Quantifying the Resiliency of Replicated Networked Control Systems to Transient Faults

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

I. Application Domain & Challenge

Networked control systems (NCS) [5]—where sensors, controllers, and actuators belonging to different control loops are connected through a shared network—are highly susceptible to both internal and external sources of electromagnetic interference (EMI), *e.g.*, engine movements, TV towers, *etc.* [7]. System engineers thus use active replication (or static redundancy) for ensuring that safety-critical NCSs are fail-operational [4, 6, 8]. Passive replication techniques, such as the use of hot/cold standbys, are insufficient in this regard because of the time-sensitive nature of NCSs.

However, coming up with a good scheme for static redundancy is a challenging problem for the following two reasons. First, any scheme should satisfy both dependability requirements and size, weight, power, and cost constraints of the platform. Second, it must take into account the inherent robustness of controllers for accurate reliability modeling. Our goal is to develop a reliability analysis that quantifies the resiliency of safety-critical, CAN-based NCSs with active replication towards EMI-induced transient failures. The analysis will provide system engineers with a method to evaluate their design choices w.r.t. the overall system reliability, and particularly evaluate any reliability bottlenecks in the designs.

II. Motivation

Consider an example CAN-based NCS that consists of multiple control systems, one of which involves remotely controlling an inverted pendulum with a single degree of freedom (Fig. 1). The pendulum (plant) is attached to an angle encoder (sensor) and a horizontal-moving cart with a servo motor (actuator), and controlled by a remote motion control task (controller). Host H_1 executes an angle encoder driver (sensor task) to receive the pulse from the angle encoder and broadcasts it over CAN, and a servo motor driver (actuator task) to receive the commands from the controller and communicate it to the servo motor. A remote host H_2, also connected to the CAN bus, executes the motion control task, along with tasks belonging to other control loops in the NCS.

Suppose that the transmission of a message from the sensor task is delayed due to EMI-induced retransmissions. The next instance of the periodic motion control task then generates commands for the servo motor based on a slightly stale feedback, resulting in over-compensation for the pendulum tilt. If message delays persist, the pendulum crashes, *i.e.*, it reaches an unrecoverable state. But if a message is successfully transmitted after a few delayed messages, the pendulum is quickly stabilized. Even for message corruptions, where the servo motor input is computed using a faulty value, occasional failures are typically tolerated using filtering algorithms or saturation of the control signal. Hence a single lost or corrupted message, or even a few in a row, do not result in immediate system failure, thanks to the inherent robustness of the NCS.

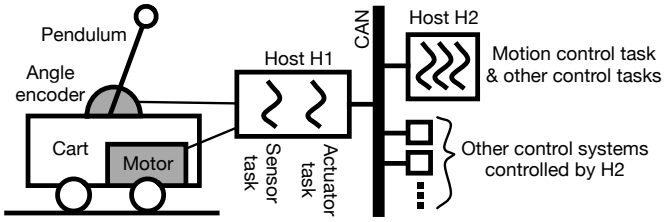

Fig. 1. Schematic diagram of a CAN-based NCS consisting of multiple control systems, including an inverted pendulum control system, which is controlled by a remote motion control unit over CAN using periodic feedback from the angle encoder and through a servo motor actuator.

This work. Unlike prior approaches, we aim to leverage the well-designed robustness in control systems, as exemplified above, and simultaneously consider the effects of active replication on the overall system reliability. That is, typical control loops tolerate a limited number of transient failures, which result in a degraded control performance, but not an unrecoverable system state. Our plan is to characterize the inherent robustness of each control loop in the NCS using an (m, k)-firm specification, *i.e.*, out of every k consecutive control loop iterations, at least m must be "correct," *e.g.*, see [2]. The (m, k)-firm specification allows us to go beyond the hard real-time assumption, which often results in excessive pessimism with regard to the "true" needs of the workload, and consequently, in under-utilized systems.

III. Problem Statement

We base our analysis on the formal model of a fault-tolerant single-input single-output (FT-SISO) control loop. The model and all the assumptions are defined below. A formal problem statement for the analysis is given in the end. For brevity, we consider only a single control loop belonging to the NCS.

SISO control loop. Let L denote a control loop consisting of plant P and a SISO controller C. The sensor output is generated periodically and broadcasted over CAN by the sensor task, which is denoted by S. The CAN message stream carrying the sensor values is denoted M_s. The controller task C is activated periodically, and upon activation, it reads the latest

sensor message received over CAN, computes a new control command for the plant, and broadcasts the control command over CAN. The CAN message stream carrying the control commands is denoted M_c. The actuator task, denoted A, is directly connected to the plant and also activated periodically. Upon activation, it reads the latest control command received from the controller, and actuates the plant accordingly.

FT-SISO control loop. To remain fail-operational despite crash failures, critical tasks in the SISO control loop may be actively replicated on independent hosts. We assume multiple functionally identical replicas of the sensor task and the controller task, denoted by sets $\overline{S} = \{S^1, S^2, \ldots\}$ and $\overline{C} = \{C^1, C^2, \ldots\}$, respectively (as a convention, we let superscripts denote replica IDs). We do not assume replicated actuators for now because it requires special hardware for the plant actuator to handle redundant control commands. Active replication results in additional message streams being transmitted over CAN. To model this redundancy, we let $\overline{M}_s = \{M_s^1, M_s^2, \ldots\}$ and $\overline{M}_c = \{M_c^1, M_c^2, \ldots\}$ denote the set of CAN message streams carrying the sensor values and the control commands sent by task replicas in $\overline{S}$ and $\overline{C}$, respectively. The redundant message streams are resolved by the receiver tasks using a simple majority voter.

Host model and register semantics. All tasks in $\overline{S}$ and $\overline{C}$, as well as task A, are deployed over the hosts in $H = \{H_1, H_2, \ldots\}$. Function $host$ defines a mapping from the set of all message streams to the set of all hosts. In particular, for any message stream M_s^i, $host(M_s^i)$ denotes the host that transmits the message stream. We assume that each CAN controller maintains a single buffer slot (a register) for each message stream, and thus old message values are overwritten by new message values. This assumption is reasonable because most CAN controllers have sufficient number of buffer slots for typical real-time workloads, *e.g.*, Atmel AT89C51CC03 has up to 15 slots. In case more buffer slots are required, software queues can be used with a similar effect [3, Chapter 3].

We do not assume any clock synchronization between the hosts, *i.e.*, any pair of clocks may drift apart arbitrarily over time. Thus, if any two tasks are deployed on separate hosts, they are not guaranteed to arrive synchronously even if they have the same period and the same release offset.

Message Failures due to EMI. *Raw transient faults*, *i.e.*, bit-flips on the network and in host memory, can manifest as program-visible transmission, crash, and commission failures. *Transmission failures* occur if messages are corrupted during transmission over CAN due to bit-flips. The CAN protocol tolerates such failures through error detection and automatic retransmission. A host experiences a *crash failure* if it suffers from EMI-induced transient corruptions that cause an exception to be raised and the system to be rebooted. Due to watchdog timers, even hangs translate into crash failures. *Commission failures* occur if messages are corrupted during preparation even before the network controller computes their checksums. We assume that the exposure interval of any message to raw transient faults is bounded. *Note that our notion of a commission failure does not refer to software bugs or malicious (or Byzantine) agents.*

We model each failure type as independent random events following a Poisson distribution, since Poisson distribution closely approximates independent random events without resulting in excessive pessimism [1]. We assume that the plant and the physical sensors and actuators execute failure-free.

Problem 1. Given the system and the failure model, what is the probability that any single iteration of the control loop "fails", *i.e.*, the plant actuation in an iteration deviates from the expected actuation in that iteration?

Problem 2. Given a solution to Problem 1, what is the probability that the entire control system "fails beyond recovery," *i.e.* its (m, k)-firm specification is violated?

IV. Fault Tree Analysis

We solve Problem 1 using a probabilistic failure analysis. The key ideas are summarized below using a fault tree (Fig. 2).

We use a fault tree to systematically deduce ways in which individual message failures result in a failed control loop iteration. Events in the fault tree are numbered as E1, E2, *etc.* and basic events are denoted using grey boxes. Since the plant and the physical actuators are assumed to execute failure-free, the failure event (E1) in our fault tree is defined w.r.t. actuator A's output, *i.e.*, whether "A's output is incorrect or delayed." The scenario occurs either if A is affected by crash failures, in which case its output is delayed (E3); if its output is incorrect due to commission failures on A's host (E4); or if it computes an incorrect output because it receives one or more incorrect inputs from the controller task replicas (E5).

Both E3 and E4 are basic events. E5's occurrence depends on whether inputs from the controller task replicas, *i.e.*, messages belonging to message streams in $\overline{M}_c$ are omitted, delayed, or incorrect. In particular, if $\overline{M}_c^O$, $\overline{M}_c^D$ and $\overline{M}_c^I$ denote message streams whose messages are omitted (E8), delayed (E9), or incorrect (E10) (respectively), then E5 may occur for each $\overline{M}_c^O, \overline{M}_c^D, \overline{M}_c^I \subseteq \overline{M}_c$ (E6) for which the simple majority results in an incorrect output (E7) (note that E5 has multiple E6-like children connected using an OR gate, and each of these children corresponds to a unique assignment for $\overline{M}_c^O, \overline{M}_c^D, \overline{M}_c^I \subseteq \overline{M}_c$). E10 is resolved further since it depends on other stages of the control loop.

E10 requires that a message belonging to each $M_c^i \in \overline{M}_c^I$ is incorrect. Thus, it has multiple E11-like children, one for each $M_c^i \in \overline{M}_c^I$, connected using an AND gate. Since M_c^i is sent by controller task C^i, E11 occurs either if a message from M_c^i is corrupted due to commission failures on $host(M_c^i)$, on which task C^i is executed (basic event E12), or if C^i received one or more incorrect inputs from the sensor task replicas, *i.e.*, messages belonging to message streams in $\overline{M}_s$ (E13). E13 is very similar to E5, except that it depends on message streams in $\overline{M}_s$ instead of $\overline{M}_c$. However, since message streams in $\overline{M}_s$ do not depend on prior stages of the control loop, but are generated by the sensor task replicas in $\overline{S}$ using the output

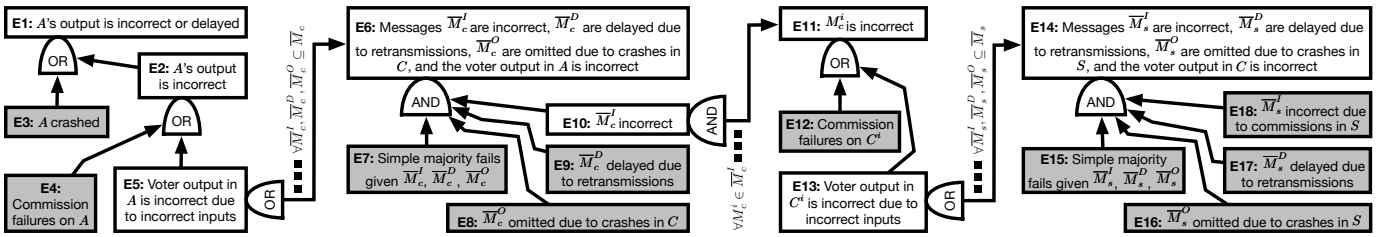

Fig. 2. Fault-tree analysis of an FT-SISO control loop. Grey-colored boxes denote basic events. For brevity, we do not illustrate all children of E5 corresponding to each $\overline{M}_c^O, \overline{M}_c^D, \overline{M}_c^I \subseteq \overline{M}_c$, but instead denote a representative child in E6 (similarly, for E10 and E13).

of the physical sensor (which is assumed to be reliable), E13 resolves into all basic events.

The fault tree analysis gives an overview of how individual message failures can result in the failure of a FT-SISO control loop iteration. But how likely is it that this failure event occurs? This requires analyzing the probability for each basic event in the fault tree, which is the subject of ongoing work.

V. Reliability Analysis

The probability that a single control loop iteration has failed, *i.e.*, solution to Problem 1, is insufficient to gauge the overall reliability of the entire NCS, mainly because a single missed or incorrect actuation does not imply that the control system has entered an unrecoverable state. In the following, we show how the (m,k)-firm robustness specification of the control loop can be used to compute its reliability, *i.e.*, solution to Problem 2. In particular, we plan to derive a *failures-in-time* (FIT) bound for each control loop in the NCS, *i.e.*, the expected number of failures in one billion operating hours, which can then be added to derive the FIT rate of the entire NCS. We discuss below the derivation of the probability density function (*p.d.f.*) for each control loop, which is the key step in the FIT analysis.

Derivation of *p.d.f.* Let F (for ***F**ailure*) denote a shorthand notation for the probability that any single iteration of a FT-SISO control loop fails and let $S = 1 - F$ (for ***S**uccess*) denote its complement. Using S and F, we define the *p.d.f.* for the control loop, *i.e.*, function $f(n)$, as the probability that the (m,k)-firm specification is not violated during the first $n-1$ iterations of A, but that it is violated for the first time during its n^{th} iteration. We derive $f(n)$ as follows.

Since the control loop violates its (m,k)-firm specification for the first time during its n^{th} iteration: **(i)** that iteration must be a failure, and **(ii)** it must be the $(k-m+1)^{\text{th}}$ failure in the last k iterations, *i.e.*, from $(n-k+1)^{\text{th}}$ to n^{th} iteration. In fact, (i) and (ii) imply that exactly $(k-m)$ iterations fail during $(n-k+1)^{\text{th}}$ to $(n-1)^{\text{th}}$ iteration of the control loop. Thus,

$$f(n) = P\left(\begin{smallmatrix}\text{no } (m,k)\text{-firm}\\ \text{violations in first}\\ n-k \text{ iterations}\end{smallmatrix}\right) P\left(\begin{smallmatrix}k-m \text{ iterations fail}\\ \text{during } (n-k+1)^{\text{th}}\\ \text{to } (n-1)^{\text{th}} \text{ iteration}\end{smallmatrix}\right) P\left(\begin{smallmatrix}n^{\text{th}}\\ \text{iteration}\\ \text{fails}\end{smallmatrix}\right).$$

Resolving this definition of $f(n)$, translating it into FIT for the FT-SISO control loop, and then translating FIT for each control loop into FIT for the NCS is our future work.

VI. Summary

As already pointed out in the earlier sections, **(i)** deriving the probabilities for each basic event in the fault tree and **(ii)** deriving the overall reliability of the NCS in terms of its FIT are still the subject of ongoing work. For (i), given any basic event in the fault tree, we must identify the exact (or a reasonable over-estimation of the) interval during which the occurrence of a message failure can cause a high-level failure. If one or more tasks share a host, then any solution to (i) must also account for correlated message omissions due to crash failures on that host. Problem (ii) is challenging because it requires deriving an exact solution for $f(n)$ and integrating it over an infinite region to compute the FIT.

The work is based on an FT-SISO model which corresponds to the most basic controller abstraction. However, complex control systems with multi-input single-output (MISO) and multi-input multi-output (MIMO) controllers, or with multi-staged controllers, are often used in practice. Ensuring that the reliability analysis applies to (or at least, can be readily extended to) each of these abstractions is another challenge.

Finally, recall that our main goal is to provide system engineers with a method to evaluate their design choices w.r.t. the overall system reliability. In this regard, we plan to demonstrate the usefulness of our FIT analysis by evaluating the following questions on a relevant control system workload: What should be the replication factor for each task? Which message stream-to-host mapping strategy should be used? Should a high-rate CAN be favored over a low-rate CAN despite its higher susceptibility to transmission failures?

References

[1] I. Broster, A. Burns, and G. Rodríguez-Navas, "Timing analysis of real-time communication under electromagnetic interference," *Real-Time Systems*, 2005.

[2] K.-H. Chen, B. Bönninghoff, J.-J. Chen, and P. Marwedel, "Compensate or ignore? Meeting control robustness requirements through adaptive soft-error handling." in *LCTES*, 2016.

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

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

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

[6] R. Isermann, R. Schwarz, and S. Stolzl, "Fault-tolerant drive-by-wire systems," *IEEE Control Systems*, 2002.

[7] J. Noto, G. Fenical, and C. Tong, "Automotive EMI shielding–controlling automotive electronic emissions and susceptibility with proper EMI suppression methods," 2010, available at https://www.lairdtech.com/sites/default/files/public/solutions/Laird-EMI-WP-Automotive-EMI-Shielding-040114.pdf.

[8] P. Sinha, "Architectural design and reliability analysis of a fail-operational brake-by-wire system from ISO 26262 perspectives," *Reliability Engineering & System Safety*, 2011.