incorrect computation errors is reasonable since real-time tasks are repeated, short workloads; thus, any generated message is equally likely to be affected by an error, and a host is equally likely to be crashed during any iteration of the task (see [31] for a mathematical basis for this argument).

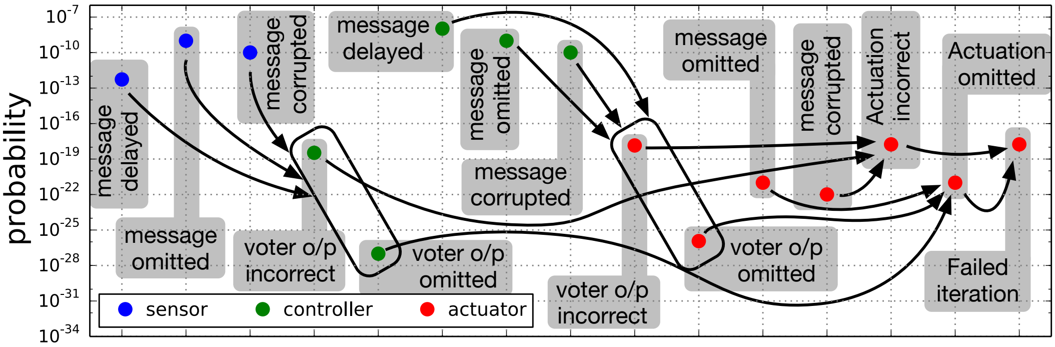


Figure 2 Error probabilities at different stages of a CAN-based wheel control loop (see §5 for details). Arrows denote dependencies among error probabilities of the different control loop stages. The error rates (per ms) are $\tau = 10^{-4}$ for the CAN bus, $\rho_i = 10^{-12}$ and $\kappa_i = 10^{-12}$ for each H_i hosting sensor and controller tasks, and $\rho_a = 10^{-24}$ and $\kappa_a = 10^{-24}$ for the actuator task's host H_a.

V_n choose a corrupted sensor value, then all messages U_n carrying the control commands are also corrupted. Messages in X_n could also be delayed or omitted due to transmission and crash failures, in which case the voters V_n work with fewer inputs. But if all the messages in X_n are either delayed or omitted, the controller jobs C_n have no inputs to work with, hence the messages U_n are not prepared. Similarly, the controller to actuator information flow may also be affected by errors, resulting in A_n's output $\mathcal{U}_n$ being corrupted or omitted. These dependencies are illustrated using an example in Fig. 2.

Based on this intuition, we next bound the probability that the final output $\mathcal{U}_n$ is corrupted or omitted, in a bottom-up fashion and in small steps of a few lemmas each. We use $P(\cdot)$ to denote *exact* probabilities and $Q(\cdot)$ to denote upper bounds on the exact probabilities. We derive the analysis as a function of the following exact (but unknown) probabilities.

For any message m, $\boldsymbol{P(m\ \ delayed)}$ denotes the exact probability of a deadline violation, $\boldsymbol{P(m\ \ corrupted)}$ denotes the exact probability of an incorrect computation due to host corruptions, and $\boldsymbol{P(m\ \ omitted)}$ denotes the exact probability of an omission due to host crashes, where m can be a sensor message in X_n, a controller message in U_n, or actuator A's output message $\mathcal{U}_n$. $\boldsymbol{P(SimpleMajority\ \ incorrect \mid \mathcal{I},\ \mathcal{C})}$ denotes the exact probability that given a set of incorrect inputs $\mathcal{I}$ and correct inputs $\mathcal{C}$ the *SimpleMajority*$(\mathcal{I} \cup \mathcal{C})$ in Algorithm 1 (Line 8) outputs an incorrect value.

In each step, we ensure that the derived probability is either independent of, or monotonic in these exact error probabilities. Thus, when instantiating the analysis using upper bounds on the exact probabilities, we implicitly guarantee that the *derived* iteration failure probability upper-bounds the *actual* iteration failure probability. Due to space constraints, we do not give a proof of monotonicity in this paper. We revisit the issue at relevant places where we explicitly add some pessimism to the analysis to ensure monotonicity.

Step 1. Analyzing correctness of $\boldsymbol{V_n^y}$'s output. We evaluate the probability that a controller voter instance V_n^y outputs an incorrect value because the majority of its inputs is corrupted. Recall from §2 that X_n denotes the set of all sensor message replicas that are inputs to each controller voter instance V_n^y.

We use a recursive procedure to compute the probability. At each step of recursion, we suppose that set X_n is *partitioned* into five subsets $\mathcal{O}_n$, $\mathcal{D}_n$, $\mathcal{I}_n$, $\mathcal{C}_n$, and $\mathcal{Z}_n$, *i.e.*, every element in X_n is *included in one* and *only one* of these five subsets. $\mathcal{O}_n$ denotes the messages that are omitted due to host crashes. $\mathcal{D}_n$ denotes the messages that are not omitted due to host crashes, but delayed due to retransmissions. $\mathcal{I}_n$ denotes the messages that are neither omitted nor delayed, but are incorrect due to host corruptions. $\mathcal{C}_n$ denotes the messages that are neither omitted, delayed, nor corrupted. Finally, $\mathcal{Z}_n$ denotes the messages whose *fate is undecided*, or in other words, each message $X_n^y \in \mathcal{Z}_n$ may still be omitted, delayed, or corrupted with some probability. Initially, $\mathcal{Z}_n = X_n$, and $\mathcal{O}_n = \mathcal{D}_n = \mathcal{I}_n = \mathcal{C}_n = \emptyset$. The recursion terminates when $\mathcal{Z}_n = \emptyset$.

The following lemma defines the recursive procedure, *i.e.*, its termination condition and the recursive step. The procedure yields the probability that voter instance V_n^y outputs an incorrect value given message sets $\mathcal{O}_n$, $\mathcal{D}_n$, $\mathcal{I}_n$, $\mathcal{C}_n$, and $\mathcal{Z}_n$.

▶ **Lemma 1.** *Given that messages in $\mathcal{O}_n$ are omitted due to host crashes, messages in $\mathcal{D}_n$ are not omitted but delayed due to retransmissions, messages in $\mathcal{I}_n$ are neither omitted nor delayed but are incorrect due to host corruptions, messages in $\mathcal{C}_n$ are neither omitted, delayed, nor corrupted, and each message $X_n^y \in \mathcal{Z}_n$ can be omitted with probability $P(X_n^y \text{ omitted})$, delayed with probability $P(X_n^y \text{ delayed})$, and corrupted with probability $P(X_n^y \text{ corrupted})$, the probability that voter instance V_n^y outputs an incorrect value is:*

$$P\left(\begin{array}{c} V_n^y \text{ output} \\ \text{incorrect} \end{array} \middle| \begin{array}{c} \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \\ \mathcal{C}_n, \mathcal{Z}_n \end{array} \right) = \begin{cases} P(\textit{SimpleMajority incorrect} \mid \mathcal{I}_n, \mathcal{C}_n) & \mathcal{Z}_n = \emptyset \\ \Gamma(\mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n) & \mathcal{Z}_n \neq \emptyset \end{cases} \tag{1}$$

where $\Gamma(\mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n) =$

$$\left(\begin{array}{l} \left(\begin{array}{l} P(X_n^s \text{ omitted}) \\ \times P(V_n^y \text{ output incorrect} \mid \mathcal{O}_n \cup \{X_n^s\}, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \setminus \{X_n^s\}) \end{array} \right) + \\ \left(\begin{array}{l} (1 - P(X_n^s \text{ omitted})) \times P(X_n^s \text{ delayed}) \\ \times P(V_n^y \text{ output incorrect} \mid \mathcal{O}_n, \mathcal{D}_n \cup \{X_n^s\}, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \setminus \{X_n^s\}) \end{array} \right) + \\ \left(\begin{array}{l} (1 - P(X_n^s \text{ omitted})) \times (1 - P(X_n^s \text{ delayed})) \times P(X_n^s \text{ corrupted}) \\ \times P(V_n^y \text{ output incorrect} \mid \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n \cup \{X_n^s\}, \mathcal{C}_n, \mathcal{Z}_n \setminus \{X_n^s\}) \end{array} \right) + \\ \left(\begin{array}{l} (1 - P(X_n^s \text{ omitted})) \times (1 - P(X_n^s \text{ delayed})) \times (1 - P(X_n^s \text{ corrupted})) \\ \times P(V_n^y \text{ output incorrect} \mid \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n \cup \{X_n^s\}, \mathcal{Z}_n \setminus \{X_n^s\}) \end{array} \right) \end{array} \right)$$

and X_n^s denotes the message with the smallest ID in $\mathcal{Z}_n$ (if $\mathcal{Z}_n \neq \emptyset$).

Proof. We prove the claim by induction over the size of message set $\mathcal{Z}_n$.

Induction base: The base case, *i.e.*, $|\mathcal{Z}_n| = 0$, is simple. When there are no messages in $\mathcal{Z}_n$, the output of V_n^y is determined only by messages in $\mathcal{I}_n$ and $\mathcal{C}_n$, since messages in $\mathcal{O}_n$ and $\mathcal{D}_n$ are either omitted or skipped and hence do not participate in voting. In particular, the output of V_n^y is determined by the output of the *SimpleMajority*$(\mathcal{I}_n \cup \mathcal{C}_n)$ procedure in Algorithm 1 (Line 8), and is given by $P(\textit{SimpleMajority incorrect} \mid \mathcal{I}_n,\ \mathcal{C}_n)$.

Induction step: We assume that for any $\mathcal{O}_n$, $\mathcal{D}_n$, $\mathcal{I}_n$, $\mathcal{C}_n$, and $\mathcal{Z}_n$, as defined above, and such that $|\mathcal{Z}_n| = k - 1 > 0$, the probability that voter V_n^y outputs an incorrect value is $P(V_n^y \textit{ output incorrect} \mid \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n)$. We prove the inductive step for $|\mathcal{Z}_n| = k$. Since $\mathcal{Z}_n \neq \emptyset$, X_n^s denotes the message with the smallest ID in $\mathcal{Z}_n$, and thus $\mathcal{Z}_n \setminus \{X_n^s\} = k - 1$. We consider four cases based on whether X_n^s is **(i)** omitted, **(ii)** not omitted but delayed, **(iii)** neither omitted nor delayed but corrupted, **(iv)** neither omitted, delayed nor corrupted.

Case (i). The case probability is $P(X_n^s \textit{ omitted})$. For this case, $\mathcal{O}_n \cup \{X_n^s\}$ denotes the new set of messages that are omitted due to host crashes, and $\mathcal{Z}_n \setminus \{X_n^s\}$ denotes the new message set where each message $X_n^y \in \mathcal{Z}_n \setminus \{X_n^s\}$ can be omitted, delayed, and corrupted with some probability. Since $|\mathcal{Z}_n \setminus \{X_n^s\}| = k - 1$, from the inductive hypothesis, the conditional probability that voter V_n^y outputs an incorrect value is $P(V_n^y \textit{ output incorrect} \mid \mathcal{O}_n \cup \{X_n^s\}, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \setminus \{X_n^s\})$.

Case (ii). The case probability is $(1 - P(X_n^s \textit{ omitted})) \times P(X_n^s \textit{ delayed})$. For this case, $\mathcal{D}_n \cup \{X_n^s\}$ denotes the new set of messages that are not omitted but delayed due to retransmissions, and $\mathcal{Z}_n \setminus \{X_n^s\}$ denotes the new message set where each message $X_n^y \in \mathcal{Z}_n \setminus \{X_n^s\}$ can be omitted, delayed, and corrupted with some probability. Thus, since $|\mathcal{Z}_n \setminus \{X_n^s\}| = k-1$, from the inductive hypothesis, the conditional probability that voter V_n^y outputs an incorrect value is $P(V_n^y \textit{ output incorrect} \mid \mathcal{O}_n, \mathcal{D}_n \cup \{X_n^s\}, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \setminus \{X_n^s\})$.

Case (iii). The case probability is $(1 - P(X_n^s \textit{ omitted})) \times (1 - P(X_n^s \textit{ delayed})) \times P(X_n^s \textit{ corrupted})$. $\mathcal{I}_n \cup \{X_n^s\}$ denotes the new set of messages that are neither omitted nor delayed but are incorrect due to host corruptions, and $\mathcal{Z}_n \setminus \{X_n^s\}$ denotes the new message set where each message $X_n^y \in \mathcal{Z}_n \setminus \{X_n^s\}$ can be omitted, delayed, and corrupted with some probability. Since $|\mathcal{Z}_n \setminus \{X_n^s\}| = k-1$, from the inductive hypothesis, the conditional probability that V_n^y outputs an incorrect value is $P(V_n^y \textit{ output incorrect} \mid \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n \cup \{X_n^s\}, \mathcal{C}_n, \mathcal{Z}_n \setminus \{X_n^s\})$.

Case (iv). The case probability is $(1 - P(X_n^s \textit{ omitted})) \times (1 - P(X_n^s \textit{ delayed})) \times (1 - P(X_n^s \textit{ corrupted}))$. For this case, $\mathcal{C}_n \cup \{X_n^s\}$ denotes the new set of messages that are neither omitted, delayed, nor corrupted, and $\mathcal{Z}_n \setminus \{X_n^s\}$ denotes the new message set where each message $X_n^y \in \mathcal{Z}_n \setminus \{X_n^s\}$ can be omitted, delayed, and corrupted with some probability. Thus, since $|\mathcal{Z}_n \setminus \{X_n^s\}| = k-1$, from the inductive hypothesis, the conditional probability that voter V_n^y outputs an incorrect value is $P(V_n^y \textit{ output incorrect} \mid \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n \cup \{X_n^s\}, \mathcal{Z}_n \setminus \{X_n^s\})$.

Combining the case probabilities and conditional probabilities for cases (i)-(iv) using law of total probability yields $\Gamma(\mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n)$, and since $\mathcal{Z}_n \neq 0$, this proves Eq. 1. ◀

Eq. 1 is, however, not monotonic in omission and delay probabilities. Its monotonicity in $P(X_n^s \textit{ omitted})$ and $P(X_n^s \textit{ delayed})$ depends on $P(X_n^s \textit{ corrupted})$. This is because the overall failure probability could be reduced by simply delaying or omitting a message, if that message is likely to be corrupted and thus can tilt the voting outcome in favor of an incorrect quorum. In order to remove this dependency on $P(X_n^s \textit{ corrupted})$, we replace $\Gamma(\mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n)$ in Lemma 1 with a slightly pessimistic term $\Gamma_\pi(\mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n)$, as stated in Corollary 2.

▶ **Corollary 2.** *Given that messages in $\mathcal{O}_n$ are omitted due to host crashes, messages in $\mathcal{D}_n$ are not omitted but delayed due to retransmissions, messages in $\mathcal{I}_n$ are neither omitted nor delayed but are incorrect due to host corruptions, messages in $\mathcal{C}_n$ are neither omitted, delayed, nor corrupted, and each message $X_n^y \in \mathcal{Z}_n$ can be omitted with probability $P(X_n^y \textit{ omitted})$, delayed with probability $P(X_n^y \textit{ delayed})$, and corrupted with probability $P(X_n^y \textit{ corrupted})$, the probability that voter instance V_n^y outputs an incorrect value is at most:*

$$Q\left(\begin{array}{c} V_n^y \textit{ output} \\ \textit{incorrect} \end{array} \middle| \begin{array}{c} \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \\ \mathcal{C}_n, \mathcal{Z}_n \end{array} \right) = \begin{cases} P(\textit{SimpleMajority incorrect} \mid \mathcal{I}_n, \mathcal{C}_n) & \mathcal{Z}_n = \emptyset \\ \Gamma_\pi(\mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n) & \mathcal{Z}_n \neq \emptyset \end{cases} \tag{2}$$

where $\Gamma_\pi(\mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n) =$

$$\left(\begin{array}{l} \left(\begin{array}{l} P(X_n^s \textit{ omitted}) \\ \times\, Q(V_n^y \textit{ output incorrect} \mid \mathcal{O}_n \cup \{X_n^s\}, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \setminus \{X_n^s\}) \end{array} \right) + \\ \left(\begin{array}{l} (1 - P(X_n^s \textit{ omitted})) \times P(X_n^s \textit{ delayed}) \\ \times\, Q(V_n^y \textit{ output incorrect} \mid \mathcal{O}_n, \mathcal{D}_n \cup \{X_n^s\}, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \setminus \{X_n^s\}) \end{array} \right) + \\ \left(\begin{array}{l} (1 - P(X_n^s \textit{ omitted})) \times (1 - P(X_n^s \textit{ delayed})) \times P(X_n^s \textit{ corrupted}) \\ \times\, Q(V_n^y \textit{ output incorrect} \mid \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n \cup \{X_n^s\}, \mathcal{C}_n, \mathcal{Z}_n \setminus \{X_n^s\}) \end{array} \right) + \\ \left(\begin{array}{l} (1 - P(X_n^s \textit{ omitted})) \times (1 - P(X_n^s \textit{ delayed})) \times (1 - P(X_n^s \textit{ corrupted})) \\ \times\, Q(V_n^y \textit{ output incorrect} \mid \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n \cup \{X_n^s\}, \mathcal{Z}_n \setminus \{X_n^s\}) \end{array} \right) + \\ \left(\begin{array}{l} \textit{max} \left(\begin{array}{l} (1 - P(X_n^s \textit{ omitted})) \times P(X_n^s \textit{ delayed}) \times P(X_n^s \textit{ corrupted}), \\ P(X_n^s \textit{ omitted}) \times (1 - P(X_n^s \textit{ delayed})) \times P(X_n^s \textit{ corrupted}) \end{array} \right) \\ \times\, Q(V_n^y \textit{ output incorrect} \mid \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n \cup \{X_n^s\}, \mathcal{C}_n, \mathcal{Z}_n \setminus \{X_n^s\}) \end{array} \right) + \end{array} \right)$$

and X_n^s denotes the message with the smallest ID in $\mathcal{Z}_n$ (if $\mathcal{Z}_n \neq \emptyset$).

Invoking Corollary 2 with the initial condition $\mathcal{Z}_n = X_n$, and $\mathcal{O}_n = \mathcal{D}_n = \mathcal{I}_n = \mathcal{C}_n = \emptyset$, *i.e.*, $Q(V_n^y \textit{ output incorrect} \mid \emptyset, \emptyset, \emptyset, \emptyset, \mathcal{X}_n)$, yields an upper bound on the probability that voter instance V_n^y outputs an incorrect value. For convenience, we let $Q(V_n^y \textit{ output incorrect}) = Q(V_n^y \textit{ output incorrect} \mid \emptyset, \emptyset, \emptyset, \emptyset, \mathcal{X}_n)$ in the following.

Step 2. Analyzing whether V_n^y omits its output. We evaluate the probability that a controller voter instance V_n^y omits its output because all its inputs were either delayed or omitted, *i.e.*, the special case in Algorithm 1 (Line 7). Once again, we use a recursive procedure to compute the probability. The procedure is similar to the one used in Step 1.

▶ **Lemma 3.** *Given that messages in $\mathcal{O}_n$ are omitted due to host crashes, messages in $\mathcal{D}_n$ are not omitted but delayed due to retransmissions, messages in $\mathcal{I}_n$ are neither omitted nor delayed but are incorrect due to host corruptions, messages in $\mathcal{C}_n$ are neither omitted, delayed, nor corrupted, and each message $X_n^y \in \mathcal{Z}_n$ can be omitted with probability $P(X_n^y \textit{ omitted})$, delayed with probability $P(X_n^y \textit{ delayed})$, and corrupted with probability $P(X_n^y \textit{ corrupted})$, the probability that voter instance V_n^y omits its output is:*

$$P\left(\begin{array}{c} V_n^y \textit{ output} \\ \textit{omitted} \end{array} \middle| \begin{array}{c} \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \\ \mathcal{C}_n, \mathcal{Z}_n \end{array} \right) = \begin{cases} 1 & \mathcal{Z}_n = \emptyset \wedge (\mathcal{I}_n \cup \mathcal{C}_n) = \emptyset \\ 0 & \mathcal{Z}_n = \emptyset \wedge (\mathcal{I}_n \cup \mathcal{C}_n) \neq \emptyset \\ \Lambda\left(\begin{array}{c} \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \\ \mathcal{C}_n, \mathcal{Z}_n \end{array} \right) & \mathcal{Z}_n \neq \emptyset \end{cases} \tag{3}$$

where $\Lambda(\mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n) =$

$$\left(\begin{array}{l} \left(\begin{array}{l} P(X_n^s \textit{ omitted}) \\ \times P(V_n^y \textit{ output omitted} \mid \mathcal{O}_n \cup \{X_n^s\}, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \setminus \{X_n^s\}) \end{array} \right) + \\ \left(\begin{array}{l} (1 - P(X_n^s \textit{ omitted})) \times P(X_n^s \textit{ delayed}) \\ \times P(V_n^y \textit{ output omitted} \mid \mathcal{O}_n, \mathcal{D}_n \cup \{X_n^s\}, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \setminus \{X_n^s\}) \end{array} \right) + \\ \left(\begin{array}{l} (1 - P(X_n^s \textit{ omitted})) \times (1 - P(X_n^s \textit{ delayed})) \times P(X_n^s \textit{ corrupted}) \\ \times P(V_n^y \textit{ output omitted} \mid \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n \cup \{X_n^s\}, \mathcal{C}_n, \mathcal{Z}_n \setminus \{X_n^s\}) \end{array} \right) + \\ \left(\begin{array}{l} (1 - P(X_n^s \textit{ omitted})) \times (1 - P(X_n^s \textit{ delayed})) \times (1 - P(X_n^s \textit{ corrupted})) \\ \times P(V_n^y \textit{ output omitted} \mid \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n \cup \{X_n^s\}, \mathcal{Z}_n \setminus \{X_n^s\}) \end{array} \right) \end{array} \right)$$

and X_n^s denotes the message with the smallest ID in $\mathcal{Z}_n$ (if $\mathcal{Z}_n \neq \emptyset$).

Proof. We prove the lemma by induction over the size of message set $\mathcal{Z}_n$. The base case ($|\mathcal{Z}_n| = 0$) is simple. When there are no messages in $\mathcal{Z}_n$, the output of V_n^y is determined only by messages in $\mathcal{I}_n$ and $\mathcal{C}_n$, since messages in $\mathcal{O}_n$ and $\mathcal{D}_n$ are either omitted or skipped and hence do not participate in voting. In particular, the output of V_n^y is omitted with probability one if $(\mathcal{I}_n \cup \mathcal{C}_n) = \emptyset$, as there are no inputs (Algorithm 1, Line 7). The output of V_n^y is omitted with probability zero if $(\mathcal{I}_n \cup \mathcal{C}_n) \neq \emptyset$, as there is at least one input, resulting in either a correct or an incorrect output (Algorithm 1, Line 8). The inductive step of the proof proceeds in the same way as in Lemma 1. ◀

Lemma 3 does not depend on the correctness of its inputs, but only on the timeliness of its inputs, unlike the simple majority procedure in Lemma 1. Hence, Eq. 3's monotonicity in $P(X_n^s \textit{ omitted})$ and $P(X_n^s \textit{ delayed})$ does not depend on $P(X_n^s \textit{ corrupted})$, unlike Eq. 1. As a result, use of a pessimistic term such as $\Gamma_\pi(\mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n)$ in Corollary 2 is not required in this case. Invoking Lemma 3 with the initial condition $\mathcal{Z}_n = X_n$, and $\mathcal{O}_n = \mathcal{D}_n = \mathcal{I}_n = \mathcal{C}_n = \emptyset$, *i.e.*, with $P(V_n^y \textit{ output omitted} \mid \emptyset, \emptyset, \emptyset, \emptyset, \mathcal{X}_n)$, yields the probability that voter instance V_n^y omits its output. For convenience, we let $P(V_n^y \textit{ output omitted}) = P(V_n^y \textit{ output omitted} \mid \emptyset, \emptyset, \emptyset, \emptyset, \mathcal{X}_n)$.

Step 3: Analyzing actuator voter instance $\boldsymbol{V_n^A}$. We bound the probability that actuator voter instance V_n^A outputs an incorrect value, because the majority of its inputs is corrupted, or that it does not choose anything, because all its inputs are either omitted or delayed.

Since all controller voter instances V_n operate on the same input values, if a correct voter instance V_n^y outputs an incorrect value because of wrong inputs, it implies that all correct voter instances in V_n output incorrect values. In such a scenario, the actuator voter V_n^A is guaranteed to get only incorrect control messages, since all of the control messages will be prepared using the corrupted sensor values.

A similar property holds for the controller voter output omission. Proper deadline and offset assignment guarantees that, in an error-free scenario, messages in X_n are transmitted before the voter instances in V_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 controller voter instance V_n^y does not choose any value because all its inputs are delayed or omitted, then all controller voter instances in V_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.

Let $Q(V_n^A \textit{ output incorrect})$ denote an upper bound on the probability that voter instance V_n^A's outputs an incorrect value, conditioned on the assumption that the sensor inputs of the controller voter instances V_n did not result in a corrupted output. Similarly, let $P(V_n^A \textit{ output omitted})$ denote the probability that voter instance V_n^A's output is omitted, conditioned on the assumption that the sensor inputs of the controller voter instances V_n did not result in an omitted output. Both $Q(V_n^A \textit{ output incorrect})$ and $P(V_n^A \textit{ output omitted})$ can be derived using the recursive procedures discussed in Steps 1 and 2, respectively, by replacing the set of voter inputs X_n with U_n (recall from §2 that U_n denotes the set of all inputs to V_n^A). The case that the sensor inputs of the controller voter instances V_n result in a corrupted or omitted output is accounted for in Step 5.

Step 4: Analyzing the final output $\boldsymbol{\mathcal{U}_n}$. We first bound the probability that the actuation during the n^{th} control loop iteration is incorrect (Lemma 4), followed by the probability that it is omitted (Lemma 5), and finally the joint probability of both events (Lemma 6). For brevity, we let $\phi_1 = Q(V_n^y \textit{ output incorrect})$, $\phi_{2a} = Q(V_n^A \textit{ output incorrect})$, $\phi_{2b} = P(\mathcal{U}_n \textit{ corrupted})$, $\omega_1 = P(V_n^y \textit{ output omitted})$, $\omega_{2a} = P(V_n^A \textit{ output omitted})$, and $\omega_{2b} = P(\mathcal{U}_n \textit{ omitted})$ in this step.

▶ **Lemma 4.** *The probability that the actuation during the n^{th} control loop iteration is incorrect is at most $\phi_1(1 + \phi_{2a}\phi_{2b}) + \phi_{2a} + \phi_{2b}$.*

Proof. We consider two cases based on whether the sensor inputs to any voter instance V_n^y results in corruption of the controller voter outputs (case 1) or not (case 2). The probability that case 1 occurs is $\phi_{case1} = P(V_n^y \textit{ output incorrect})$. For this case, since the sensor inputs to voter instance V_n^y results in corruption of its output, voter instances in all controller tasks choose an incorrect output. Thus, all control commands transmitted were incorrect, thus it is guaranteed that the actuation during the n^{th} control loop iteration is incorrect. Thus, the conditional probability in this case is $\phi_{cond1} = 1$.

The probability that case 2 occurs is $\phi_{case2} = 1 - \phi_{case1}$. For this case, the conditional probability that the actuation during the n^{th} control loop iteration is incorrect depends on two sources: **(a)** voter instance V_n^A's output can be incorrect, and **(b)** A's host can be affected by incorrect computation errors. The probability for case (a) is $\phi_{case2a} = P(V_n^A \textit{ output incorrect})$. The probability for case (b) is $\phi_{case2b} = P(\mathcal{U}_n \textit{ corrupted})$. Cases (a) and (b) are independent: (a) occurs because inputs to V_n^A were corrupted due to incorrect computation errors on the controller tasks' hosts, whereas (b) occurs due to incorrect computation errors on the actuator task's host. Thus, using theorem $P(A_1 \cup A_2) = P(A_1) + P(A_2) - P(A_1) \cdot P(A_2)$ for independent events A_1 and A_2, 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 the actuation during the n^{th} control loop iteration is incorrect is given by $\phi_{case1}\,\phi_{cond1} + \phi_{case2}\,\phi_{cond2}$. Upon expanding ϕ_{cond1}, ϕ_{cond2}, and ϕ_{case2}, and then rearranging the resulting expression w.r.t. ϕ_{case1}, we get

$$\phi_{case1}\,\phi_{cond1} + \phi_{case2}\,\phi_{cond2} = \left(\begin{array}{l} \phi_{case1} \times (1 - \phi_{case2a} - \phi_{case2b} + \phi_{case2a} \cdot \phi_{case2b}) \\ + \phi_{case2a} + \phi_{case2b} - \phi_{case2a} \cdot \phi_{case2b} \end{array} \right).$$

Further, upon dropping any negative terms for monotonicity, and since $\phi_{case1} \leq \phi_{1}$, $\phi_{case2a} \leq \phi_{2a}$, and $\phi_{case2b} = \phi_{2b}$, we have the following upper bound:

$$\phi_{case1}\,\phi_{cond1} + \phi_{case2}\,\phi_{cond2} \leq \phi_{1} \times (1 + \phi_{2a} \cdot \phi_{2b}) + \phi_{2a} + \phi_{2b}. \quad ◀$$

▶ **Lemma 5.** *The probability that the actuation during the n^{th} control loop iteration is delayed or omitted is at most $\omega_{1}(1 + \omega_{2a}\omega_{2b}) + \omega_{2a} + \omega_{2b}$.*

The proof of Lemma 5 is analogous to that of Lemma 4 and is thus omitted. In Lemma 6, we compose the probabilities derived in Lemmas 4 and 5 to derive the probability that the n^{th} control loop iteration fails, *i.e.*, that the actuation during this iteration is either incorrect or delayed (or omitted). We do not assume that the probabilities derived in Lemmas 4 and 5 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 the actuation is incorrect.

▶ **Lemma 6.** *The probability that the n^{th} control loop iteration fails is at most*

$$Q\left(n^{th} \textit{ control loop iteration fails}\right) = \left(\begin{array}{c} \phi_{1}(1 + \phi_{2a}\phi_{2b}) + \phi_{2a} + \phi_{2b} + \\ \omega_{1}(1 + \omega_{2a}\omega_{2b}) + \omega_{2a} + \omega_{2b} \end{array} \right). \tag{4}$$

Proof. Follows from Lemmas 4 and 5. ◀

In summary, Steps 1-4 account for all direct and indirect dependencies between the individual message error events and the final actuation of the controlled plant, and the derived $Q\left(n^{th} \textit{ control loop iteration fails}\right)$ automates propagation of the failure probability along this dependency tree. Although the analysis has exponential time complexity in the number of sensor message streams $|X_n|$ and the number of controller message streams $|U_n|$ (due to the branching recursions in Lemmas 1–3), since the number of replicas of any task is likely small, *i.e.*, typically under ten, the analysis can be quickly computed.

Upper-bounding the failure probability. Since exact message error probabilities are impossible to obtain, we instantiate the above analysis with upper bounds on the exact probabilities. Due to the monotonicity property of the analysis, the analysis remains sound despite the use of these upper bounds. We next define upper bounds on the message error probabilities for any sensor message X_n^y. The bounds for any control message U_n^y and actuator task's output message $\mathcal{U}_n$ are analogously defined.

The probability that any sensor message X_n^y is delayed beyond its deadline is bounded by $P(X_n^y \textit{ delayed}) \leq B(X^y)$ (as defined in §3). Let the host on which X_n^y's sender task is deployed be denoted H_a. Regarding message omission, suppose X_n^y is expected to be scheduled for transmission at the earliest by time t and at the latest by time $t + J$ (where J denotes the maximum release jitter of the message). Since R_a is the maximum time to recover from a crash error on host H_a, if there is at least one crash error during the interval

$[t-R_a,\ t+J)$, X_n^y's arrival may be skipped. Thus, $P(X_n^y\ \mathit{omitted}) \leq \sum_{x>0} \mathcal{P}(x,\ R_a+J,\ \rho_a)$. Regarding message corruptions due to incorrect computation errors, recall from §2 that the exposure interval for sensor message X_n^y is upper-bounded by $E(X^y)$. Thus, X_n^y may be corrupted if there is at least one incorrect computation error in this interval. Thus, $P(X_n^y\ \mathit{corrupted}) \leq \sum_{x>0} \mathcal{P}(x,\ E(X^y),\ \kappa_a)$.

The probability $P(\mathit{SimpleMajority\ incorrect} \mid \mathcal{I},\ \mathcal{C})$ is upper-bounded as follows, by making the worst-case assumption that incorrect inputs in $\mathcal{I}$ are identically faulty.

▶ **Lemma 7.** *If $\mathcal{C}$ and $\mathcal{I}$ denote the sets of correct and incorrect inputs, respectively, to the simple majority procedure in Algorithm 1 (Line 8), the probability that it outputs an incorrect value is at most:*

$$Q\left(\begin{array}{c} \mathit{SimpleMajority} \\ \mathit{incorrect} \end{array} \middle| \ \mathcal{I},\ \mathcal{C} \right) = \begin{cases} 1 & (n_i > n_c) \vee (n_i = n_c \neq 0 \wedge s_0 \in \mathcal{I}) \\ 0 & (n_i = n_c \neq 0 \wedge s_0 \in \mathcal{C}) \vee (n_i < n_c) \vee (n_i = n_c = 0) \end{cases}$$

where $n_c = |\mathcal{C}|$, $n_i = |\mathcal{I}|$, and $s_0 \in \mathcal{C} \cup \mathcal{I}$ denotes the message in $\mathcal{C} \cup \mathcal{I}$ with the smallest ID.

Proof. If $n_i > n_c$, the largest-sized quorum belongs to incorrect messages, and Algorithm 1's output is incorrect with probability 1. If $n_i = n_c \neq 0$, there are two largest-sized quorums. If message s_0 with the smallest ID is incorrect ($s_0 \in \mathcal{I}$), Algorithm 1 chooses an incorrect output with probability 1. Otherwise ($s_0 \in \mathcal{C}$), it chooses an incorrect output with probability 0. If $n_i < n_c$, the largest-sized quorum belongs to correct messages, and Algorithm 1's output is correct, *i.e.*, incorrect with probability 0. If $n_i = n_c = 0$, the voter has received no inputs, so the probability of choosing an incorrect output is 0. ◀

The IID property. Since each of the upper-bounds defined above is independent of n, Eq. 4 can be iteratively unfolded until it consists only of terms that are independent of n. The bound is thus identical for any control loop iteration. In addition, the upper bounds are derived under worst-case assumptions with respect to interference from other messages on the network [8, 11]. Failure of the n^{th} control loop iteration, defined as a deviation from an error-free execution of that iteration, is independent of whether past iterations encountered any failures or not. Thus, the bounds obtained using Eq. 4 for any two iterations n_1 and n_2 are mutually independent as well. As a result, when $Q(n^{th}\ \mathit{control\ loop\ iteration\ fails})$, which is monotonic in the error rates, is instantiated with the aforementioned upper-bounds on the error rates, it satisfies the IID property with respect to n.

FIT analysis. We use the probability of a failed control loop iteration, *i.e.*, the result of Lemma 6, 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 §1 that a control failure occurs if the control loop 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 [27], and leverage existing results [47] (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. Given an MTTF lower bound $MTTF_{LB}$ in hours, the FIT rate is computed as $10^9/MTTF_{LB}$ [50]. The full derivation and evaluation of the FIT analysis is available online [20].

5 Evaluation

The objective of the evaluation is threefold. First, in order to understand the accuracy of our approach, we compare the proposed analysis with simulations (§5.1). Second, we demonstrate the ability of our analysis to reveal and quantify non-obvious differences in the reliability of workloads with different (m, k) requirements and subject to error rates (§5.2). And third, we illustrate the utility of our analysis in a design-space exploration context by comparing FITs of different replication schemes (§5.3).

To implement the analysis, we extended the SchedCAT [5] library to support our system model for CAN-based NCSs, and implemented the proposed analysis on top. All computations related to the analysis were carried out at a precision of 200 decimal places using the *mpmath* Python library for arbitrary precision arithmetic [26]. As the underlying timing analysis of the network, we used Broster *et al.*'s probabilistic response-time analysis for CAN [7]. We also implemented a simulation of a CAN-based NCS that mimics the system model described in §2 along with CAN's network transmission protocol (see [13] for a detailed description).

We based our experiments on an active-suspension workload previously studied by Anta and Tabuada [2]. The system consists of four control loops (L_1, L_2, L_3, and L_4) corresponding to the control of four wheels (W_1, W_2, W_3, and W_4) with magnetic suspensions (period $1.75\,ms$), 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 $10\,ms$). In addition, we assumed the presence of a clock synchronization message with a period of $50\,ms$ [17] and a soft real-time message responsible for logging with a period of $100\,ms$. 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 40%. The clock synchronization message stream had 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 was set to $R_h = 1\,s$ for each host $H_h \in H$, and the exposure interval of each message stream was set to ten times its period to reflect the possibility of latent errors. The error rates and the (m, k) specifications used in each experiment are mentioned in the corresponding graphs. All error rates in the following are reported as the mean number of errors per *ms*. For context, Ferreira *et al.* [16] and Rufino *et al.* [44] reported peak transmission error rates range from 10^{-4} in aggressive environments to 10^{-10} in lab conditions, and as per Hazucha and Svensson [23], a 4 Mbit SRAM chip has a fault rate of approximately 10^{-12}. The error rates used in the following experiments have similar orders of magnitudes. Since the actuator task is not replicated, its host was assumed to be heavily shielded and thus assigned negligibly low crash and incorrect computation error rates.

5.1 Experiment 1: Simulation vs. Analysis

To assess the accuracy of the proposed analysis, we compared our proposed iteration-failure probability bound (§4) with an estimate of the actual mean iteration failure probabilities obtained through simulation. The pessimism incurred by the FIT analysis was already evaluated in prior work [20] and found to be acceptably small, and is not considered here.

Recall from §4 that: **(1)** we bound the message error probabilities based on the Poisson fault model in the form of $B(X^y)$ (for deadline violation errors), $P(X_n^y$ *corrupted*$)$ (for incorrect computation errors), and $P(X_n^y$ *omitted*$)$ (for crash-induced omission errors); and then **(2)** we use these message error probability bounds to upper-bound the iteration failure probability. In order to separately evaluate the pessimism incurred in steps (1) and (2), we used two separate simulator versions `Sim-v1` and `Sim-v2`.

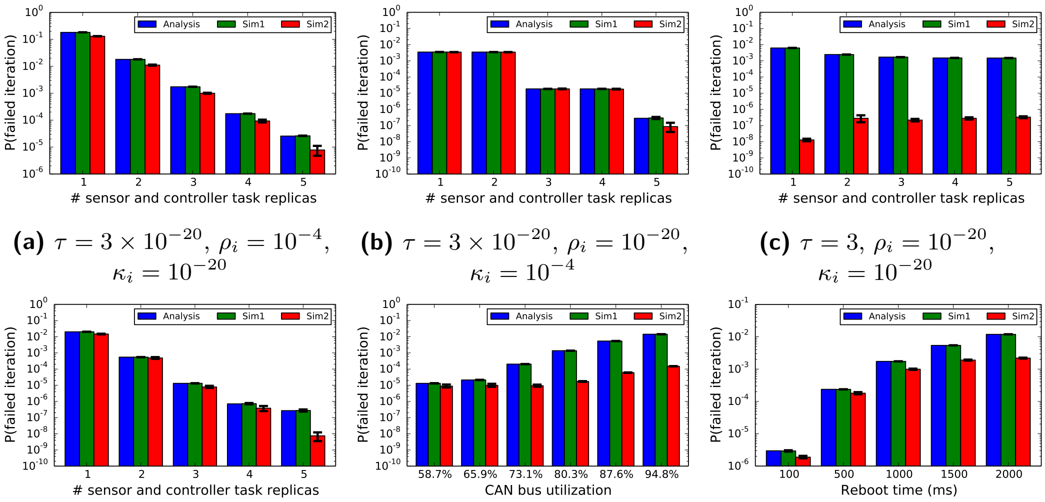
In the simple version (`Sim-v1`), for each sensor message X_n^y (and similarly for the control messages), the message error probabilities $B(X^y)$, $P(X_n^y$ *corrupted*$)$, and $P(X_n^y$ *omitted*$)$ were known to the simulator. Thus, each time any message $M_{i,n}^x$ is activated, the simulator draws a number uniformly at random from the range $[0, 1]$, compares it with the respective message error probabilities to decide whether the message is affected by that error type, and if the message is affected, simulates the corresponding error scenario. Thus, Sim-v1 does not actually simulate Poisson processes, nor does it simulate the CAN protocol, but it helps to isolate the pessimism incurred in step (2).

Sim-v2 is more complex than Sim-v1, and simulates the entire NCS along with the CAN transmission protocol. Separate Poisson processes are used to generate the respective fault events on each host and on the network. These fault events may manifest as message errors if they coincide with the message's lifetime, *e.g.*, as an incorrect computation error if they coincide with the message's exposure interval and a retransmission error if they coincide with the message's network transmission interval. Sim-v2 evaluates the pessimism incurred when upper-bounding the message error probabilities as a function of the raw transient fault rates, *e.g.*, when using the CAN timing analysis [8, 11] to determine $B(X^y)$, or when determining $P(X_n^y\ \mathit{corrupted})$ as in §4. It also evaluates whether this pessimism significantly impacts the overall iteration failure probability bound. Both Sim-v1 and Sim-v2 make the worst-case assumption that any two faulty message copies are identical, as in the analysis.

We compared the analysis, Sim-v1, and Sim-v2 for four different sets of error rates and replication factors. We used higher error rates for this experiment than can be realistically expected (and much higher than those used in the later experiments) as otherwise the simulations would be extremely time-consuming. To understand the effects of individual error types, we first compared three scenarios in which respectively only one of the three error types was assigned a significant rate, *i.e.*, $\rho_i = 10^{-4}$, $\kappa_i = 10^{-4}$, and $\tau = 3$, respectively, whereas the others were assigned negligible values, *i.e.* $\approx 10^{-20}$. Additionally, we evaluated a fourth scenario where all three error types have significant rates, *i.e.*, $\rho_i = 10^{-5}$, $\kappa_i = 10^{-5}$, and $\tau = 3 \times 10^{-1}$. Finally, to understand the effects of bus utilization and reboot time on the analysis, we compared the analysis, Sim-v1, and Sim-v2 for different CAN bus utilizations (by assuming increased message payload sizes) and for different reboot times ($100\,ms$-$2000\,ms$), with a replication factor of three. The results are shown in Figs. 3a–3f.

Observations. Several trends can be clearly seen. First, in *all* evaluated scenarios, the analysis results always track Sim-v1 extremely closely, which indicates that any pessimism introduced in step (2) to ensure monotonicity of the model with respect to the error rates is negligible. The results shown in Figs. 3a, 3b, and 3d further show that the analysis tracks Sim-v2 quite closely, too, provided that the underlying CAN timing analysis is not the bottleneck (*i.e.*, if message delays are not the dominant source of failures). Specifically, we observe that the full analysis, including step (1), results in less than an order of magnitude difference between the predicted and observed failure probabilities if crash or incorrect computation errors are non-negligible. This confirms the overall accuracy of the approach for the intended use cases: the proposed analysis closely tracks and soundly bounds the actual iteration failure probabilities in the presence of crashes, retransmissions, and message corruptions.

However, as is evident from Figs. 3a, 3b, and 3d, there exist cases where the analysis diverges significantly from Sim-v2. The common factor in these scenarios is that the underlying CAN analysis is the dominating factor. Most prominently, this is visible in Fig. 3c, which focuses exclusively on transmission faults: while the analytical failure bound is initially large and then decreases gradually with increasing replication factor, the observed failure probability is several orders of magnitude smaller than the analytical bound and actually indicates the opposite trend—the analysis is not at all a good predictor of actual failure rates in this scenario. Fig. 3e indicates that the gap between Sim-v2 and the analysis increases with CAN bus utilization. And even in Fig. 3a, when the replication factor is increased to five (resulting in high network contention), Sim-v2 begins to deviate from the analysis.



(a) $\tau = 3 \times 10^{-20}$, $\rho_i = 10^{-4}$, $\kappa_i = 10^{-20}$

(b) $\tau = 3 \times 10^{-20}$, $\rho_i = 10^{-20}$, $\kappa_i = 10^{-4}$

(c) $\tau = 3$, $\rho_i = 10^{-20}$, $\kappa_i = 10^{-20}$

(d) $\tau = 3 \times 10^{-1}$, $\rho_i = 10^{-5}$, $\kappa_i = 10^{-5}$

(e) $\tau = 3 \times 10^{-1}$, $\rho_i = 10^{-5}$, $\kappa_i = 10^{-5}$

(f) $\tau = 3 \times 10^{-20}$, $\rho_i = 10^{-4}$, $\kappa_i = 10^{-20}$

Figure 3 Comparing the iteration failure probability bound derived from the analysis with the estimates derived from simulation versions `Sim-v1` and `Sim-v2`. The vertical errors bars along with the simulation estimates denote 99% confidence intervals. Insets **(a)**, **(b)**, **(c)**, and **(d)** illustrate the variation in the iteration failure probability when the number of sensor and controller task replicas of L_1 are increased from one to five, for different sets of error rates. Insets **(e)** and **(f)** illustrate the impact of increasing CAN bus utilization and reboot times, respectively, for three replicas.

We attribute the pessimism caused by the timing analysis to the fact that not every message instance experiences worst-case interference during transmission (*i.e.*, not every message is released at a *critical instant*), and consequently, the derived deadline violation probability is extremely pessimistic for most message instances.

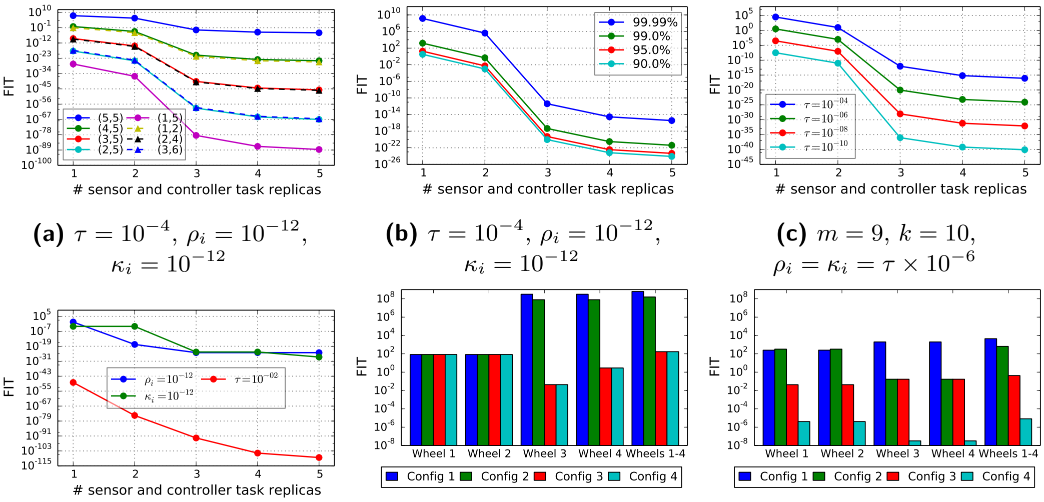
We conclude that the pessimism incurred by the current CAN timing analysis is significant, however, this has a measurable effect only in cases where the network becomes the dominant reliability bottleneck. As we will show with the next experiment, this is rather unlikely in the case of realistic fault rates (in contrast to the extremely high rates assumed in this experiment for the sake of simulation speed, which exaggerate the impact of the CAN analysis).

Finally, Fig. 3f indicates that the pessimism incurred by step (1) also increases with the reboot time, which is also an exaggerated trend due to the extremely high rate of crash failures in this scenario (*i.e.*, $\rho_i = 10^{-4}$ per millisecond, which means a reboot is expected every 10 seconds on average). As a result, with increasing reboot times, it becomes more likely that a crash fault affects an already-crashed host while it is rebooting—which "masks" in part the effects of the prior crash, which our analysis does not exploit. For more realistic crash rates, the effect is negligible, and even in this exaggerated setup, the analysis stays within an order of magnitude of the observed failure rate (note the y-axis scale in Fig. 3f).

In summary, we conclude from the overall small gap to the `Sim-v2` baseline that the incurred pessimism is not significant in cases where the crash and incorrect computation error rates are non-negligible, and where network congestion is not the sole dominating bottleneck.

5.2 Experiment 2: FIT for Different Parameters

To evaluate the impact of different replication factors, (m, k) requirements, and environmental conditions, we next evaluated control loop L_1's FIT while varying the number of sensor and controller task replicas. Figs. 4a–4d present the results.



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

Figure 4 **(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.

In Fig. 4a, m and k were varied as follows: $1 \leq m \leq 5$, and $k = 5$ or $k = 2m$; and in Fig. 4b, m/k is 90%, 95%, 99%, or 100% (while minimizing m and k).

A hard specification, *i.e.*, where $m = k$, yields a much higher 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. 4b, if the desired reliability threshold is 10 FIT, a hard real-time analysis (*i.e.*, requiring a 100% iteration success rate) requires the use of three replicas, whereas if a 90% success rate is sufficient, then our analysis indicates that no replication is required. Fig. 4a 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 (and where k can hence be chosen to be arbitrarily large) can be easily supported by our analysis. Interestingly, different (m, k) specifications can result in very similar FIT rates, *e.g.*, the curves of $(3, 5)$ and $(2, 4)$ in Fig. 4a overlap.

Next, we varied the transmission error rate τ across 10^{-4}, 10^{-6}, 10^{-8}, and 10^{-10}. The crash and incorrect computation error rates were set to $\rho_i = \kappa_i = \tau \times 10^{-6}$. The results are illustrated in Fig. 4c. As expected, the FIT rates decrease as the error rates are lowered. The FIT rates also decrease with increasing replication, but this decrease is significant only up to three replicas. Confirming earlier observations [19], active replication in real-time systems results in diminishing returns or becomes counterproductive after some point, as it reduces the slack available for fault-induced retransmissions and results in increased FIT rates. In general, graphs such as these can help engineers identify the maximum reliability that they can extract out of a system by increasing its replication factor, or conversely, the minimum number of replicas needed to achieve a desired level of reliability.

To better understand the effects of individual error types, we computed FIT values for three different scenarios. In each scenario, only one of the three error types was assigned a significant rate, *i.e.*, $\rho_i = 10^{-12}$, $\kappa_i = 10^{-12}$, and $\tau = 10^{-2}$, respectively, whereas the others were assigned negligible values, *i.e.*, 10^{-48}. As apparent in Fig. 4d, the FIT rates are higher for crash and incorrect computation errors, but very low for transmission errors, despite a relatively high retransmission rate and even at high utilization (five replicas). This indicates the relative importance of tolerating host errors, at least when hard timeliness is not required, and also puts in perspective the pessimism observed in §5.1—while the CAN analysis is the single biggest source of pessimism, its overall contribution to the overall failure probability is relatively minor for realistic retransmission rates.

Fig. 4d also shows that active replication helps tolerate incorrect computation errors only if the number of replicas is odd (*i.e.*, the curve for $\kappa_i = 10^{-12}$ does not improve when going from 1 to 2, or 3 to 4, replicas), in contrast to crash errors, which become less relevant already with the first added replica (until a point of diminishing returns is reached at 3 replicas). This is expected, *e.g.*, with two replicas, the voting algorithm is unable to distinguish between a correct and an incorrect input, which is significant if the messages are frequently corrupted.

In summary, Experiment 2 demonstrates that the analysis allows engineers to quantify and explore an NCS's reliability under various environmental conditions (*i.e.*, for varying peak error rates) and for different levels of robustness (by varying (m, k) specifications).

5.3 Experiment 3: FIT for Different Replication Schemes

To demonstrate that the analysis is useful for identifying reliability bottlenecks with respect to resource constraints, and for identifying opportunities to significantly increase a system's reliability at modest costs, we conducted a case study in which we analyzed different replication schemes of the workload. Our objective was to identify a replication scheme with a FIT rate under 10. That is, if such an active suspension workload is deployed in, say, 100 million cars, then as per Mancuso's calculations (discussed in §1), no more than about one vehicle per day will experience a failure in its active suspension NCS.

We considered the following error rates: $\tau = 10^{-4}$, $\rho_i = 10^{-8}$, and $\kappa_i = 10^{-12}$. To model practical design constraints, we assumed that the rear wheels W_1 and W_2 were close to many electromechanical parts, and assigned the hosts of the respective sensor tasks an order of magnitude higher crash and incorrect computation error rates. The different configurations are summarized in Table 1 and the their FIT bounds are illustrated in Figs. 4e and 4f.

Given a period of $1.75\,ms$ and an (m, k)-firm specification of (9, 10) for each control loop, the bound on the total FIT rate without any replication is greater than 10^{10}. Can we find a replication scheme with a FIT rate under 10 and with as few replicas as possible?

To answer the question, we conducted an exhaustive search over all possible replication schemes, varying the replication factor of each task from one to five, ignoring any scheme that did not result in a schedulable system. While we do not report the results of this exhaustive search due to space constraints, we observed that all feasible replication schemes can be partitioned into a few groups, where each group corresponds to schemes that result in FIT rate bounds of roughly the same order of magnitude. Thus, for each group, we report only the scheme with the minimum number of replicas, as given by Configurations 1–4 in Table 1 and Fig. 4e (configurations 5-8 and the corresponding Fig. 4f are discussed below).

Unfortunately, none of the feasible replication schemes yields a FIT rate under 10. Configuration 1 contains two copies of the sensor and controller tasks for L_1 and L_2, which helps reduce their respective FIT rate to under 10^2, but the system's total FIT rate still remains high ($\approx 10^8$) owing to L_3 and L_4's high individual FIT rates. Adding an extra replica of the sensor task for L_3 and L_4 (Configuration 2) does not help reduce this difference, but adding an extra copy of both sensor and controller tasks for L_3 and L_4 (Configuration 3) reduces the total FIT to around 10^2. In fact, while L_3 and L_4 are the bottleneck in Configuration 1 and Configuration 2, the bottleneck in Configuration 3 is L_1 and L_2. At this point, it seems that adding another pair of replicas for the rear wheel sensors (Configuration 4) to tolerate the relatively higher fault rates might be sufficient to bring down the total FIT rate under 10. However, this does not yield any significant benefit, and since we have maxed

Config.	**Wheel 1**	**Wheel 2**	**Wheel 3**	**Wheel 4**	**Period**	$\mathbf{(m, k)}$	**Util.**
1	$2S$, $2C$	$2S$, $2C$	$1S$, $1C$	$1S$, $1C$	$1.75\,ms$	(9, 10)	59%
2	$2S$, $2C$	$2S$, $2C$	$2S$, $1C$	$2S$, $1C$	$1.75\,ms$	(9, 10)	68%
3	$2S$, $2C$	$2S$, $2C$	$2S$, $2C$	$2S$, $2C$	$1.75\,ms$	(9, 10)	77%
4	$4S$, $2C$	$4S$, $2C$	$2S$, $2C$	$2S$, $2C$	$1.75\,ms$	(9, 10)	96%
5	$2S$, $1C$	$2S$, $1C$	$1S$, $1C$	$1S$, $1C$	$1.25\,ms$	(3, 5)	68%
6	$2S$, $2C$	$2S$, $2C$	$2S$, $2C$	$2S$, $2C$	$2.50\,ms$	(19, 20)	55%
7	$3S$, $2C$	$3S$, $2C$	$2S$, $2C$	$2S$, $2C$	$2.50\,ms$	(19, 20)	61%
8	$3S$, $3C$	$3S$, $3C$	$3S$, $3C$	$3S$, $3C$	$2.50\,ms$	(19, 20)	81%

Table 1 Different replication schemes. Parameters xS and yC denote that x and y replicas were provisioned for the sensor and the controller task of the respective wheel control loops.

out the bus utilization, we cannot add any more replicas. This shows that with the current set of parameters, we cannot guarantee a FIT of under 10, which would have been difficult to realize without the proposed analysis.

Can we instead relax the parameters of the control loops at the cost of slightly affecting their *instantaneous quality-of-control* [2]? For example, does **(i)** a shorter period of $1.25\,ms$ with a relaxed (m, k)-firm specification of (3, 5), or alternatively, **(ii)** a relaxed period of $2.5\,ms$ with a stricter (m, k)-firm specification of $(19, 20)$ allow designing the system with the desired levels of reliability, *i.e.*, with a FIT rate of 10 or less? To answer this question, we once again exhaustively generated FIT bounds for all schedulable replication schemes and report four representative cases here (see Configurations 5–8 in Table 1 and Fig. 4f).

For case (i), the best possible FIT bound ($\approx 10^3$) is obtained when two copies of the L_1 and L_2 sensor tasks are provisioned (Configuration 5). While we could add a few more replicas to Configuration 5 without saturating the bus, this does not help to reduce the FIT bound any further. Case (ii), however, allows us to add many more replicas (Configurations 6–8) because of the relaxed period, yielding much better FIT bounds despite the stricter (m, k)-firm specification. In particular, Configuration 7 yields a total FIT bound under 1 and Configuration 8 yields a total FIT bound of around 10^{-5}. Thus, while case (i) is not a useful alternative, case (ii) shows clear reliability benefits. In fact, the substantial FIT reduction in case (ii) makes it a worthwhile tradeoff, despite the slightly degraded control quality [2], whereas case (i) would give up control quality for no appreciable gain in reliability.

In general, this case study highlights the importance of quantifying system reliability for design-space exploration and for identifying and strengthening the weakest link of a system (*e.g.*, in this study, L_3 and L_4 in Configurations 1–2, and L_1 and L_2 in Configuration 3), and that the proposed analysis is an effective aid in this process.

6 Related Work

The (m, k)-firm model was first studied in the context of real-time streams and control applications [22, 43]. 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 [6]). We use (m, k)-firm specifications to model control system robustness, where the specification is a function of control loop iteration failures in both the time and value domains.

With regard to real-time networks, several reliability analyses have been proposed to date, particularly of the CAN bus under EMI-induced retransmission errors, *e.g.*, [7, 11, 38, 42, 52]. For example, Punnekkat *et al.* [42] and Broster *et al.* [7] proposed analyses to bound the response time of CAN messages assuming a sporadic and a Poisson model of EMI, respectively, and recently, Sebastian *et al.* [46] proposed the use of hidden Markov models in this context. Our prior work [19] proposed an analysis to bound the probability of successful transmission of a single logical message stream over CAN assuming host failures. In this work, like some of the prior work, we use the Poisson model of EMI, but unlike all aforementioned analyses, we evaluate the reliability of an end-to-end NCS system model.

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.*, [9, 25, 32, 39]. In contrast, we abstract the control specifics and solve the related, but 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 controlled plant may enter an unrecoverable state.

In related work targeting overall system reliability, Dugan and Van Buren [14] evaluated the reliability of a specific system, namely a 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 reliability of each of the individual states. It is possible to extend our analysis for systems with passive replication in a similar manner. Sinha [48] proposed a reliability analysis of a fail-operational brake-by-wire system networked with CAN and FlexRay buses. Sinha's approach differs substantially from ours since it is not defined at the granularity of individual messages. In contrast to these works that focus on specific systems, our analysis targets broadcast-based NCSs in general.

An alternative approach for quantifying the reliability of NCSs faced with transient component failures is the use of probabilistic model checkers such as PRISM [29] and Storm [12]. This approach has been adopted in a number of works for reliability analysis of simple networked systems [1, 15, 28, 30, 34, 51]. While such model-checking approaches are very general, their weak spot is generally the question of scalability. In contrast, our analysis is specific to the presented NCS model, but in return does not suffer from state-space explosion issues. Due to space constraints, we omit a detailed comparison of our NCS-specific method with more general model checkers from this paper, but plan to carry out to a thorough assessment of the respective benefits and limitations in future work.

7 Conclusion

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) control loop can be analyzed by modifying Steps 1 and 2 in §4 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 [14].

References

1 Masakazu Adachi, Yiannis Papadopoulos, Septavera Sharvia, David Parker, and Tetsuya Tohdo. An approach to optimization of fault tolerant architectures using hip-hops. *Software: Practice and Experience*, 41(11):1303–1327, 2011.

2 Adolfo Anta and Paulo Tabuada. On the benefits of relaxing the periodicity assumption for networked control systems over CAN. In *Real-Time Systems Symposium, 2009, RTSS 2009. 30th IEEE*, pages 3–12. IEEE, 2009.

3 Robert B Ash. *Basic probability theory*. Courier Corporation, 2008.

4 Rainer Blind and Frank Allgöwer. Towards networked control systems with guaranteed stability: Using weakly hard real-time constraints to model the loss process. In *Decision and Control (CDC), 2015 IEEE 54th Annual Conference on*, pages 7510–7515. IEEE, 2015.

5 Björn B Brandenburg. The schedulability test collection and toolkit, 2017. Available at `https://people.mpi-sws.org/~bbb/`.

6 Ian Broster, Guillem Bernat, and Alan Burns. Weakly hard real-time constraints on controller area network. In *Real-Time Systems, 2002. Proceedings. 14th Euromicro Conference on*, pages 134–141. IEEE, 2002.

7 Ian Broster, Alan Burns, and Guillermo Rodríguez-Navas. Probabilistic analysis of can with faults. In *Real-Time Systems Symposium, 2002. RTSS 2002. 23rd IEEE*, pages 269–278. IEEE, 2002.

8 Ian Broster, Alan Burns, and Guillermo Rodriguez-Navas. Timing analysis of real-time communication under electromagnetic interference. *Real-Time Systems*, 30(1-2):55–81, 2005.

9 Ahmet Cetinkaya, Hideaki Ishii, and Tomohisa Hayakawa. Networked control under random and malicious packet losses. *IEEE Transactions on Automatic Control*, 62(5):2434–2449, 2017.

10 Cristian Ionut Chihaia. *Active fault-tolerance in wireless networked control systems*. PhD thesis, Universität Duisburg-Essen, Fakultät für Ingenieurwissenschaften» Elektrotechnik und Informationstechnik» Automatisierungstechnik und komplexe Systeme, 2010.

11 Robert I Davis, Alan Burns, Reinder J Bril, and Johan J Lukkien. Controller area network (CAN) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*, 35(3):239–272, 2007.

12 Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. A storm is coming: A modern probabilistic model checker. *preprint arXiv:1702.04311*, 2017.

13 Marco Di Natale, Haibo Zeng, Paolo Giusto, and Arkadeb Ghosal. *Understanding and using the controller area network communication protocol: theory and practice*. Springer Science & Business Media, 2012.

14 Joanne Bechta Dugan and Randy Van Buren. Reliability evaluation of fly-by-wire computer systems. *Journal of Systems and software*, 25(1):109–120, 1994.

15 Jonas Elmqvist and Simin Nadjm-Tehrani. Formal support for quantitative analysis of residual risks in safety-critical systems. In *High Assurance Systems Engineering Symposium, 2008. HASE 2008. 11th IEEE*, pages 154–164. IEEE, 2008.

16 Joaquim Ferreira, Arnaldo Oliveira, Pedro Fonseca, and José Fonseca. An experiment to assess bit error rate in CAN. In *Proceedings of 3rd International Workshop of Real-Time Networks (RTN2004)*, pages 15–18, 2004.

17 Martin Gergeleit and Hermann Streich. Implementing a distributed high-resolution real-time clock using the CAN-bus. In *1st international CAN-Conference*, volume 94, 1994.

18 Alain Girault, Hamoudi Kalla, and Yves Sorel. An active replication scheme that tolerates failures in distributed embedded real-time systems. In *Design Methods and Applications for Distributed Embedded Systems*, pages 83–92. Springer, 2004.

19 Arpan Gujarati and Björn B Brandenburg. When is CAN the weakest link? a bound on failures-in-time in CAN-based real-time systems. In *Real-Time Systems Symposium, 2015 IEEE*, pages 249–260. IEEE, 2015.

20 Arpan Gujarati, Mitra Nasri, and Björn Brandenburg. Lower-bounding the MTTF for systems with (m,k) constraints and IID iteration failure probabilities. In *Workshop on Security and Dependability of Critical Embedded Real-Time Systems (CERTS), 2017*.

21 Rachana A Gupta and Mo-Yuen Chow. Overview of networked control systems. In *Networked Control Systems*, pages 1–23. Springer, 2008.

22 Moncef Hamdaoui and Parameswaran Ramanathan. A dynamic priority assignment technique for streams with (m, k)-firm deadlines. *IEEE transactions on Computers*, 44(12):1443–1451, 1995.

23 Peter Hazucha and Christer Svensson. Impact of CMOS technology scaling on the atmospheric neutron soft error rate. *IEEE Transactions on Nuclear science*, 47(6):2586–2594, 2000.

24 Rolf Isermann, Ralf Schwarz, and Stefan Stolzl. Fault-tolerant drive-by-wire systems. *IEEE Control Systems*, 22(5):64–81, 2002.

25 Ning Jia, Ye-Qiong Song, and Rui-Zhong Lin. Analysis of networked control system with packet drops governed by (m, k)-firm constraint. In *Fieldbus Systems and Their Applications 2005*, pages 63–70. Elsevier, 2006.

26 Fredrik Johansson. mpmath - Python library for arbitrary-precision floating-point arithmetic, 2017. Available at `http://mpmath.org/`.

27 Way Kuo and Ming J Zuo. *Optimal reliability modeling: principles and applications*. John Wiley & Sons, 2003.

28 Marta Kwiatkowska, Gethin Norman, and David Parker. Controller dependability analysis by probabilistic model checking. *Control Engineering Practice*, 15(11):1427–1434, 2007.

29 Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *International conference on computer aided verification*, pages 585–591. Springer, 2011.

30 Florian Leitner-Fischer. *Causality Checking of Safety-Critical Software and Systems*. PhD thesis, 2015.

31 Xiaodong Li, Sarita V Adve, Pradip Bose, and Jude A Rivers. Architecture-level soft error analysis: Examining the limits of common assumptions. In *Dependable Systems and Networks, 2007. DSN'07. 37th Annual IEEE/IFIP International Conference on*, pages 266–275. IEEE, 2007.

32 Feng-Li Lian, James Moyne, and Dawn Tilbury. Analysis and modeling of networked control systems: MIMO case with multiple time delays. In *American Control Conference, 2001. Proceedings of the 2001*, volume 6, pages 4306–4312. IEEE, 2001.

33 George MA Lima and Alan Burns. A consensus protocol for CAN-based systems. In *Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE*, pages 420–429. IEEE, 2003.

34 Yu Lu. *Probabilistic verification of satellite systems for mission critical applications*. PhD thesis, University of Glasgow, 2016.

35 Renato Mancuso. *Next-generation safety-critical systems on multi-core platforms*. PhD thesis, University of Illinois at Urbana-Champaign, 2017.

36 Shubhendu S Mukherjee, Christopher Weaver, Joel Emer, Steven K Reinhardt, and Todd Austin. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, pages 29–40. IEEE, 2003.

37 Nithin Nakka, Giacinto Paolo Saggese, Zbigniew Kalbarczyk, and Ravishankar K Iyer. An architectural framework for detecting process hangs/crashes. In *European Dependable Computing Conference*, pages 103–121. Springer, 2005.

38 Nicolas Navet, Y-Q Song, and Françoise Simonot. Worst-case deadline failure probability in real-time applications distributed over controller area network. *Journal of Systems Architecture*, 2000.

39 Johan Nilsson et al. Real-time control systems with delays. 1998.

40 John Noto, Gary Fenical, and Colin Tong. Automotive EMI Shielding–Controlling Automotive Electronic Emissions and Susceptibility with Proper EMI Suppression Methods. *Laird Technologies, White Paper*, 2010.

41 Stefan Poledna. *Fault-tolerant real-time systems: The problem of replica determinism*, volume 345. Springer Science & Business Media, 2007.

42 Sasikumar Punnekkat, Hans Hansson, and Christer Norstrom. Response time analysis under errors for CAN. In *Real-Time Technology and Applications Symposium, 2000. RTAS 2000. Proceedings. Sixth IEEE*, pages 258–265. IEEE, 2000.

43 Parameswaran Ramanathan. Overload management in real-time control applications using (m, k)-firm guarantee. *IEEE Transactions on Parallel and Distributed Systems*, 10(6):549–559, 1999.

44 José Rufino, Paulo Verissimo, Guilherme Arroz, Carlos Almeida, and Luıs Rodrigues. Fault-tolerant broadcasts in CAN. In *Fault-Tolerant Computing, 1998. Digest of Papers. Twenty-Eighth Annual International Symposium on*, pages 150–159. IEEE, 1998.

45 Indranil Saha, Sanjoy Baruah, and Rupak Majumdar. Dynamic scheduling for networked control systems. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, pages 98–107. ACM, 2015.

46 Maurice Sebastian, Philip Axer, and Rolf Ernst. Utilizing hidden markov models for formal reliability analysis of real-time communication systems with errors. In *Dependable Computing (PRDC), 2011 IEEE 17th Pacific Rim International Symposium on*, pages 79–88. IEEE, 2011.

47 M. Sfakianakis, S. Kounias, and A. Hillaris. Reliability of a consecutive k-out-of-r-from-n:F system. *IEEE Transactions on Reliability*, 41(3):442–447, 1992.

48 Purnendu Sinha. Architectural design and reliability analysis of a fail-operational brake-by-wire system from ISO 26262 perspectives. *Reliability Engineering & System Safety*, 96(10):1349–1359, 2011.

49 Fedor Smirnov, Michael Glaß, Felix Reimann, and Jürgen Teich. Formal reliability analysis of switched ethernet automotive networks under transient transmission errors. In *Design Automation Conference (DAC), 2016 53nd ACM/EDAC/IEEE*, pages 1–6. IEEE, 2016.

50 Susan Stanley. MTBF, MTTR, MTTF & FIT explanation of terms. *IMC Networks*, pages 1–6, 2011.

51 Anton Tarasyuk, Elena Troubitsyna, and Linas Laibinis. Augmenting formal development of control systems with quantitative reliability assessment. In *Proceedings of the 2nd International Workshop on Software Engineering for Resilient Systems*, pages 61–70. ACM, 2010.

52 Ken Tindell and Alan Burns. Guaranteeing message latencies on control area network (CAN). In *Proceedings of the 1st International CAN Conference*. Citeseer, 1994.

53 Nicholas J Wang, Justin Quek, Todd M Rafacz, and Sanjay J Patel. Characterizing the effects of transient faults on a high-performance processor pipeline. In *Dependable Systems and Networks, 2004 International Conference on*, pages 61–70. IEEE, 2004.