TOWARDS "ULTRA-RELIABLE" CPS: RELIABILITY ANALYSIS OF DISTRIBUTED REAL-TIME SYSTEMS

THESIS APPROVED BY
THE DEPARTMENT OF COMPUTER SCIENCE
TECHNISCHE UNIVERSITÄT KAISERSLAUTERN
FOR THE AWARD OF THE DOCTORAL DEGREE

DOCTOR OF ENGINEERING (DR.-ING)

TO

ARPAN GUJARATI

DATE OF DEFENSE: 22.10.2020
DEAN: PROF. DR. JENS SCHMITT
REVIEWER: DR. BJÖRN B. BRANDENBURG
REVIEWER: PROF. DR. RUPAK MAJUMDAR
REVIEWER: PROF. DR. LINH THI XUAN PHAN

D 386

To my parents, Tanuja and Bharat.

ABSTRACT

In the avionics domain, “ultra-reliability” refers to the practice of ensuring *quantifiably negligible* residual failure rates in the presence of transient and permanent hardware faults. If autonomous Cyber-Physical Systems (CPS) in other domains, e.g., autonomous vehicles, drones, and industrial automation systems, are to permeate our everyday life in the not so distant future, then they also need to become ultra-reliable. However, the rigorous reliability engineering and analysis practices used in the avionics domain are expensive and time consuming, and cannot be transferred to most other CPS domains. The increasing adoption of faster and cheaper, but less reliable, Commercial Off-The-Shelf (COTS) hardware is also an impediment in this regard.

Motivated by the goal of ultra-reliable CPS, this dissertation shows how to soundly analyze the reliability of COTS-based implementations of actively replicated Networked Control Systems (NCSs)—which are key building blocks of modern CPS—in the presence of transient hardware faults. When an NCS is deployed over field buses such as the Controller Area Network (CAN), transient faults are known to cause host crashes, network retransmissions, and incorrect computations. In addition, when an NCS is deployed over point-to-point networks such as Ethernet, even *Byzantine* errors (i.e., inconsistent broadcast transmissions) are possible. The analyses proposed in this dissertation account for NCS failures due to each of these error categories, and consider NCS failures in both time and value domains. The analyses are also provably free of *reliability anomalies*. Such anomalies are problematic because they can result in unsound failure rate estimates, which might lead us to believe that a system is safer than it actually is.

Specifically, this dissertation makes four main contributions. (1) To reduce the failure rate of NCSs in the presence of Byzantine errors, we present a hard real-time design of a Byzantine Fault Tolerance (BFT) protocol for Ethernet-based systems. (2) We then propose a quantitative reliability analysis of the presented design in the presence of transient faults. (3) Next, we propose a similar analysis to upper-bound the failure probability of an actively replicated CAN-based NCS. (4) Finally, to upper-bound the long-term failure rate of the NCS more accurately, we propose analyses that take into account the temporal robustness properties of an NCS expressed as *weakly-hard* constraints.

By design, our analyses can be applied in the context of full-system analyses. For instance, to certify a system consisting of multiple actively replicated NCSs deployed over a BFT atomic broadcast layer, the upper bounds on the failure rates of each NCS and the atomic broadcast layer can be composed using the *sum-of-failure-rates* model.

ACKNOWLEDGMENTS

Like any Ph.D., I have had my share of ups and downs, and many people have helped me through this seven-year long journey.

First of all, I would like to thank my advisor, Björn B. Brandenburg, for accepting me into the prestigious Ph.D. program at MPI-SWS. When I started my Ph.D., I had zero research experience. Björn taught me the nuts and bolts of research. His continuous guidance and support has made me into the researcher that I am today. I would like to thank Björn for his patience while advising me, for accommodating my research interests, for always encouraging me to give my best, and for supporting my efforts towards an academic career.

I would also like to thank Rupak Majumdar and Linh Phan for reviewing my thesis and for providing me valuable feedback. I am grateful to Rupak for guiding me through one of my research projects during the later part of my Ph.D. I would like to express my thanks and appreciation to Sameh Elnikety, Yuxiong He, and Kathryn S. McKinley for mentoring me during my internship at Microsoft Research and for continuing to collaborate with me thereafter until the conclusion of my internship project. In addition, I would like to thank all my other co-authors, Mitra Nasri, Felipe Cerqueira, Manohar Vanga, Malte Appel, and Sergey Bozhko. I learnt a lot while working with everyone. Mitra, in particular, guided me through some tough times before deadlines. Also, her enthusiasm for research was incomparable, which made her stay at MPI-SWS very encouraging for me.

Many thanks also to members of the SysNets group at MPI-SWS, including our faculty members Peter Druschel, Krishna Gummadi, Deepak Garg, Allen Clement, Jonathan Mace, and Antoine Kaufmann, and senior students and colleagues Pramod Bhatotia, Bimal Viswanath, Paarijaat Aditya, Anjo Vahldiek-Oberwagner, Pedro Fonseca, Reinhard Munz, and Natacha Crooks, from whom I have learnt a lot over the years through both formal and informal discussions.

I would like to thank Felipe and Manohar again, for they were not just my co-authors and colleagues, but also my office-mates and friends, my train buddies each day I went from SB to KL, and my travel buddies at conferences. We joined the real-time systems research group around the same time, and have had loads of fun since then, including experiences that will stay with me forever. I would also like to thank my BITSian and school friends back home – conversations with you guys, even for the slightest of time, have always uplifted my mood.

Without the wonderful and extremely helpful staff at MPI-SWS, I would have had no time for research! Mary-Lou, Claudia, Brigitta, Vera, Ros, Susanne, Mouna, Annika, Gretchen, and Corinna have been very helpful in troubleshooting all kinds of problems I have had during my stay at MPI-SWS, be it regarding university admissions, residence in Germany, or even apartment renting. Rose's presentation and writing courses, and her feedback on my drafts, has helped me tremendously improve my soft skills. Tobias, Christian, and Carina have always been prompt at helping me with any IT-related issues. I would like to thank everyone for their help during my Ph.D.

Most importantly, I would like to thank my amazing family. I thank my parents, *mummy* and *pappa*, for their unwavering support, understanding, and encouragement, both during graduate school and the years before. They provided me means to get good education, and opportunities to learn beyond the school system, through books and travel, which has led me to where I am today. I also feel very blessed for receiving immense love, support, and goodwill from my extended family – *ba*, *nana*, *dadi*, *dada*, *foi*, *fua*, *kaki*, *kaka*, *mami*, *mama*, cousins, nephews, and nieces. I would also like to thank my parents-in-law, *mummy* and *daddy*, for their blessings and encouragement. Finally, I am deeply thankful to my wife, Aastha, for her love, support, and encouragement throughout my Ph.D. She has stood with me like a pillar through tough times, and has cheered me up every time I was disappointed. I simply cannot imagine finishing this journey without her – time to move on to further adventures in life!

PUBLICATIONS

Parts of this dissertation have appeared in the following publications.

[1] M. Appel, A. Gujarati, and B. B. Brandenburg. "A Byzantine Fault-Tolerant Key-Value Store for Safety-Critical Distributed Real-Time Systems." In: *2nd Workshop on the Security and Dependability of Critical Embedded Real-Time Systems (CERTS 2017)*. URL: https://certs2017.uni.lu/wp-content/uploads/sites/39/2017/11/certs_2017-proceedings.pdf.

[2] A. Gujarati and B. B. Brandenburg. "When Is CAN the Weakest Link? A Bound on Failures-in-Time in CAN-Based Real-Time Systems." In: *36th IEEE Real-Time Systems Symposium (RTSS 2015)*. San Antonio, Texas, pp. 249–260. ISBN: 978-1-4673-9507-6. DOI: 10.1109/RTSS.2015.31. URL: http://ieeexplore.ieee.org/document/7383582/.

[3] A. Gujarati, M. Nasri, R. Majumdar, and B. B. Brandenburg. "From Iteration to System Failure: Characterizing the FITness of Periodic Weakly-Hard Systems." In: *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*. Vol. 133. Leibniz International Proceedings in Informatics (LIPIcs). Stuttgart, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 9:1–9:23. ISBN: 978-3-95977-110-8. DOI: 10.4230/lipics.ecrts.2019.9. URL: http://drops.dagstuhl.de/opus/volltexte/2019/10746/.

[4] A. Gujarati, M. Nasri, and B. B. Brandenburg. "Lower-Bounding the MTTF for Systems with (m,k) Constraints and IID Iteration Failure Probabilities." In: *2nd Workshop on the Security and Dependability of Critical Embedded Real-Time Systems (CERTS 2017)*. URL: https://certs2017.uni.lu/wp-content/uploads/sites/39/2017/11/certs_2017-proceedings.pdf.

[5] A. Gujarati, M. Nasri, and B. B. Brandenburg. "Quantifying the Resiliency of Fail-Operational Real-Time Networked Control Systems." In: *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*. Vol. 106. Leibniz International Proceedings in Informatics (LIPIcs). Barcelona, Spain: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 16:1–16:24. ISBN: 978-3-95977-075-0. DOI: 10.4230/lipics.ecrts.2018.16. URL: http://drops.dagstuhl.de/opus/volltexte/2018/8988/.

[6] A. Gujarati, S. Bozhko, and B. B. Brandenburg. “Real-Time Replica Consistency over Ethernet with Reliability Bounds.” In: *26th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2020)*. Sydney, Australia, pp. 376–389. ISBN: 978-1-72815-499-2. DOI: `10.1109/RTAS48715.2020.00012`. URL: `https://ieeexplore.ieee.org/document/9113102/`.

[7] A. Gujarati, M. Appel, and B. B. Brandenburg. “Achal: Building Highly Reliable Networked Control Systems.” In: *15th ACM SIGBED International Conference on Embedded Software Companion (EMSOFT 2019)*. New York, New York: ACM Press, 2019, pp. 1–2. ISBN: 978-1-4503-6924-4. DOI: `10.1145/3349568.3351545`. URL: `http://dl.acm.org/citation.cfm?doid=3349568.3351545`.

CURRICULUM VITAE

EDUCATION

2014-2019	Ph.D. in Computer Science (dissertation phase) *MPI-SWS & TU Kaiserslautern, Germany*
2012-2014	Ph.D. in Computer Science (preparatory phase) *MPI-SWS & Saarland University, Germany*
2007–2011	B.E. (with Honors) in Computer Science *BITS Pilani, India*

WORK EXPERIENCE

2020	Postdoctoral Researcher, *MPI-SWS*
2012-2019	Ph.D. Student, *MPI-SWS*
2015	Research Intern, *Microsoft Research USA*
2011–2012	Software Development Engineer, *Citrix R&D India*
2011	Software Development Intel, *Intel India*

HONORS AND AWARDS

2020	Distinguished Paper Award *26th IEEE Real-Time and Embedded Technology and Applications Symposium*
2018	Best Presentation Award *30th Euromicro Conference on Real-Time Systems*
2017	Best Student Paper Award *18th ACM/IFIP/USENIX International Middleware Conference*
2014	Young Researcher *2nd Heidelberg Laureate Forum*
2013	Outstanding Paper Award *25th Euromicro Conference on Real-Time Systems*

PROFESSIONAL ACTIVITIES

Technical Program Committee	*RTEST WiP* (2018) *RTAS BP* (2019, 2020), *ECRTS AE* (2019), *Middleware DW* (2020)
Journal Reviewer	*TECS* (2019), *TDSC* (2019)
External Reviewer	*EuroSys* (2013, 2016, 2019), *RTSS* (2013, 2016, 2018, 2020), *RTAS* (2013, 2014, 2016), *ECRTS* (2013-2015, 2019), *RTNS* (2014-2016), *Systor* (2015, 2016), *Middleware* (2018), *EMSOFT* (2020)

TEACHING EXPERIENCE

2017	Teaching Assistant, Operating Systems *MPI-SWS & Saarland University*
2016	Teaching Assistant, Distributed Systems *MPI-SWS & Saarland University*
2014	Teaching Assistant, Foundations of CPS *MPI-SWS & TU Kaiserslautern*
2010	Teaching Assistant, Data Structures and Algorithms *BITS Pilani*

ADVISING

2017-2018	Malte Appel, Undergraduate Thesis *MPI-SWS & Saarland University*
2016	Rohith R, Summer Internship, *MPI-SWS*
2015	Akshay Aggarwal, Summer Internship, *MPI-SWS*

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

ABS	Anti-lock Braking System
ACC	Advanced Cruise Control
AER	Almost-Everywhere to Everywhere
AI	Artificial Intelligence
AIPS	Advanced Information Processing System
AP	Application Processor
API	Application Programming Interface
AR	Active Replication
AURIX	Automotive Realtime Integrated NeXt Generation Architecture
AVB	Audio Video Bridging
BFT	Byzantine Fault Tolerance
BG	Byzantine Generals
BIU	Bus Interface Unit
BLAS	Basic Linear Algebra Subprograms
CAN	Controller Area Network
CBS	Credit-Based Shaper
CDF	Cumulative Density Function
CDT	Control Data Traffic
CMOS	Complementary Metal–Oxide–Semiconductor
CNI	Communication Networking Interface
COTS	Commercial Off-The-Shelf
CPA	Compositional Performance Analysis
CPS	Cyber-Physical Systems
CPU	Central Processing Unit
CQL	Cassandra Query Language

CRC	Cyclic Redundancy Check
DECTED	Double-Error-Correcting Triple-Error-Detecting
DMAC	Deadline-Miss-Aware Control
DMR	Dual Modular Redundancy
DNN	Deep Neural Network
ECC	Error-Correcting Code
ECU	Electronic Control Unit
EIG	Exponential Information Gathering
EMI	Electromagnetic Interference
FCC	Flight Control Computer
FIFO	First In, First Out
FIT	Failures-In-Time
FLP	Fisher, Lynch, and Paterson
FMEA	Failure Mode and Effect Analysis
FP	Fixed Priority
FRT	Firm Real-Time
FTA	Fault-Tree Analysis
FTMP	Fault-Tolerant Multiprocessor
FT-SISO	Fault-Tolerant Single-Input Single-Output
GEDF	Global Earliest Deadline First
GNU	GNU’s Not Unix!
HRT	Hard Real-Time
IC	Interactive Consistency
IID	Independent and Identically Distributed
I/O	Input/Output
LAPACK	Linear Algebra PACKage
LET	Logical Execution Time
LSF	Link Shielding Factor
MAC	Message Authentication Codes

MAFT	Multiprocessor Architecture for Fault-Tolerance
MARS	Maintainable Real-Time System
MART	MARTingale approach
MIMO	Multi-Input Multi-Output
MISO	Multi-Input Single-Output
MPFR	Multiple Precision Floating-Point Reliably
MTBF	Mean Time Between Failures
MTTF	Mean Time To Failure
NCS	Networked Control System
NSF	Node Shielding Factor
OC	Operations Controller
ORDC	On-Demand Replica Consistency
OS	Operating System
OSF	Overall Shielding Factor
PBFT	Practical Byzantine Fault Tolerance
PE	Processing Element
PID	Proportional Integral Derivative
PMC	Probabilistic Model Checking
PMC-E	Probabilistic Model Checking (Exact)
PMC-P	Probabilistic Model Checking (Parametric)
PMF	Probability Mass Function
POSIX	Portable Operating System Interface
PTP	Precision Time Protocol
PTPd	Precision Time Protocol daemon
QMR	Quadruple Modular Redundancy
QoS	Quality of Service
RBD	Reliability Block Diagram
RBFT	Redundant Byzantine Fault Tolerance
ROBUS	Reliable Optical Bus

RPC	Remote Procedure Calls
RTS	Real-Time Systems
RTOS	Real-Time Operating Systems
RR	Round-Robin
SAP	Sound APproximation
SchedCAT	Schedulability test Collection And Toolkit
SECDED	Single-Error-Correcting Double-Error-Detecting
SIFT	Software Implemented Fault Tolerance
SISO	Single-Input Single-Output
SMR	State Machine Replication
SOFR	Sum-Of-Failure-Rates
SPIDER	Scalable Processor-Independent Design for Electromagnetic Resilience
SPoF	Single Point of Failure
SRP	Stream Reservation Protocol
SRT	Soft Real-Time
TKVS	Temporally-aware Key-Value Service
TMR	Triple Modular Redundancy
TSN	Time-Sensitive Networking
TTP	Time-Triggered Protocol
TTP/C	Time-Triggered Communication Protocol
UDP	User Datagram Protocol
UTC	Coordinated Universal Time
VAN	Vehicle Area Network
WCET	Worst-Case Execution Time
WCRT	Worst-Case Response Time

Part I

MOTIVATION AND BACKGROUND

1 INTRODUCTION

What is "ultra-reliability"? When a commercial aircraft is developed, multiple fault tolerance mechanisms are designed to mitigate the effects of faults that may arise once the aircraft is deployed. In particular, these mechanisms tolerate many unpreventable faults that result from exposure to radiation and electromagnetism. In addition, reliability analyses are applied to validate that the overall failure probability of the aircraft remains under a certified threshold even if faults occur at the maximum expected rates. To succinctly describe these practices, researchers in the early seventies defined "ultra-reliability" as the practice of ensuring *quantifiably negligible* residual failure rates using a combination of reliability engineering and analysis techniques [149], which is essential for building trustworthy safety-critical systems.

Unfortunately, safety-critical systems in other domains—such as autonomous vehicles, industrial automation systems, delivery and fire-safety drones, and telesurgery robots—are not engineered as rigorously as commercial aircraft, and therefore are not as reliable. For example, a recent study by Banerjee et al. [18] reports that autonomous cars tested in California between 2014 and 2017 are at least $4.22\times$ less reliable than airplanes per trip. The same study also estimates that autonomous cars are likely to make $10{,}000\times$ more trips than airlines in the future. In general, over the next few decades, the use of fully-autonomous CPS and their impact on human lives is expected to grow substantially. It will then become imperative to ensure that all safety-critical CPS are designed to be ultra-reliable. That is, if autonomous CPS are to permeate our everyday life in the not so distant future, then they need to become at least as trustworthy as airplanes.

However, with current practices, it is simply not yet possible to reach ultra-reliability in most CPS domains. A major impediment is the sheer cost of replicating reliability engineering practices from the avionics domain in other CPS domains. For example, the use of custom fault-tolerant hardware with triple-modular-redundant components and an additional set of spares is common in the avionics domain, but not affordable for the automotive industry, which runs on small cost margins. Two recent trends further impede achieving aircraft-like reliability targets: the push towards the use of faster and cheaper, but less reliable, Commodity Off-The-Shelf (COTS) hardware, such as Ethernet; and the tremendous increase in the complexity of workloads used for next-generation CPS, e.g., the use of Deep Neural Networks (DNNs) for self-driving cars. When such CPS are deployed by the millions, for catastrophic consequences to occur, it suffices if just one of them experiences a faulty execution. Therefore, new and rigorous reliability analyses are necessary. To this end, the subject of this dissertation is reliability analysis of networked control systems, which are key building blocks of modern CPS.

1.1 PROBLEM STATEMENT

A *Networked Control System (NCS)* constitutes one or more control systems wherein the control and feedback signals are exchanged among *distributed* components through a communication network. To ensure a minimum quality of control, an NCS typically requires that the underlying infrastructure provides strong temporal guarantees (also referred to as *hard real-time* guarantees).

However, implementing ultra-reliable NCS in a cost-effective manner using COTS processors and networks is far from trivial. One aspect that makes the problem particularly difficult is the effect of the harsh environments in which CPS are often deployed. Environmental disturbances cause *transient faults* (bit flips) in hardware. When an NCS is deployed over field buses such as Controller Area Network (CAN),[1] transient faults are known to cause host crashes, network retransmissions, and incorrect computations. In addition, when an NCS is deployed over point-to-point networks such as Ethernet, even *Byzantine* errors (i.e., inconsistent broadcast transmissions) are possible. Although these errors occur with extremely low probabilities, they must nonetheless be tolerated to build trustworthy CPS.

In this dissertation, we show how to analyze the reliability of NCS applications—more generally, *distributed real-time systems*—in the presence of such environmentally induced transient faults. We use the *Failures-In-Time (FIT)* metric for reporting the NCS reliability. It is an industry standard metric for measuring device reliability, and is defined as the expected number of failures in one billion operating hours of the device [214]. In particular, our objective is to:

> *Quantify the reliability of CAN- and Ethernet-based implementations of NCSs in terms of upper bounds on their FIT rates.*

1.2 ANALYSIS APPROACH

Computing the FIT rate of an NCS using empirical techniques is straightforward. For instance, the NCS implementation can be simu-

1 Field bus refers to computer network protocols that are used for real-time distributed control in industry, e.g., for connecting instruments in manufacturing plants. These are broadly specified using the IEC 61158 standard [108]. Controller Area Network (CAN) is a field bus standard that is widely used in automotive and industrial automation domains (see Section 2.1.3.1 for details).

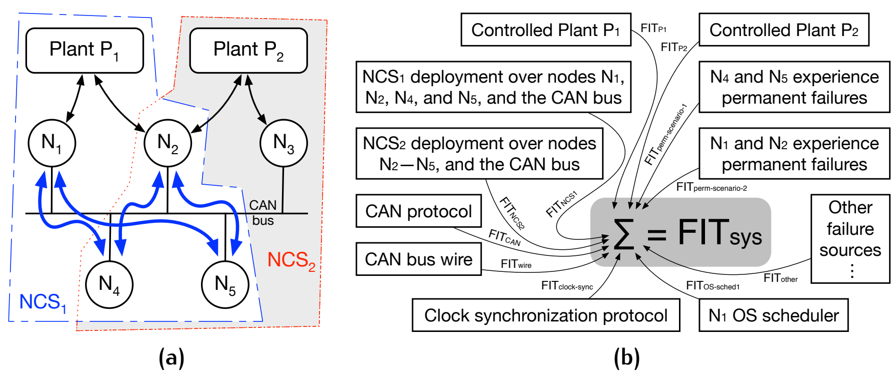


Figure 1.1: **(a)** Two NCSs are deployed in Dual Modular Redundancy (DMR) configurations over a set of nodes that communicate over a CAN bus. **(b)** Decomposing FIT analysis of the CAN-based distributed real-time system shown in (a) into FIT analysis of its sub-components using the Sum-Of-Failure-Rates (SOFR) model.

lated for a finite period of time, and the average number of failures experienced over multiple simulation trials can be used to estimate the FIT rate. However, empirical simulation-based approaches scale poorly when evaluating low-probability events, and can under-approximate the true failure rate, especially in the presence of *reliability anomalies*, when worst-case component-specific fault rates do not yield the worst-case system-wide failure rate (see Section 5.1 for a detailed explanation). On the other hand, computing such a metric analytically and in a sound manner for a complex implementation consisting of multiple critical software modules is not trivial.

In this dissertation, we address the FIT analysis problem for NCSs in a divide and conquer approach using the *Sum-Of-Failure-Rates (SOFR)* model [222]. The SOFR model is widely used in industry to compute the failure rate of a system as the sum of the failure rates of its different sub-components. It assumes that the sub-components constitute a *series failure system*. That is, the first instance of any sub-component failing, because of any failure mechanism, causes the entire system to fail. Srinivasan et al. [211], for instance, used the SOFR model to compute the failure rate of the processor as an aggregate of the failure rates of its arithmetic logic units, floating-point units, register files, branch predictor, caches, load-store queue, and reorder buffer. Similarly, in this work, we decompose the problem of computing the FIT rate of a distributed real-time system (that hosts one or more NCS applications) into multiple sub-problems. Each sub-problem corresponds to computing a safe upper bound on the FIT of one of the many software modules that are critical to the system's functional safety.

We illustrate the approach using a simple example. Consider a distributed system consisting of five nodes, N_1, N_2, N_3, N_4, and N_5, networked over a single CAN bus (see Fig. 1.1a). Two NCS applications NCS_1 and NCS_2 are deployed over these nodes to control plants

P_1 and P_2. Both applications rely on some form of Dual Modular Redundancy (DMR). In the case of NCS_1, both N_1 and N_2 sense and actuate plant P_1, and both N_4 and N_5 are used to compute the control commands; and analogously, in the case of NCS_2, both N_2 and N_3 sense and actuate plant P_2, and both N_4 and N_5 are used to compute the control commands.

The reliability of this system depends on the correct functioning of the two control plants P_1 and P_2, which in turn depends on several factors. For instance, the hardware components within the control plants must function correctly, the CAN bus wire must not fail, and nodes N_1–N_5 should not experience permanent failures beyond what the DMR configuration can tolerate (e.g., N_4 and N_5 should not both fail). In addition, the software components, including the controller replicas, the operating system service on each node, and the clock synchronization protocol (if the replicas rely on it) must function both correctly and timely despite environmentally-induced transient faults.

To account for all such failure scenarios, we decompose the system-wide FIT analysis into separate independent FIT analyses (e.g., as shown in Fig. 1.1b). Intuitively, the idea is to separately analyze orthogonal concerns like the failure rate of the CAN bus wire and the failure rate of the control plant hardware, but jointly analyze tightly coupled components like the DMR protocol that spans across multiple nodes. Although separately analyzing the failure rate of the DMR protocol instance on each node is conceptually also a sound alternative, it may yield a very pessimistic upper bound on the overall FIT. For example, applying the SOFR approach to the failure rates of the DMR protocol instances on nodes N_4 and N_5 will double the expected failure rate. Instead, jointly analyzing the DMR protocol failure rate across these two nodes (i.e., considering only those scenarios when the DMR protocol execution fails on both nodes during the same control loop iteration) actually yields a more accurate upper bound.

1.3 THESIS CONTRIBUTIONS

In the context of the SOFR model for FIT analysis, the overall contribution of this dissertation is a set of analyses to derive upper bounds on the FIT rates of actively replicated NCS subsystems. FIT analysis of the remaining critical components, like the controlled plant, the CAN bus protocol, the wire, etc., is thus orthogonal. In particular, (i) we consider simple DMR configurations as well as more sophisticated Byzantine fault tolerant configurations for active replication; (ii) we model errors affecting the active replication protocols at the granularity of message exchanges; and as mentioned before, (iii) our objective is to evaluate NCS implementations for field buses like CAN and point-to-point networks like Ethernet. Specifically, the dissertation is divided into four main research problems, which we summarize below.

1.3.1 Tolerating Byzantine Errors in CPS

Point-to-point technologies like Ethernet are fundamentally different from field buses like CAN. Lack of an atomic broadcast primitive (unlike in CAN) exposes Ethernet-based systems to the risk of environmentally-induced Byzantine errors. Byzantine fault tolerance (BFT) protocols can mitigate such errors to a large extent; but is the cost of using a BFT protocol (e.g., network bandwidth and compute capacity used) offset by the gain in reliability? Is one network topology necessarily or significantly more reliable than another? The first step towards answering such questions is understanding how BFT protocols can be implemented over COTS networks like Ethernet while satisfying hard real-time constraints, which are required by many NCS.

To this end, we first focus on the problem of designing a BFT distributed real-time system for the CPS domain, which is suitable for hosting safety-critical NCS applications (Chapter 4). Prior work on Byzantine fault tolerance in safety-critical domains relied either on custom processors or custom networks, or both; whereas BFT solutions developed for general-purpose computing systems were not designed from the perspective of hard real-time applications.

We propose a hard real-time design of a BFT protocol based on the periodic task model for Ethernet-based systems. To evaluate the proposed design, we built a prototype implementation of a BFT key-value store, called *Achal*, for coordinating distributed NCS replicas. Our results indicate that while Achal's latency is predictable and satisfies hard real-time constraints, the latencies of BFT key-value stores based on BFT-SMaRt and Cassandra (both well-known general-purpose systems) are unpredictable and frequently violate these constraints. Achal's design thus provides a basis for building BFT hard real-time applications, and a model for analytically quantifying the reliability of NCS applications that are exposed to Byzantine errors.

1.3.2 Reliability Analysis of a BFT Protocol

Classical Byzantine safety guarantees (e.g., $3f+1$ processes can tolerate up to f Byzantine faults) do not take into account non-uniform fault rates across different system components that arise due to environmental disturbances. They also abstract from the underlying network topology despite its strong influence on actual failure rates. To address this gap, we present in Chapter 5 the first quantitative reliability analysis of a hard real-time implementation of a BFT atomic broadcast protocol over Ethernet in the presence of stochastic transient faults. Most importantly, the presented analysis is free of *reliability anomalies*, which can result in non-monotonic increases in a system's overall failure rate despite local decreases in an individual component's failure rate. This is the first work to formalize and propose techniques to eliminate reliability anomalies in a hard real-time setting.

1.3.3 Reliability Analysis of an NCS Iteration

The contributions summarized in Sections 1.3.1 and 1.3.2 can together be used to implement an ultra-reliable atomic broadcast service over point-to-point networks for building trustworthy NCS. Such an analysis-driven implementation can be configured to provide comparable levels of reliability to that of conventional field buses like CAN.

However, for a full-system reliability analysis (recall the example presented in Fig. 1.1), we must also upper-bound the FIT rate of the actively replicated NCS implementation that is implemented on top of the atomic broadcast layer (either over CAN or Ethernet). In particular, despite the atomic broadcast properties of the underlying network, errors due to transient faults, such as a crash and reboot error, may keep a host unavailable for a small amount of time; and corruption errors may affect the integrity of certain messages. In an actively replicated NCS, such errors may not always affect the final actuation. Hence, especially for actively replicated NCS, a fine-grained reliability analysis is needed to more accurately capture the benefits of replication.

Thus, we present in Chapter 6 the reliability analysis of an actively replicated NCS in the presence of transient faults at the granularity of network messages. We focus on in this chapter on NCS implementations over CAN; but we also briefly discuss how the presented analysis can be modified for NCS implementations over Ethernet with a software atomic broadcast layer.

1.3.4 Reliability Analysis of Weakly-Hard Systems

The above analysis provides an implementation-specific upper bound on the failure probability of a *single* NCS iteration. A simplistic and the conventional approach to obtain metrics like FIT from such single iteration estimates is to calculate the time to first fault. However, this approach is excessively pessimistic for NCS applications, which are routinely designed to be *temporally robust*, i.e., remain functional despite a few skipped or misbehaving control loop iterations.

Instead, we propose analyses that account for the temporal robustness of NCS using multi-state models such as the widely used *weakly-hard constraints*, resulting in more accurate FIT estimates. In particular, we present in Chapter 7 three different techniques based on *probabilistic model checking*, *martingale theory*, and *sound approximation* to address the expressiveness, accuracy, and scalability requirements of larger and more complicated NCS applications. We also provide a systematic exploration and empirical evaluation of these techniques for different points in the weakly-hard constraint space.

1.4 ORGANIZATION

The remainder of this dissertation is organized as follows. We review relevant background on distributed real-time systems and reliability engineering principles in Chapter 2. We discuss the fault model and related assumptions in Chapter 3. The four main contributions summarized above are presented in detail in Chapters 4 to 7, respectively. Finally, we conclude and discuss future work in Chapter 8. Appendices A to C provide detailed proofs and implementation details.

2 BACKGROUND

In this chapter, we provide the necessary background on distributed real-time systems and reliability engineering practices.

2.1 DISTRIBUTED REAL-TIME SYSTEMS

Real-Time Systems (RTS) must guarantee response times under specified thresholds, also known as *deadlines*. RTS are integral to the CPS domain, since interaction with the physical world is often subject to strict timing constraints. For example, airbags in a passenger vehicle must be deployed within 70 ms from the time of impact [79], which requires timely execution of multiple events, including crash sensing, deciding the airbag deployment rate, and inflating the airbag [47].

In many cases, an RTS may consist of sensors, controllers, and actuators that are physically apart. The components in such a distributed system need to communicate and coordinate with each other over a shared network to ensure timely and correct responses [121]. For example, in an Advanced Cruise Control (ACC) subsystem (see Fig. 2.1 below), distributed sensors for sensing the wheel speeds and for sensing any adjacent vehicles need to periodically communicate with the central ACC unit, so that the ACC unit in turn can coordinate the actions of the braking and throttle actuators, if required.

Distributed RTS are also crucial for tolerating common cause failures due to host crashes, as opposed to standalone RTS. For example, in a RTS designed to tolerate crash faults, independent hosts can serve as a primary-backup pair, where the backup replica is redundant but, nonetheless, remains active, i.e., continues to service requests like the

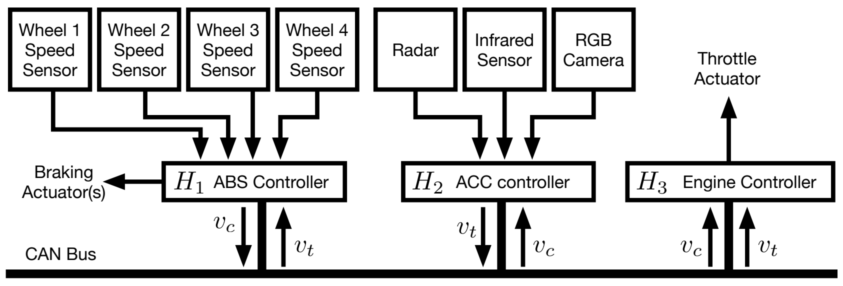


Figure 2.1: Schematic diagram of an ACC subsystem, as presented in [3]. Current and target velocities are denoted v_c and v_t, respectively. ABS denotes an Anti-lock Braking System.

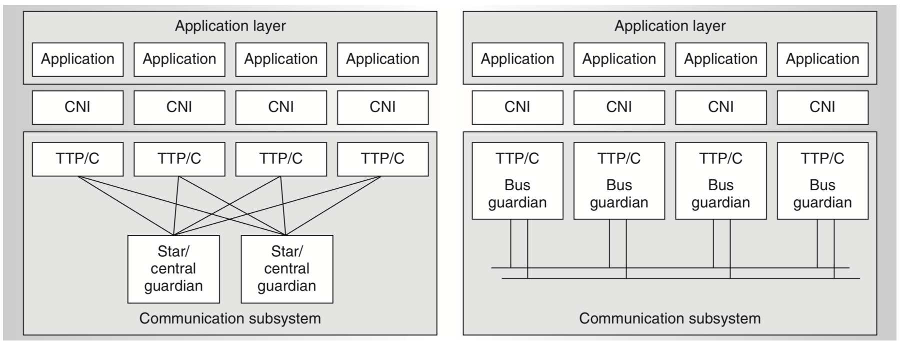


Figure 2.2: © Copyright 2002 IEEE [140]. Example of an actively replicated system with central guardians (left) and with local bus guardians (right). CNI denotes a Communication Networking Interface and TTP/C denotes the Time-Triggered Communication Protocol.

primary, in case the primary crashes [87, 195]. The two hosts, although independent, are connected in order to maintain a consistent state through regular state transfer, or else the system functionality is hindered in case of a host crash. Fig. 2.2 illustrates one such architecture.

In the following, we describe in detail the concepts from distributed systems, real-time systems, and networking that are most relevant for understanding this dissertation.

2.1.1 Distributed Systems

A central problem in a distributed system is timely and correct coordination among the distributed processes (possibly replicas) despite their "distributedness" and despite the resulting faults. To this end, two key primitives—fault-tolerant agreement and clock synchronization—are fundamental. We describe below each of these primitives in detail.

2.1.1.1 *The Agreement Problem*

In a distributed system, the distributed processes are typically designed to realize a single global objective. This often requires coordination among the distributed processes and agreement on one or more values. For example, in a spaceship, all processes controlling the spaceship engines should decide to either "proceed" or "abort" in an unanimous manner; or when funds are transferred from one bank account to another, the involved processes must consistently agree to perform the respective debit and credit. Hence, the agreement problem is a fundamental problem for distributed systems, and has been formalized in the literature in different ways. These include the *consensus* problem, the *Byzantine Generals (BG)* problem, and the *Interactive Consistency (IC)* problem.

We provide below formal definitions of these problems. Our definitions are based on the definitions by Coulouris et al. [50]. Suppose there are N_p processes. The distributed processes are denoted p_1, p_2, and so on. Further, each process is classified as either faulty or correct, based on whether it executes erroneously or not. The objective is therefore to ensure that correct processes achieve agreement despite a subset of processes being faulty.

DEFINITION 2.1. *The consensus problem.* Each process p_i proposes a single value v_i. The processes communicate with each other, after which each process p_i decides the value of a local decision variable d_i. The following conditions must hold to solve the consensus problem:

- *Termination*: Each process sets its decision variable eventually.
- *Agreement*: If processes p_i and p_k are correct and have decided d_i and d_k (respectively), then $d_i = d_k$.
- *Integrity*: If all correct processes proposed the same value v, and if process p_i is correct and has decided d_i, then $d_i = v$.

DEFINITION 2.2. *The BG problem.* Only the leader process, say p_0, proposes a value, say v. The processes communicate with each other, after which each process p_i decides the value of a local decision variable d_i. The following conditions must hold to solve the BG problem:

- *Termination*: Each process sets its decision variable eventually.
- *Agreement*: If processes p_i and p_k are correct and have decided d_i and d_k (respectively), then $d_i = d_k$.
- *Integrity*: If the leader process p_0 is correct, and if any process p_i is correct and has decided d_i, then $d_i = v$.

DEFINITION 2.3. *The IC problem.* Each process p_i proposes a single value v_i. The processes communicate with each other, after which each process decides the value of a local decision vector D_i of length N_p. The following conditions must hold to solve the IC problem:

- *Termination*: Each process sets its decision vector eventually.
- *Agreement*: If processes p_i and p_k are correct and have decided D_i and D_k (respectively), then $D_i = D_k$.
- *Integrity*: If process p_i is correct, and if any process p_k is also correct and has decided D_k, then $D_k[i] = v_i$.[1]

These three problems are equivalent in the sense that it is possible to derive a solution for one of the problems using a solution for one of the other problems (see [50] for details). However, finding a solution to any of these problems that works despite faults is challenging. We describe a range of faults that are likely to occur in a distributed

1 $D_k[i]$ denotes the i^{th} element of vector D_k.

PROPERTY	CHARACTERISTIC	FAVORABLE?
Processors	Asynchronous	No
	Synchronous	Yes
Communication	Asynchronous	No
	Synchronous	Yes
Message Order	Asynchronous	No
	Synchronous	Yes
Transmission Mechanism	Point-to-point	No
	Broadcast	Yes
Receive/Send	Separate	No
	Atomic	Yes

Table 2.1: System characteristics affecting the agreement problem based on prior work by Dolev et al. [63]. The last column specifies whether it is favorable to solve the agreement problem for the given system characteristic.

real-time CPS in Chapter 3. We also discuss classic solutions for these problems, and particularly for the IC problem that is the focus of this dissertation, in Chapter 4.

Next, we give an overview of certain system characteristics that determine whether it is easy or difficult to solve the agreement problem, and in the end, describe the characteristics that we assume in the rest of this dissertation for our analyses. Our characterization is based on a prior work by Dolev et al. [63]. See Table 2.1 for a quick summary.

In the case of processors and communication, asynchronous (or synchronous) behavior is characterized by unbounded (or bounded) delays between consecutive processor steps and between consecutive message delivery events, respectively.

In contrast, in the case of message order, asynchronous and synchronous behaviors imply whether the messages can be delivered out of order or if in-order delivery is guaranteed, respectively. In particular, a synchronous message order implies that if process p_i sends a message m_1 to process p_j at time t_1, and if process p_k also sends a message m_2 to process p_j at time t_2, such that $t_2 > t_1$, then p_j receives m_1 before m_2. Here, p_i, p_j, and p_k are not necessarily distinct, and time-stamps t_1 and t_2 refer to the wall-clock time, which is external to the system. In general, any kind of asynchronous behavior makes it difficult to solve the agreement problem.

The last two properties in Table 2.1 correspond to the atomicity of send and receive operations. With a point-to-point transmission mechanism, a processor can send a message to at most one processor atomically, whereas a broadcast mechanism enables a processor to send messages to all processors in a single atomic step. Similarly, a processor can receive and send messages either as part of the same atomic step, or separately. Like asynchronous behaviors, lack of atomicity also hinders solving the agreement problem.

In this dissertation, we analyze distributed real-time systems with synchronous processors and communication. Processors are synchronized using a clock synchronization protocol (which is discussed in Section 2.1.1.2), and communication is synchronized using time-sensitive networking standards (which are discussed in Section 2.1.3). We also assume that send and receive are separate operations. Finally, we consider both point-to-point and broadcast-based systems, and provide separate analyses for each (see Parts ii and iii of the dissertation, respectively). For point-to-point systems, we analyse Pease et al.'s solution [173] for the IC problem (reviewed in Section 4.2.1); and for broadcast-based systems (where it is easier to solve the agreement problem), we analyze a simple active replication protocol.

2.1.1.2 *Clock Synchronization*

As discussed above, asynchronous processors make it difficult to solve the agreement problem. In fact, in a fully asynchronous system, because it is impossible to distinguish between a faulty processor and a slow processor, it is generally impossible to reach consensus, as shown by Fischer et al. [74].[2] Thus, distributed systems are often designed to behave synchronously. However, this is quite challenging because computer clocks, even if initialized to the same value, tend to diverge over time. The oscillators underlying the crystal clocks are subject to physical variations; and as a result, their frequencies differ. When these differences accumulate over many oscillations, the differences between the clock values can be significant.

The solution is to synchronize each clock in a distributed system regularly. If the objective is to synchronize each clock with an external authoritative source of time, such as the Coordinated Universal Time (UTC), *external synchronization* is needed. Suppose that C_i and $C_i(t)$ denote the ith clock and its readings at absolute time t, respectively; S and $S(t)$ denote the UTC time source and its readings at absolute time t, respectively; and $D > 0$ denotes the synchronization bound.

DEFINITION 2.4. External synchronization requires that $|S(t) - C_i(t)| < D$ for each clock C_i and at all times t.

If the objective is simply to mutually synchronize the different components of a distributed system, *internal synchronization* suffices.

DEFINITION 2.5. Internal synchronization requires that $|C_i(t) - C_k(t)| < D$ for each pair of clocks C_i and C_k, and at all times t.

In general, even if any one clock in a distributed system is externally synchronized with a bound of D, internal synchronization guarantees

2 This result is also called the FLP impossibility proof after its authors Michael J. Fischer, Nancy Lynch, and Mike Paterson.

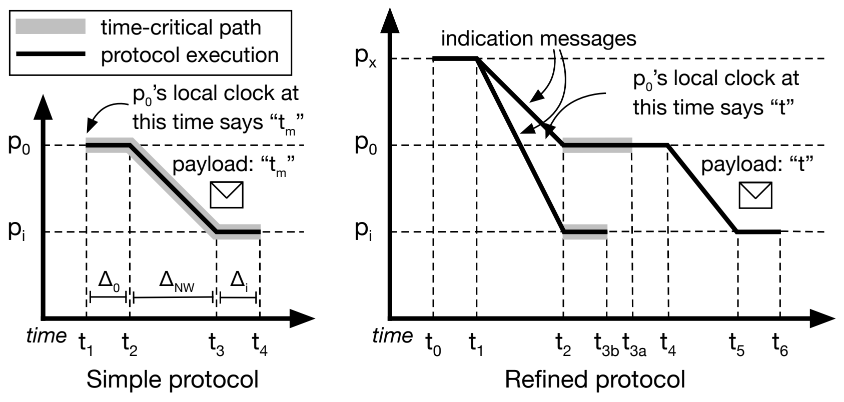


Figure 2.3: A simple and a refined clock synchronization protocol for CAN-like networks. Process p_0 is the leader process, whereas processes p_i and p_x are follower processes.

that all other clocks in the distributed system are externally synchronized as well, although with a bound of $2 \times D$. Coulouris et al. [50] explains these synchronization modes in detail.

In the following, we explain the intuition behind two types of internal clock synchronization algorithms on which our evaluation workloads in Chapters 5 and 6 are based. We start by explaining the protocols by Gergeleit and Streich [84], each of which is designed in the form of a leader/follower algorithm, and specified for broadcast networks (such as CAN) that satisfy the following three conditions:

1. Network messages are delivered to all nodes with a fixed and known delay. The delay can be a function of the receiving node.

2. The delay from the transmission (and similarly, from the reception) of a network message to an interrupt service routine that timestamps this transmission (reception, respectively) is known and has a very small variance. This delay can also be a function of the receiving node.

3. An upper bound on the time between two valid clock synchronization messages can be guaranteed in advance.

The protocols are illustrated in Fig. 2.3 and explained below.

In the simple protocol, the leader process, say p_0, first reads the value of its local clock at absolute time t_1 (let this clock value be denoted t_m) and broadcasts it at absolute time t_2. All follower processes then receive the broadcast message at absolute time t_3 and adjust their local clocks accordingly at absolute time t_4. The adjustment done by each follower process p_i ($i \neq 0$) depends on the latency of the complete path, i.e., each p_i sets its local clock to:

$$t_{i,new} = t_m + t_4 - t_1 = t_m + \Delta_0 + \Delta_{NW} + \Delta_i. \tag{2.1}$$

Δ_0, Δ_{NW}, and Δ_i denote upper bounds on $t_2 - t_1$, $t_3 - t_2$, and $t_4 - t_3$, and can be computed in advance. The adjustment results in the synchronization of p_0 and p_i's clocks since $t_{i,new}$ also denotes leader process p_0's clock value at absolute time t_4. The synchronization bound D depends on the accuracy of the upper bounds Δ_0, Δ_{NW}, and Δ_i.

There are two main drawbacks of this simple protocol. The entire path from t_1 to t_4 (as highlighted in Fig. 2.3) is time-critical and must be deterministic; whereas in practice, the path length is affected by runtime characteristics, like the payload-based bit stuffing introduced by the CAN protocol (Section 2.1.3.1 describes the CAN protocol in detail). As a result, the clock synchronization accuracy is significantly affected. In addition, it is expected that the time between t_1 and t_4 is at least an order of magnitude smaller than the desired granularity. However, for CAN-like broadcast networks, this time duration can be very high. For example, on a CAN bus with a bit transmission rate of 125 kbit/s, the transmission time of an 8 byte time-stamp is about 1 ms, which implies that even a millisecond-granularity clock synchronization is not possible using this protocol.

To overcome these limitations, Gergeleit and Streich [84] defined another protocol that does not rely on such a long time-critical path (see the refined protocol in Fig. 2.3). The key idea is to exploit the synchrony of the broadcast network. First, an arbitrary process, say p_x ($x \notin \{0, i\}$), decides to broadcast an empty *indication* message at absolute time t_0. The indication message is broadcast at absolute time t_1, and received by all processes synchronously at absolute time t_2. Let us focus only on the leader process p_0 and a follower process p_i. Upon receiving the indication message, processes p_0 and p_i take local time-stamps. Suppose that these local time-stamps are denoted t and t', respectively, and that their time-stamping procedure completes at absolute times t_{3a} and t_{3b}, respectively. The leader process p_0 then broadcasts its time-stamp t at absolute time t_4. Upon receiving this message, process p_i simply needs to compare its own time-stamp for the indication message (i.e., t') with the leader process p_0's time-stamp for the indication message (i.e., t). In other words, $t' - t$ denotes the difference between p_0 and p_k's local clocks. Therefore, p_k can synchronize its clock with p_0's clock by making the corresponding adjustment. Notice that both processes p_0 and p_i, when determining t and t' (respectively), need to account for their processor-local execution delays, i.e., for time durations $t_{3a} - t_2$ and $t_{3b} - t_2$, respectively.

In summary, the refined protocol by Gergeleit and Streich [84] depends only on short time-critical paths, i.e., process p_0's execution from t_2 to t_{3a}, and process p_i's execution from t_2 to t_{3b}, which are also independent of the network. Hence, this protocol allows processes to synchronize clocks at a higher granularity and with better accuracy. In fact, Gergeleit and Streich further optimized this protocol by treating the second message by the leader process p_0

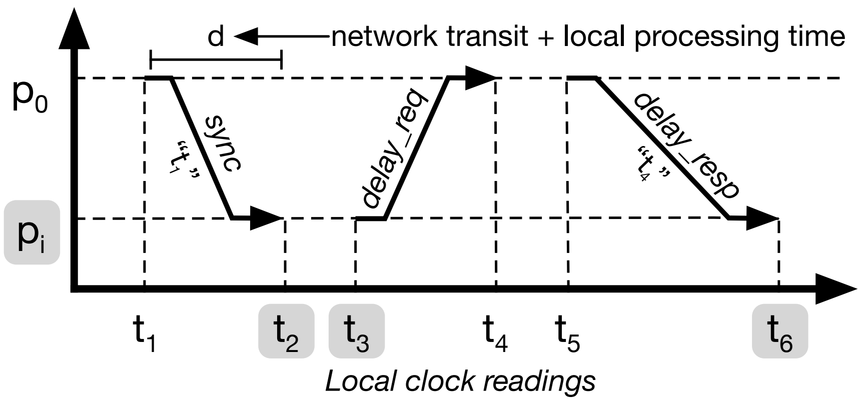


Figure 2.4: The key mechanism underlying PTP [106].

as an indication message for the next round, thereby reducing the protocol bandwidth by a factor of two. They also explain how system-dependent inaccuracies (due to the interrupt routine software and the hardware) in determining the message reception or transmission times can be accounted for by the protocols. Unfortunately, none of the protocols proposed by Gergeleit and Streich [84] apply to Ethernet (or point-to-point networks, in general). Thus, we give an overview of another internal synchronization protocol in the following.

We explain the key steps involved in the Precision Time Protocol (PTP) [106], which is widely used by Ethernet-based distributed real-time systems in the CPS domain for achieving clock synchronization in the sub-microsecond range. The objective is for any process p_i $(i \neq 0)$ to find the offset $O(t) = C_i(t) - C_0(t)$ between the time measured by its local clock C_i and the time measure by the leader process p_0's local clock C_0 at absolute time t. If $O(t)$ is accurately known, p_i can correct its clock so that it agrees with the leader process p_0's clock. However, in an Ethernet-like network (and unlike in CAN-like networks), where networking delays cannot be determined in advance, computing $O(t)$ requires two steps (which are illustrated in Fig. 2.4). In the first step, the leader process p_0 broadcasts a *sync* message at its local time t_1, which contains the clock value t_1 as its payload and which is received by p_i at its local time t_2. In the second step, process p_i tries to determine the network transit time d between itself and the leader process p_0 (d also includes any local processing delays incurred on the respective processors). It sends a *delay_req* message at its local time t_3 to the leader process p_0, which time-stamps the receipt of this message at its local time t_4 and further responds with a *delay_resp* message containing the time-stamp t_4. By the end of this exchange, process p_i learns about the leader process p_0's time-stamps t_1 and t_4, and it also keeps a record of its own time-stamps t_2 and t_3. Hence, assuming that the offset between processes p_0 and p_i's clocks is constant over the period during which this message exchange happens, process p_i can determine this offset, denoted o, as follows:

$$t_2 - t_1 = o + d \text{ and } t_4 - t_3 = -o + d \tag{2.2}$$

$$\implies o = \frac{1}{2}(t_2 - t_1 - t_4 + t_3). \tag{2.3}$$

Once o is known, p_i can synchronize it's clock with leader process p_0's clock. The synchronization accuracy depends on the accuracy with which the processes can measure the time at which they send or receive messages, and also on whether the transit times from p_0 to p_i and from p_i to p_0 are identical.

Throughout this dissertation, we trust clock synchronization algorithms to synchronize the distributed clocks in a trustworthy manner. We then analyze the reliability of a synchronous Byzantine Fault Tolerance (BFT) protocol and an actively replicated Networked Control System (NCS), both of which rely on the clock synchronization primitive. However, a similar reliability analysis of such clock synchronization algorithms (or their fault-tolerant versions) is also needed. We plan to investigate this problem in future work (see Chapter 8).

2.1.2 Real-Time Systems

We describe Real-Time Systems (RTS) fundamentals including the commonly used task models, different Quality of Service (QoS) guarantees, and different scheduling policies [31, 53, 201].

2.1.2.1 *Task Models*

Real-time applications typically consist of a set of recurring *tasks* that are designed to execute their function either periodically based on timer interrupts or sporadically based on external inputs. To formalize these two execution types, Liu and Layland [137] proposed the *periodic task model* and Mok [153] later proposed the *sporadic task model*, respectively. We describe these models in the following.

Suppose that the real-time application consists of a set of tasks $T = \{T_1, T_2, \ldots, T_n\}$. Each task executes a sequential piece of code; in other words, intra-task parallelism is not permitted. The Worst-Case Execution Time (WCET) of each task T_i is assumed to be known in advance and denoted C_i. As mentioned above, the tasks are invoked based on timer interrupts or external events, possibly for an infinite number of times. Hence, each task T_i is modeled as a set of infinitely many *jobs* (or invocations), and the k^{th} job is denoted $J_{i,k}$.

The *arrival time* of each job $J_{i,k}$, i.e., the time at which job $J_{i,k}$ is ready to be scheduled, is denoted $a_{i,k}$. Under the periodic task model, if P_i denotes the *period* or the *time period* of task T_i, then its jobs' arrival times are related as $a_{i,k+1} = a_{i,k} + P_i$ $(k > 0)$. If all periodic tasks in T arrive simultaneously in the beginning, also referred to as a *synchronous* arrival, then for each pair of tasks T_i and T_k in T, $a_{i,1} = a_{k,1}$. Alternatively, the task arrival times may be separated by fixed *offsets* (which is referred to as an *asynchronous* arrival). In this case, each task T_i first arrives at time $a_{i,1} = \phi_i$, where ϕ_i denotes an offset from a synchronous time point.

Unlike in the periodic task model, P_i under the sporadic task model denotes the *minimum inter-arrival time* between any two jobs of task T_i; hence, $a_{i,k+1} \geqslant a_{i,k} + P_i$ $(k > 0)$. Also, for analysis purposes, it is assumed that the dependence, if any, between the arrival times of sporadic jobs of different tasks is not known in advance.

Each task T_i is also characterized by a *relative deadline* parameter D_i, which determines the range of its acceptable response times. In particular, each job $J_{i,k}$ must finish its execution no later than time $a_{i,k} + D_i$. In many cases, deadlines are *implicit*, i.e., $D_i = P_i$, to ensure that when a new job arrives, the old job has finished. However, depending on the application requirements, the deadlines may also be more *constrained*, i.e., $D_i \leqslant P_i$, or relaxed, i.e., $D_i > P_i$. The term *arbitrary* deadlines is typically used to denote the union of these cases.

In summary, the periodic and sporadic task models characterize each task T_i using the tuple (C_i, P_i, D_i), where $C_i \leqslant P_i$ and $C_i \leqslant D_i$. The *task utilization* U_i of each task T_i determines its processor demand and is defined as C_i/P_i. Similarly, the *task set utilization* U determines the cumulative processor demand of all tasks in T, and is defined as $U = \sum_{T_i \in T} U_i$. In this dissertation, we assume the periodic task model with implicit deadlines when modeling real-time applications. Our contributions, though, are not limited by this assumption; alternative task models with corresponding timing analyses could be used as well.

2.1.2.2 *Different QoS Guarantees*

Given the periodic and sporadic task models for real-time applications, we next define different Quality of Service (QoS) guarantees that an RTS may expose to the application. We also discuss common real-time scheduling policies that can be used to enforce these guarantees.

In a real-time system, QoS guarantees refer to guarantees on the temporal correctness of task executions, and more precisely, on the response times of task invocations. Let $R_{i,k}$ denote the Worst-Case Response Time (WCRT) of job $J_{i,k}$ in the presence of delays due to the task scheduling policy and the execution of other tasks on the system. We define below three types of QoS guarantees that have been commonly used in the literature.

In the following definitions, $\mathbb{N} = \{1, 2, 3, \ldots\}$ and $\mathbb{W} = \mathbb{N} \cup \{0\}$ denote the set of *natural* numbers and *whole* numbers, respectively.

DEFINITION 2.6. A *Hard Real-Time (HRT)* guarantee implies that task deadlines are never violated, i.e.,

$$\forall T_i \in T, k \in \mathbb{N} : R_{i,k} \leqslant D_i. \tag{2.4}$$

DEFINITION 2.7. A *Soft Real-Time (SRT)* guarantee implies that task deadlines may be violated, but the violations are bounded, i.e.,

$$\exists B \in \mathbb{W} \text{ s.t. } \forall T_i \in T, k \in \mathbb{N} : R_{i,k} \leqslant D_i + B. \tag{2.5}$$

Like the SRT guarantee, a *Firm Real-Time (FRT)* guarantee also allows task deadlines to be violated. However, unlike the SRT guarantee, a FRT guarantee allows only a bounded number of violations in every finite execution history of the task. For example, we define below an (m, k) guarantee, which is a widely used FRT guarantee.

DEFINITION 2.8. An (m, k) FRT guarantee $(1 \leqslant m \leqslant k)$ implies that the number of violations among every k consecutive jobs of a task is bounded by $k - m$, i.e.,

$$\forall T_i \in T, j \in \mathbb{N} : \sum_{x=0}^{x=k-1} V_{i,j+x} \leqslant k - m, \tag{2.6}$$

where $V_{i,j} = 0$ if $R_{i,j} \leqslant D_i$ and $V_{i,j} = 1$ otherwise.

HRT guarantees are needed for applications if even a single deadline violation can cause a total system failure, which in turn can result in catastrophic consequences. For example, consider the airbag deployment process in a passenger vehicle, which has an end-to-end deadline of less than $70\,ms$ [79]. Any task constituting this end-to-end process cannot miss its deadline. On the other hand, FRT guarantees [210] are sufficient if infrequent deadline misses are tolerable by the application and if the usefulness of a deadline is zero after its deadline. Control systems work well with FRT guarantees, since occasional deadline misses may only slightly degrade the quality of control. The (m, k) guarantee [95, 183] defined in Definition 2.8 is just one way to define FRT guarantees. Bernat et al. [21] provide other variants of the (m, k) guarantee. Finally, SRT guarantees [206] are useful for applications that benefit from a task's execution even if it was delayed beyond its deadline, but at the cost of some loss in the application's service, e.g., audio-video systems are often soft real-time.

Enforcing any of the aforementioned QoS guarantees requires a combination of a runtime scheduling algorithm and an offline *schedulability analysis*. The latter is useful for validating whether a specified workload when scheduled using a specified scheduling algorithm experiences any QoS violations. In general, a scheduling algorithm designed to schedule periodic and sporadic tasks with HRT guarantees can be used in the case of SRT and FRT guarantees as well. However, the schedulability analyses may vary in each case.

The most simple scheduling algorithm is the Fixed Priority (FP) scheduling algorithm. As per this algorithm, each task T_i is assigned a unique fixed priority. At runtime, among all the jobs that are ready to execute, the job belonging to the highest priority task is scheduled first. The job is scheduled either *preemptively* or *non-preemptively* [83]. Under preemptive scheduling, if a higher-priority job arrives, the scheduler preempts the currently running job and schedules the highest priority job. In contrast, under non-preemptive scheduling, the currently running job executes to its completion. In addition, on a multiprocessor system, FP scheduling can either be implemented *globally* [24], i.e., jobs are dispatched to each core from one global priority-ordered queue, or in a *partitioned* manner [32], i.e., each task is assigned to an individual core in advance and jobs are then dispatched to each core from the respective core-local priority-ordered queue. Task priorities can be assigned using different heuristics such as *rate monotonic* [138, 200] or *deadline monotonic* [14] (where the task with the shortest period or the shortest relative deadline gets the highest priority, respectively).

In this work, we assume partitioned FP scheduling with preemption and rate monotonic priorities, unless specified otherwise.

2.1.3 Time-Sensitive Networks

Next, we provide a background on the Controller Area Network (CAN) field bus [58] and provisions in Ethernet for time-sensitive networking [161]. CAN has been widely used for CPS (especially in the automotive domain) in the past three decades. On the other hand, time-sensitive variants of Ethernet, such as many automotive Ethernet standards [148], are likely to find widespread use in future distributed real-time systems due to their high speed and bandwidth.

2.1.3.1 *Controller Area Network*

Traditional point-to-point networking solutions became increasingly expensive and cumbersome as the number of Electronic Control Units (ECUs) in an automobile grew beyond 40s. Hence, Robert Bosch GmbH in the late eighties developed CAN—an inexpensive message-based protocol that is both robust and predictable—for networking ECUs inside an automobile [40]. Over time, CAN became the de facto standard field bus for the automotive industry, and was also widely used in other CPS domains [131]. We discuss below its main properties that make it useful for distributed real-time systems in general. For a comprehensive overview, see the book by Di Natale et al. [58].

The data frame format of a CAN message frame is illustrated in Fig. 2.5 for reference. The message transmission protocol relies on a bit-level synchronization protocol so that every host agrees on the value of the currently transmitted bit [77]. This enables CAN to use a bit-wise arbitration method for contention resolution. That is, during

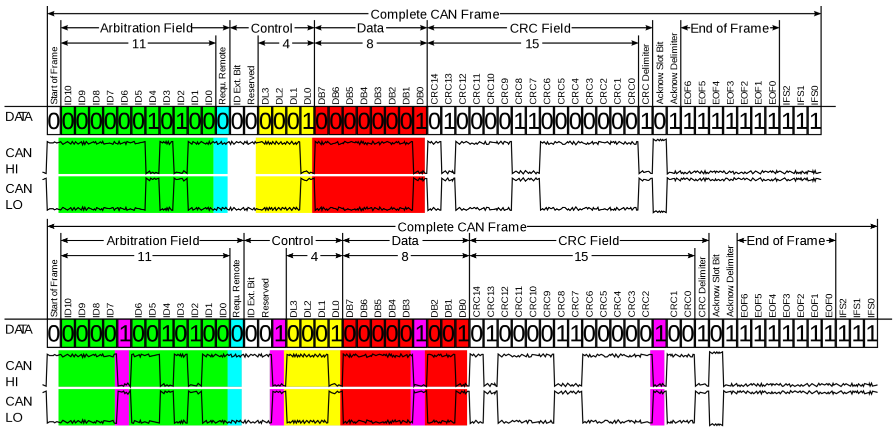


Figure 2.5: The data frame format of a CAN message, with (bottom) and without (top) bit-stuffing. The payload (data field) can range from 0 to 8 bytes. Image source: Wikipedia Commons [230].

the arbitration phase, multiple message *identifiers* may be simultaneously broadcast, and the message with the lowest identifier wins the arbitration. Message identifiers must be unique. Otherwise, two hosts may continue transmission beyond the end of the arbitration phase, causing an error. Unique identifiers are guaranteed by partitioning the identifier space across all messages during design time [52].

To ensure message integrity, i.e., that messages are not corrupted due to bit-flips on the bus, the CAN protocol incorporates message checksum and acknowledgement slots in the data frame. If any host detects an error, it transmits an error frame to signal the sender. In addition, since bit patterns '000000'and '111111' are used to signal errors, the CAN protocol also incorporates a *bit-stuffing* mechanism to avoid the use of these bit patterns during an error-free transmission of the data frame [159]. That is, whenever five bits of the same polarity are transmitted, a bit of the opposite polarity is immediately inserted by the transmitter. Upon error detection, the sender host (in particular, its CAN controller) then schedules that message for retransmission.

While the arbitration and error detection mechanisms in CAN make it both predictable and robust, it was the response-time analysis of CAN messages that enhanced its credentials as a real-time network. Tindell et al.'s seminal work [220, 221] on mapping the problem of upper-bounding CAN message response times to the problem of upper-bounding task response times in a uniprocessor FP setting played a significant role in this regard. More recently, Davis et al. [54] proposed a revised and corrected version of this analysis.

In summary, from the point of view of building reliable Networked Control Systems (NCSs), which is the focus of this dissertation, CAN's atomic broadcast property (which is a consequence of its bit synchronization and error detection mechanisms) [175] is most important. In particular, our reliability analysis of actively replicated NCS applications (in Chapter 6) assumes that the underlying networking layer guarantees atomic broadcast of messages. Therefore, our claim is that the reliability analysis applies to all CAN-based NCSs as well.

Although it has been shown that CAN's atomic broadcast properties can be violated under rare circumstances [111, 188], i.e., faults in the last two bits of the End of Frame delimiter may lead to inconsistent message delivery or message duplicates, we consider the problem of analyzing these rare events as orthogonal to the larger problem that constitutes the reliability analysis of the active replication protocol. That is, we treat the probability of extremely rare events like these (which is already evaluated by Rufino et al. [188]) as a separate, additive failure source in the system-wide SOFR analysis.

Our analyses also assume that PEs connected to the CAN bus are synchronized (notice that CAN's bit-level synchronization does not imply clock synchronization). This can be efficiently achieved using CAN-specific clock synchronization protocols, such as the protocol by Gergeleit and Streich [84], as discussed in Section 2.1.1.2.

Finally, the CAN standard is a representative of similar other field buses that are also designed for predictability and robustness, like the Vehicle Area Network (VAN) [107].

2.1.3.2 *Ethernet TSN*

We focus on Ethernet TSN (Time-Sensitive Networking), which refers to a set of standards defined for timely and robust transmission of data over Ethernet with the objective of supporting real-time control systems and automation applications. The Ethernet TSN task group is a continuation of its earlier Audio Video Bridging (AVB) task group [80, 215], which was also constituted for designing low-latency and reliable solutions for switched Ethernet networks.

The TSN standards include a range of mechanisms for improving the management, control, integrity, and synchronization of TSN flows (i.e., end-to-end unicast or multicast connections through a TSN-capable network). However, the core idea behind achieving timely and predictable response times for traffic flows is the use of prioritized traffic classes [161]. In particular, up to eight traffic classes, each with a dedicated FIFO queue, are allowed, as summarized in Table 2.2. In addition, under the Stream Reservation Protocol (SRP) [104], high priority traffic can be throttled so that lower-priority traffic does not starve. This is achieved using a Credit-Based Shaper (CBS) [105], as per which, the throttled queue is eligible for transmission only if it has non-negative credits. The credits increase at a rate of *idleSlope* when there is at least one waiting frame in the queue, and decrease at a rate of *sendSlope* when a frame is transmitted.

Unlike CAN, which consists of just one networking element (i.e., the bus), an Ethernet network consists of multiple links and switches. Hence, upper-bounding the response time of TSN flows requires schedulability analysis of each arbitration point in the flow, as well as

PRIORITY	GUIDELINES FOR PRIORITY ASSIGNMENT
0	Background
1	Best effort
2	Excellent effort
3	Critical application
4	“Video”, less than 100 ms latency and jitter
5	“Voice”, less than 10 ms latency and jitter
6	Inter-network control, e.g., IP routing protocols
7	Control Data Traffic (CDT) from real-time applications

Table 2.2: Description of Ethernet traffic classes based on [161]. 0 denotes the lowest priority level, and 7 denotes the highest priority level.

an end-to-end analysis that takes into account all path dependencies. Diemer et al. [59, 60] have shown that Compositional Performance Analysis (CPA) [98] can be used in this regard. In a nutshell, each output port in an Ethernet switch is modeled as a processing resource with an associated arbitration policy, which accounts for the FIFO priority classes and the CBS traffic shaping policy; TSN frame processing on each output port is modeled as a sporadic task, which is activated either due to timer events, external inputs, or based on inter-task dependencies; and CPA then computes an upper bound on the end-to-end latency of all task chains, which also implies an upper bound on the response time of the TSN flows. Chapter 5 on the reliability analysis of an Ethernet-based Interactive Consistency (IC) protocol employs this scheduling model.

2.1.4 Realization on COTS Platforms

Recall from Chapter 1 that our goal is to propose reliability analyses for COTS distributed real-time systems. The phrase “COTS distributed real-time systems” may seem self-contradictory, since COTS software and hardware systems are not designed in the first place to satisfy any hard-real time assumptions. However, despite this, a plethora of companies today sell autonomous COTS-based CPS, including rovers, drones, and robots, for commercial purposes. In general, even though most COTS platforms are not designed for timeliness, they can be enhanced to behave in a real-time friendly manner by using an RTOS. For example, Linux-based platforms can be enhanced with the PREEMPT_RT patch [150] in order to provision real-time workloads (see [34] for a tutorial). Based on this assumption, we rely on hard real-time schedulability analyses as the basis for our reliability analyses. In other words, the proposed reliability analyses themselves do not introduce any uncertainty in the timing. Therefore, if they are used in the context of COTS-based distributed real-time systems, such as systems based on Linux, the overall result is no more or less "hard" than the real-time workloads realized on these COTS platforms.

2.2 RELIABILITY ENGINEERING

Reliability engineering of safety-critical systems deals with identifying the causes of failures through systematic testing, validation, and verification procedures, preventing or reducing the likelihood of failures through fault-tolerance mechanisms (e.g., by redundancy), and then analyzing the expected reliability of new designs for certification purposes [26, 152]. Naturally, reliability engineering techniques cut across a number of different disciplines (computer science, statistics, engineering, etc.). In this section, we discuss fault-tolerance techniques commonly used for safety-critical distributed real-time systems, standard reliability metrics such as MTTF and FIT, and reliability analysis approaches commonly used in the industry to estimate such metrics.

2.2.1 Fault Tolerance

We first discuss three computer architecture designs (lockstep execution, ECC memory, and watchdog timers) for detecting transient faults and taking corrective actions upon detection.

Lockstep execution refers to perfectly synchronous execution of, typically, dual- or triple-modular redundant systems [176]. Synchronization among processors executing in lockstep happens at the hardware level and at instruction granularity, driven by a common clock source. In an error-free scenario, the processors receive identical inputs, execute identical operations, and output identical values. Hence, any discrepancy between the outputs of the redundant processors helps detect an erroneous execution. In case of Triple Modular Redundancy (TMR), the erroneous execution by a faulty processor can also be automatically corrected through majority voting (assuming that other processors were not affected by faults). Some early examples of lockstep processors include Stratus [226], Sequoia [23], and the V60 microprocessor [158]. More recently, Infineon's AURIX family of micro-controllers [1], which are targeted at the automotive industry, have been designed for lockstep execution.

Lockstep processors are highly reliable. However, the lockstep execution approach is considered a centralized approach rather than a distributed approach to fault tolerance [119, 176]. This is because faults can affect the synchronized processors in a correlated fashion, and they can result in a common-cause failure. The high cost of present-day lockstep processors (such as Infineon's AURIX family of processors) is also a concern.

In comparison to lockstep execution, use of *Error-Correcting Code memory* (ECC memory) is more common when it comes to detecting and correcting faults. An ECC memory stores a k-bit word as an n-bit code ($n > k$), where the extra $n - k$ bits are for checking parity. Using these parity bits, the original message can be extracted as long as up to t bits in the code are corrupted (t varies with the type of coding mechanism used); if $t + 1$ bits are corrupted, corruption is detected but the errors cannot be corrected; and if more than $t + 1$ bits are corrupted, some errors may neither be detected nor corrected [135].

Most ECC memories use the Single-Error-Correcting Double-Error-Detecting (SECDED) codes [103], for which $t = 1$. More resilient codes such as Double-Error-Correcting Triple-Error-Detecting (DECTED) and ChipKill [30, 56] incur much higher storage and performance overheads than SECDED codes. Hence, most ECC memory can only detect a double bit-flip, but cannot correct it. In this case, the CPU raises a machine check exception to the OS whenever a double bit-flip is detected, resulting in an application crash. In addition, there is also a residual possibility of silent data corruption when more than two bit-flips affect ECC memory. Alternatively, in the absence of ECC memory or hardware support for ECC, or if the likelihood of double and triple bit-flips are high, software-defined error detection and correction mechanisms could also be used [86, 203].

Finally, all safety-critical systems, and most processors these days, are also equipped with a *watchdog timer* [155, 156, 194]. The watchdog timer is a piece of hardware whose output is directly connected to the processor's reset signal. The counter in the watchdog timer is initialized to a positive value, and simply counts down to zero. The software is expected to restart the counter before it reaches zero. Otherwise, the system is assumed to be either hung or functioning incorrectly, and the processor is restarted. The watchdog timer thus helps tolerate hangs due to transient faults or software anomalies, especially in safety-critical systems that are not accessible to human operators and that must be reset in a timely manner.

The aforementioned mechanisms are not foolproof. They mitigate the effects of transient faults, but cannot completely prevent fault-induced failures. Hence, designers often also introduce redundancy at the highest level, in the form of software fault-tolerance techniques like active replication and passive replication (using hot and cold standbys). We describe these techniques in the following.

Active replication is similar to lockstep execution in the sense that all replicas (redundant processors) execute the same set of procedures in parallel, and in reaction to the same set of inputs [96, 176, 231]. However, unlike lockstep execution, the active replicas are synchronized loosely at periodic time points using message-based information exchange protocols; hence, there is no Single Point of Failure (SPoF).

Active replication also requires some sort of redundancy suppression mechanism before the replica outputs are forwarded to an actuator.

In contrast, under *passive* replication, only one replica (the *primary*) generates outputs, which removes the needed for redundancy suppression; the other replicas (the *secondaries* or *backups*) remain in a standby mode [112, 195]. Passive replication can be implemented with either hot or cold standbys. A *hot standby* remains active but does not produce outputs until the primary fails and it is promoted to become a primary; whereas a *cold standby* simply tries to remain consistent with the primary's state. In fact, the cold standby can remain completely inactive: the primary's state can be periodically logged into a shared storage device; when the primary fails, the cold standby first reads these logs and updates its state, and only then begins executing requests and producing outputs.

While passive replication is resource efficient, the recovery time from a single crash is non-zero, and in the case of a cold standby, also quite significant. In contrast, active replication works seamlessly despite a crash. It can also tolerate corruption errors that passive replication cannot. Since we target NCS applications that might operate at high frequencies, we consider only active replication in this dissertation.

2.2.2 Reliability Metrics

Reliability is defined as the probability that a system will perform its intended functions for a specified period of time under specified operating conditions [124]. It may be defined for a single component or for a system consisting of many components, e.g., the latter in case of distributed real-time systems. To compute reliability, the lifetime of the system is treated as a random variable. Further, the operating conditions under which the system is expected to operate must be specified. For example, reliability analysis of distributed real-time systems requires that the application workload and peak transient fault rates due to environmental factors be specified in advance.

More formally, let T be a continuous and non-negative random variable representing the lifetime of a system. Its distribution can be described by its *probability density function* $f(t)$ or its *cumulative distribution function* $F(t)$. Given either of these metrics, the reliability function of the system, denoted by $R(t)$, is given by

$$R(t) = \Pr(T > t) = 1 - F(t) = \int_t^\infty f(x)\,dx. \tag{2.7}$$

In words, $R(t)$ is the probability that the system's lifetime is larger than t, the probability that the system will survive beyond time t, or the probability that the system will fail after time t. Hence, $R(0) = 1$ and $R(\infty) = 0$. Also, the function $R(t)$ is a non-increasing function of t, and is referred to as the *survivor function* by some authors.

While the reliability function $R(t)$ is adequate to specify a system's lifetime distribution entirely, it does not directly convey whether the system is expected to fail within one year, or whether at least one of a thousand similar systems is expected to fail in the next one year given their combined operation time is roughly, say, a million hours. Thus, product reliability in industry is typically reported using more intuitive metrics, such as the Mean Time To Failure (MTTF) or the Failures-In-Time (FIT) rate of the system, which are defined below.

$$\mathrm{MTTF} = E(T) = \int_0^\infty t \cdot f(t)\,dt \tag{2.8}$$

$$FIT = \frac{10^9}{MTTF\ in\ hours} \tag{2.9}$$

In general, MTTF denotes the expected life of a system, i.e., the expected value or the mean of its lifetime T, whereas the FIT rate is the expected number of failures in one billion operating hours. FIT has an added advantage over MTTF that the component FIT rates can be simply added to derive an upper bound on the overall FIT rate of the entire system. Notice that systems that are repairable may go through several failures before they are scrapped. For such systems, the MTTF represents the mean time to the first failure. After it is repaired and put into operation again, the average time to the next failure is indicated by the Mean Time Between Failures (MTBF). For safety-critical distributed real-time systems, which is the subject of this dissertation, since we care only about their first failure, we use the MTTF and FIT metrics.

More specifically, since we focus on periodic systems (recall the periodic task model from Section 2.1.2.1), we use in this dissertation a discrete definition of the MTTF instead of the definition provided in Eq. (2.8). In particular, when analyzing the reliability of a periodic system S, instead of relying on its probability density function $f(t)$, which is a continuous variable, we rely on its *stopping time* $N(S)$. The stopping time $N(S)$ of the periodic system S is a discrete variable that defines the first iteration of S during which it fails. Using $N(S)$, we define the MTTF of S as follows:

$$\mathrm{MTTF} = T \sum_{n=0}^{\infty} n \cdot \Pr[N(S) = n]. \tag{2.10}$$

The stopping time of a system depends on its failure semantics and robustness specification. For example, the stopping time of an NCS may depend on a finite part of its execution history. In our MTTF analysis for NCS applications, we thus define stopping times based on the widely used *weakly hard* robustness specification (see Chapter 7).

2.2.3 Reliability Analysis

Industry engineers estimate full-system reliability through a set of deductive or inductive reasoning tools. For example, Fault-Tree Analysis (FTA) [189, 212, 224] and related tools are widely used for deductive reasoning using Boolean logic [28], where low-level events are connected to a higher-level event through logic gates. Reliability Block Diagrams (RBDs) [61] are an alternative (or rather an aid) to FTA, since they constitute diagrammatic methods for illustrating the relationship between a complex system and its components. FTA and RBDs are thus very good at estimating a complex system's failure rate given a set of known faults. In contrast, Failure Mode and Effect Analysis (FMEA) [213] helps identify all possible potential failure models in a complex system, their causes, and their effects by analyzing as many components as possible.

We explain the FTA in detail since it is widely used in practice and since its deductive approach is, in principle, analogous to the analyses proposed in this dissertation. A typical fault tree consists of the *failure event* being analyzed, *intermediate events* that lead to the failure event either directly or indirectly (i.e., via other intermediate events), and *basic events* that are like intermediate events, but that cannot be resolved further. These events are connected using *logic gates* (AND, OR, XOR) that determine the causal relationships between events.

For example, consider the use of FTA by Dugan and Van Buren [66] for reliability evaluation of fly-by-wire computer systems. They evaluated an Airbus A310 subsystem consisting of four diverse (but functionally identical) software versions v_1–v_4 deployed on four independent processors h_1–h_4. The four processors constitute two diverse pairs of identical processors. One pair, say, (h_1, h_2) constitutes the primary Flight Control Computer (FCC), whereas the other pair (h_3, h_4) constitutes the backup (hot standby) FCC. The outputs of the identical processors in the primary FCC are compared by an independent decider node d before they are transmitted to the actuators. If the decider finds any discrepancies, the backup FCC is made the new primary. Given this system model, Dugan and Van Buren consider five types of basic fault tree events, which are listed in Table 2.3. Using FTA, they evaluate different combinations of these events that can result in a complete failure of this highly redundant architecture (see Fig. 2.6).

If the individual basic event probabilities in Fig. 2.6 are known in advance, the probability of a full-system failure can be estimated. However, these probabilities are typically estimated empirically through testing and simulation. In general, FTA and similar other tools and techniques consider the individual software implementation as a black box. In contrast, the reliability analyses proposed in this dissertation are more fine-grained since they explore fault propagation at the level of message exchanges between different components of the system.

EVENT	MEANING
H_i	Hardware transient fault in processor h_i
V_i	Independent software fault activation in software version v_i
D	Independent fault activation in decider node d
RV2	Related fault between two software versions
RVA	Related fault between all software versions

Table 2.3: Basic fault events considered by Dugan and Van Buren [66] in their analysis of an Airbus A310 subsystem for fly-by-wire control.

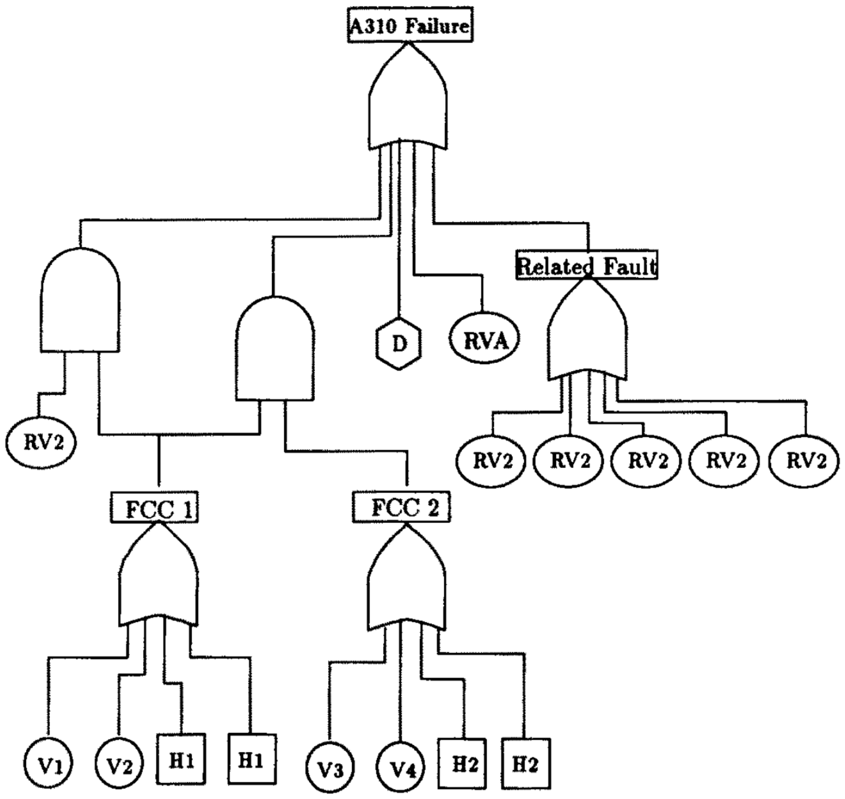


Figure 2.6: Reprinted from [66] with permission from Elsevier. The FTA by Dugan and Van Buren [66] combines the basic events listed in Table 2.3 to evaluate the probability of a full-system failure.

3 FAULT MODEL

As mentioned in Chapter 1, this dissertation deals with the problem of quantitative reliability analysis of CAN- and Ethernet-based distributed real-time systems in the presence of stochastic transient faults.

Any such analysis hinges on two key ingredients: a conservative modeling of transient faults, and an understanding of how the internal state and externally visible outputs of the analyzed system diverge in the presence of transient faults (with respect to a fault-free scenario). Thus, to lay a foundation for the proposed reliability analyses, we discuss in this chapter a widely used probabilistic model of transient faults that we reuse in this work (Section 3.2), and how transient faults affect the functioning of distributed real-time systems, which have strict timing requirements (Sections 3.3 and 3.4).

We start by providing a background on the common terminology used in the dependable computing literature (Section 3.1).

3.1 FAULTS, ERRORS, AND FAILURES

Based on prior work [15], we provide precise definitions of the terms “faults”, “errors”, and “failures” to remove any ambiguity with respect to their interpretation in the rest of this dissertation.

As per Avizienis et al. [15], a *system* denotes a set of components (hardware, software, mechanical components, etc.) that interacts with other components, including humans and the physical world. The *behavior* of a system is a sequence of system states, each of which includes the following: computation, communication, stored information, interconnection, and physical condition. We consider that the system renders *correct service* when it implements the system function or when its behavior adheres to the functional specification of the system. Similarly, we consider that the system renders *timely service* when its behavior adheres to its temporal specification, irrespective of adherence to its functional specification. A *service failure* (or just a *failure*) is thus a deviation of the system’s service from its intended service: incorrect service, untimely service, or no service at all.

Failures affect the system’s behavior outside the system boundary, i.e., as perceived by its environment or its users. In contrast, we define *errors* as deviations of the system’s behavior from its intended behavior inside the system boundary. Thus, failures occur due to one or more errors. The root causes of errors, such as bit flips in memory buffers or defects in the system, are denoted *faults*.

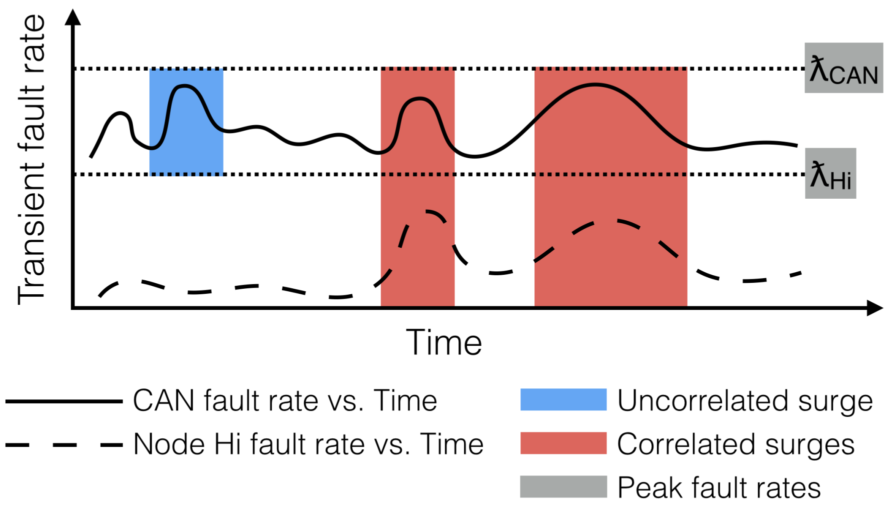


Figure 3.1: Example transient fault rate variation of the CAN bus and host H_i over time. Since our model relies on peak rates, it implicitly accounts for any correlated surges in the transient fault rate across both the CAN bus and host H_i, as shown in the figure.

3.2 TRANSIENT FAULTS

Transient faults (also known as *soft errors*) are temporary faults that arise in digital circuits or networks due to a variety of internal and external noise sources, such as power supply noise, electromagnetic interference, energetic radiation particles, thermal effects, etc. [113, 191, 225]. In contrast to permanent faults, the effect of transient faults lasts for a relatively short duration of time (often, less than a nanosecond) [232]. In this work, we model transient faults as single bit flips independent of the specific causes, as described next.

We assume that the peak rate at which each component in the system may experience bit flips is known in advance (see Fig. 3.1 for a schematic diagram). This assumption is reasonable because system engineers typically determine transient fault rates under worst-possible operating conditions (through a combination of empirical measurements and environmental modeling). The reported rates, in addition, also include safety margins as deemed appropriate by reliability engineers or domain experts.

For example, a rescue robot for nuclear disaster response is designed to tolerate very high degrees of radiation and therefore a high rate of radiation-induced bit flips. In contrast, ECUs used inside a passenger vehicle are not designed to sustain such high bit-flip rates; rather, engineers design such ECUs taking into consideration the worst-case operating conditions expected for a passenger vehicle, e.g., when the vehicle is driven near a radio tower. Formally, we use $\lambda(\text{comp})$ to denote the peak rate at which any component *comp* in the system is expected to experience bit flips based on its operating environment.

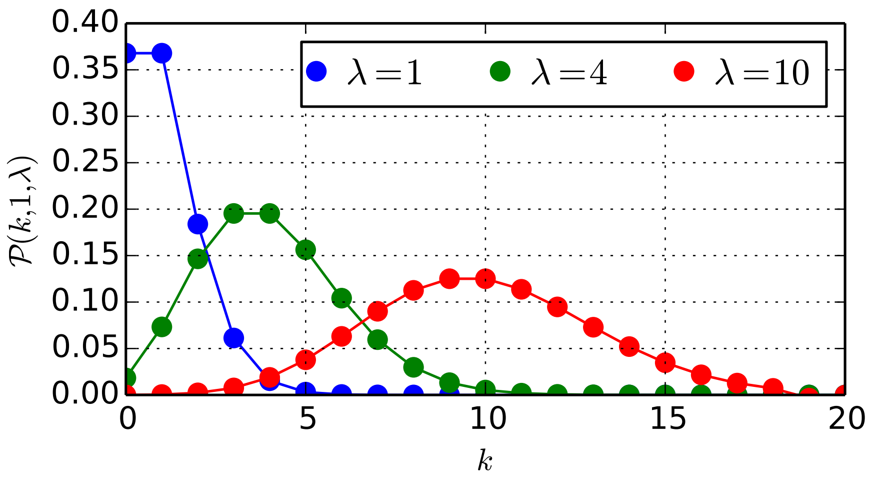


Figure 3.2: Probability Mass Function (PMF) of the Poisson distribution. A variable that is Poisson distributed takes only integer values. Thus, the PMF is defined only at integer values of k; the lines connecting the markers are only to guide the reader.

In other words, $\lambda(\text{comp})$ denotes the peak bit-flip rate that component *comp* is designed to withstand during system operation. For example, $\lambda(\text{CAN})$ and $\lambda(H_i)$ respectively denote the peak bit-flip rates that the CAN bus and host H_i are expected to experience during operation.

As a next step, given the peak transient fault rate for each component, we model the respective fault arrival pattern, i.e., how the transient faults affecting that component vary with time. For this, researchers in the past have relied either on deterministic models such as sporadic fault models with bursts [179, 220], or on probabilistic models such as time-invariant Poisson processes [39, 94, 163] and time-dependent Markov models [198]. In this dissertation, we model the arrival pattern of raw transient faults as random events following a Poisson distribution. As remarked by Broster et al. [39], a Poisson process is a good *approximation* of the worst-case scenario if the mean fault rates used in the Poisson model are obtained from high interference periods, which is the case here since we rely on peak fault rates. Thus, given $\lambda(\text{comp})$ and the *probability mass function* (PMF) of the Poisson distribution defined as follows [11] (see Fig. 3.2 for an illustration),

$$\mathcal{P}(x, \delta, \lambda(\text{comp})) = \frac{e^{-\delta \cdot \lambda(\text{comp})} \cdot (\delta \cdot \lambda(\text{comp}))^x}{x!}, \tag{3.1}$$

we define the probability that x bit flips affect the system component *comp* in any interval of length δ as $\mathcal{P}(x, \delta, \lambda(\text{comp}))$.

Mathematically, the Poisson modeling of transient bit flips implies the following. Since the peak transient fault rate $\lambda(\text{comp})$ for any component *comp* is likely to exceed any transient fault rate τ_{comp} experienced by the component in practice, the probability that the

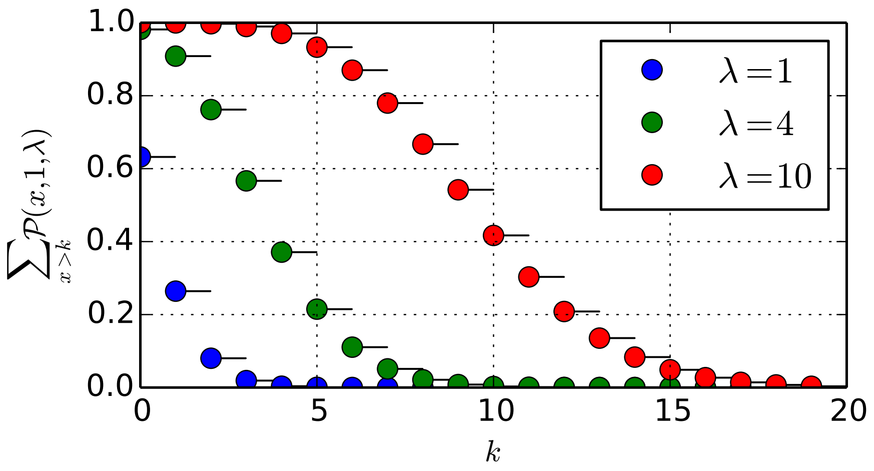


Figure 3.3: $\sum_{x>k} \mathcal{P}(x, \delta, \lambda(\text{comp})) = 1 - \text{CDF}$, where CDF denotes the Cumulative Density Function of the Poisson distribution. Since a variable that is Poisson distributed takes only integer values, the CDF is discontinuous at integer values and flat everywhere else.

component experiences more than k transient faults (for any k) in any interval of length δ as per the assumed Poisson model is also likely to be higher than that in practice. In other words, as illustrated in Fig. 3.3, if $\lambda(\text{comp}) > \tau_{\text{comp}}$, then[1]

$$\sum_{x>k} \mathcal{P}(x, \delta, \lambda(\text{comp})) > \sum_{x>k} \mathcal{P}(x, \delta, \tau_{\text{comp}}). \tag{3.2}$$

Our model assumes environmentally-induced transient faults to be independent based on the stochastic nature of physical sources of transient faults. However, it does implicitly account for correlated surges in the transient fault rates across all components of the system (e.g., when a UAV flies through a strong radar beam), since the Poisson distributions are based on peak transient fault rates (as shown in Fig. 3.1).

3.3 FAULT-INDUCED BASIC ERRORS

Transient faults may manifest as different types of errors based on a system's design and configuration. We focus exclusively on designs and configurations that are used in the safety-critical CPS domain. We introduce first a classification of errors from prior work (Section 3.3.1). Based on this classification, we then specify and model all CPS-specific *basic errors*, i.e., errors which are not application-specific and which can be modeled as independent events based on the stochastic nature of transient faults (Sections 3.3.2 and 3.3.3, respectively).

1 Eq. (3.2) can be easily proven by representing the CDF of the Poisson distribution in the form of an *upper incomplete gamma function* [5, 82, 164].

3.3.1 Classification of Node and Network Errors

We rely on prior work by Barborak et al. [19] and Dwork et al. [67] for understanding the different categories of *processing element* (PE) errors (i.e., host or node errors) and network errors, respectively.[2]

The PE errors are categorized into multiple classes with the property that a stronger class is a subset of a weaker class, i.e., compared to a weaker class, a stronger class imposes more constraints on how a faulty PE can deviate from the correct behavior. The classes, from strongest to weakest, are defined in the following and illustrated in Fig. 3.4.

A *fail-stop* error causes a PE to cease operation, but other PEs are alerted of its failure. In contrast, in a *crash* error, a PE loses its internal state or halts, and hence other PEs are not immediately alerted of its failure. A PE experiences an *omission* error if it fails to meet a deadline or begin a task. More generally, if a PE never completes a task, or completes it either before or after its specified time frame, it experiences a *timing* error. When a PE fails to produce the correct results in response to correct inputs, the error is classified as an *incorrect computation* error. An arbitrary or a malicious error, e.g., when one PE sends differing messages during a broadcast to its neighbors, but that cannot imperceptibly alter an authenticated message[3], is termed as an *authenticated Byzantine* error. *Byzantine* errors represent the universal set comprising every error possible in the system model.

Among these, the crash, omission, and timing error classes refer to problems that occur in the time domain and that are detectable in the time domain. The incorrect computation error class is a superset of the crash, omission, and timing failure classes because a miscalculation may take place in time or space. It is a subset of the Byzantine failure classes since an error due to an incorrect computation is consistent to all PEs in the system. The need for defining a class for authenticated messages arises only for Byzantine failures; no other failure class allows a PE to make false claims about values sent to it by other PEs.

The effect of network errors is commonly abstracted into different types of communication models. For example, Dwork et al. [67] specify

2 Barborak et al. [19] and Dwork et al. [67] classify ways in which a faulty PE or a faulty network may deviate from its correct behavior, respectively. For a distributed system consisting of multiple PEs and multiple network elements, these deviations correspond to an internal system state. Hence, based on the terminology presented in Section 3.1, we refer to the presented material as a classification of *errors*, as opposed to a classification of *faults*, which is the terminology used in [19, 67].

3 Message authentication can be achieved if each PE *cryptographically signs* the messages that it sends. For example, suppose that PEs A and B share a secret key K. PE A can digitally sign any message M by computing a secure hash or a *message authentication code* using key K and attaching it with the original message. Upon receiving the message, B can verify using the shared key K that the message originated at A and that its contents, M, have not subsequently been altered. For more details on cryptographic authenticators and their use in distributed systems, refer to the book by Coulouris et al. [50, Chapter 11]. Note that in case of environmentally-induced non-malicious Byzantine errors, which are the focus of this dissertation, a strong checksum may suffice as a message authenticator, instead of a cryptographic authenticator.

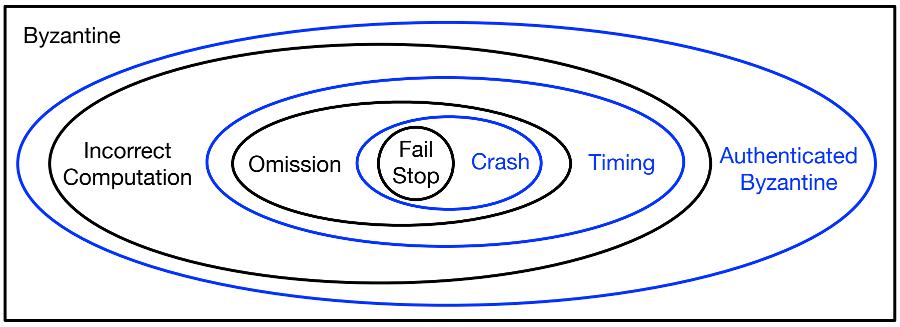


Figure 3.4: Classification of PE errors [19].

three different abstractions for network communication under faults: *synchronous*, *asynchronous*, and *partially synchronous*. Synchronous communication implies that there is a fixed upper bound Δ_{delivery} on the delivery time of messages across the network, and that this upper bound is known a priori. Asynchronous communication implies absence of such an upper bound. A partially synchronous communication lies between synchronous and asynchronous communication. Depending on the specific use case, it may imply that upper bound Δ_{delivery} exists but is not known a priori, or that Δ_{delivery} is known a priori but that it may be violated occasionally due to faults.

3.3.2 Basic Errors in Safety-Critical CPS

In a safety-critical CPS, multiple safeguarding mechanisms are deployed, which typically restart the system upon fault detection to bring it back to its pristine state. For example, if the OS or the hardware detects a fault-induced transient corruption in the PE's memory, it causes an exception that results in a reboot. Since most safety-critical system architectures are equipped with watchdog timers, e.g., see [160], even unbounded hangs due to transient corruption (e.g., loops that never terminate due to a bit flip in the termination condition) eventually trigger a system reboot. Thus, depending on whether other PEs are notified during system reboots and whether the distributed system uses a heartbeat or other monitoring mechanisms, each error resulting in a system reboot falls into the category of either a fail-stop error or a crash error. In fact, a recent study by Schuster et al. [196] of control flow checking schemes in embedded systems reported that more than 91% of faulty computations (induced by bit flips) are caught by OS or hardware mechanisms. Hence, in safety-critical CPS, fail-stop and crash errors are the most likely outcome of transient faults.

However, in some cases, bit flips in memory, or in certain instruction or data registers, may not trigger a kernel exception and instead silently alter the control flow, such that the application task produces no output at all (omission errors) or produces a delayed but correct output (timing errors). A single timing error may further cascade into multiple timing errors if it violates some safety invariant of the scheduler, such as the assumption that tasks do not exceed their statically determined worst-case execution times.

In this work, we assume that appropriate monitoring mechanisms are in place (e.g., [85, 208]) that detect such timing violations, initiate a system reboot, and reset the system state. Based on this assumption, we model any sort of timing violations, i.e., any instance of a fail-stop, crash, omission, or a timing error as a generic crash and reboot error resulting in message omissions for a bounded interval of time. This interval includes the maximum time required to detect the appropriate error before a reboot, as well as the maximum time required to resynchronize state, if any, after a reboot. In other words, our error model conservatively maps every timing violation error to the worst-case scenario where the system remains unavailable for the duration of its reboot, even though this might not always be the case.

Although relatively infrequently, instead of resulting in timing errors, bit flips affecting program memory can also result in generation of wrong output (i.e., incorrect computation errors). For example, with respect to message exchanges between different PEs, a message can be corrupted during preparation before the network controller computes the payload checksum (to be included in the network frame header) due to bit flips in registers or memory of the network controller.

Incorrect computation errors depend on the mechanisms in place to tolerate (or avoid) *latent faults* (i.e., state corruptions that have not yet been detected). In particular, for stateful tasks such as a PID controller, the message computation relies on both the current input and the application state, and the latter can be affected by latent faults. Thus, with each message, we associate an interval during which it is at risk of corruption, known as its *exposure interval*.

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 case of a mismatch), e.g., [203]. 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 in our error model that exposure intervals of application tasks and messages are bounded and that the respective upper bounds can be determined in advance.

Timing errors and incorrect computation errors are sufficient to model all program-visible effects of transient faults in a standalone (i.e., a single-host) system. However, in a distributed system, transmission or network errors also come into play, especially since transient fault rates on networks are typically higher than transient fault rates in PEs. While prior work (see Section 3.3.1) abstracts the effect of network transient faults as communication models with different levels of synchrony, we explicitly consider different types of transmission errors based on the networking standard being used. For example, the CAN protocol has a robust error detection and correction mechanism in place [58]. Erroneous messages are detected using checksums and automatically queued for retransmission. Hence, in case of CAN-based systems, we model the effect of bit flips on the wire as retransmission errors. Similarly, in case of Ethernet-based systems, we model frame corruption and frame omission errors for each Ethernet link, and timing and incorrect computation errors for each Ethernet switch.

Finally, for safety-critical systems, we must also account for the manifestation of transient faults as Byzantine errors, despite the small likelihood of such errors. However, Byzantine errors are not basic errors, but result due to a combination of one or more incorrect computation errors. For example, suppose that a PE broadcasts a message m to all other PEs. In this case, incorrect computation errors in the network layer can result in an inconsistent broadcast of the message, i.e., it is possible that while some PEs receive a pristine copy of message m, the remaining PEs receive a faulty copy $m_{\text{incorrect}}$; or alternatively, different PEs receive distinct copies each [64, 188]. In general, Byzantine errors depend on the implementation of the communication protocol between the distributed PEs, and fundamentally arise due to the lack of an atomic broadcast primitive. Hence, we account for them by analyzing the reliability of an atomic broadcast service implemented in software.

3.3.3 Probabilistic Modeling of Basic Errors

Having defined the basic errors, we next model the occurrence of these errors, similar to the probabilistic modeling of transient faults. Prior studies have shown that a large fraction of transient faults has no negative effects [225]. We thus assume a *derating factor* (also known as the *architectural vulnerability factor*) that accounts for masked transient faults, which can be determined empirically [154]. We let $f_{err}(comp)$ denote the derating factor for basic error type *err* and component *comp*. Accounting for it, the peak rate at which component *comp* experiences a basic error of type *err* is given by

$$\gamma_{err}(comp) = f_{err}(comp) \cdot \lambda(comp), \tag{3.3}$$

e.g., if $f_{crash}(H_i)$ denotes the derating factor for crash errors on host H_i, the peak rate of crash errors on host H_i is $\gamma_{crash}(H_i) = f_{crash}(H_i) \cdot \lambda(H_i)$.

Like the peak transient fault rate, the derating factors are also computed considering the worst-case scenarios and include appropriate safety margins. For example, in case of retransmissions over CAN, it is common to assume that *every* bit flip causes a retransmission, i.e., a derating factor of $f_{retrans}(CAN) = 1$, which is a simplifying but safe overestimation (since a transient fault may occur when the bus is idle and multiple transient faults may result in a single retransmission).

Since real-time tasks are repeated, short workloads, any generated message is equally likely to be affected by an error, and a PE is equally likely to be crashed during any iteration (see [134] for a mathematical basis for this argument). Thus, we model error occurrences as random events following a Poisson distribution. As per this model, we define the probability that x instances of basic errors of type *err* affect component *comp* in any interval of length δ as $\mathcal{P}(x, \delta, \gamma_{err}(comp))$. For example, the probability that x crash errors occur in any interval of length δ on host H_i is given by $\mathcal{P}(x, \delta, \gamma_{crash}(H_i))$.

Similar to transient fault modeling, the probabilistic model for basic errors also guarantees that $\sum_{x>k} \mathcal{P}(x, \delta, \gamma_{err}(comp))$ upper-bounds the probability that any component *comp* experiences more than k error events of type *err* in any interval of length δ. In addition, since we only consider basic errors due to environmentally induced transient faults, we consider them to be independent, like transient faults. We do account explicitly for correlated errors that arise from the system model, e.g., such as situations in which deterministic replicas produce the same wrong output if given the same wrong input.

3.4 SERVICE FAILURES

As mentioned in Section 3.1, a distributed real-time system experiences a failure if it fails to deliver both correct and timely service. However, precise definitions of correct and timely service depend on the application or the workload being analyzed. In this dissertation, since our objective is to analyze two different software layers with different characteristics—an atomic broadcast service over Ethernet and an NCS application over a reliable network—we defer a detailed discussion of their failure models to the respective chapters. In a nutshell, failure of an atomic broadcast service depends on the violation of any of the atomic broadcast invariants (agreement, validity, and timeliness), and is discussed in Section 5.3.1. Failure of an NCS application over a reliable network depends on the frequency and recent history of its failed iterations, i.e., iterations where the final actuation was incorrect, delayed, or skipped, and is formally modeled in Section 7.2.

3.5 RELIABILITY ASSUMPTIONS

Our fault model does not account for failures in the operating system and its scheduling mechanism, or the clock synchronization mechanism. In general, while analyzing the failure rate of an atomic broadcast service or an NCS application, we assume that other system components are reliable, even though the analyzed service may directly depend on them. This does not imply that the proposed analysis is not useful if a dependent component fails; rather it provides a FIT rate for the analyzed service, which can then be composed with the FIT rates of other dependent, dependee, or unrelated subsystems using a fault tree analysis (recall the example FTA from Section 2.2.3). This is a common way of decomposing the reliability analysis of a complex system into manageable subproblems. In fact, extremely rare events like bit flips affecting the scheduler's priority bits occur with such low likelihood that they are best modeled as a separate, additive failure source and accounted for using a separate FIT analysis. For instance, in a fault-tree analysis of a complete system, orthogonal concerns (e.g., a failing power supply vs. loss of network connectivity) are represented by separate branches of the fault tree, whereas tightly coupled components form a single branch and must be analyzed jointly.

Examples of tightly coupled components include tasks constituting an end-to-end NCS iteration and replica coordination protocols, which are analyzed in this dissertation. If an FTA is used to analyze the failure rate of such distributed protocols, it would yield grossly pessimistic estimates, since it can only account for boolean combinations of independent failure probabilities. In contrast, we explicitly consider the dependencies arising from the message passing sequence of the distributed protocols, and also take into account their *temporal robustness* properties (explained in Chapter 7), which in turn results in a more accurate estimate of the protocol failure rate, which can then be used as an input to the full-system FTA.

Part II

BYZANTINE FAULT TOLERANCE

4 TOLERATING BYZANTINE ERRORS IN CPS*

* This chapter is based on our CERTS 2017 [10] and EMSOFT 2019 [93] papers.

Byzantine errors (recall Section 3.3.2) represent complex error scenarios that result from environmentally induced timing and incorrect computation errors affecting specific locations at specific instants of time [64, 188]. For example, in a distributed real-time system networked over Ethernet, a Byzantine error may result from multiple transient bit flips in the controller of a network switch (such that the quantum of corruption due to the bit flips is just enough for Ethernet's checksum-based error detection to fail), at a time when an application message is being queued in the switch.

Detecting and tolerating such errors with hard real-time constraints and with low latency is challenging. Quantifying a system's reliability in the presence of such errors is even more challenging, since the reliability analysis must account for the various possible sources of Byzantine errors as well as timing requirements of the application.

In this chapter, we focus on the first problem, i.e., the design of a BFT distributed real-time system for the CPS domain. To this end, we first survey prior work related to this problem (Section 4.1). We then identify a specific BFT protocol that we believe is ideal for the CPS domain (Section 4.2.1) and present a hard real-time design for implementing the chosen BFT protocol (Section 4.2.2). Finally, using a case study, we compare the performance of the hard real-time design with that of other general-purpose BFT systems (Section 4.2.3). Reliability analysis of the proposed protocol design and of an NCS application deployed on top of such a protocol is the subject of Chapters 5 and 6.

4.1 PRIOR WORK

Driscoll et al. [64] recently revisited the problem of Byzantine errors from a practitioner's perspective. They emphasized that—although the problem of Byzantine errors was first presented by Lamport et al. [129] in the form of a "traitorous anthropomorphic" model (i.e., involving a disloyal human entity) and despite the intuitive notion that processors "have no volition" and that they do not lie—Byzantine errors are real, they can be caused by common hardware faults such as a CMOS bridging fault [130], and they occur far more frequently than commonly expected (in some systems, more than 10^{-5} times per operational hour). Safety-critical aerospace systems, on the other hand, are expected to be designed with a *maximum* failure probability of 10^{-9} failures per hour [190]. Thus, with the goal of safety certification,

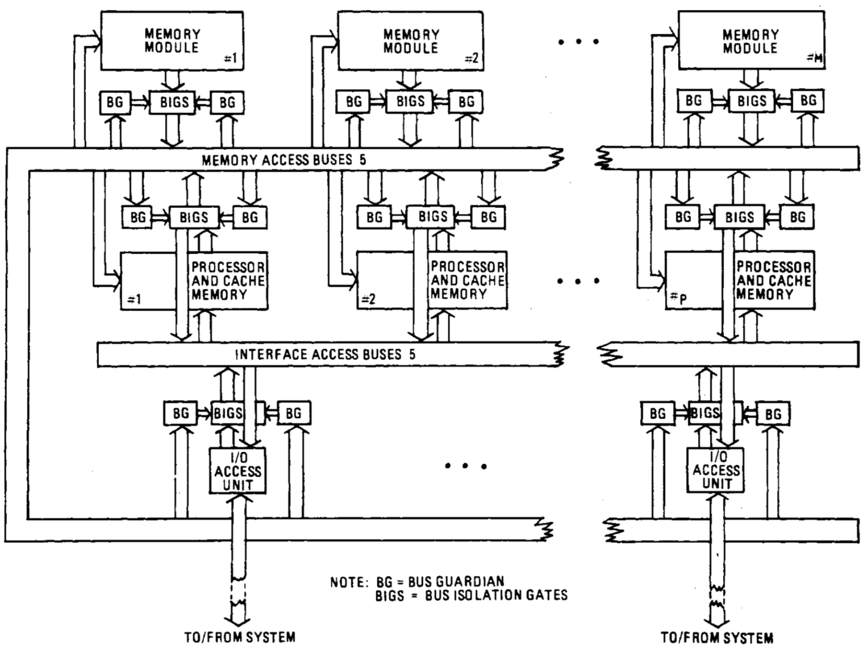


Figure 4.1: © Copyright 1978 IEEE [100]. Simplified diagram of the FTMP consisting of M redundant memory modules, P redundant PEs, as well as redundant I/O access units. Each of these is further associated with two independent bus guardian nodes.

the avionics domain has been the first to acknowledge and tackle the problem of Byzantine fault tolerance in a practical and systematic manner. Hence, in the first part of our survey, we summarize some of the early and prominent work on the development of BFT architectures in the avionics domain. In the second part, we explore some more recent BFT solutions developed for general-purpose computing systems.

4.1.1 BFT in the Avionics Domain

In 1970, Hopkins [101] proposed an information processing system concept for manned space vehicles with the goal of realizing long autonomous flights. The proposed design consisted of a hierarchical distributed system with redundant processors and buses, in which information was processed at various levels based on the peak load, bandwidth, and reaction time requirements at each level. Hopkins's design paved the way for future highly reliable architectures, with more systematic redundancy management backed by analytical reasoning, and with Byzantine fault tolerance.

For instance, Hopkins et al. [100] later proposed the Fault-Tolerant Multiprocessor (FTMP) architecture, which was specifically designed to achieve a failure rate of under 10^{-10} failures per hour on a ten-hour flight without any maintenance. Analyses revealed that Triple Modular Redundancy (TMR) is insufficient to achieve such high reliability without replacement of failed modules. Thus, in addition to employing TMR, FTMP employs an arbitrary number of spares (see Fig. 4.1 for an illustration), and the hardware and software necessary to manage the redundancy, including fault detection, reconfiguration, and recovery mechanisms. Byzantine fault tolerance in FTMP is achieved through hardware-implemented bit-by-bit voting of all transactions, which is made possible by the use of a fault-tolerant clock system. In addition, independent bus guardian nodes are associated with each module to detect and silence any active transmission by the faulty nodes.

Another influential architecture, Software Implemented Fault Tolerance (SIFT), was proposed by Wensley et al. [228] around the same time as FTMP and with similar objectives. Unlike FTMP, though, SIFT does not rely on bit-by-bit voting, but employs voting on the state data of the computer system only at the beginning of each task iteration. Hence, it suffices to ensure that different processors allocated to a task are executing the same iteration (and not necessarily the same instruction), using loosely synchronized clocks. Wensley et al. preferred loose synchronization also because it reduces the likelihood of correlated failures in presence of transient faults, as task replicas may not necessarily execute the same instruction at the same time. Overall (as its name suggests), SIFT uses software intensive implementations for many of its *executive functions*, such as voting and synchronization, whereas FTMP provides hardware assistance for these.

Despite being highly reliable, both FTMP and SIFT were inefficient, since executive functions consumed up to 80 and 60 percent of their system throughput, respectively. To get rid of these performance bottlenecks, Keichafer et al. [115] proposed the Multiprocessor Architecture for Fault-Tolerance (MAFT). As illustrated in Fig. 4.2, each logical module in MAFT is segregated into two separate processors, an Operations Controller (OC) for managing the executive functions and a simple Application Processor (AP). The OCs are networked via a fully connected broadcast network, and also run a Byzantine agreement algorithm [173] to tolerate Byzantine errors affecting critical system parameters. In contrast, each AP is connected to sensors, actuators, and to its respective OC through a dedicated channel to prevent any interference from the executive functions.

The fault-tolerant multiprocessor designs presented above relied on entirely custom architectures (i.e., on specialized processing and networking elements). In contrast, Somani and Bagha [207] and Miner et al. [151] proposed designs where executive functions related to fault tolerance (distributed voting, fault detection, diagnosis, reconfiguration, and recovery) were instead placed "inside" the communication bus, and, therefore, use of *simplex* (i.e., without any redundancy) general-purpose PEs was sufficient. For example, the *MeshKin* architecture by Somani and Bagha [207] (illustrated in Fig. 4.3) relies on a set of fault-tolerant Bus Interface Units (BIUs), which form the intersection

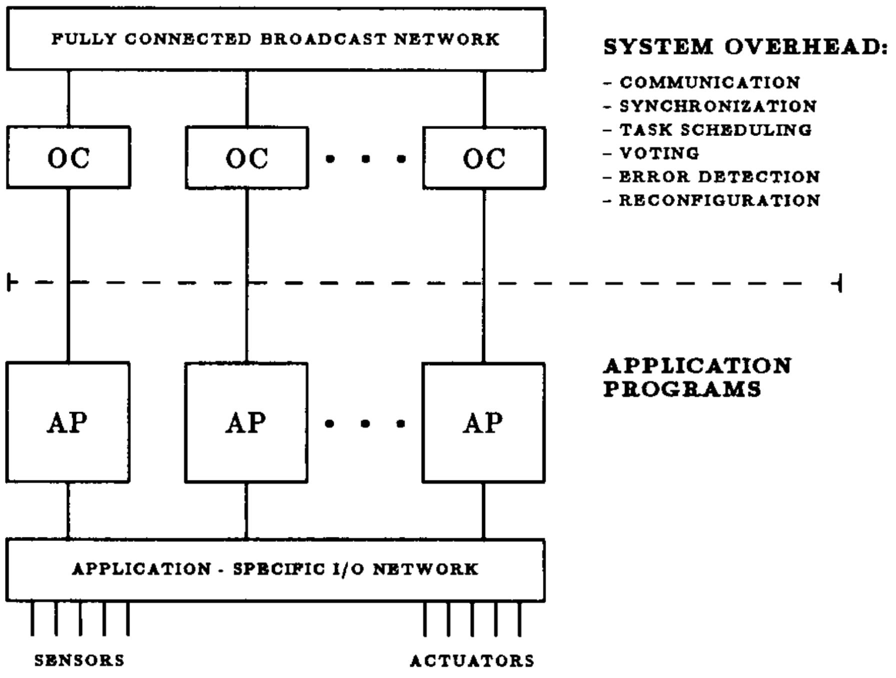


Figure 4.2: © Copyright 1988 IEEE [115]. Simplified diagram of the MAFT system architecture consisting of redundant Operations Controller (OC) and Application Processor (AP) modules. The executive functions run on OC modules and communicate over a separate broadcast network (see top half of the figure). Hence, unlike in FTMP and SIFT, they do not interfere with the application programs (see bottom half of the figure).

points of a grid connecting commodity compute and I/O processing units. Similarly, the core of the Scalable Processor-Independent Design for Electromagnetic Resilience (SPIDER) by Miner et al. [151] is the Reliable Optical Bus (ROBUS), which implements an interactive consistency protocol, a fault-tolerant clock synchronization mechanism, and a consistent diagnosis manager to facilitate the development of a BFT configuration using general-purpose PEs.

Many other architectures have been proposed for safety-critical distributed real-time systems, which mainly differ in the placement and management of redundant elements. For example, the Advanced Information Processing System (AIPS) [128] improved upon FTMP's design; Thompson's work [218] provided similar guarantees using the Inmos transputer family of devices [229], which were specifically designed for parallel processing; the Maintainable Real-Time System (MARS) [120] uses temporal redundancy in addition to spatial redundancy, i.e., each message is transmitted n times, either in parallel over n buses or sequentially over a single bus or a combination thereof; the SAFEbus architecture [102] divides all connected PEs into subsets, and uses private exchanges inside the subsets whereas simplified exchanges between the subsets; and the Time-Triggered Protocol (TTP) in star topology [122] employs a centralized filtering mechanism to

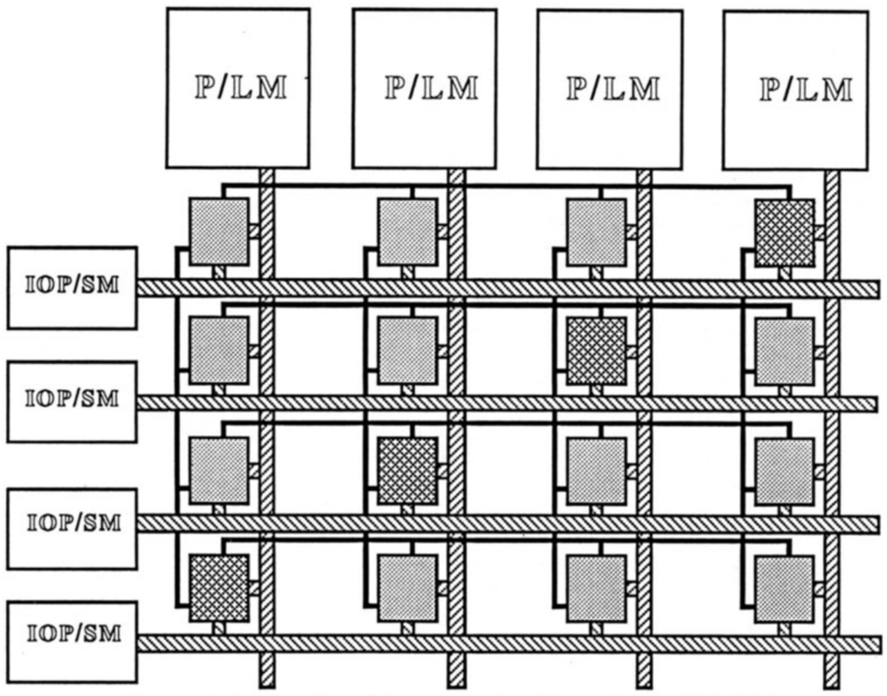


Figure 4.3: Reprinted by permission from Springer Nature Customer Service Centre GmbH. © Springer-Verlag Berlin Heidelberg 1989 [207]. *MeshKin* configuration with Quadruple Modular Redundancy (QMR). Each Processor (ℙ), Local Memory (𝕃𝕄), System Memory (𝕊𝕄), and I/O Interface (𝕀𝕆ℙ) module is replicated four times, and connected via redundant horizontal and vertical buses along with a BIU at every juncture. BIUs colored in dark gray are responsible for controlling the respective bus traffic.

remove the asymmetric manifestation of a Byzantine fault. Smith and Yelverton [205] provide a comparison of some of these alternatives.

In summary, all designs listed above were motivated by the need for fast reaction times (for timely actuation of control systems) and safety certification (which suggested the use of synchronous designs, since adequately validating asynchronous systems was expected to be much more difficult). Thus, each design uses custom hardware—either a specially designed fault-tolerant PE or a specially designed networking layer with redundant buses and custom reconfiguration logic, or both—and relies on a synchronous time base for tolerating Byzantine errors in the presence of transient faults. On the other hand, COTS-based CPS—which are the focus of this dissertation—are not typically made up of such fault-toleant components. Hence, we also survey general-purpose BFT systems, which can be trivially implemented on top of COTS components, next.

4.1.2 General-Purpose BFT Systems

There exists a plethora of work on Byzantine fault tolerance in the cloud computing domain that focuses on general-purpose systems, e.g., Rampart [186], SecureRing [116], Practical Byzantine Fault Tolerance (PBFT) [43], Zyzzyva [123], Spinning [223], Aardvark [46], Raft [166], Redundant Byzantine Fault Tolerance (RBFT) [13], On-Demand Replica Consistency (ORDC) [62], etc. Unlike in the CPS domain, though, the objective of these systems is to protect highly available replicated services from malicious attackers and in presence of software errors, in addition to errors due to environmentally induced bit flips. In the case of attacks and errors that are not environmentally induced, the Byzantine failure model can be applied if the security violations and software errors across replicated nodes are ensured to be independent, e.g., by running different implementations of the service code and OS on each replica [57, 136, 165].

In addition, the synchronous network assumption commonly used in the CPS domain is not always applicable in the cloud computing domain. Realizing fault-tolerant clock synchronization is much harder, and denial-of-service attacks are relatively easier, in such loosely-coupled distributed systems. BFT systems in the cloud are thus designed assuming an asynchronous network model. In particular, since, as per Fischer et al.'s impossibility result [74], solving the Byzantine fault tolerance problem deterministically in an asynchronous setting is impossible, BFT systems in the cloud actually rely on slightly weaker notions of asynchrony, e.g., PBFT by Castro and Liskov [43] relies on *eventual* synchrony for liveness. However, such techniques to circumvent the impossibility result (see [49] for a survey) are typically designed with the objective of achieving high throughput (for instance, by optimizing the fault-free scenario). Properties like low latency and predictability are not always achieved in such designs, which impedes their use for safety-critical distributed real-time systems.

Furthermore, cloud BFT systems commonly employ leader-based designs, where a single *primary* replica is assigned the role of communicating with the clients, and which must be replaced immediately upon failure. If such systems are used for safety-critical time-sensitive applications, the primary can easily become a reliability bottleneck [6, 7]. That is, if the mechanism to switch the primary (upon its failure) takes up to Δ_{switch} time units, but a critical data item in a high-frequency control loop needs to be synchronized among its replicas in less than Δ_{switch} time units, a primary failure can render the control loop unavailable for one or more iterations. Since Δ_{switch} (on commodity platforms) is on the order of at least a few milliseconds or more, whereas control loops may need to execute at a frequency of one iteration per millisecond (or even faster), the probability of failure of a single control loop iteration would actually be proportional to the failure probability of a *single* replica (the primary), despite other active replicas still being functional.

Most interesting among cloud BFT systems, from a CPS perspective, are thus *quorum-based* systems, e.g., [2, 51, 146], which are fundamentally leaderless, and therefore have predictable performance even in the presence of faulty replicas. Quorums of appropriate sizes (BFT quorums) can also be easily configured on top of read and write operations to implement Byzantine fault tolerance in a key-value store (and similar other topic-based abstractions). For example, suppose that N denotes the number of datastore replicas, and R and W denote the number of replicas that must acknowledge each read and each write (respectively). If up to f replicas can be faulty, by ensuring that each read and each write operation intersect in at least $f+1$ nodes i.e., if $R+W \geqslant N+f+1$, every read is guaranteed to intersect with every write in at least one correct replica, which in turn ensures soundness despite Byzantine errors. As for liveness, since the f faulty replicas may not respond, it must be ensured that both $R \leqslant N-f$ and $W \leqslant N-f$. Thus, in order to tolerate up to $f=1$ Byzantine replica, and with $N=4$, read and write BFT quorums sizes are defined as $R=3$ and $W=3$. Both Cassandra [9] and ScyllaDB [197]—which are leading open-source key-value stores—offer `QUORUM` consistency as a configurable option, which corresponds to the use of BFT quorums.

Another class of BFT protocols that are also interesting from a CPS perspective are non-deterministic BFT protocols (notice that the protocols discussed above are all deterministic). In particular, since Fischer et al.'s impossibility result [74] applies only to deterministic protocols, non-deterministic or randomized BFT protocols [20, 182] were proposed to circumvent the impossibility result, or to improve upon asymptotic performance bounds, e.g., the number of rounds required for agreement being lower-bounded by $f+1$ when tolerating up to f failures [73]. The key idea is to weaken one of the correctness properties expected of a deterministic BFT protocol by replacing it with a similar property, but which must hold only with a certain probability. For example, in the $(1-\epsilon)$-terminating protocol by Patra et al. [170], a correct task terminates with probability $(1-\epsilon)$, and in the *almost-everywhere to everywhere* (AER) algorithm proposed by Braud-Santoni et al. [36], agreement is guaranteed for all but $O(\log^{-1} n)$ correct tasks. Protocols such as these can be designed to minimally affect the overall system reliability (i.e., the probability with which the correctness properties are violated is validated a priori to be within acceptable thresholds). However, their inherent non-determinism makes them unfavorable for safety-critical CPS with real-time requirements, since temporal correctness certification becomes challenging. In this dissertation, we therefore focus on the analysis of deterministic BFT protocols (see Section 4.2.1); that is, environmentally-induced transient faults are the only source of non-determinism in our system models.

4.2 HARD REAL-TIME DESIGN

Like the avionics domain BFT systems discussed in Section 4.1.1, fast reaction times and safety certification are also desired from the systems analyzed in this dissertation. However, our primary objective is to validate the reliability of BFT distributed real-time systems built entirely using COTS platforms. Unfortunately, as concluded from Section 4.1.2, prior work on general-purpose BFT systems does not directly apply to distributed real-time systems. Therefore, in this dissertation, we focus on analyzing the key building blocks used in avionics domain BFT architectures—i.e., use of synchronous time base and synchronous information exchange protocols for Byzantine fault tolerance—but in the context of COTS processors and networks, which can then be used to build future BFT systems over COTS platforms.

In particular, with the goal of building BFT systems for real-time applications, we propose in this section a straightforward hard real-time implementation of a BFT information exchange protocol for COTS platforms (which is specified next), and in the subsequent chapters analyze the reliability of the proposed implementation. Reliability analysis of a fault-tolerant clock synchronization protocol for maintaining a synchronous time base is the subject of future work (as discussed in Chapter 8).

4.2.1 Interactive Consistency Protocol

BFT protocols are designed to solve fundamental distributed agreement problems. More complex services such as key-value stores or replicated state machines are then built on top of these foundational primitives. In this dissertation, we analyze a BFT protocol for the classical Interactive Consistency (IC) problem, since interactive consistency is the most generic version of the distributed agreement problem [50]. Formally, the IC problem is defined as follows. Consider a distributed system consisting of N_p processes $\Pi = \{\Pi_1, \Pi_2, \ldots, \Pi_{N_p}\}$, each deployed on an independent PE denoted E_i. Each process Π_i has a private value ν_i and seeks to compute a vector V_i such that for $1 \leqslant k \leqslant N_p$, item $V_i[k]$ corresponds to the private value of process Π_k. The objective of an IC protocol, i.e., which solves the IC problem, is to ensure that $V_i[k] = V_j[k]$ for any two correct processes $\Pi_i, \Pi_j \in \Pi$, and if process Π_k is also correct, then $V_i[k] = V_j[k] = \nu_k$.[1]

1 The IC problem was originally defined for synchronous systems. For asynchronous systems, a similar problem is often denoted as the *vector consensus* problem [48]. In particular, a solution to the IC problem requires a consensus on a vector with values from all correct processes. However, in an asynchronous system, values from all correct process cannot be guaranteed to arrive on time. Therefore, a solution to the vector consensus problem requires consensus on a vector with only $f + 1$ values (assuming up to f faulty replicas).

The IC problem definition is ideally suited for embedded applications that must deal with noisy sensor values. For example, voting procedures or Kalman filters [76] that fuse the private data of all processes into a single consistent value can be trivially implemented as a post-processing step of an IC protocol. In fact, the MAFT [115] and SPIDER [151] architectures discussed earlier (Section 4.1.1) also rely on an IC protocol for this purpose. In contrast, if a more specific version of the distributed agreement problem is used (e.g., such as the *Byzantine Agreement* problem, solving which requires that all processes agree on a single process's private value), multiple instances of an agreement protocol need to be run before their respective outputs can be fused.

We consider the synchronous IC protocol proposed by Pease et al. [173], and actually analyze a generalized version of the protocol. That is, we do not upper-bound the number of faulty processes beforehand and, conversely, also do not lower-bound the number of message exchange rounds. Instead, we parameterize the protocol in terms of an arbitrary number of participating processes N_p and protocol rounds N_r. The reason for this generalization is that, in the presence of environmentally induced transient faults, each process may behave erroneously at different times with non-zero probability. Therefore, depending on the program-visible effects of transient faults, additional protocol rounds or processes do not always increase the chances of solving the IC problem successfully.

Intuitively, the protocol works as follows. Each process Π_i first informs every other process about its private value; in the second round, each process informs every other process about the information received in the first round; in the third round, each process informs every other process about the information received in the second round, and so on. After N_r rounds, each process reduces the collected information to estimate all other processes' private values.

The precise IC protocol executed by each process Π_i is given in Algorithm 4.1. Process Π_i gathers all received information in the form of a tree, called the Exponential Information Gathering (EIG) tree [29], and denoted EIG_i. Each node in EIG_i is a $\langle label, value \rangle$ pair, where the label is an ordered sequence of one or more process identities. In the beginning (Line 2), EIG_i is initialized with the root node $\langle \epsilon, v_i \rangle$, where ϵ denotes an empty label and v_i denotes the private value of Π_i.

For each of the N_r rounds thereafter, Π_i executes up to three steps. During the *sending step* in round r (Lines 5–7), Π_i sends to other processes all nodes in the $(r-1)^{st}$ level of its tree (i.e., all nodes with $|\alpha| = r-1$), except any nodes with value $\perp$ (as explained later, these correspond to omitted messages) and nodes whose labels contain Π_i (to avoid cycles in the EIG tree labels).

The next step is the *state transition step* during which Π_i updates its EIG tree based on the received messages (Lines 9–16). In particular, during round r, for every level-$(r-1)$ node $\langle \alpha, v \rangle$ in EIG_i (i.e., for

Algorithm 4.1 Achieving IC in a synchronous system (Π_i's version).

```
 1: procedure INITIALIZATION
 2:     $EIG_i.addRoot(\langle \epsilon, v_i \rangle)$
 3: procedure ROUND($r$)
 4:     ▷ sending step
 5:     for all $\langle \alpha, v \rangle \in EIG_i$.nodes s.t. $|\alpha| = r - 1$ do
 6:         if $\Pi_i \notin \alpha \wedge v \neq \perp$ then
 7:             send $\langle \alpha, v \rangle$ to all processes in $\Pi \setminus \{\Pi_i\}$
 8:     ▷ state transition step
 9:     for all $\langle \alpha, v \rangle \in EIG_i$.nodes s.t. $|\alpha| = r - 1$ do
10:         for all $\Pi_j \in \Pi$ s.t. $\Pi_j \notin \alpha$ do
11:             if $\Pi_i = \Pi_j$ then
12:                 $EIG_i.addChild(\langle \alpha, v \rangle, \langle \alpha\Pi_j, v \rangle)$
13:             else if $\langle \alpha, v' \rangle$ is received from $\Pi_j$ then
14:                 $EIG_i.addChild(\langle \alpha, v \rangle, \langle \alpha\Pi_j, v' \rangle)$
15:             else
16:                 $EIG_i.addChild(\langle \alpha, v \rangle, \langle \alpha\Pi_j, \perp \rangle)$
17:     ▷ reduction step
18:     if $r = N_r$ then
19:         for all $\langle \alpha, v \rangle \in EIG_i$.nodes from $|\alpha| = N_r - 1$ to $|\alpha| = 1$ do
20:             candidates $= \emptyset$, $v_{majority} = \perp$
21:             for all $\langle \alpha\Pi_j, v' \rangle \in EIG_i.getChildren(\langle \alpha, v \rangle)$ do
22:                 if $v' \neq \perp$ then
23:                     candidates $=$ candidates $\cup \{v'\}$
24:             if candidates $\neq \emptyset$ then
25:                 $v_{majority} = simpleMajority(candidates)$
26:             $EIG_i.updateValue(\langle \alpha, v \rangle, v_{majority})$
27:             if $\alpha = \Pi_k$ then                                  ▷ if level-1 node
28:                 $V_i[k] \leftarrow v_{majority}$                      ▷ update the decision vector
```

every node with $|\alpha| = r - 1$), Π_i expects other processes to send their corresponding level-$(r-1)$ nodes. If Π_i indeed receives a message of the form $\langle \alpha, v' \rangle$ from another process Π_j, it adds the pair $\langle \alpha\Pi_j, v' \rangle$ as a child of node $\langle \alpha, v \rangle$; if Π_i does not receive such a message, it adds a dummy pair $\langle \alpha\Pi_j, \perp \rangle$ to register an error-induced omission. Note that Π_i does not expect a message from Π_j if $\Pi_j \in \alpha$ (Line 10), since cycles in the EIG tree labels are avoided during the sending step.

Information gathering as described above goes on for N_r rounds, where N_r is a freely configurable parameter.[2] In the last round, the state transition step is followed by a *reduction step*, during which a reduction function is recursively applied to each sub-tree of EIG_i's

2 In principle, the quantum of information exchanged among processes reduces in each round, since each process Π_i never sends any node in its EIG tree whose label already contains the process identity Π_i (see the $\Pi_i \notin \alpha$ condition in Line 6). Hence, for all practical purposes, $N_r \leqslant N_p$, i.e., beyond $N_r = N_p$ rounds, unless these is a corruption in the process state, no messages are exchanged.

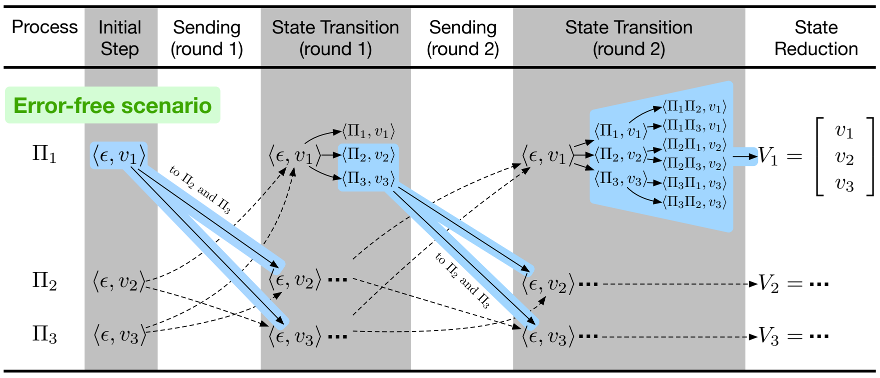

Process
Initial Step
Sending (round 1)
State Transition (round 1)
Sending (round 2)
State Transition (round 2)
State Reduction
Error-free scenario
Π_1
Π_2
Π_3
$\langle\epsilon, v_1\rangle$
$\langle\epsilon, v_2\rangle$
$\langle\epsilon, v_3\rangle$
to Π_2 and Π_3
$\langle\Pi_1, v_1\rangle$
$\langle\Pi_2, v_2\rangle$
$\langle\Pi_3, v_3\rangle$
$\langle\Pi_1\Pi_2, v_1\rangle$
$\langle\Pi_1\Pi_3, v_1\rangle$
$\langle\Pi_2\Pi_1, v_2\rangle$
$\langle\Pi_2\Pi_3, v_2\rangle$
$\langle\Pi_3\Pi_1, v_3\rangle$
$\langle\Pi_3\Pi_2, v_3\rangle$
$V_1 = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$
$V_2 = \cdots$
$V_3 = \cdots$

Figure 4.4: IC protocol execution for $N_p = 3$ and $N_r = 2$ in an error-free scenario. EIG trees for processes Π_2 and Π_3 are not shown for brevity. Regions highlighted in blue denote the EIG tree segments that are sent over the network or used during the reduction step.

root node (Lines 19–28). If any node $\langle\alpha, v\rangle$ is a leaf node (i.e., $|\alpha| = N_r$), its value does not change; otherwise, if v_{majority} denotes the majority among the values of node $\langle\alpha, v\rangle$'s children, $\langle\alpha, v\rangle$ is updated to $\langle\alpha, v_{\text{majority}}\rangle$ (Line 26). The decision vector V_i is finally determined by the level-1 nodes (Line 28). Message exchanges in the IC protocol for $N_p = 3$ and $N_r = 2$ in an error-free scenario are illustrated in Fig. 4.4.

4.2.2 Realization using the Periodic Task Model

The IC protocol described in the previous section can be realized in many ways. However, a hard real-time implementation is most beneficial for safety-certification and typically expected when building safety-critical CPS. For instance, many CPS applications, including control applications, rely on strong temporal properties of the underlying infrastructure to ensure a minimum quality of service [81]. Moreover, hard real-time predictability is where prior literature on Byzantine fault tolerance falls short, which is why it is important to sketch a design that we know for sure to be analyzable. Hence, we map the IC protocol to Liu and Layland's periodic task model [137], which has been widely studied in the real-time systems community and which, therefore, provides a solid foundation for temporal certification.

In particular, we propose a design where the execution of the IC protocol by each process Π_i is modeled using multiple periodic tasks deployed on the respective PE. The proposed design depends on two assumptions. First, we assume that PE clocks are synchronized, which can be ensured on commodity PEs using clock synchronization protocols such as the Precision Time Protocol (PTP) [75]. Second, we assume that network latency is predictable, which can be ensured using time-sensitive networking standards, e.g., Ethernet's Time-Sensitive Networking (TSN) standard [161].

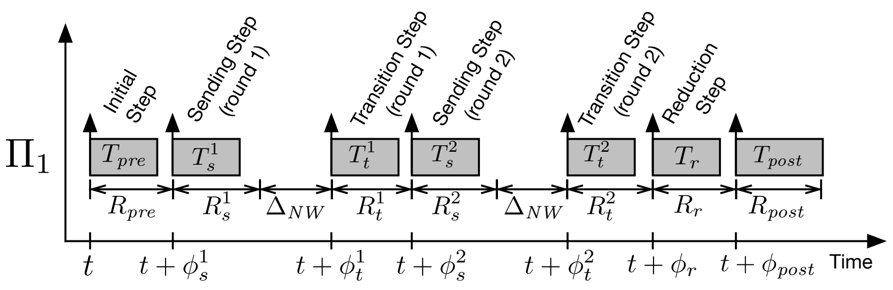


Figure 4.5: Periodic tasks as part of process Π_1 corresponding to the IC protocol execution starting at time t.

In the following, we present the detailed task model, which is also illustrated in Fig. 4.5. Since the IC protocol is symmetric for all processes, identical task sets are deployed on each PE; therefore, we omit the process index i from the notations to reduce clutter.

Recall that the protocol consists of N_r rounds, each consisting of a sending step and a state transition (or receiving) step, and the last round also consisting of a reduction step. Hence, we realize each process by a set of tasks $T_s = \{T_s^1, T_s^2, \ldots, T_s^{N_r}\}$ that execute the N_r sending steps (respectively), a set of tasks $T_t = \{T_t^1, T_t^2, \ldots, T_t^{N_r}\}$ that execute the N_r state transition steps (respectively), and a task T_r that executes the reduction step in the end. Additionally, we model tasks T_{pre} and T_{post} that execute at the beginning and end of the protocol, respectively, and which interface the IC protocol with the application. T_{pre} is also responsible for initializing the EIG tree.

We assume that the IC protocol is invoked periodically with time period P, i.e., a new protocol instance with the objective of achieving interactive consistency over a new set of values is initiated every P time units. Hence, all tasks are assigned a time period of P, and each new activation of the task set corresponds to a new IC protocol instance.

To ensure that the tasks are activated in the order required by the IC protocol, each task is also assigned an appropriate *release offset*. Task T_{pre}, which is expected to execute before any other IC protocol tasks, is released periodically with release offset 0 on each PE, i.e., T_{pre} becomes ready for execution at time instants 0, P, $2P$, and so on. Suppose that R_{pre} denotes the *global worst-case response time* of T_{pre} across all PEs, i.e., the periodic invocations of T_{pre} on all PEs finish their executions at the latest by time instants R_{pre}, $P + R_{pre}$, $2P + R_{pre}$, and so on, respectively. As per Algorithm 4.1, task T_s^1, which is responsible for executing the sending step of round one, must follow task T_{pre}. Thus, T_s^1 is assigned a release offset of $\phi_s^1 = R_{pre}$, i.e., T_s^1 becomes ready for execution at time instants R_{pre}, $P + R_{pre}$, $2P + R_{pre}$, and so on. We omit differences due to clock skew in these absolute time instants to avoid clutter; this can be fixed by adding the maximum clock skew between any two clocks, which is known from the clock synchronization protocol, to these time instants.

The next step as per Algorithm 4.1 is the state transition step of round one, which is executed by task T_t^1. Task T_t^1 must also wait for the messages sent during the preceding sending step to be transmitted. Thus, task T_t^1 is assigned a release offset of $\phi_t^1 = \phi_s^1 + R_s^1 + \Delta_{NW}$, where R_s^1 denotes task T_s^1's global worst-case response time and Δ_{NW} denotes the worst-case latency for the exchange of IC protocol messages over the network. This assignment ensures that, in an error-free scenario, the sending step of round one has finished sending all messages and that these messages have been transmitted before the state transition step of round one begins. Other tasks are assigned their release offsets in a similar manner (see Fig. 4.5).

The task organization discussed above works only if all the tasks with their respective parameters can be integrated *successfully* on the host platforms, i.e., without any deadline misses. This requires the use of a predictable scheduler at runtime and an a priori schedulability analysis of the task set. In this work, we consider the partitioned fixed-priority scheduling policy as our predictable scheduling policy, which is supported on all major real-time platforms such as VxWorks and QNX, and also on Linux (via `SCHED_FIFO` and suitably chosen processor affinity masks). For schedulability analysis, the existing literature on real-time scheduling theory for periodic task models [53] provides a rich foundation for checking if each task meets its *implicit deadline*, i.e., finishes before the next task instance arrives.

The proposed task modeling breaks down the IC algorithm into smaller tasks to ensure that the pessimism incurred in the schedulability analysis is minimal. An alternative design where the entire IC algorithm is implemented as one periodic task with suspensions (while awaiting network I/O) requires use of suspension-aware schedulability analyses [44], which are prone to substantial pessimism. Another alternative design where the periodic tasks are implemented without suspensions (i.e., when tasks spin while waiting for I/O) is extremely inefficient in terms of CPU usage. Further, note that these alternatives pertain only to the modeling of the protocol implementation. An actual implementation can still realize all tasks (model entities) within a single sequential process (OS facility).

4.2.3 Case Study: Key-Value Store

To evaluate the feasibility of the hard real-time IC protocol design presented in Section 4.2.2, we implemented a BFT key-value service on top, which we refer to as *Achal*, and compared its performance against state-of-the-art general-purpose BFT systems. In particular, we compared Achal's performance against Cassandra [9] configured with BFT quorums, and against a key-value service implemented on top of BFT-SMaRt [25] (which is a state-of-the-art library for implementing

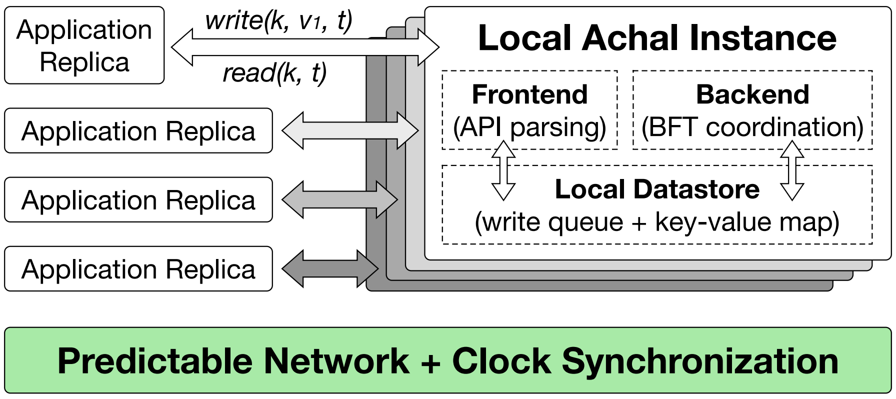


Figure 4.6: Overview of Achal's architecture

State Machine Replication (SMR) [195] with Byzantine fault tolerance). We start with a brief description of Achal's overall design.

4.2.3.1 *Achal: A Hard Real-Time Key-Value Service*

Fig. 4.6 shows an overview of Achal's architecture. Each PE hosts a *local instance* of Achal consisting of a *frontend* that interfaces with the application replicas hosted on that PE, a *backend* that interfaces with the local Achal instances on other PEs, and an in-memory *local datastore*.

Achal's frontend offers application replicas the usual read and write interface of a key-value service, but enhanced with an absolute time parameter t. In particular, the $\mathrm{write}(k, v, t)$ operation writes the value v to key k with absolute *publishing time* t; and the $\mathrm{read}(k, t)$ operation returns the latest value v with publishing time no earlier than t for which consensus has been achieved. Thus, a written value becomes visible to applications only at time t, that is, no read of k prior to time t will return v. Conversely, a read operation returns the latest value for key k that was published at or later than time t.

The absolute time parameter allows both operations to be *non-blocking*. That is, the write operation stores the given value and publishing time to the local write queue (part of the local datastore) and then immediately returns; coordination with other replicas occurs asynchronously. In fact, depending on its publishing time t, coordination for a write operation can be delayed to accommodate other more urgent operations (e.g., another write with an earlier publishing time $t' < t$). Similarly, the read operation translates into a synchronous lookup from the key-value map in the local datastore, and thus immediately yields a value for which coordination has already completed or an error signaling the absence of any matching value.

As an example, we illustrate in Algorithm 4.2 a PID control loop programmed over Achal. Active replicas of the PID controller synchronize the *error* and *integral* variables (which are used across iterations, i.e., which denote the control loop's global state) using Achal. For clar-

Algorithm 4.2 Periodic task of a PID controller for balancing an inverted pendulum, programmed over Achal.

```
1:  procedure PeriodicTaskActivation
2:      time ← timeOfLastActivation()              ▷ compute freshness constraint
3:      current ← getSensorData()                  ▷ get latest angle encoder value
4:      error ← target − current                   ▷ compute absolute error
5:      ▷ update cumulative error and rate of change of error
6:      integral ← Achal.read("integralKey", time) + error
7:      derivative ← error − Achal.read("errorKey", time)
8:      ▷ compute actuation force as a wighted sum of ...
9:      ▷ absolute error, cumulative error, and rate of change of error
10:     force ← kp ∗ error + ki ∗ integral + kd ∗ derivative
11:     time ← timeOfNextActivation()              ▷ compute publishing time
12:     ▷ synchronize state with other replicas (if any)
13:     Achal.write("errorKey", error, time)
14:     Achal.write("integralKey", integral, time)
15:     actuate(force)                             ▷ apply force on the pendulum cart
```

ity, error handling has been omitted. Notice that Achal's API enables a programmer to make definitive statements about when written data is available in the system and ready to be read by tasks on different PEs. The resulting data determinism eliminates execution-time dependent race conditions, and is thus ideal for CPS domain applications.[3]

While Achal's frontend presents itself as one logical datastore to the application, the backend ensures write propagation and takes care of BFT replica coordination using the predictable hard real-time design of the IC protocol, which was presented in Section 4.2.2. As a result, the system is able to reject operations with infeasible publishing times in advance. Specifically, if $\Delta_{\text{coord}} = P + \phi_{\text{post}} + R_{\text{post}}$ denotes a deployment-specific upper bound on the maximum time required to coordinate among all replicas (based on the periodic task model presented in Section 4.2.2), and if an application executes the operation $\text{write}(k, v, t)$ at time t_{now}: Achal rejects the write if $t_{now} + \Delta_{\text{coord}} > t$. Similarly, the system rejects read operations that specify a time in the future. In other words, the predictable hard real-time design helps ensure that in an error-free scenario, application reads never fail.

4.2.3.2 *Setup, Configuration, and Methodology*

All the experiments were performed on a cluster of four Raspberry Pi 3 Model B+ units [185], each equipped with a 1.4GHz Cortex-A53 quad-core processor and 1 GB of memory. The four Pis were connected over IEEE 802.3ab Gigabit Ethernet using a 1 Gbps Ethernet connection.

3 The publishing time parameter is inspired by the Logical Execution Time (LET) paradigm proposed by Henzinger et al. [99], which decouples the read and write time of global data used by a task from the actual execution time of the task. See the book chapter by Kirsch and Sokolova [117] for a detailed explanation.

Since the Ethernet controller is internally connected via USB 2.0, the effective maximum throughput was limited to 300 Mbps.

The Pis were running Linux kernel 4.14.27 applied with both Raspberry Pi and PREEMPT_RT patches.[4] To synchronize their clocks, the Pis were running the Precision Time Protocol daemon (PTPd) version 2.3.2 [68] (an open source implementation of PTP for Unix-like computers), resulting in an observed clock skew of around 10 µs. PTPd was configured to execute in a hybrid mode that utilizes both multicast and unicast so as to reduce the amount of PTP messages per client.

Achal was implemented using a set of POSIX processes. To realize partitioned fixed-priority scheduling on Linux, processor affinities and the SCHED_FIFO scheduling policy were used.[5] The memory required by the tasks and the shared data structures was locked into physical memory at startup (i.e., pre-faulted and excluded from paging using memlockall). The tasks communicated via unicast UDP to realize point-to-point message channels.

We deployed Cassandra with the recommended settings, with the four Pis configured as one rack in one datacenter. For a fair comparison with Achal, we also modified some system parameters to improve Cassandra's predictability. In particular, Cassandra was configured to use the jemalloc library, its *cache save* intervals were set high enough so that they did not interfere with the experiments, RLIMIT_MEMLOCK was set to unlimited to allow Cassandra to lock a sufficient amount of memory, and all employed Cassandra Query Language (CQL) insert and select statements were prepared (i.e., pre-compiled) on system startup to minimize query parsing overheads. The above memory-related settings were also applied to BFT-SMaRt.

We also ensured that Achal, Cassandra and BFT-SMaRt have equivalent semantics and provide the same level of fault tolerance. In particular, Achal was configured to tolerate up to $f = 1$ faulty replicas, by using $3f + 1 = 4$ replicas, but without any cryptographic message authenticators. Thus, for parity, i.e., to ensure that the baselines do not incur additional overheads, we did not use a hardened version

4 The PREEMPT_RT patch [150] minimizes the amount of non-preemptible kernel code by reducing the number and the length of critical sections in the kernel that mask interrupts or disable preemptions. Thus, the PREEMPT_RT patch improves the scheduling latency of real-time user threads. In fact, Linux with the PREEMPT_RT patch is also considered the de facto standard real-time variant of Linux.

5 Each thread in Linux has an associated scheduling policy [235]. *Normal* threads are associated with either the SCHED_OTHER, SCHED_IDLE, or the SCHED_BATCH scheduling policy, whereas *real-time* threads are associated with either the SCHED_FIFO, SCHED_RR, or the SCHED_DEADLINE scheduling policy. The real-time threads are also associated with a static priority. When a real-time thread becomes runnable, it immediately preempts any currently running normal threads or lower-priority real-time threads. While SCHED_FIFO schedules threads with same priority in a First In, First Out (FIFO) manner, SCHED_RR uses Round-Robin (RR) (with a fixed maximum time quantum) instead. SCHED_DEADLINE schedules threads using Global Earliest Deadline First (GEDF) in conjunction with a constant bandwidth server [71]. It requires that each thread is modeled as a sporadic task (recall different real-time task models from Section 2.1.2.1).

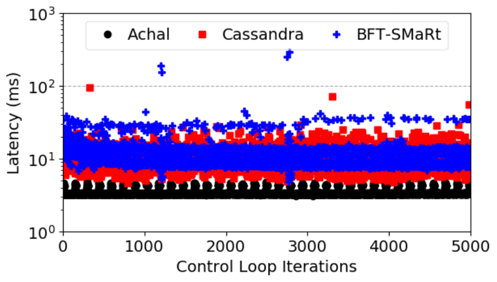
of Cassandra [78] and disabled the use of MAC-based signatures in BFT-SMaRt. In addition, to let Cassandra tolerate Byzantine failures, all Cassandra queries were executed with the `QUORUM` consistency level. We also implemented a thin proxy layer on top of Cassandra's CQL and on top of BFT-SMaRt's RPC library to expose an Achal-like temporally-aware API over Cassandra and BFT-SMaRt.

After configuring the three systems, we evaluated them in terms of their read and write latencies using a periodic PID control loop as the application workload. In each iteration, the application program first reads a value that was written in the previous iteration, and then writes a new value that will be read in the subsequent iteration. However, while a write request in Achal returns immediately after writing to the local write queue (non-blocking), a write request in Cassandra and BFT-SMaRt returns only after the write request has been propagated to other replicas (blocking). We thus required different measurement approaches for each system.

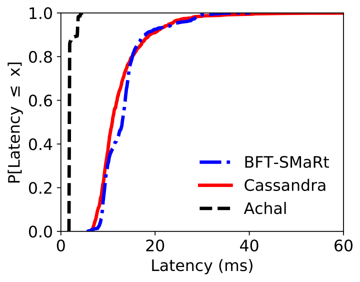
The read and write latencies for Cassandra and BFT-SMaRt were simply measured by computing the time to execute their read and write operations, respectively (since these are blocking operations). The obtained values are thus independent of the application time period and the publishing time. For Achal, as explained in Section 4.2.3.1, the effective write latency depends on both the application time period and the period of the Achal tasks. Thus, to obtain the minimum possible write latency in Achal, we minimized the period subject to maintaining temporal correctness (i.e., we used the shortest period possible without missing any deadlines), and then measured the time to finish replica coordination. In other words, we first estimated using profiling the worst-case time to execute each Achal task and worst-case network delay with a relaxed period, and then use these estimations to run Achal with a tighter period. In practice, for safety certification, the profiling would need to be replaced by the use of tools such as aiT [233] and SymTA/S [219] for sound worst-case execution time and network analysis, respectively. The read latency in Achal is measured in the same way as in Cassandra and BFT-SMaRt since a read operation in Achal simply involves reading a value locally.

4.2.3.3 *Evaluation Results*

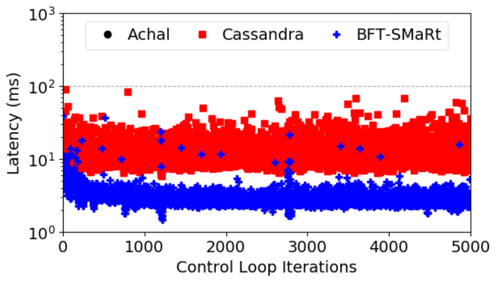
We start with single-key experiment results. To evaluate the latency profiles of Achal and the baselines in order to estimate their predictability, we measured the read and write latency for each system using a single application control loop accessing one key per iteration. In case of Achal, only one instance mapped to a single core was running per Pi, since we want to evaluate its single core performance first. BFT-SMaRt and Cassandra instances, in contrast, were allowed to use up to all four cores on the Pi, since they are multi-threaded by design. The latency scatter plots and CDFs are illustrated in Fig. 4.7.



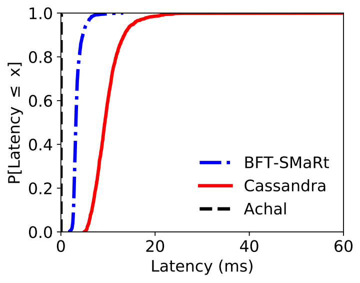
(a) Write latency (scatter plot)



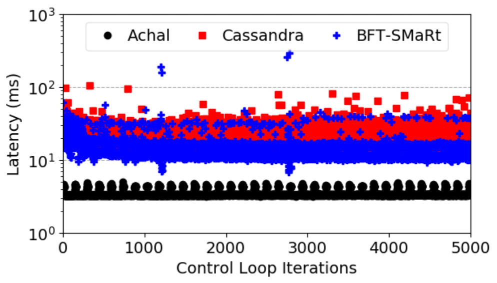
(b) Write latency (CDF)



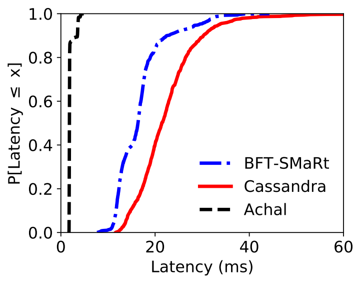
(c) Read latency (scatter plot)



(d) Read latency (CDF)



(e) Total latency (scatter plot)



(f) Total latency (CDF)

Figure 4.7: Single-key experiment results. Achal's read latency was consistently under 100 microseconds, hence not visible in (c) and (d).

Achal, BFT-SMaRt, and Cassandra's write latency distributions (see Figs. 4.7a and 4.7b) each follow a unique pattern. The write latency of Achal always remains between 3 ms and 5 ms. This was expected since Achal's latency depends on and is upper-bounded by the time period of Achal tasks by design (which was 5 ms in this case). In contrast, the write latency of Cassandra and BFT hovers between 10 ms and 30 ms for a majority of iterations, but exceeds 100 ms occasionally.

The read latency distributions of Achal, BFT-SMaRt, and Cassandra (see Figs. 4.7c and 4.7d) vary differently from their write latency distributions. Achal's read latency was consistently under 100 µs (and hence not visible when using the log scale in Fig. 4.7c). This was again expected since Achal's read operation reads the key from the local datastore and does not require any coordination. The read latency of BFT-SMaRt is also low (a couple of milliseconds). In contrast, Cassandra's read latency is significant, averaging in excess of 10 ms.

BASELINE	TKVS	WITHOUT TKVS	WITHOUT TKVS AND AR
BFT-SMaRt	3.47 ms	3.41 ms	3.12 ms
Cassandra	12.58 ms	11.82 ms	12.53 ms

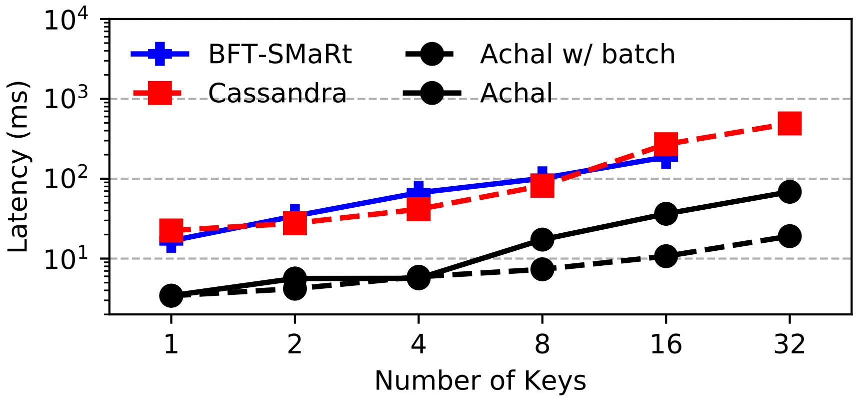
Table 4.1: Average read latency in BFT-SMaRt and Cassandra for a single key and for three different configurations.

We attribute this to the use of a BFT quorum during read operations, which requires that a value be read from $2f + 1$ replicas (to tolerate f faulty replicas).

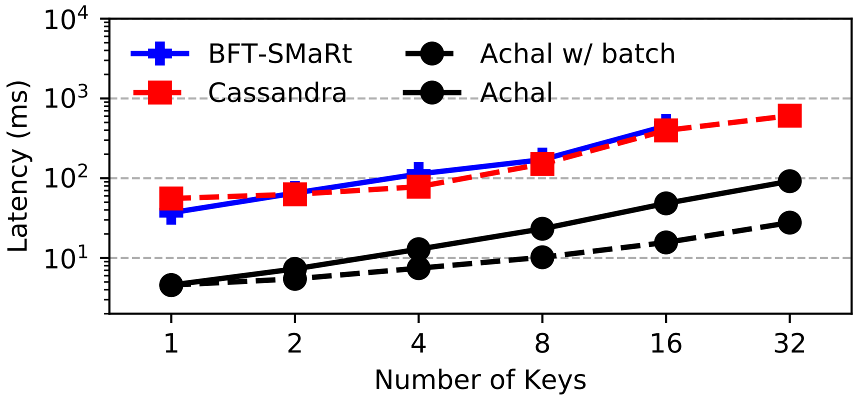
We also report the sum of the individual read and write latencies (see Figs. 4.7e and 4.7f), since it lower-bounds the minimum achievable time period (i.e., maximum possible frequency) of an actively replicated periodic control loop deployed on top of Achal, BFT-SMaRt, and Cassandra. The results clearly show Achal to be more capable in this regard. In addition, the frequent spikes in the latency results for Cassandra and BFT-SMaRt expose the inherent unpredictability in their throughput-oriented designs. In contrast, Achal exhibits little latency variability, which reflects its predictable design.

To verify that BFT-SMaRt and Cassandra's read latencies were not severely affected by the proxy layer that was used to implement Achal-like temporally-aware semantics, we also evaluated their read latencies without the proxy layer, and also without the active replication of application control loop (i.e., with only the datastore instances replicated). The results are summarized in Table 4.1; TKVS denotes the use of a Temporally-aware Key-Value Service API like Achal, and AR denotes active replication. For both baselines, the overhead due to implementation of the time-aware semantic layer was negligible (see column "*without TKVS*"). In fact, the overhead due to multiple values being written by active replicas of the application control loop, as opposed to a single write per key, was also negligible (see column "*without TKVS and AR*"). We thus attribute the read latencies to the respective coordination protocols.

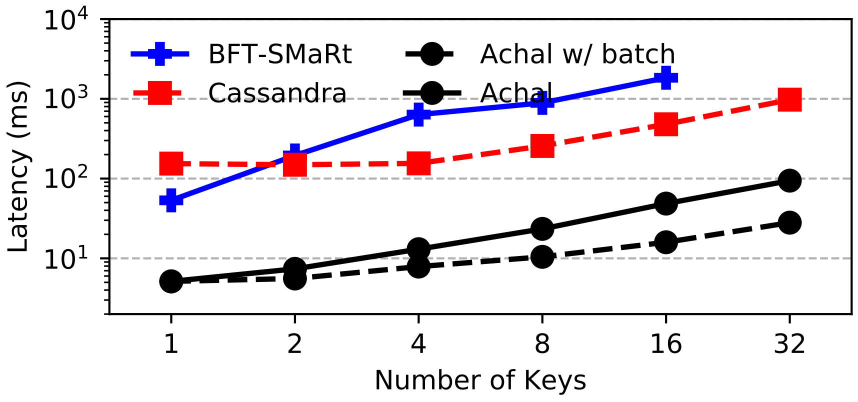
Next, we discuss the multi-key experiment results. Our objective is to evaluate whether Achal scales well with the number of keys, and whether the observations made for the single-key experiments also hold when the application writes (and reads) multiple keys to (and from) the datastore. In this case, we also evaluated an extended version of Achal with *explicit batching*, where the application can deliver all writes together using a batch API. We measured read and write latencies, and report the average, the 99th, and the maximum aggregate latencies (i.e., the sum of read and write latency). The results are illustrated in Figs. 4.8a to 4.8c. Once again, the order of magnitude difference between the latency of Achal and the baselines is apparent irrespective of the number of keys, and even in the average case. For all systems, the latency scales proportionally to the



(a) Average latency vs. #keys



(b) 99th percentile latency vs. #keys



(c) Maximum latency vs. #keys

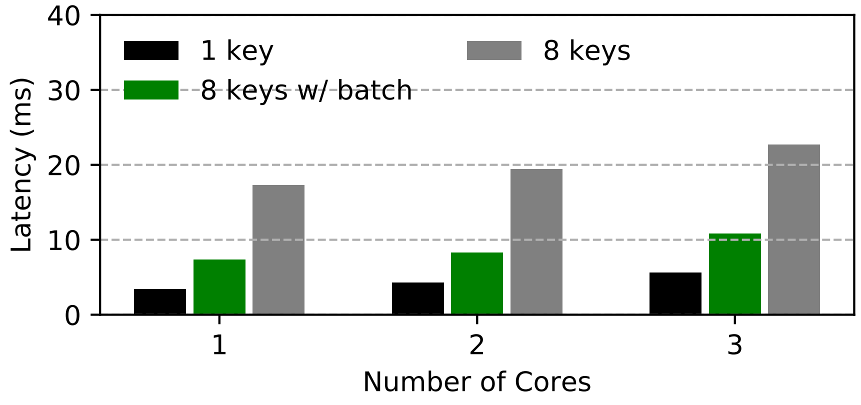
Figure 4.8: Average, 99th percentile, and maximum latency results when multiple keys are written.

number of keys (note the log scale in the figures). Achal with explicit batching performs slightly better than Achal without batching; similar application-side batching could also be done for BFT-SMaRt and Cassandra. In terms of the maximum latency (see Fig. 4.8c), which is the most relevant metric for real-time systems, BFT-SMaRt performs the worst among all systems. For 32 keys, in fact, it frequently timed out during the experiments.

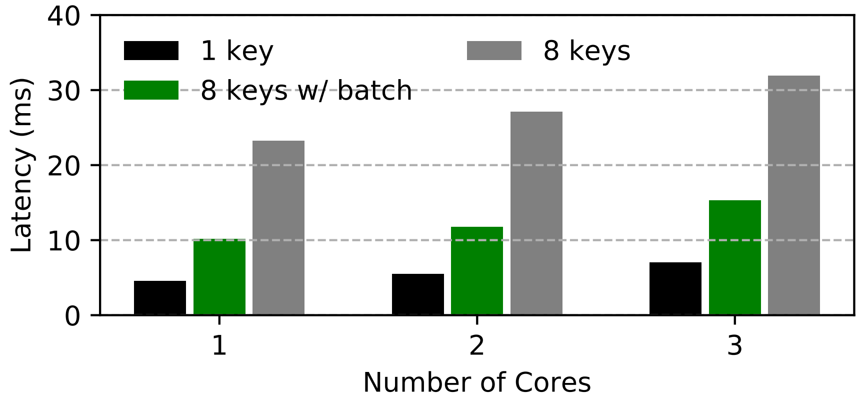
In the experiments discussed so far, only one instance of Achal was running on each host. To evaluate the overheads due to contention on the network or kernel resources by parallel instances of Achal, we also compared Achal's read and write latency in a multiprocessor scenario. Three separate instances of Achal and the application control loop were running on three cores of each host, whereas one core was left unoccupied to run the PTP clock synchronization protocol at high priority. The Achal tasks were released synchronously to emulate the worst-case scenario. The results (in terms of aggregate latency) from this experiment for a single key and eight keys are illustrated in Figs. 4.9a to 4.9c. Latency grows linearly with the number of cores, mainly due to the fact that the networking layer needs to deal with a proportionally increasing number of messages. In case of Achal without application-side batching, the *maximum* latency with eight keys almost doubles from around 20 ms to 40 ms, but is still below Cassandra and BFT-SMaRt's *average* latency with eight keys (as reported in Fig. 4.8a).

We also evaluated how the three systems react to faults. In particular, we are interested in crash faults since they reveal the limitations of leader-based protocols with regards to servicing time-sensitive queries. For this, we ran the three systems for about 500 iterations, with a time period of 100 ms each, and introduced a crash fault into one of the replicas around the 250th iteration. The resulting latency observations are illustrated in Fig. 4.10. As expected, Achal incurs no latency spikes at all upon a crash, due to relying on a leaderless BFT protocol. In fact, its latency decreases owing to the reduced number of messages transmitted per iteration. In contrast, both BFT-SMaRt and Cassandra experience extreme latency spikes when the crash is introduced, and an interval of high latency fluctuation persists for a few consecutive iterations, amounting to an unavailability interval of around one second. (the skipped iterations are indicated with zero latency). Since BFT-SMaRt uses a classical SMR design with primary and backups, this was expected. Surprisingly, Cassandra also evoked a similar result, which we attribute to its reliance on a centralized coordinator node.

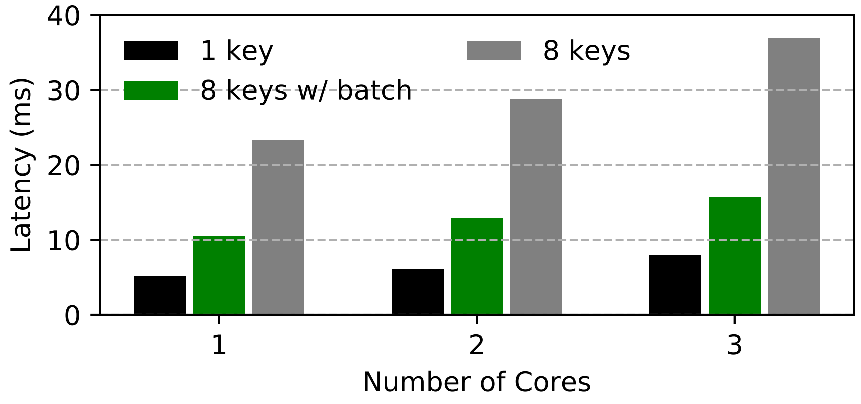
In summary, on embedded platforms with limited CPU, memory, and network resources, Achal is more efficient than the state-of-the-art systems BFT-SMaRt and Cassandra, which are primarily designed for server-scale machines. Achal's predictable latency helps in validating temporal constraints prior to deployment, and is helpful when



(a) Average latency vs. #cores



(b) 99th percentile latency vs. #cores



(c) Maxium latency vs. #cores

Figure 4.9: Average, 99th percentile, and maximum latency results for the multiprocessor scenario.

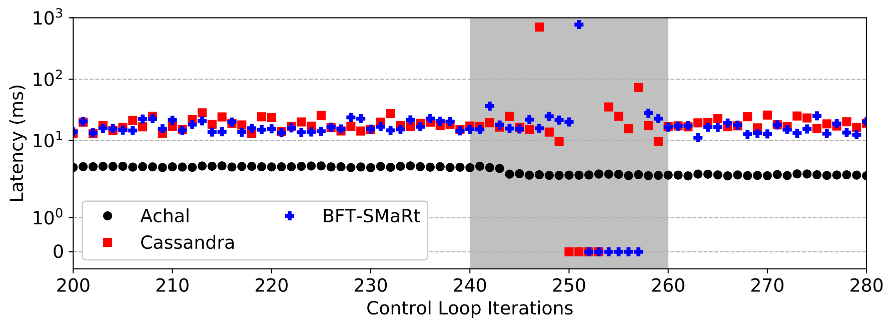


Figure 4.10: Latency distributions for Achal, BFT-SMaRt, and Cassandra, when a crash fault is introduced into one of the replicas around the 250th iteration (see the region colored gray). Iterations in BFT-SMaRt and Cassandra that were skipped completely due to the crash fault are indicated with zero latency.

targeting high-frequency applications with strict timing constraints. In the next chapter, we analyze the reliability of the hard real-time IC protocol implementation, which lies at the core of Achal.

5 RELIABILITY ANALYSIS OF A BFT PROTOCOL*

* This chapter is based on our RTAS 2020 [92] paper.

In the previous chapter, we presented a hard real-time design for an Interactive Consistency (IC) protocol, which can be easily implemented over COTS platforms, and which can tolerate environmentally-induced Byzantine errors as well. However, recall from Chapter 1 that our goal is to build ultra-reliable CPS, i.e., systems that are highly reliable with negligible failure rates and quantifiable reliability guarantees. Hence, we must also supplement the hard real-time IC protocol design from Section 4.2.2 with a corresponding reliability analysis. To this end, we present in this chapter a quantitative reliability analysis of an Ethernet-based implementation of the protocol.

5.1 PRIOR WORK AND RELIABILITY ANOMALIES

Although there is plenty of work on reliability analysis of distributed hard real-time systems, e.g., [39, 54, 198, 204], much of it primarily focuses on the analysis of low-level properties. For instance, Broster et al.'s analysis [39] upper-bounds the probability that any individual message is transmitted on time over CAN despite delays due to fault-induced retransmissions. Our objective is to leverage such fine-grained message-level analyses to evaluate the failure rate of a more complex, higher-level, and multi-round protocol.

In this regard, prior work has proposed logics to formally verify the correctness of round-based BFT protocols, e.g., [65]. However, these results, too, are orthogonal to our requirements since our objective is to provide quantifiable bounds on reliability (rather than a binary result), and also account for time-domain failures (rather than just value-domain failures).

Also, prior work on Byzantine fault tolerance is oblivious to non-uniform fault rates across different components of the system that arise in presence of transient faults due to environmental disturbances (e.g., classical Byzantine guarantees such as $3f + 1$ processes can tolerate up to f Byzantine faults is abstract from the underlying network topology). The presented analysis overcomes these limitations by considering timing delays in the correctness definitions, and by explicitly modeling PE nodes, network switches, and network links (including the network topology) while considering the possibility of non-uniform fault rates across these components.

Simulations and model checking are alternative techniques to solve the reliability analysis problem. However, these techniques suffer from

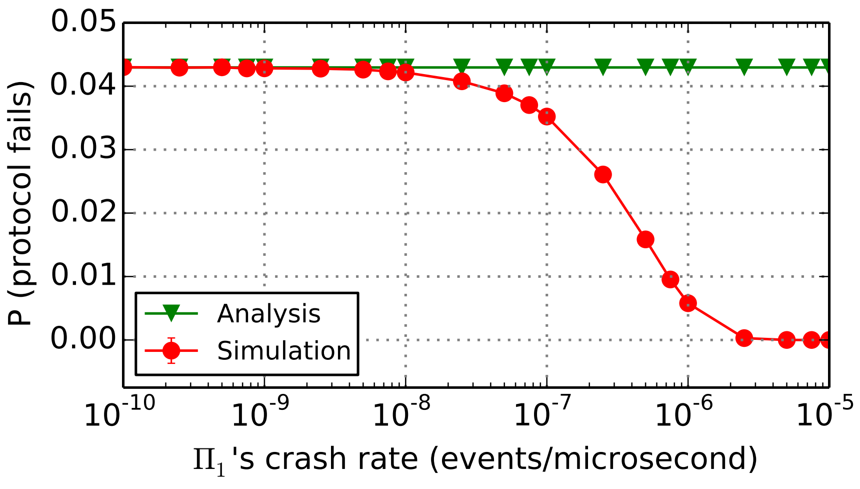


Figure 5.1: The failure probability decreases when Π_1's crash rate is increased from 10^{-10} *events/µs* to 10^{-5} *events/µs*, given a message corruption rate of 10^{-5} *events/µs*. The proposed analysis is designed to analytically account for such anomalies and, therefore, always bounds the worst-case failure probability.

scalability issues when the error probabilities are very small. In particular, simulations must be run for excessively long durations to estimate the failure rate with high confidence, and probabilistic model checkers such as PRISM [125] need to fall back on exact model checking to avoid incorrect results due to floating-point noise. For example, evaluating the reliability of even a very basic distributed system [126] using PRISM takes up to a few hours when exact representations are used (whereas otherwise, it takes only a few seconds).

Most importantly, though, prior work does not account for *reliability anomalies*, which can result in non-monotonic increases in a system's overall failure rate despite local decreases in a component's failure rate. For example, consider processes Π_1, Π_2, and Π_3 executing a two-round IC protocol (as explained in Section 4.2.1). Suppose that Π_1 is susceptible to crashes and message corruptions, which may occur at a maximum rate of 10^{-05} $events/\mu s$ each, whereas the other processes execute error-free. Simulation results (illustrated in Fig. 5.1) show that if Π_1 experiences crashes at a reduced (i.e., better) rate of only 10^{-10} $events/\mu s$, the protocol failure probability is actually higher than it is for the scenario in which the crashes occur at the peak rate. This counter-intuitive behavior occurs because crash-induced message omissions at Π_1 prevent the transmission of possibly faulty messages, thereby preventing other processes from making a wrong decision.

The presence of reliability anomalies poses a significant problem in practice because simulating errors at peak rates does not necessarily yield a safe upper bound on the overall failure rate and since it is infeasible to simulate or exhaustively evaluate all possible error rates. In contrast, as shown in Fig. 5.1, the analysis introduced in this chapter is *sound* despite such reliability anomalies, i.e., it reports the maximum possible failure probability without exhaustively evaluating all possible crash rates. In fact, to the best of our knowledge, this is the first work to formalize the concept of reliability anomalies, and to propose techniques to eliminate such anomalies in a hard real-time setting.

In the rest of this chapter, we start by giving an overview of the proposed analysis (Section 5.2); present a probabilistic analysis to upper-bound the failure probability of the IC protocol as a function of basic system error probabilities (Section 5.3); and then provide implementation-specific analyses to upper-bound the probability with which these basic system errors occur, which in turn helps upper-bound the implementation-specific failure rate of the IC protocol (Section 5.4). Finally, we report on a case study to evaluate the pessimism incurred by our analysis and to demonstrate its utility in identifying non-trivial and non-obvious reliability trade-offs (Section 5.5).

5.2 ANALYSIS OVERVIEW

Recall from Chapter 2 that we adopt the Failures-In-Time (FIT) rate—which is an industry-standard reliability metric denoting the number of failures expected in one billion device operating hours [214]—for measuring reliability in presence of transient faults. From its definition, 1 FIT implies that at most one IC protocol instance is expected to violate the correctness criterion in one billion operating hours. Hence, in a real-time context, since the maximum frequency at which the IC protocol is invoked is known in advance, the IC protocol's FIT rate can be derived simply by analyzing a single invocation of the protocol. In particular, to bound the IC protocol's FIT rate, it is sufficient to **(i)** derive an upper bound on the failure probability of a single invocation of the IC protocol, **(ii)** use this upper bound to compute a lower bound on the mean time to the first failed execution of the protocol, which is also known as its MTTF (see [124, Section 2.2] for a detailed discussion), and then **(iii)** derive an upper bound on the FIT rate as an inverse of the MTTF lower bound. Among these, steps (i) and (ii) can be trivially addressed. Our objective is thus to address the first step, i.e., the single-invocation failure probability problem, in a sound manner such that the analysis is free from reliability anomalies, and for a hard real-time implementation of the IC protocol (from Section 4.2.2).

In this regard, our analysis is split into two parts. In the first part (Section 5.3), we abstract the effect of basic errors, which were discussed in Section 3.3, into different types of *protocol-specific message errors*, such as crash-induced message omissions; exhaustively evaluate all scenarios in which one or more protocol-specific message errors result in a failed execution of the IC protocol; and present a reliability anomaly-free upper bound on the failure probability of a single invocation of the IC protocol. Since the first part of the analysis is implementation-oblivious, when deriving the upper bound, we make an unrealistic assumption that the exact probabilities with which different protocol-specific message errors occur are known in advance. In the second part (Section 5.4), we determine upper bounds on these exact probabilities for an Ethernet-based hard real-time implementation of the IC protocol, and use these bounds to obtain an implementation-specific upper bound on the IC protocol failure probability (and also its FIT). The second part is safe because we ensure that the proposed failure probability analysis is free from reliability anomalies.

5.3 PROBABILISTIC ANALYSIS

Recall the objective of the IC protocol from Section 4.2.1. In a distributed system consisting of N_p processes $\Pi = \{\Pi_1, \Pi_2, \ldots, \Pi_{N_p}\}$, if each process Π_i seeks to compute a vector V_i such that item $V_i[k]$ (for $1 \leqslant k \leqslant N_p$) corresponds to the private value of process Π_k: the objective is to ensure that $V_i[k] = V_j[k]$ for any two correct processes $\Pi_i, \Pi_j \in \Pi$, and if process Π_k is also correct, then $V_i[k] = V_j[k] = \nu_k$. However, the stated objective does not take into account any application semantics, and is therefore insufficient to determine if an erroneous execution of the protocol causes the application to fail.

For example, an embedded application may use the IC protocol to achieve *input consistency* over redundant sensor values, where the processes may *fuse* their respective decision vectors using a noise filtering function and forward the results to an actuator, which in turn may use a simple majority hardware for redundancy suppression. In this case, the application reliability depends on the IC protocol execution as well as on the fuse function used by the processes. In general, every application may rely on a different set of correctness criteria requiring a slightly different set of reliability analyses. We define below two correctness criteria that form the basis of our analysis.

5.3.1 Correctness Criteria

We consider a *strong* and a *weak* correctness criterion to define a failed execution of the IC protocol. In the first case, we assume that every process Π_i determines the *quorum majority* as its fuse function, which we denote as f_{quorum}. That is, $f_{\text{quorum}}(V_i)$ returns either the quorum majority over all values in V_i (in which case the returned value equals at least $\lfloor N_p/2 \rfloor + 1$ elements in V_i) or $\perp$ (if no such majority exists). With a focus on real-time applications, we also assume that $f_{\text{quorum}}(V_i) = \perp$ if Π_i fails to produce V_i on time. This is possible if environmentally-induced faults delay the execution of IC protocol steps (i.e., deadline misses in the hard real-time realization of the IC protocol, which was provided in Section 4.2.2). Suppose that at the end of an error-free execution of the IC protocol, $f_{quorum}(V_i) = f_{correct} \neq \bot$. Let Π be partitioned into the following three sets:

$$\begin{aligned} \mathcal{S}_{correct} &= \{\Pi_i \in \Pi \mid f_{quorum}(V_i) = f_{correct}\}, \\ \mathcal{S}_{skipped} &= \{\Pi_i \in \Pi \mid f_{quorum}(V_i) = \bot\}, \text{ and} \\ \mathcal{S}_{faulty} &= \{\Pi_i \in \Pi \mid f_{quorum}(V_i) \neq f_{correct} \wedge f_{quorum}(V_i) \neq \bot\}. \end{aligned}$$

The strong correctness criterion requires that $|\mathcal{S}_{correct}| \geqslant \lfloor N_p/2 \rfloor + 1$. This criterion resembles the guarantees offered by traditional BFT protocols for general-purpose systems.

In the second case, we assume that each Π_i uses *simple majority* as its fuse function, denoted as g_{simple}. The simple majority function $g_{simple}(V_i)$ breaks ties deterministically using process IDs, and returns $\bot$ only if all values in V_i are $\bot$ or if Π_i failed to produce vector V_i on time. Once again, suppose that $g_{simple}(V_i) = g_{correct} \neq \bot$ denotes the output at the end of an error-free execution of the IC protocol. Let Π be partitioned into the following three sets:

$$\begin{aligned} \mathcal{W}_{correct} &= \{\Pi_i \in \Pi \mid g_{simple}(V_i) = g_{correct}\}, \\ \mathcal{W}_{skipped} &= \{\Pi_i \in \Pi \mid g_{simple}(V_i) = \bot\}, \\ \text{and } \mathcal{W}_{faulty} &= \{\Pi_i \in \Pi \mid g_{simple}(V_i) \neq g_{correct} \wedge g_{simple}(V_i) \neq \bot\}. \end{aligned}$$

The weak correctness criterion requires that $|\mathcal{W}_{correct}| > |\mathcal{W}_{faulty}|$. It is particularly useful for embedded applications which are not concerned if redundant outputs are skipped, as long as at least one correct output is delivered on time. For example, the weak correctness criterion is ideal for the embedded application mentioned above that relies on a simple majority hardware for redundancy suppression.

5.3.2 Basic Errors to Message Errors

As part of our fault model, we introduced in Section 3.3 crash and incorrect computation (corruption) errors due to transient faults. In case of Ethernet-based distributed real-time systems, both hosts (on which the application processes are deployed) and network switches constitute the set of all PEs. Hence, we further classify the crash errors into *host crashes* and *switch crashes*. Similarly, we also further classify corruption errors into *host corruption* and *frame corruption* errors. In the following, we model the effect of these basic errors on successful transmission of IC protocol messages.

Recall the IC protocol from Section 4.2.1. Notice that in each round of the protocol, the processes exchange one or more nodes belonging to their respective EIG trees; and in the final round, they process the exchanged information locally to determine their respective decision vectors. Suppose that $M_{i,k}(\alpha)$ denotes the message sent by process Π_i

#	ERROR EVENT	REMARK
1	"round r msgs. omitted at source E_i"	Omission
2	"round r msgs. omitted at switch S_l"	
3	"round r msgs. omitted at dest. E_k"	
4	"round r frame from Π_i to Π_k omitted by NW"	
5	"round r msgs. corrupted at source E_i"	Corruption
6	"round r frame from Π_i to Π_k corrupted by NW"	

Table 5.1: Message error events due to transient faults.

to another process Π_k carrying information about the node labeled α. Since $|\alpha| = r - 1$ (Line 5), $M_{i,k}(\alpha)$ is one of the messages sent by process Π_i to another process Π_k during the $(r-1)^{st}$ round.

Each message $M_{i,k}(\alpha)$ can be affected by crash or corruption errors. In particular, $M_{i,k}(\alpha)$ can be omitted if its *sender* (i.e., the host on which process Π_i is deployed) crashes, if one of the switches through which the message is routed crashes, or if its *receiver* (i.e., the host on which process Π_k is deployed) crashes. $M_{i,k}(\alpha)$ can also be omitted (or rather explicitly dropped) if the Ethernet frame carrying the message is corrupted during transmission and if Ethernet's checksum mechanism successfully detects this corruption. Finally, $M_{i,k}(\alpha)$ can also be corrupted if it was incorrectly prepared in the first place due to corruptions on the sender side, if the Ethernet frame carrying that message is corrupted during transmission but the corruptions are not detected by Ethernet's checksum mechanism, or if it is affected by corruptions on the receiver side just before being delivered. We denote these events as *protocol-specific message errors*.

However, unlike transient faults and basic errors, which are mutually independent (recall our transient fault and basic error modeling from Sections 3.2 and 3.3, respectively), the protocol-specific message errors are not mutually independent. For example, all messages from Π_i to Π_k during round r (i.e., each $M_{i,k}(\alpha)$ with $|\alpha| = r - 1$) are typically batched together into a single Ethernet frame; hence, they are simultaneously dropped if the frame gets corrupted and if the corruption is successfully detected. Similarly, if the common payload that is carried by all message frames originating from Π_i during round r is corrupted during preparation, even before its checksum has been computed, the corruptions go undetected and are passed on to every message containing the payload.

Hence, for the purpose of this analysis, we model error events that are defined at a coarser granularity in terms of sets of dependent messages (at the cost of slight pessimism). These are summarized in Table 5.1. Events 1, 2, and 3 denote message omissions due to host and switch crashes during the r^{th} round's sending step of the IC protocol.

Events 4 and 6 denote frame omissions and frame corruptions due to perceptible and imperceptible corruption during transmission, respectively. Event 5 denotes the message corruption due to corruption on the host. Unlike omission errors, we do not consider corruption errors at destinations since these are implicitly accounted for as corruption errors at the source of subsequently sent messages.

5.3.3 Message Errors to Protocol Failure

By exhaustively enumerating all possible cases based on whether each protocol-specific message error event (belonging to one of the six types listed in Table 5.1) occurs or does not occur, the overall IC protocol failure probability can be derived. Suppose that the respective event probabilities, which are required for such an exhaustive case analysis, are known in advance. That is, for each error event x, suppose that the exact probability $P(x)$ with which the event occurs is known in advance (we relax this assumption in Section 5.3.4). In this section, we propose a recursive analysis that computes the IC protocol failure probability $P($IC failure$)$ as a function of each $P(x)$, while taking into account all relevant scenarios, and while ensuring soundness in presence of reliability anomalies.

In particular, although we modeled in a coarse-grained fashion only six different types of protocol-specific message errors (Table 5.1), considering all rounds, PEs, switches, and message frames, altogether tens of errors events need to be evaluated. Furthermore, the total number of cases is exponential in the number of error events. Hence, for efficiency, we identify and prune certain scenarios that are not possible in practice, without compromising the analysis accuracy and safety. For example, if a message is omitted at its source, it cannot be omitted by the network. Therefore, the scenario corresponding to the omission of a message at source and also by the network can be safely excluded from the exhaustive enumeration. Similarly, corruption of a message that is eventually omitted is irrelevant in practice. Therefore, scenarios where an omitted message is corrupted and not corrupted can be merged. The analysis is explained in detail below.

RECURSIVE ANALYSIS OVERVIEW The analysis pseudocode is provided in (and split across) Algorithms 5.1 and 5.2. Let $\mathcal{M}$ denote the set of all messages exchanged between the processes in an error-free scenario, i.e., $\mathcal{M} = \bigcup_{\Pi_i \in \Pi, 1 \leqslant r \leqslant N_r} M^r_{i,*}$. To evaluate the probability of a failed protocol instance, we perform a recursive case analysis over all possible error combinations for each message in $\mathcal{M}$.

We choose one message at a time from $\mathcal{M}$, consider all scenarios in which this message may be affected by the errors listed in Table 5.1, assign case probabilities for each of these scenarios, and recursively evaluate the error possibilities for the next message in $\mathcal{M}$. The recursion terminates when all messages in $\mathcal{M}$ (and hence all possible cases) have been accounted for. We consider messages from round one first, followed by messages from round two, and so on, because message errors during an earlier round may impact message transmissions during subsequent rounds. For example, if it can be determined from the protocol structure that omission of a first round message $M_{i,k}(\alpha)$ guarantees the omission of a second round message $M_{k,l}(\beta)$, the recursive steps that deal with the analysis of whether $M_{k,l}(\beta)$ is omitted or corrupted can be ignored for all cases where message $M_{i,k}(\alpha)$ is omitted in the first place. The ordering of messages from the same round is arbitrary since there is no causal relationship among message errors in the same round.

The analysis maintains the following message sets for bookkeeping. Message set $\mathcal{U}$ is initialized to $\mathcal{M}$. Messages are removed from $\mathcal{U}$ and analyzed one at a time. Every message that is omitted is inserted into message set $\mathcal{O}$. Similarly, every message that is not omitted (and hence delivered on time) but incorrectly computed is inserted into message set $\mathcal{C}$. If a message is neither omitted nor corrupted, it is still removed from $\mathcal{U}$ but inserted into message set $\mathcal{P}$, denoting that it is in pristine condition. Sets $\mathcal{O}$, $\mathcal{C}$, and $\mathcal{P}$ are eventually used during the terminating step of the recursion. We also maintain an event log $\mathcal{E}$ to keep track of the message error events that have already been accounted for in the earlier stages of the recursion, and that must not be accounted for again. Sets $\mathcal{O}$, $\mathcal{C}$, $\mathcal{P}$, and event log $\mathcal{E}$ are initially empty (Line 2).

RECURSIVE CASES The probability that an IC protocol instance fails is denoted $P(\text{IC failure})$ and computed recursively by the function ProbAnalysisRec (Line 4). First, we obtain a message from $\mathcal{U}$ using GetEarliestMessage (Line 7), which returns messages from round one, followed by messages from round two, and so on. Let $M_{i,k}(\alpha)$ denote this message. Suppose that it belongs to round r, i.e., $|\alpha| = r-1$. Probabilities P_{fail} and P_{prefix}, which keep track of the cumulative failure probability and the case probability prefix (explained below), are then initialized to zero and one, respectively (Line 9).

Based on the error events in Table 5.1, we consider six cases in which $M_{i,k}(\alpha)$ is affected by errors and one case in which $M_{i,k}(\alpha)$ is transmitted error-free. Case 1 implies that $M_{i,k}(\alpha)$ experienced an error of type 1. Case 2 implies that $M_{i,k}(\alpha)$ did not experience an error of type 1, but experienced an error of type 2, Case 3 implies that $M_{i,k}(\alpha)$ did not experience errors of type 1 and 2, but experienced an error of type 3, and so on. In other words, our analysis explicitly ignores all scenarios that do not adhere to this rule. As a result, all omission errors (event types 1–4) are analyzed first, which is sound since message corruption probabilities contribute to the failure probability only if the message is not omitted. Similarly, an omission at the source (event

Algorithm 5.1 Recursive analysis to estimate the failure probability of an IC protocol execution. Procedures OmissionCases, CorruptionCases, and ErrorFreeCase are defined in Algorithm 5.2.

```
procedure ProbAnalysisInit
    $P(\text{IC failure}) \leftarrow \text{ProbAnalysisRec}(\mathcal{M}, \emptyset, \emptyset, \emptyset, \emptyset)$

procedure ProbAnalysisRec($\mathcal{U}, \mathcal{O}, \mathcal{C}, \mathcal{P}, \mathcal{E}$)
    if $\mathcal{U} = \emptyset$ then return $P(\text{IC failure} \mid \mathcal{O}, \mathcal{C}, \mathcal{P})$        ▷ termination case
    ▷ get the message to be analyzed
    $M_{i,k}(\alpha) \leftarrow \text{GetEarliestMessage}(\mathcal{U})$
    $r \leftarrow |\alpha| + 1$                                ▷ compute the IC protocol round
    $P_{\text{fail}} \leftarrow 0$ , $P_{\text{prefix}} \leftarrow 1$                     ▷ initialize probabilities

    $X \leftarrow \langle\rangle$                           ▷ an empty FIFO-ordered sequence
    ▷ Case 1 event string
    X.enqueue("round r msgs. omitted at source $E_i$")
    ▷ Case 2 event strings
    for all $S_l \in \text{route}_{i,k}$ do
        X.enqueue("round r msgs. omitted at switch $S_l$")
    ▷ Case 3 and Case 4 event strings
    X.enqueue("round r frame from $\Pi_i$ to $\Pi_k$ omitted by NW")
    X.enqueue("round r msgs. omitted at dest. $E_i$")
    $P_{\text{fail}}, P_{\text{prefix}}, \mathcal{E} \leftarrow \text{OmissionCases}(\mathcal{U}, \mathcal{O}, \mathcal{C}, \mathcal{P}, \mathcal{E}, X, P_{\text{fail}}, P_{\text{prefix}})$

    $X \leftarrow \langle\rangle$                           ▷ an empty FIFO-ordered sequence
    ▷ Case 5 event string
    X.enqueue("round r msgs. corrupted at source $E_i$")
    ▷ Case 6 event string
    X.enqueue("round r frame from $\Pi_i$ to $\Pi_k$ corrupted by NW")

    $P_{\text{fail}}, P_{\text{prefix}}, \mathcal{E} \leftarrow \text{CorruptionCases}(\mathcal{U}, \mathcal{O}, \mathcal{C}, \mathcal{P}, \mathcal{E}, X, P_{\text{fail}}, P_{\text{prefix}})$

    $P_{\text{fail}} \leftarrow \text{ErrorFreeCase}(\mathcal{U}, \mathcal{O}, \mathcal{C}, \mathcal{P}, \mathcal{E}, P_{\text{fail}}, P_{\text{prefix}})$        ▷ Case 7
    return $P_{\text{fail}}$
```

type 1) is considered first since that determines whether the message is even exposed to omissions by the network (event type 3).

Finally, the case that corresponds to an error-free transmission of message $M_{i,k}(\alpha)$ is evaluated last.

CASES 1–4 These cases are evaluated by calling the OmissionCases procedure (Line 20). Since an error event may affect multiple messages, it is possible that an error event that might affect $M_{i,k}(\alpha)$ has already been accounted for while analyzing another message in an earlier recursion stage. Thus, each case is evaluated only if the corresponding error event has not already been evaluated before, in which case, it is

Algorithm 5.2 Probabilistic analysis of an IC protocol instance.

```
31: procedure OmissionCases($\mathcal{U}, \mathcal{O}, \mathcal{C}, \mathcal{P}, \mathcal{E}, X, P_{fail}, P_{prefix}$)
32:     while X is not empty do
33:         $x \leftarrow X.dequeue()$
34:         if $x \notin \mathcal{E}$ then                    ▷ analyze event x if not analyzed before
35:             $\mathcal{E} \leftarrow \mathcal{E} \cup \{x\}$              ▷ update $\mathcal{E}$ to prevent repeated analysis
36:             $P_{case} \leftarrow P_{prefix} \times P(x)$              ▷ compute case probability
37:             $P_{prefix} \leftarrow P_{prefix} \times \overline{P(x)}$     ▷ update prefix for subsequent cases
38:             ▷ compute dependent messages
39:             $\mathcal{S}_o \leftarrow$ OmittedMessagesGiven($x$)
40:             ▷ compute conditional probability using the recursive call
41:             $P_{cond} \leftarrow$ ProbAnalysisRec($\mathcal{U} \setminus \mathcal{S}_o, \mathcal{O} \cup \mathcal{S}_o, \mathcal{C}, \mathcal{P}, \mathcal{E}$)
42:             $P_{fail} \leftarrow P_{fail} + P_{case} \times P_{cond}$        ▷ update failure probability
43:     return $P_{fail}, P_{prefix}, \mathcal{E}$     ▷ return params needed in the subsequent cases
44:
45: procedure CorruptionCases($\mathcal{U}, \mathcal{O}, \mathcal{C}, \mathcal{P}, \mathcal{E}, X, P_{fail}, P_{prefix}$)
46:     ▷ similar to the while loop in OmissionCases, except Line 53
47:     while X is not empty do
48:         $x \leftarrow X.dequeue()$
49:         if $x \notin \mathcal{E}$ then
50:             $\mathcal{E} \leftarrow \mathcal{E} \cup \{x\}$
51:             $P_{case} \leftarrow P_{prefix} \times P(x)$
52:             $P_{prefix} \leftarrow P_{prefix} \times \overline{P(x)}$
53:             $\mathcal{S}_c \leftarrow \{M_{i,k}(\alpha)\}$
54:             $P_{cond} \leftarrow$ ProbAnalysisRec($\mathcal{U} \setminus \mathcal{S}_c, \mathcal{O}, \mathcal{C} \cup \mathcal{S}_c, \mathcal{P}, \mathcal{E}$)
55:             $P_{fail} \leftarrow P_{fail} + P_{case} \times P_{cond}$
56:     return $P_{fail}, P_{prefix}, \mathcal{E}$
57:
58: procedure ErrorFreeCase($\mathcal{U}, \mathcal{O}, \mathcal{C}, \mathcal{P}, \mathcal{E}, P_{fail}, P_{prefix}$)
59:     $\mathcal{S} \leftarrow \{M_{i,k}(\alpha)\}$
60:     $P_{cond} \leftarrow$ ProbAnalysisRec($\mathcal{U} \setminus \mathcal{S}, \mathcal{O}, \mathcal{C}, \mathcal{P} \cup \mathcal{S}, \mathcal{E}$)
61:     $P_{case} \leftarrow P_{prefix}$
62:     $P_{fail} \leftarrow P_{fail} + P_{case} \times P_{cond}$
63:     return $P_{fail}$
```

not in the event log $\mathcal{E}$ (Line 34). If the event is indeed being evaluated for the first time, it is first inserted into $\mathcal{E}$ (Line 35). The case analysis is then executed as follows. First, the case probability is computed as the product of the probability that prior cases do not occur (given by the latest value of P_{prefix}) and probability $P(x)$ with which the analyzed case occurs (Line 36). Probability P_{prefix} is then updated to account for the negation of the analyzed case, so that it can be reused during the analysis of subsequent cases (Line 37). All messages in M that are omitted either directly or indirectly due to X are computed as $\mathcal{S}_o =$ OmittedMessagesGiven(X) (Line 39). The conditional failure probability is then computed using a recursive call to ProbAnalysis-

REC with the updated values of $\mathcal{U}$ and $\mathcal{O}$, where the set of omitted messages $\mathcal{S}_o$ is excluded from $\mathcal{U}$ and added to $\mathcal{O}$ (Line 41). In the end, the conditional probability is multiplied with the case probability, and added to the cumulative failure probability (Line 42).

CASES 5–7 These cases are evaluated by calling the CORRUPTION-CASES procedure (Line 27), and their analysis is similar to the analysis of Cases 1–4 except for the computation of the conditional failure probability. That is, unlike Cases 1–4, the corrupted message $M_{i,k}(\alpha)$ is removed from $\mathcal{U}$ and added to $\mathcal{C}$ while invoking the recursive call to PROBANALYSISREC (Line 54). The last case corresponds to the scenario where $M_{i,k}(\alpha)$ is transmitted error-free (Line 29). In this case, $M_{i,k}(\alpha)$ is removed from $\mathcal{U}$ and inserted into set $\mathcal{P}$ that consists of all pristine messages (Line 60). Unlike Cases 1–6, the case probability for the last case is simply the probability that Cases 1–6 do not occur, given by the latest value of P_{prefix} (Line 61).

TERMINATING CASE The recursion terminates when $\mathcal{U}$ is empty, since each message has been assigned to either $\mathcal{O}$, $\mathcal{C}$, or $\mathcal{P}$ based on whether it is affected by any fault-induced error in this case. What remains is a computation of the conditional probability given $\mathcal{O}$, $\mathcal{C}$, and $\mathcal{P}$ that the IC protocol instance fails. We denote this conditional probability as $P(\text{IC failure} \mid \mathcal{O}, \mathcal{C}, \mathcal{P})$ (Line 5).

Since it is impossible to estimate $P(\text{IC failure} \mid \mathcal{O}, \mathcal{C}, \mathcal{P})$ without knowing the exact contents of the corrupted messages, we derive an upper bound on it through worst-case analysis. In a nutshell, since all messages in $\mathcal{M}$ are already partitioned into sets $\mathcal{O}$, $\mathcal{C}$, and $\mathcal{P}$, we can deterministically apply the reduction procedure in the IC protocol to these messages and map the conditional failure probability for the termination case to either zero or one. We assume as a worst-case scenario that all faulty messages are identically corrupted.

LAST MILE ERRORS A protocol instance may also fail if, at the last moment, say, just after the reduction step, the decision vectors are corrupted or the host crashes. Since the proposed analysis is based on the analysis of message errors, to account for such *last-mile errors*, we use dummy messages that are sent back to the same host. To avoid clutter, Algorithm 5.1 does not discuss dummy messages; it can be updated as follows. **(i)** The dummy messages are denoted using our regular notation $M_{i,k}(\alpha)$, but with $i = k$ (the value of α is irrelevant for these dummy messages); **(ii)** they are incorporated into the recursive analysis by adding them to message set $\mathcal{M}$ during initialization; **(iii)** function GETEARLIESTMESSAGE (Line 7) is modified to return one of these dummy messages only if all other regular messages have been analysed; and finally, **(iv)** cases 4 and 6 corresponding to network

LABEL	ERROR EVENT x IN LINE 33	$P(x)$	P_{cond}
X_1	"round r msgs. omitted at source E_i"	P_1	C_1
X_2	"round r msgs. omitted at switch S_l"	P_2	C_2
X_3	"round r msgs. omitted at dest. E_k"	P_3	C_3
X_4	"round r frame from Π_i to Π_k omitted by NW"	P_4	C_4
X_5	"round r msgs. corrupted at source E_i"	P_5	C_5
X_6	"round r frame from Π_i to Π_k corrupted by NW"	P_6	C_6
-	"msg. $M_{i,k}(\alpha)$ is transmitted error-free"	-	C_7

Table 5.2: Shorthand notation for the exact message error probabilities and intermediate conditional failure probabilities used in Algorithms 5.1 and 5.2. $M_{i,k}(\alpha)$ is assumed to be routed through a single switch S_l. C_1–C_4 refer to P_{cond} at Line 41, C_5 and C_6 refer to P_{cond} at Line 54, and C_7 refers to P_{cond} at Line 60.

errors are not applied to the dummy messages (since these messages are local to each host).

5.3.4 Reliability Anomalies

Algorithms 5.1 and 5.2 define a recursive procedure to compute $P(\text{IC failure})$ as a function of the exact message error probabilities defined in Section 5.3.2. However, P_{fail} returned at the end of function ProbAnalysisRecSM$(\mathcal{U}, \mathcal{O}, \mathcal{C}, \mathcal{P}, \mathcal{E})$ may not be monotonically increasing in all exact probabilities (as evident from the use of $\overline{P(x)}$ in Lines 37 and 52). As a result, probability $P(\text{IC failure})$, which is computed by invoking this recursive function, is not monotonic in all exact probabilities. In presence of such anomalies, $P(\text{IC failure})$ cannot be safely upper-bounded by simply replacing the exact message error probabilities with their respective upper bounds. We thus derive non-negative correction terms that are added to the analysis to mask such anomalies.

For brevity, we first introduce a shorthand notation (see Table 5.2) to denote the exact error probabilities and the conditional failure probabilities used in Algorithms 5.1 and 5.2. For each P_i in Table 5.2, we let $\overline{P_i} = 1 - P_i$. Note that the shorthand notation is defined with respect to the specific iteration of the recursive analysis, i.e., pertaining to the analysis of message $M_{i,k}(\alpha)$ specifically.

Also, we assume in this section that message $M_{i,k}(\alpha)$ is routed through a single switch S_l. The results can be trivially extended to more general cases, as discussed in the end.

Using the shorthand notation, P_{fail} returned at the end of function PROBANALYSISRECSM($\mathcal{U}, \mathcal{O}, \mathcal{C}, \mathcal{P}, \mathcal{E}$) is defined as follows. If the condition $x \in \mathcal{E}$ (in Line 34) evaluates to *false* for each $x \in X$, then

$$P_{\text{fail}} = \left(\begin{array}{l} P_1 \times C_1 \\ + \overline{P_1} \times P_2 \times C_2 \\ + \overline{P_1} \times \overline{P_2} \times P_3 \times C_3 \\ + \overline{P_1} \times \overline{P_2} \times \overline{P_3} \times P_4 \times C_4 \\ + \overline{P_1} \times \overline{P_2} \times \overline{P_3} \times \overline{P_4} \times P_5 \times C_5 \\ + \overline{P_1} \times \overline{P_2} \times \overline{P_3} \times \overline{P_4} \times \overline{P_5} \times P_6 \times C_6 \\ + \overline{P_1} \times \overline{P_2} \times \overline{P_3} \times \overline{P_4} \times \overline{P_5} \times \overline{P_6} \times C_7 \end{array} \right). \tag{5.1}$$

In case an event in X has already been analyzed during an earlier stage of the recursion, the corresponding case analysis is skipped, since $x \in \mathcal{E}$ would evaluate to *true* (see Line 34). In this case, if, say, event X_1 = "round r msgs. omitted at source E_i" has already been analyzed, P_{fail} for this recursion step is defined by setting probabilities P_1 and C_1 to zero in Eq. (5.1).

Clearly, it is not apparent from Eq. (5.1) if P_{fail} is monotonic in all P_i's, since P_{fail} relies on complementary probability terms $\overline{P_i}$'s. Thus, as a first step, we express P_{fail} in a canonical form consisting of only P_i's, i.e., where all $\overline{P_i}$'s in Eq. (5.1) are replaced with $1 - P_i$, as follows:

$$\begin{aligned} &P_{\text{fail}} = T_1 + T_2 + T_3 + T_4 + T_5 + T_6 + T_7, \text{ where} \\ &T_1 = C_7, \\ &T_2 = \sum_{i=1}^{6} P_i(C_i - C_7), \\ &T_3 = -\sum_{i=1}^{5} \sum_{j=i+1}^{6} P_i P_j (C_j - C_7), \\ &T_4 = \sum_{i=1}^{4} \sum_{j=i+1}^{5} \sum_{k=j+1}^{6} P_i P_j P_k (C_k - C_7), \\ &T_5 = -\sum_{i=1}^{3} \sum_{j=i+1}^{4} \sum_{k=j+1}^{5} \sum_{l=k+1}^{6} P_i P_j P_k P_l (C_l - C_7), \\ &T_6 = \sum_{i=1}^{2} \sum_{j=i+1}^{3} \sum_{k=j+1}^{4} \sum_{l=k+1}^{5} \sum_{m=l+1}^{6} P_i P_j P_k P_l P_m (C_m - C_7), \\ &T_7 = -P_1 P_2 P_3 P_4 P_5 P_6 (C_6 - C_7). \end{aligned} \tag{5.2}$$

In Eq. (5.2), P_{fail}'s monotonicity in all P_i's depends on the relation between each C_i (for $i \in \{1, 2, \ldots, 6\}$) and C_7. However, this relationship cannot be determined in advance. Even though C_7 corresponds to the conditional failure probability in an error-free scenario whereas each

i	$T_{i,pos}$
1	C_7
2	$\sum_{i=1}^{6} P_i B_i \lvert C_i - C_7 \rvert$
3	$\sum_{i=1}^{5} \sum_{j=i+1}^{6} P_i P_j (1 - B_j)(\lvert C_j - C_7 \rvert)$
4	$\sum_{i=1}^{4} \sum_{j=i+1}^{5} \sum_{k=j+1}^{6} P_i P_j P_k B_k (\lvert C_k - C_7 \rvert)$
5	$\sum_{i=1}^{3} \sum_{j=i+1}^{4} \sum_{k=j+1}^{5} \sum_{l=k+1}^{6} P_i P_j P_k P_l (1 - B_l)(\lvert C_l - C_7 \rvert)$
6	$\sum_{i=1}^{2} \sum_{j=i+1}^{3} \sum_{k=j+1}^{4} \sum_{l=k+1}^{5} \sum_{m=l+1}^{6} P_i P_j P_k P_l P_m B_m (\lvert C_m - C_7 \rvert)$
7	$P_1 P_2 P_3 P_4 P_5 P_6 (1 - B_6)(\lvert C_6 - C_7 \rvert)$

Table 5.3: Definition of each $T_{i,pos}$ used in Eq. (5.5).

C_i corresponds to a conditional failure probability in an error scenario, C_i can be smaller than C_7 because of the anomaly that omission errors can sometimes reduce the failure chances. Instead, for each C_i, we rely on a boolean value B_i that can be evaluated at analysis runtime to denote whether $C_i \geqslant C_7$, based on the following definition.

$$B_i = \begin{cases} 1 & \text{if } C_i \geqslant C_7 \\ 0 & \text{otherwise} \end{cases} \tag{5.3}$$

Using Eq. (5.3), we can rewrite each term $C_i - C_7$ as

$$C_i - C_7 = B_i \cdot \lvert C_i - C_7 \rvert - (1 - B_i) \cdot \lvert C_i - C_7 \rvert. \tag{5.4}$$

Next, using Eq. (5.4), and relying on the fact that all probabilities (i.e., each P_i and C_i) are positive, we split P_{fail} defined in Eq. (5.2) into two terms $P_{fail,pos}$ and $P_{fail,neg}$, such that $P_{fail,pos}$ is guaranteed to be non-negative and $P_{fail,neg}$ is guaranteed to be non-positive. That is,

$$\begin{aligned} &P_{fail} = P_{fail,pos} + P_{fail,neg}, \text{ where} \\ &P_{fail,pos} = \sum_{i=1}^{7} T_{i,pos}, \quad P_{fail,neg} = \sum_{i=1}^{7} T_{i,neg}, \\ &\text{and } T_{i,neg}, T_{i,neg} \text{ are defined as in Tables 5.3 and 5.4.} \end{aligned} \tag{5.5}$$

Given Eq. (5.5), it is trivial to come up with an over-estimation of P_{fail}

i	$T_{i,neg}$
1	0
2	$-\sum_{i=1}^{6} P_i(1-B_i)(\lvert C_i - C_7\rvert)$
3	$-\sum_{i=1}^{5}\sum_{j=i+1}^{6} P_iP_jB_j(\lvert C_j - C_7\rvert)$
4	$-\sum_{i=1}^{4}\sum_{j=i+1}^{5}\sum_{k=j+1}^{6} P_iP_jP_k(1-B_k)(\lvert C_k - C_7\rvert)$
5	$-\sum_{i=1}^{3}\sum_{j=i+1}^{4}\sum_{k=j+1}^{5}\sum_{l=k+1}^{6} P_iP_jP_kP_lB_l(\lvert C_l - C_7\rvert)$
6	$-\sum_{i=1}^{2}\sum_{j=i+1}^{3}\sum_{k=j+1}^{4}\sum_{l=k+1}^{5}\sum_{m=l+1}^{6} P_iP_jP_kP_lP_m(1-B_m)(\lvert C_m - C_7\rvert)$
7	$-P_1P_2P_3P_4P_5P_6B_6(\lvert C_6 - C_7\rvert)$

Table 5.4: Definition of each $T_{i,neg}$ used in Eq. (5.5).

(since under-approximation is unsafe) that is also monotonic in each P_i by negating all the negative terms, as defined below:

$$P_{\text{fail-mono}} = P_{\text{fail}} - \sum_{i=1}^{7} T_{i,neg} = \sum_{i=1}^{7} T_{i,pos}. \tag{5.6}$$

The aforementioned procedure can be generalized to any number of switches. In particular, with every extra switch in the route from Π_i to Π_k (recall that $M_{i,k}(\alpha)$ is the message being analyzed), we need to deal with one extra error probability term, and thus the definition of

P_{fail} in Eq. (5.2) would be updated accordingly. For example, with one additional switch, P_{fail} would be defined as follows:

$$\begin{aligned}
&P_{\text{fail}} = T_1 + T_2 + T_3 + T_4 + T_5 + T_6 + T_7 + T_8, \qquad (5.7)\\
&\text{where } \ T_1 = C_8,\\
&T_2 = \sum_{i=1}^{7} P_i(C_i - C_8),\\
&T_3 = -\sum_{i=1}^{6}\sum_{j=i+1}^{7} P_iP_j(C_j - C_8),\\
&T_4 = \sum_{i=1}^{5}\sum_{j=i+1}^{6}\sum_{k=j+1}^{7} P_iP_jP_k(C_k - C_8),\\
&T_5 = -\sum_{i=1}^{4}\sum_{j=i+1}^{5}\sum_{k=j+1}^{6}\sum_{l=k+1}^{7} P_iP_jP_kP_l(C_l - C_8),\\
&T_6 = \sum_{i=1}^{3}\sum_{j=i+1}^{4}\sum_{k=j+1}^{5}\sum_{l=k+1}^{6}\sum_{m=l+1}^{7} P_iP_jP_kP_lP_m(C_m - C_8),\\
&T_7 = -\sum_{i=1}^{2}\sum_{j=i+1}^{3}\sum_{k=j+1}^{4}\sum_{l=k+1}^{5}\sum_{m=l+1}^{6}\sum_{n=m+1}^{7} \begin{pmatrix} P_iP_jP_kP_lP_mP_n \\ (C_m - C_8) \end{pmatrix},\\
&\text{and } \ T_8 = P_1P_2P_3P_4P_5P_6P_7(C_7 - C_8).
\end{aligned}$$

This concludes the first part of the reliability analysis. In our evaluation (Section 5.5), we show that the correction terms added to ensure monotonicity have a very small impact on the overall failure probability, in the sense that they do not result in appreciable pessimism. Next, we use the proposed analysis to upper-bound the overall failure rate of the IC protocol, given a specific hard real-time implementation.

5.4 ANALYSIS INSTANTIATION

The recursive analysis in Section 5.3 relies on *exact* protocol-specific message error probabilities, which are unknown in practice. In the second part of the reliability analysis, we instantiate the recursive analysis with implementation-specific upper bounds on these exact probabilities, which is safe since we have explicitly addressed all reliability anomalies. For the instantiation, we rely on the hard real-time design of the IC protocol (which was presented in Section 4.2.2) and on the probabilistic modeling of basic errors (which was presented in Section 3.3.3). We start by summarizing all notations and assumptions from the previous sections that are relevant to the following analysis.

In Section 4.2.2, we mapped the IC protocol steps to a periodic task model. We realized the N_r sending steps using tasks $T_s^1, T_s^2, \ldots, T_s^{N_r}$,

NOTATION	REMARK
$\mathcal{P}(x, \delta, \gamma_{crash}(E_i))$	PMF for crash errors on host E_i
$\mathcal{P}(x, \delta, \gamma_{crash}(S_l))$	PMF for crash errors on switch S_l
$\mathcal{P}(x, \delta, \gamma_{corrupt}(E_i))$	PMF for corruption errors on host E_i
$\mathcal{P}(x, \delta, \gamma_{corrupt}(S_l))$	PMF for corruption errors on switch S_l
$\mathcal{P}(x, \delta, \gamma_{corrupt}(L_k))$	PMF for corruption errors on network link L_k

Table 5.5: Probability Mass Function (PMF) for different types of basic errors.

the N_r state transition steps using tasks $T_t^1, T_t^2, \ldots, T_t^{N_r}$, and the reduction step using task T_r. To ensure that these tasks are activated in the order required by the IC protocol, they were assigned appropriate release offsets. The release offsets were defined as a function of the network latency bound Δ_{NW} (which denotes the worst-case latency for the exchange of IC protocol messages over the network) and response-time bounds R_s^r and R_t^r (which denote the global worst-case response time of each task T_s^r and T_t^r, respectively). Both these bounds can be easily derived using network- and host-specific schedulability analyses. Thus, the hard real-time implementation, besides ensuring timeliness, ensures that tasks always execute in deterministic intervals, which helps us upper-bound different message error probabilities.

To quantify the probability of non-zero crash or corruption events in any given interval of time, we proposed in Section 3.3.3 a Poisson-based arrival model for the basic errors. As per the model, the probability that x instances of any basic error type *err* affect any component *comp* in any interval of length δ, given the peak error rate $\gamma_{err}(\text{comp})$, is denoted $\mathcal{P}(x, \delta, \gamma_{err}(\text{comp}))$. In the case of an Ethernet-based system, which we analyze in this chapter, the crash and corruption error probabilities on each host E_i, switch S_l, and network link L_k are modeled using a similar notation (see Table 5.5).

The proposed implementation-specific upper bounds also relies on the following set of assumptions regarding crash and corruption errors. 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 of each PE E_i and switch S_i is upper-bounded by $\Delta_{reboot}(E_i)$ and $\Delta_{reboot}(S_i)$, respectively, and that any messages queued in a switch are lost upon a crash. Regarding corruption errors, we assume that process states are checked at least once between consecutive activations of the protocol. Thus, an IC protocol task cannot be affected by memory corruptions that occur prior to the end of the previous protocol instance. Finally, as also mentioned in Section 3.3.3, we consider all basic errors as independent events based on their stochastic nature (recall that we focus exclusively on basic errors due to environmentally-induced transient faults).

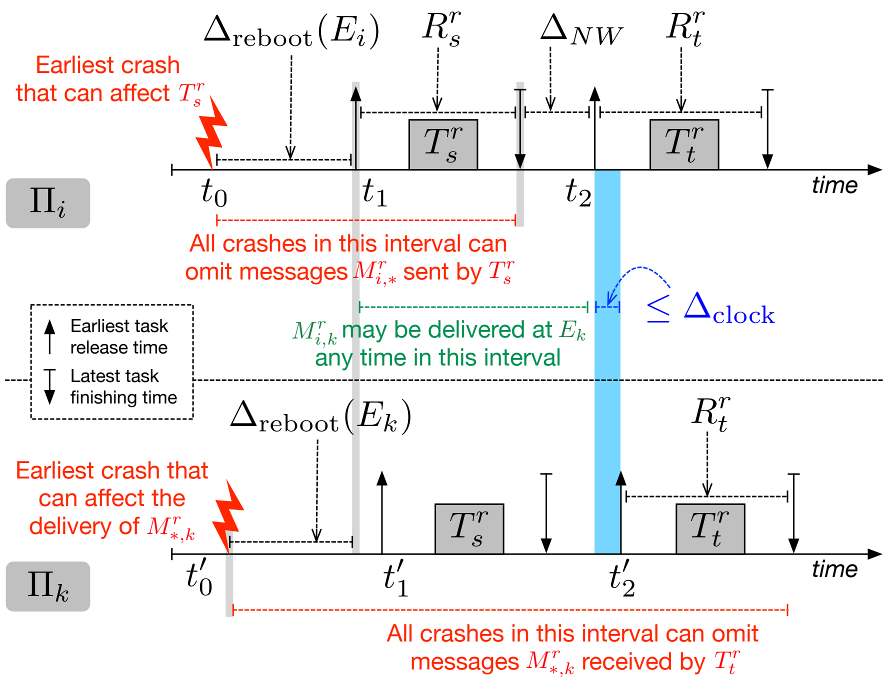

$\Delta_{\text{reboot}}(E_i)$
R_s^r
Δ_{NW}
R_t^r
Earliest crash that can affect T_s^r
T_s^r
T_t^r
Π_i
t_0
t_1
t_2
time
All crashes in this interval can omit messages $M_{i,*}^r$ sent by T_s^r
$\leq \Delta_{\text{clock}}$
Earliest task release time
Latest task finishing time
$M_{i,k}^r$ may be delivered at E_k any time in this interval
$\Delta_{\text{reboot}}(E_k)$
R_t^r
Earliest crash that can affect the delivery of $M_{*,k}^r$
T_s^r
T_t^r
Π_k
t_0'
t_1'
t_2'
time
All crashes in this interval can omit messages $M_{*,k}^r$ received by T_t^r

Figure 5.2: Illustration to demonstrate when a crash on the sender side (or the receiver side) may result in the omission of messages that are to be sent from (or delivered to) that host.

5.4.1 Upper-Bound Node Error Probabilities

Using the Poisson arrival model, we first upper-bound the probability of node errors. Let t_1 and t_1' denote the release time of task T_s^r (responsible for the sending step in round r) on PEs E_i and E_k, respectively. Similarly, let $t_2 = t_1 + R_s^r + \Delta_{NW}$ and $t_2' = t_1' + R_s^r + \Delta_{NW}$ denote T_t^r's release time on PEs E_i and E_k, respectively. Since the IC protocol rounds on all PEs execute synchronously and since the PE clocks differ by at most Δ_{clock} time units, $|t_1' - t_1| \leqslant \Delta_{\text{clock}}$ and $|t_2' - t_2| \leqslant \Delta_{\text{clock}}$. These parameters along with error scenarios are illustrated in Fig. 5.2.

The sending step on PE E_i may be omitted if node E_i is crashed at any time during task T_s^r's scheduling window $[t_1, t_1 + R_s^r)$. Thus, the event "round r msgs. omitted at source E_i" may occur if at least one crash occurs during interval $[t_1 - \Delta_{\text{reboot}}(E_i),\ t_1 + R_s^r)$, i.e.,

$$P(\text{"round } r \text{ msgs. omitted at source } E_i\text{"}) \\ \leqslant 1 - \mathbb{P}(0,\ R_s^r + \Delta_{\text{reboot}}(E_i),\ \gamma_{\text{crash}}(E_i)). \tag{5.8}$$

The round r messages sent from Π_i to Π_k may arrive at Π_k any time during $[t_1, t_1 + R_s^r + \Delta_{NW})$. These messages are then used to update the EIG tree on E_k any time during task T_t^r's scheduling window $[t_2', t_2' + R_t^r)$. Thus, the event "round r msgs. omitted at dest. E_k" may occur if at least one crash occurs during $[t_1, t_2' + R_t^r)$. Since time t_2

and t_2' may differ by at most Δ_{clock} (as also shown in Fig. 5.2), the event "round r msgs. omitted at dest. E_k" may occur if at least one crash occurs during interval $[t_1, t_2 + \Delta_{\text{clock}} + R_t^r)$, i.e.,

$$\begin{aligned}&P(\text{"round r msgs. omitted at dest. } E_k\text{"})\\ &\leqslant 1 - \mathbb{P}\left(0, \left(\begin{array}{c} R_s^r + \Delta_{NW} + R_t^r + \\ \Delta_{\text{clock}} + \Delta_{\text{reboot}}(E_k) \end{array}\right), \gamma_{\text{crash}}(E_k)\right). \end{aligned} \tag{5.9}$$

Furthermore, all round r messages sent from E_i that are routed through switch S_l can be omitted if switch S_l is crashed at any time during the interval $[t_1, t_1 + R_s^r + \Delta_{NW})$. Similarly, any round r message sent from E_k that is routed through switch S_l may be omitted if switch S_l is crashed at any time during time interval $[t_1', t_1' + R_s^r + \Delta_{NW})$. Since these two intervals are expected to be offset by at most Δ_{clock} time units, by generalizing across all round r messages that are routed through S_l, we get the following upper bound.

$$\begin{aligned}&P(\text{"round r msgs. omitted at switch } S_l\text{"})\\ &\leqslant 1 - \mathbb{P}(0,\ R_s^r + \Delta_{NW} + \Delta_{\text{clock}} + \Delta_{\text{reboot}}(S_l),\ \gamma_{\text{crash}}(S_l))\end{aligned} \tag{5.10}$$

The recovery interval of nodes from crashes is typically significantly larger than the task scheduling windows. Hence, for each of the omission errors whose probability is upper-bounded above, we conservatively assume that if a crash error occurs once during the protocol instance, it affects all subsequent tasks (and rounds) on that node in the remaining part of the protocol instance.

To upper-bound the probability of corruption errors, we need to argue about their exposure intervals. The entire message broadcast $M_{i,*}^r$ may be corrupted if the common payload is corrupted during preparation as part of the sending task T_s^r's execution. The payload corruption may even depend on state corruption during earlier rounds of the same protocol instance (e.g., corruption of the EIG tree). However, due to memory protection mechanisms (recall from Section 3.3.2), latent errors prior to the beginning of the protocol instance do not affect the protocol's execution. Hence, since T_s^r's release offset ϕ_s^r denotes the time since the start of the IC protocol instance, and since R_s^r denotes the maximum response time of T_s^r, the event "round r msgs. corrupted at source E_i" may occur if at least one corruption error occurs during an interval of length $\phi_s^r + R_s^r$.

$$\begin{aligned}&P(\text{"round r msgs. corrupted at source } E_i\text{"})\\ &\qquad\leqslant 1 - \mathbb{P}(0,\ \phi_s^r + R_s^r,\ \gamma_{\text{corrupt}}(E_i))\end{aligned} \tag{5.11}$$

5.4.2 Upper-Bound Network Error Probabilities

Next, we upper-bound the probability of network errors. Upper-bounding the probability of message omission or corruption by the network layer is non-trivial because the network itself is constituted of multiple components (links and switches), each of which may experience different rates of transient faults.

The standard 32-bit Cyclic Redundancy Check (CRC) used in Ethernet networks successfully detects every message corruption with three or fewer bit flips [118], whereas error detection becomes increasingly more difficult with larger numbers of bit-flips. Thus, if the message frame carrying $M^r_{i,k}$ suffers up to three bit-flips during transmission, the corruption is detected and the frame is dropped. In contrast, if the message frame experiences more than three bit-flips, the corruption may remain undetected. Hence, if events A_1 and A_2 hold, where

- A_1 denotes event "$M^r_{i,k}$ suffers no corruption on any of the Ethernet links in $\text{route}_{i,k}$" and
- A_2 denotes event "$M^r_{i,k}$ suffers no corruption on any of the switches in $\text{route}_{i,k}$,"

$M^r_{i,k}$ is guaranteed to not be omitted during transmission, i.e.,

$$P\left(\begin{array}{c}\text{"round } r \text{ frame from } \Pi_i \\ \text{to } \Pi_k \text{ omitted by NW"}\end{array}\right) \leqslant 1 - P(A_1)\cdot P(A_2). \tag{5.12}$$

Supposing that $\text{route}_{i,k} = \langle L_{l_1} S_{l_1} L_{l_2} S_{l_2} \ldots L_{l_{n-1}} S_{l_{n-1}} L_{l_n}\rangle$ consists of n hops, and using the independence assumption, the probabilities of events A_1 and A_2 are defined as

$$\begin{aligned} P(A_1) &= \prod_{1\leqslant x\leqslant n} \mathbb{P}\left(0, \Delta_{\text{link}}(M^r_{i,k}), \gamma_{\text{corrupt}}(L_{l_x})\right) \quad \text{and} \\ P(A_2) &= \prod_{1\leqslant x< n} \mathbb{P}\left(0, R^+(M^r_{i,k}, S_{l_x}), \gamma_{\text{corrupt}}(S_{l_x})\right), \end{aligned} \tag{5.13}$$

where $R^+(M^r_{i,k}, S_{l_x})$ denotes the maximum queuing delay of message frame $M^r_{i,k}$ on switch S_{l_x} and $\Delta_{\text{reboot}}(S_{l_x})$ denotes the recovery time of switch S_{l_x} from a fault-induced reboot.

Frame $M^r_{i,k}$ is corrupted by the network only if it is undetectably corrupted (i.e., with four or more bit-flips). To accurately upper-bound its probability, we must account for two factors.

1. If $M^r_{i,k}$ is undetectably corrupted once, any corruptions later on the network path need not be accounted for (as a worst case, we assume that more bit-flips later do not reverse previous bit-flips and do not render the corruption detectable).
2. Before $M^r_{i,k}$ is undetectably corrupted for the first time, it does not suffer any detectable corruptions so as to cause its omission.

Thus, we define the probability upper bound as a sum of the probabilities of events $C_{1,x}$ and $C_{2,y}$ for each $1 \leqslant x \leqslant n$ and $1 \leqslant y < n$,

- where $C_{1,x}$ denotes event "the first undetectable corruption occurs on the x^{th} link in $\text{route}_{i,j}$" and
- $C_{2,y}$ denotes event "the first undetectable corruption occurs on the y^{th} switch in $\text{route}_{i,j}$," i.e.,

$$\begin{aligned}&P(\text{"round } r \text{ frame from } \Pi_i \text{ to } \Pi_k \text{ corrupted by NW"})\\ &\qquad \leqslant \sum_{1 \leqslant x \leqslant n} P(C_{1,x}) + \sum_{1 \leqslant y < n} P(C_{2,y}).\end{aligned} \tag{5.14}$$

Like $P(A_1)$ and $P(A_2)$, the probabilities of events $C_{1,x}$ and $C_{1,y}$ are defined using the independence assumption:

$$\begin{aligned}P(C_{1,x}) &= \left(\begin{array}{l} \left(\prod_{1 \leqslant y < x} (\mathbb{P}(0, L_{k_y})\mathbb{P}(0, S_{k_y}))\right) \\ \times\, \mathbb{P}(4^+, L_{k_x}) \end{array} \right) \text{ and} \\ P(C_{2,y}) &= \left(\begin{array}{l} \left(\prod_{1 \leqslant z < y} (\mathbb{P}(0, L_{k_z})\mathbb{P}(0, S_{k_z}))\right) \\ \times\, \mathbb{P}(0, L_{k_y})\mathbb{P}(4^+, S_{k_y}) \end{array} \right),\end{aligned} \tag{5.15}$$

$$\begin{aligned}\text{where} \quad \mathbb{P}(0, L_{k_y}) &= \mathbb{P}(0, \Delta_{\text{link}}(M^r_{i,k}), \gamma_{\text{corrupt}}(L_{k_y})), \\ \mathbb{P}(0, S_{k_y}) &= \mathbb{P}(0, R^+(M^r_{i,k}, S_{k_y}), \gamma_{\text{corrupt}}(S_{k_y})), \\ \mathbb{P}(4^+, L_{k_y}) &= \sum_{i \geqslant 4} \mathbb{P}(i, \Delta_{\text{link}}(M^r_{i,k}), \gamma_{\text{corrupt}}(L_{k_y})), \\ \mathbb{P}(4^+, S_{k_y}) &= \sum_{i \geqslant 4} \mathbb{P}(i, R^+(M^r_{i,k}, S_{k_y}), \gamma_{\text{corrupt}}(S_{k_y})).\end{aligned}$$

The network analyses above are defined assuming a 32-bit CRC. However, they can be trivially modified if an alternative CRC is being used. We only require that the number of bit-flips up to which the CRC guarantees detection is known in advance. More generally, the analysis can be defined for any predictable networking standard, as long as corresponding timing anaylses are available.

5.5 EVALUATION

We evaluate first the pessimism incurred due to the correction factors added to compensate for reliability anomalies. For this, we compared our monotonic probabilistic analysis with simulations. Second, we demonstrate that the analysis can be used to reveal and quantify non-obvious differences in the reliability of workloads with different design parameters and subject to varying error rates.

We implemented the recursive analysis (Algorithms 5.1 and 5.2) in C++ using the GNU MPFR library [217]. To ensure correct rounding

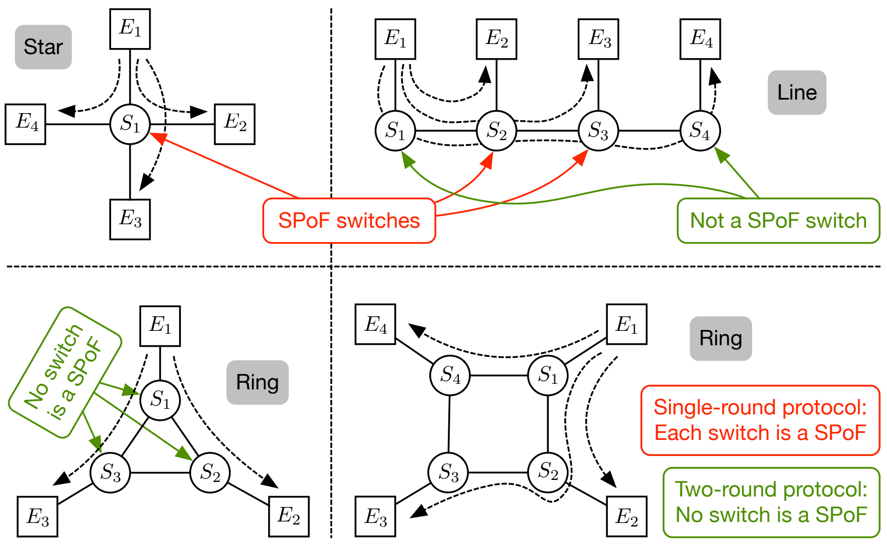


Figure 5.3: Network topologies with static routes (dotted arrows) from E_1 to other PEs. SPoF denotes "Single Point of Failure".

in floating-point computations involving very small probabilities, all analysis computations were carried out at a precision of 200 decimal places.[1] For the timing analysis of the network layer (i.e., to upper-bound message transmission jitters on switches), we modeled each Ethernet output port as a resource with non-preemptive fixed-priority scheduling, and computed message queuing delays on each port using Compositional Performance Analysis (CPA) [59, 98, 227]. Recall from Section 2.1.3.2 that Ethernet allows only up to eight distinct priority classes, and that messages of equal priorities are stored in and serviced from a dedicated FIFO queue. All experiments were carried out on Intel Xeon E7-8857 v2 machines (48 cores, 1.5 TB of memory) clocked at 3 GHz. While each analysis instance executed sequentially, the multi-core machines were used to run multiple instances of the analysis in parallel.

The analyzed workload consisted of up to four processes on four different PEs periodically executing a hard real-time IC protocol instance every $P = 100\,\mathrm{ms}$. The global worst-case response time for each IC protocol task was assumed to be $1\,\mathrm{ms}$, based on a prototype implementation of the protocol used for the key-value service case study (see Section 4.2.3). The PEs were assumed to be connected via a single switch (*star* topology) or via multiple switches arranged in either a *line* or a *ring* topology (Fig. 5.3). We used a transfer rate of $100\,\mathrm{Mbps}$ for each port and a wire delay of $330\,\mathrm{ns}$ for each link. The PEs also periodically exchanged PTP messages for clock synchronization, which were assigned to the highest priority network class. Based on PTPd version 2.3.2, these messages have a payload of $76\,\mathrm{bytes}$ each and a period of $500\,\mathrm{ms}$. We assumed periodic exchanges of maximum-sized

1 The *precision* of a variable indicates the number of bits used to store its significand.

frames between PEs to model lower-priority traffic, which results in worst-case blocking delay for the IC protocol messages at each switch.

The crash recovery times were set to 1 s. Error rates are reported as the *mean number of errors per microsecond*. Unless mentioned otherwise, experiments assume the strong correctness criterion (which was defined in Section 5.3.1).

5.5.1 Analysis vs. Simulation

We compared `Unsafe-Analysis` and `Mono-Analysis` (i.e., with and without reliability anomalies, respectively) with simulation baselines `Sim-v1` and `Sim-v2` to evaluate the pessimism incurred due to reliability anomalies elimination.

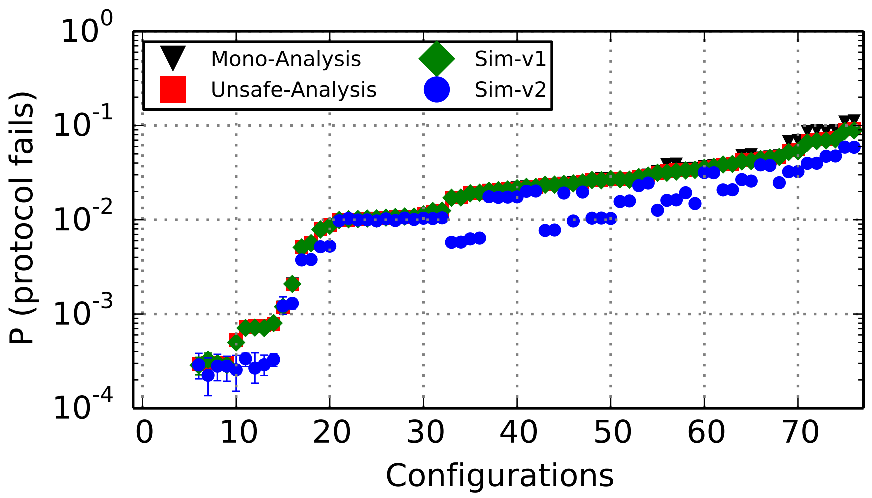
`Sim-v1` knows in advance the message error probabilities for each IC protocol message. Thus, for every error type, `Sim-v1` draws a number uniformly at random from the range $[0, 1]$, compares it with the respective error probability to decide whether the error is encountered or not, and if the error is encountered, simulates the corresponding error scenario. `Sim-v1` thus helps to isolate the pessimism incurred (if any) in our recursive analysis procedure.

In contrast, `Sim-v2` simulates FIFO priority queues at the network layer and uses Poisson processes to generate the respective fault events on each host and on the network. These 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. `Sim-v2` evaluates the pessimism incurred when upper-bounding the message error probabilities as a function of the raw transient fault rates using the Poisson model.

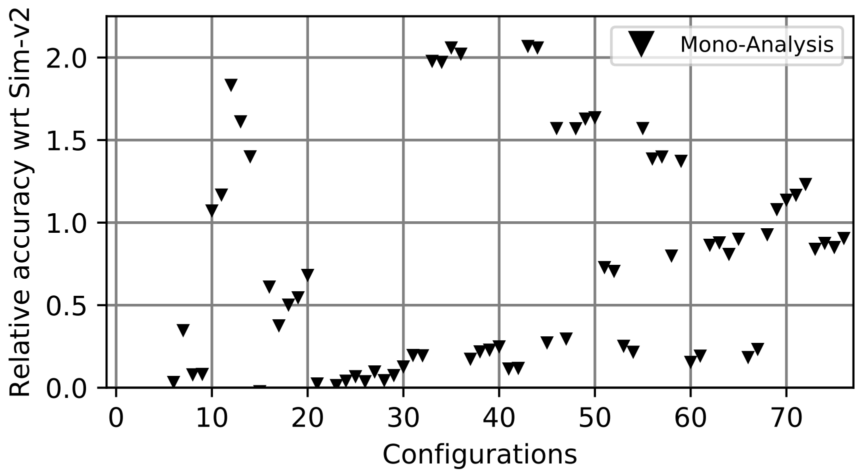
Both `Sim-v1` and `Sim-v2` make the worst-case assumption that any two faulty message copies are identical, as in the analysis. The simulations were run as a discrete event simulation for 100 000 iterations each to ensure that the 99th percentile confidence intervals were of negligible magnitude relative to the absolute values.

We compared the four baselines `Unsafe-Analysis`, `Mono-Analysis`, `Sim-v1`, and `Sim-v2` for different topologies, PE crash error rates (0 or 10^{-8}), switch crash error rates (0 or 10^{-8}), PE corruption rates (0 or 10^{-5}), switch corruption rates (0 or 10^{-5}), and link corruption rates (0 or 0.001). We used higher error rates than can be realistically expected in practice as otherwise the simulations would be extremely time-consuming. In Fig. 5.4a, we illustrate the results absolute failure probabilities for $N_p = 3$ processes and $N_r = 2$ rounds. In Fig. 5.4b, we illustrate only the failure probabilities for `Mono-Analysis`, but normalized with respect to the failure probabilities obtained using `Sim-v2`.

`Unsafe-Analysis` exactly tracks `Sim-v1`, which indicates that the recursive analysis presented in Section 5.3.3 incurs no substantial pessimism. `Mono-Analysis` also closely tracks `Unsafe-Analysis` and



(a)



(b)

Figure 5.4: **(a)** Failure probabilities estimated by analyses `Unsafe-Analysis` and `Mono-Analysis`, and using simulation versions `Sim-v1` and `Sim-v2` (sorted in increasing order of `Unsafe-Analysis` results). **(b)** Failure probabilities estimated by `Mono-Analysis` normalized with respect to failure probabilities estimated by `Sim-v2`, for the same set and order of configurations as in (a).

`Sim-v1`, (i.e., it does not exhibit notable pessimism), which we attribute to the anomaly correction terms having negligibly small magnitudes when not needed.

In contrast, the analysis results do not closely track `Sim-v2` for some configurations (with the maximum observed relative error of about $2.066\times$, as shown in Fig. 5.4b). Higher pessimism results from the analysis to upper-bound PE corruption errors (Section 5.4.1), since the exposure intervals for different protocol tasks overlap, i.e., the exposure interval of each task includes the time since the start of the IC protocol instance.

5.5.2 Reliability Trade-offs

The next set of experiments were conducted to understand the benefits (if any) of using the weak correctness criterion (whenever the application permits), and the effects of non-uniform fault rates and different network topologies on the protocol reliability. The error rates used are much smaller than those used in the previous section, since the analysis runtime (unlike simulations) does not depend on the magnitude of error rates. In particular, we use realistic error rates derived from prior studies on transient fault rates [12, 97]. In addition, in the following experiments, we report FIT rates, which can be derived from the failure probability of a single IC protocol invocation using the steps outlined in Section 5.2.

5.5.2.1 *Experiment 1*

We evaluated FIT bounds for the strong and weak correctness criteria for six configurations with $N_p \in \{2, 3, 4\}$ and $N_r \in \{1, 2\}$. We only considered crash errors in this experiment (each PE has a crash rate of 10^{-15}), since the two criteria differ in terms of how they treat omissions, which in turn are aggravated by crash errors.

The results in Fig. 5.5 show that the FIT bounds for the strong criterion are orders of magnitude higher than the FIT bounds for the weak criterion, which indicates that the protocol is much more likely to violate the strong criterion (as expected). Therefore, an effective reliability analysis should account for the weak criterion whenever it suffices for an application, to obtain more accurate failure rates. In addition, when the number of processes is increased from two to three, while the FIT bounds for the weak criterion decrease, the FIT bounds for the strong criterion remain the same. This observation corroborates the findings from classical BFT theory (which relies on the strong correctness criterion) that going from an odd number of replicas to an even number of replicas does not yield any reliability benefits.

Surprisingly, the results in Fig. 5.5 indicate that additional rounds seemingly never help. This is a consequence of crash errors, which dominate in these scenarios, since a crash is likely to keep the node

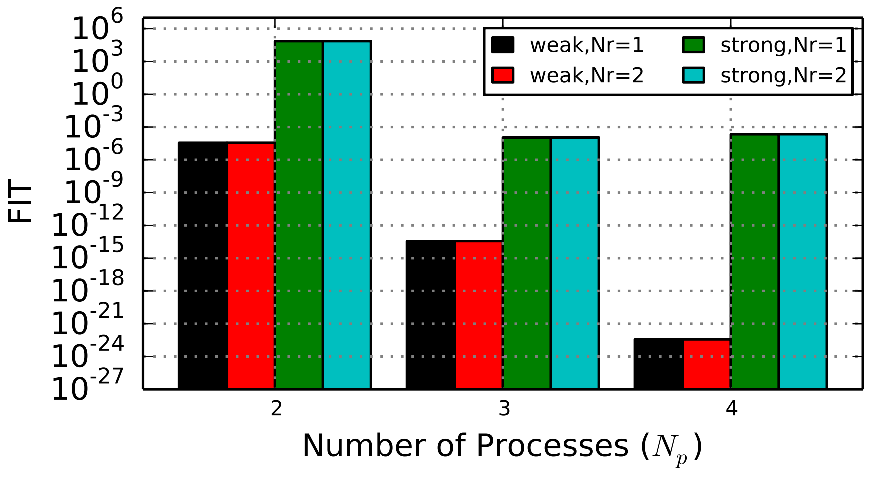


Figure 5.5: FIT bounds estimated in the presence of PE crash errors.

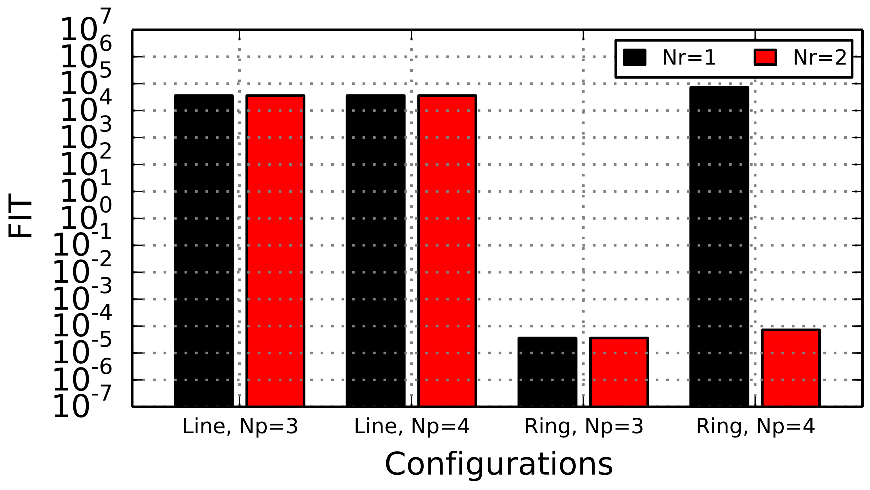


Figure 5.6: FIT bounds estimated in the presence of switch crash errors.

unavailable for all rounds of the protocol. We repeated a similar experiment for $N_p = 3$ while considering only network corruption errors. The resulting FIT bounds for $N_r = 1$ and $N_r = 2$ were 3.623×10^{-5} and 6.993×10^{-14} (respectively), clearly indicating the benefit of multiple rounds when the dominant error sources affect different rounds independently.

5.5.2.2 *Experiment 2*

Next, we sought to understand the impact of different network topologies as well as non-uniform error rates on the evaluated FIT analysis. Therefore, we considered only switch crash errors in this experiment, assigned a crash error rate of 10^{-15} to switches S_1 and S_2 (see Fig. 5.3 for reference), whereas other switches were assumed to execute error-free. Assuming the strong correctness criterion, we computed FIT bounds for eight different configurations: line and ring topology, $N_p \in \{3,4\}$, and $N_r \in \{1,2\}$. The results are illustrated in Fig. 5.6. We observe that all configurations with line topology have very high FIT bounds with negligible differences. This is because switch S_2 is a single point of failure (SPoF) in a line topology with three or four PEs (two PEs cannot form a quorum if $N_p = 4$). In contrast, if three PEs are arranged in a ring topology, the FIT bounds are low since no single switch is a Single Point of Failure (SPoF) (failure results only if both S_1 and S_2 crash). Interestingly, four PEs benefit from the ring topology only if $N_r = 2$. We attribute this to a combination of two factors: static routing and asymmetric IC protocol rounds. Static routing prevents switches from immediately moving to an alternate route. Thus, every single switch becomes a SPoF for $N_r = 1$. However, for $N_r = 2$, if Π_3 misses a message from Π_1 in the first round owing to S_2's crash, it still gets a chance to receive Π_1's private value from Π_4 in the second round, since messages from Π_4 and Π_1 are not routed through S_2.

5.5.2.3 *Experiment 3*

In the final experiment, we applied different *shielding factors* (that lead to reduced error rates) with the aim of simulating practical tradeoffs between using more resilient processors, better quality links, or just better casing (each of which helps to reduce environmental effects) versus auxiliary factors (e.g., cost, weight, power, etc.).

In absence of any shielding, the node crash rates, node corruption rates, and link corruption rates are 10^{-15}, 10^{-17}, and 10^{-7}, respectively. We considered Node Shielding Factors (NSF), Link Shielding Factors (LSF), and Overall Shielding Factors (OSF) that lead to reduced error rates across nodes, links, or the entire system, respectively (e.g., an LSF of 10 indicates that the link corruption rates are 10 times smaller, i.e., 10^{-8} instead of 10^{-7}). The results are illustrated in Fig. 5.7.

In case of the star topology, since the switch denotes a SPoF, its crash rate is the determining factor. Nonetheless, given a reliability objective in terms of a maximum acceptable FIT, the analysis can help determine appropriate levels of shielding. In contrast, the FIT bounds for the ring topology vary in complex ways, and given a FIT objective, multiple shielding options can be used (e.g., to achieve a FIT of under 10^{-4} with ring topology, either better quality casing is needed so that the OSF exceeds 10^3, or simply more resilient nodes could be used that provide a NSF greater than 100).

To conclude, we presented in this chapter the first quantitative reliability analysis of a hard real-time IC protocol over Ethernet in the presence of environmentally induced Byzantine errors (which are the most general kind). Our analysis explicitly models PE nodes, network switches, and network links and considers the effect of transient faults in any of them. Importantly, our analysis is free from reliability

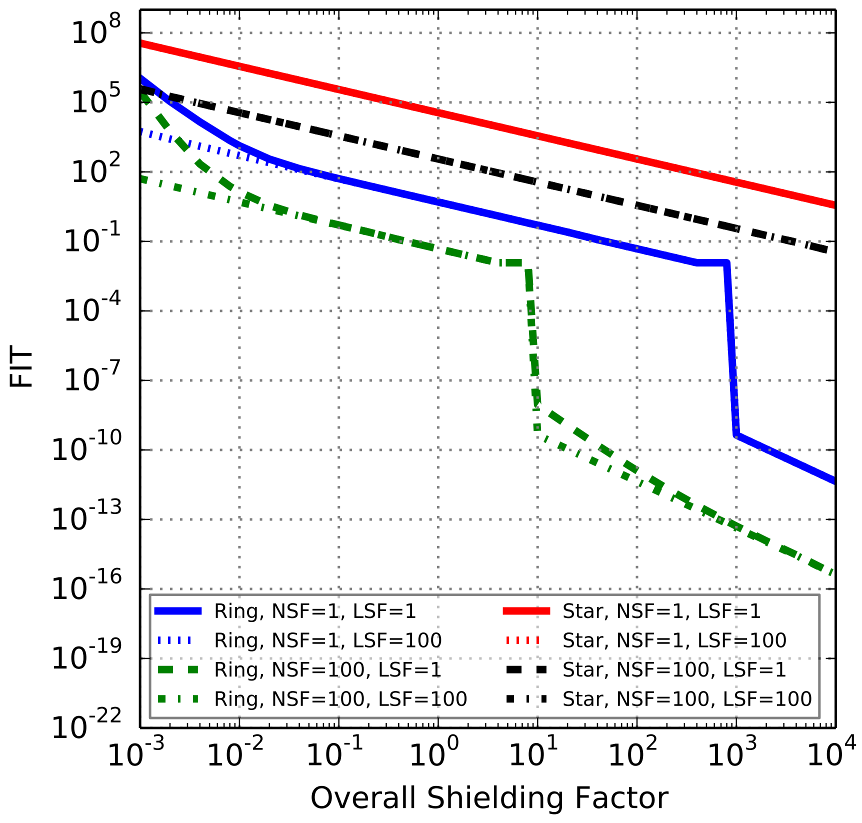


Figure 5.7: FIT bounds for different shielding factors.

anomalies, i.e., when a non-maximal fault rate in some component can counter-intuitively result in an increase of the system's overall failure rate. In fact, to the best of our knowledge, this is the first work to formalize the concept of reliability anomalies, and to propose techniques to eliminate such anomalies in a hard real-time setting. Our evaluation has demonstrated the proposed analysis to reveal non-obvious reliability trade-offs and to closely track simulation results.

In future work, it would be interesting to evaluate a practical prototype of the analyzed protocol, and to incorporate recent advances in real-time Ethernet standards related to flow integrity, such as different stream reservation and path control protocols, into our quantitative reliability analysis framework.

Part III

NETWORKED CONTROL SYSTEMS

6 RELIABILITY ANALYSIS OF AN NCS ITERATION*

* This chapter is based on our ECRTS 2018 [91] paper.

Chapters 4 and 5 dealt with Byzantine error scenarios, which are possible in distributed real-time systems that are connected over point-to-point networks, such as Ethernet. We presented a hard real-time design for a classical IC protocol and a corresponding reliability analysis, which can together be used to implement an ultra-reliable atomic broadcast service over COTS networks for building safety-critical CPS. Such an analysis-driven implementation can be configured to provide comparable levels of reliability as that of conventional field buses like CAN (see Section 2.1.3.1), or as that of customized bus architectures like those used in MeshKin [207] and SPIDER [151] (see Section 4.1.1), each of which implicitly provides atomic broadcast guarantees.

However, for a full-system reliability analysis (recall our goals from Chapter 1), we must also upper-bound the FIT rate of other critical software components, even if they are deployed on top of an atomic broadcast network layer. To this end, in this chapter and the following chapter, we present analyses to safely upper-bound the FIT rate of one or more NCS applications.

We focus on NCS applications since they constitute a major share of all safety-critical applications in CPS devices. Other critical services that are also of primary interest from a safety perspective include, for instance, the clock synchronization module and the operating system, which we however do not consider here as they are orthogonal concerns (recall the SOFR approach discussed in Chapter 1).

Prior work on the analysis of actively replicated NCS has focussed on very coarse-grained methods that analyze the probability of permanent host failures, evaluate the system state transitions arising out of such failures (e.g., from a highly redundant TMR configuration to a less redundant DMR configuration), and report the expected lifetime of the system. Examples include Dugan and Van Buren's [66] reliability analysis of a fly-by-wire system with passive replication and Sinha's [202] reliability analysis of a fail-operational brake-by-wire system networked with CAN and FlexRay buses. Fine-grained analyses have also been proposed, but they do not report full-system (or end-to-end NCS) reliability. For example, prior studies on the effect of EMI on CAN-based systems [38, 54, 163, 179, 198, 221] only analyze the response times of individual CAN messages.

In this dissertation, we evaluate the reliability of an actively replicated NCS in the presence of transient faults at the granularity of network messages, like we did for the IC protocol. Errors due to transient faults, such as crash and reboot errors, may keep a host unavailable for a small amount of time. Corruption errors may affect the integrity of certain messages. However, in an actively replicated NCS, such errors do not affect the final actuation if masked by the redundancy. Even if they do, in most cases, the control might be robust enough to withstand a few skipped or incorrect actuations. Hence, especially for actively replicated NCS applications, a fine-grained reliability analysis is needed to more accurately capture the benefits of active replication, which we present in this and the following chapter.

The remainder of this chapter is organized as follows. We first provide a formal model of an NCS with active replication that is connected using a network with atomic broadcast guarantees (Section 6.1). Following the system model, we provide an overview of the reliability analysis (Section 6.2) and describe the detailed analysis (Sections 6.3 and 6.4). For brevity, we defer all soundness proofs to Appendix A. Finally, we evaluate the pessimism incurred in our analysis by comparing its results with simulation results for a CAN-based active suspension workload (Section 6.5).

In our evaluation, we emphasize on NCS applications based on CAN since the bandwidth limitations in CAN (unlike in Ethernet) can seriously impact the reliability of a time-sensitive system. Also, the use of CAN is still prevalent in the development of many safety-critical CPS, especially in subsystems that are attached to the physical sensors and actuators. For example, the architecture of Care-O-bot 4, which is a next-generation service robot developed by Fraunhofer IPA, uses Ethernet to bridge the different hosts, but CAN buses to bridge each host with sensors and actuators [42, 139, 187].

6.1 SYSTEM MODEL AND ASSUMPTIONS

We model and analyze a Single-Input Single-Output (SISO) control loop with active replication (as described below), which is necessary for fault tolerance. The SISO control loop is hence also referred to as an FT-SISO control loop. In the end (Section 6.5), we consider extensions for more complex system models with multi-input single-output (MISO) and multi-input multi-output (MIMO) controllers.

The FT-SISO networked control loop, denoted L, is 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 of the analyzed FT-SISO loop with all notations is illustrated in Fig. 6.1. The notations are explained next, and summarized in Table 6.1 for quick reference.

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

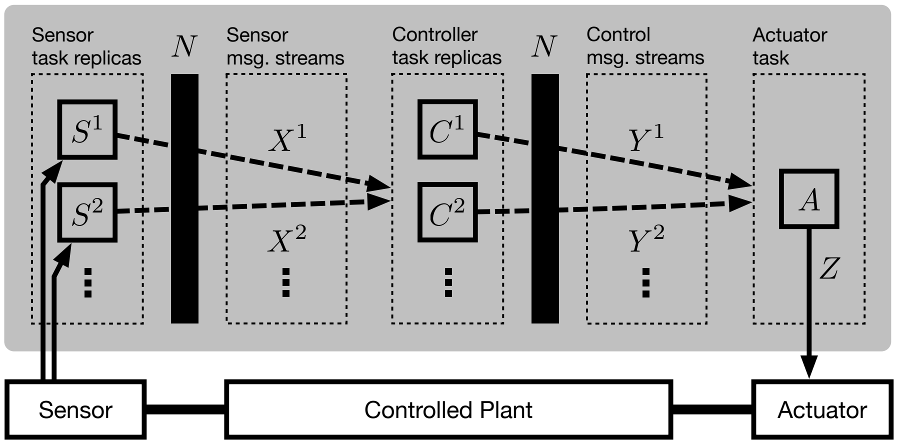


Figure 6.1: An FT-SISO networked control loop. Solid boxes denote hosts. Each dashed box denotes a task replica set or a set of message streams transmitted by a task replica set. Dashed arrows denote message streams broadcast over the shared network N, e.g., X^1 and X^2 are received by all tasks in C.

the sensor values of the ith 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 $Y = \{Y^1, Y^2, \ldots\}$.

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, A is not replicated since it requires special hardware in the plant actuator to handle redundant inputs [109].

All tasks and messages in the control loop have a period of T time units. The nth runtime activations or jobs of sensor task replicas in $S = \{S^1, S^2, \ldots\}$ and controller task replicas in $C = \{C^1, C^2, \ldots\}$ are denoted $S_n = \{S^1_n, S^2_n, \ldots\}$ and $C_n = \{C^1_n, C^2_n, \ldots\}$, respectively; and the nth job of actuator task A is denoted A_n. Similarly, the nth messages in sensor message streams $X = \{X^1, X^2, \ldots\}$ and controller message streams $Y = \{Y^1, Y^2, \ldots\}$ are denoted $X_n = \{X^1_n, X^2_n, \ldots\}$ and $Y_n = \{Y^1_n, Y^2_n, \ldots\}$, respectively. In general, as a convention, we let subscripts denote the job ID (or iteration).

Finally, we let Z_n denote the actuator command applied to the physical plant in the nth iteration, i.e., output of job A_n, and let

PURPOSE	MAIN SYMBOL	REP i, ALL ITERS	ALL REPS, ITER n	REP i, ITER n
FT-SISO loop	L	-	-	-
Network	N	-	-	-
Host	H	H_i	-	-
Sensor task	S	S^i	S_n	S_n^i
Controller task	C	C^i	C_n	C_n^i
Actuator task	A	-	A_n	-
Sensor message	X	X^i	X_n	X_n^i
Control command	Y	Y^i	Y_n	Y_n^i
Actuation	Z	-	Z_n	-
Controller voter o/p	U	U^i	U_n	U_n^i
Actuator voter o/p	V	-	V_n	-

Table 6.1: Summary of notations. REP denotes replica, and ITER denotes iteration. The main symbol corresponds to a union of per-replica partitions or a union of per-iteration partitions (if applicable), e.g., $S = \cup_{\forall i} S^i = \cup_{\forall n} S_n$. The per-replica and per-iteration notations correspond to a union of per-replica per-iteration partitions (if applicable), e.g., $S^i = \cup_{\forall n} S_n^i$ and $S_n = \cup_{\forall i} S_n^i$.

$Z = \{Z_1, Z_2, \ldots\}$ denote the ordered set of such commands applied to the physical plant across all iterations.

To suppress redundancy, we assume that each task resolves redundant inputs at the start of every iteration through voting (Algorithm 6.1). We let $U_n = \{U_n^1, U_n^2, \ldots\}$ denote the voter outputs after resolving the redundant inputs for controller jobs $C_n = \{C_n^1, C_n^2, \ldots\}$, respectively. Similarly, we let V_n denote the voter output after resolving the redundant inputs for the actuator job A_n. 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 6.1). 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). We assume that old inputs from previous iterations are not reused. While computing the simple majority (Line 8), any ties in quorum size are broken deterministically using message IDs, i.e., the message with the smallest ID is favored.

Inputs to the voters may be corrupted. However, whether or not the corrupted inputs (messages) are likely to be identical is highly system- and application-specific. Transient faults normally do not cause identically corrupted patterns and many systems use end-to-end checksums; the likelihood of identically corrupted messages is

Algorithm 6.1 Voting procedure before the activation of any controller task C_n^i. The voting procedure before any actuator task A_n is defined similarly by replacing the input set X_n with Y_n.

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

thus small. In contrast, if the application payload is of boolean type or encoded using only a few bits, the likelihood of identically corrupted messages is non-negligible. In this work, we (pessimistically) assume that corrupted message replicas are identically corrupted because it is a worst-case scenario with respect to the voting protocol. That is, 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.

We also require that all tasks that are part of the networked control application are deterministic. That is, given identical inputs and identical states, any two sensor (controller) task replicas produce identical sensor messages (control messages, respectively), unless one is affected by memory corruption.

Regarding the underlying platform, we make three important assumptions. First, we assume that NCS hosts are synchronized using a clock synchronization protocol, such as PTP [106], and that task and message offsets have been chosen to account for the maximum clock synchronization error. Without this assumption, and without any other explicit replica determinism protocol (such as Achal, which was presented in Section 4.2.3), it is much more challenging to ensure replica determinism [176]. Simply assigning appropriate offsets to tasks and messages is insufficient.

Second, we assume that the underlying network always guarantees atomic broadcast, even in the presence of faults. As mentioned in Section 3.5, this strong assumption does not compromise the safety of the proposed analysis. In fact, any flaw in the implementation of the underlying network protocol that might cause a violation of the atomic broadcast guarantee can be analyzed separately and accounted for in the overall failure rate computation.

In other words, even though we derive the atomic broadcast assumption from the use of BFT middleware like Achal (Section 4.2.3) or from the protocol descriptions of CAN and similar other field buses, these networking layers may not provide atomic broadcast every single time. For example, Rufino et al. [188] identify one such corner case in the CAN specification that is triggered when there are bit-slips in specific bits of the CAN message frame. Similarly, middleware like Achal, when configured with a replication factor of four, may fail if more than one replica behaves erroneously during a single instance of the protocol. The probability of such corner cases, or any flaws in the protocol implementations, could be separately computed (such as the failure analysis of Achal's atomic broadcast protocol in Chapter 5) and associated with a full-system failure in the worst case using the SOFR model (recall the discusssion in Section 1.2); the resulting failure rate can then be composed with the failure rate of the NCS (assuming perfect atomic broadcast) derived from the proposed analysis. A similar argument also holds for the failure of the assumed clock synchronization algorithm.

Third, we assume that a message that is delayed beyond its deadline is discarded by its receivers, or not transmitted by its sender in the first place (the latter scenario is possible if the start time of the message transmission is delayed beyond its latest start time). Clock synchronization can be leveraged in such cases to ensure that a message is safely discarded on all hosts. Broster and Burns [37] discuss multiple ways to achieve this in the context of CAN.

The deterministic tasks and the atomic broadcast assumption together ensures that all correct, functionally identical replicas in the NCS produce the same output and fail in the same manner (since correct replicas can fail only due to faulty inputs). We explicitly handle this correlation in our analysis.

6.2 ANALYSIS OVERVIEW

We analyze the probability that the n^{th} iteration of the control loop fails, for any n. Thus, we mostly use the notations in the last two columns of Table 6.1 while defining the analysis. In the end, we argue that the derived probability is, in fact, identically and independently distributed (IID) with respect to n. The IID property is leveraged by the MTTF/FIT analyses in Chapter 7.

Due to clock synchronization and the atomic broadcast property of the underlying network, and due to the deterministic nature of NCS tasks, message replicas function identically in an error-free scenario. That is, the messages in X_n carry identical sensor values and the messages in Y_n carry identical control commands in the absence of any errors. However, due to incorrect computation errors, one or more

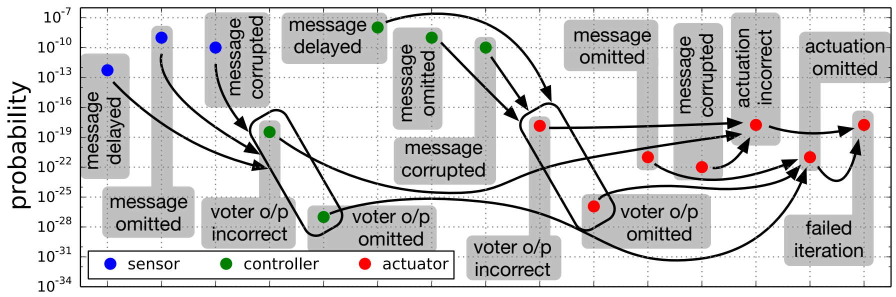


Figure 6.2: Propagation of error probabilities in a CAN-based wheel control loop (see Section 6.5 for details). Arrows denote dependencies among error probabilities of the different control loop stages.

messages in X_n may be corrupted. Due to transmission errors and crash-induced omissions, one or more messages in X_n may also be delayed or omitted. Thus, the controller voter instances may have to work with fewer inputs and/or incorrect inputs.

In such a scenario, depending on whether the controller voter instances choose a corrupted sensor value as their outputs (i.e., whether each U_n^i is incorrect), and whether the controller tasks experience fault-induced incorrect computation errors themselves (resulting in some U_n^i being incorrect), some or all of the messages in Y_n carrying the control commands may also be corrupted. In the worst case, if all the message in X_n are either delayed or omitted, the controller voter instances have no inputs to work with, and consequently the messages in Y_n would not be prepared in the first place.

Similarly, the controller to actuator information flow may be affected by errors, resulting in A_n's output Z_n being corrupted or omitted.

These dependencies between different events during the n^{th} control loop iteration are illustrated using an example in Fig. 6.2, along with the event probabilities associated with each event (which are a byproduct of the proposed analysis). The objective of our analysis is to capture these dependencies accurately without compromising the soundness requirement. We start with an overview of the analysis.

In a nutshell, the proposed analysis is similar to that of an IC protocol instance, which was presented in Section 5.3. That is, **(i)** we start with a set of message errors; **(ii)** quantify the NCS iteration failure probability in the presence of these errors, and as a function of exact message error probabilities; **(iii)** eliminate the effect of reliability anomalies on the derived failure probability bound (as in Section 5.3, we add correction terms to each analysis step whose output is not monotonic in the exact message error probabilities); and **(iv)** instantiate the analysis using implementation-specific upper bounds on the message error probabilities. However, unlike the IC protocol analysis in Section 5.3, we also deal with correlated errors in case of NCS applications, which result due to clock synchronization and the atomic broadcast property of the underlying network (as explained above).

In addition, since the active replication protocol is simpler than the IC protocol, we define the analysis steps in more detail in this chapter.

Throughout the analysis, we use $P(\cdot)$ to denote *exact* probabilities and $Q(\cdot)$ to denote upper bounds on the corresponding exact probabilities. This distinction is necessary to simplify reasoning about the analysis safety, that is, to ensure that the derived probability of an iteration failure is indeed an upper bound. Thus, while we freely use the complement $1 - P(\cdot)$ of any exact probability $P(\cdot)$ in our analysis definitions, we ensure that the complementary probability $1 - Q(\cdot)$ of any probability upper bound $Q(\cdot)$ is never used, since it denotes a lower bound. Also, for brevity, we let $\overline{P(\cdot)} = 1 - P(\cdot)$.

The analysis proceeds as follows. First, we define the following three exact (but unknown) message error probabilities for each message m based on the fault model description provided in Chapter 3.

DEFINITION 6.1. $P(m \text{ omitted})$ denotes the exact probability with which message m is omitted.

DEFINITION 6.2. $P(m \text{ delayed})$ denotes the exact probability with which message m suffers a deadline violation.

DEFINITION 6.3. $P(m \text{ corrupted})$ denotes the exact probability with which message m is incorrectly computed.

In the above definitions, m can denote a message carrying a sensor value (i.e., one of the messages in X_n), a message carrying a control command (i.e., one of the messages in Y_n), or the final actuation command Z_n that is applied to the physical plant (although, since the final actuation Z_n is not applied over the shared network N, probability $P(Z_n \text{ delayed})$ is not defined). In addition to these, since the effect of message corruption on Algorithm 6.1's output also depends on the application-specific message payload, the analysis initially also assumes the following exact (but unknown) probability.

DEFINITION 6.4. $P(\text{SimpleMajority incorrect} \mid \mathcal{I}, \mathcal{C})$ denotes the exact probability with which the SIMPLEMAJORITY$(\mathcal{I} \cup \mathcal{C})$ procedure in Algorithm 6.1 (Line 8) outputs an incorrect value, given a set $\mathcal{I}$ of incorrect inputs and set $\mathcal{C}$ of correct inputs.[1]

Given the aforementioned exact probabilities, we derive the per-iteration failure probability (Section 6.3). For safety reasons, i.e., to avoid reliability anomalies, the derived probability must be either independent of or increasing in these exact error probabilities. In the second part of the analysis (Section 6.4), we provide upper bounds for the exact probabilities in Definition 6.1-Definition 6.4 (since their exact

1 We use the terms *corrupted* and *incorrect* differently. A corrupted message is directly affected by incorrect computation errors, whereas an incorrect message simply refers to a message that differs from the corresponding message in an error-free scenario. Therefore, to denote a voter output that is corrupted because a majority of inputs to the voter instance were corrupted, we use the term incorrect.

values are unknown), and then instantiate the per-iteration failure probability derived in Section 6.3 with these upper bounds in place of the exact probabilities. As a result of the monotonicity property, despite this replacement, it is implicitly guaranteed that the resulting per-iteration failure probability upper-bounds the actual per-iteration failure probability. Therefore, the proposed analysis is safe even in the event that error probabilities experienced in practice are lower than those used for the analysis (which is usually the case).

6.3 PROBABILISTIC ANALYSIS

We estimate the probability that the final actuation output Z_n is either corrupted or omitted, in a bottom-up fashion, and in small steps of a few lemmas each. We analyze the controller voter instance output in Section 6.3.1, the actuator voter instance output in Section 6.3.2, and the final actuation output in Section 6.3.3.

6.3.1 Controller Output

In this section, we analyze output U_n^y of the controller voter instance (that executes before controller task instance C_n^y). In particular, we separately analyze the probability that U_n^y is either incorrect or omitted in Sections 6.3.1.1 and 6.3.1.2, respectively.

Recall from the system model that X_n denotes the set of all sensor message replicas that are inputs to this voter instance. Fault-induced errors in each message in X_n can affect U_n^y's since the message can be omitted due to timing errors or delayed due to retransmission errors. Even if the message is transmitted on time, the received message could have been corrupted due to incorrect computation errors. To model all such possibilities, we represent the *error status* of messages in X_n using an ordered 5-tuple $\mathcal{E}(X_n)$, which is defined as follows.

DEFINITION 6.5. The *error status* of messages in X_n is defined as $\mathcal{E}(X_n) = \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle$ where

- sets $\mathcal{O}_n$, $\mathcal{D}_n$, $\mathcal{I}_n$, $\mathcal{C}_n$, and $\mathcal{Z}_n$ *partition* the message set X_n;
- $\mathcal{O}_n$ denotes the set of messages that are omitted;
- $\mathcal{D}_n$ denotes the set of messages that are not omitted, but delayed due to retransmissions;
- $\mathcal{I}_n$ denotes the set of messages that are neither omitted nor delayed, but are incorrectly computed;
- $\mathcal{C}_n$ denotes the set of messages that are neither omitted, delayed, nor incorrectly computed; and

- $\mathcal{Z}_n$ denotes the set of messages whose status is unknown.

The distinction made in Definition 6.5 among different elements of the 5-tuple helps reduce pessimism in the analysis. For example, the definition ignores events such as a message being both omitted and incorrectly computed, since whether a message is corrupted or not is inconsequential once the message has been omitted. In general, $\mathcal{Z}_n$ denotes the set of messages whose *fate is undecided*, or in other words, each message $X_n^y \in \mathcal{Z}_n$ may still be omitted with probability $P(X_n^y \text{ omitted})$, delayed with probability $P(X_n^y \text{ delayed})$, and incorrectly computed with probability $P(X_n^y \text{ corrupted})$. As a result, there can be multiple valid definitions of the error status tuple. For example, $\langle \emptyset, \emptyset, \emptyset, \emptyset, X_n \rangle$ is a valid definition denoting that none of the messages in X_n is guaranteed to be omitted, delayed, or incorrectly computed, but that each message in X_n can be affected by any message error.

6.3.1.1 *Analyzing the Correctness of* U_n^y

Using the error status in Definition 6.5, and using the exact probabilities in Definition 6.1-Definition 6.4, we first define the probability that the output of controller task C_n^y's voter instance U_n^y is incorrect.

DEFINITION 6.6. The probability that U_n^y is incorrect is given by $P(U_n^y \text{ incorrect} \mid \langle \emptyset, \emptyset, \emptyset, \emptyset, X_n \rangle)$, where

$$P\left(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle\right) = \begin{cases} P(\text{SimpleMajority incorrect} \mid \mathcal{I}_n, \mathcal{C}_n) & \mathcal{Z}_n = \emptyset \\ \Gamma_1 + \Gamma_2 + \Gamma_3 + \Gamma_4 & \mathcal{Z}_n \neq \emptyset \end{cases},$$

$$\begin{aligned} \Gamma_1 =& P(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n \cup \{X_n^s\}, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \setminus \{X_n^s\} \rangle) \\ & \times P(X_n^s \text{ omitted}), \end{aligned}$$

$$\begin{aligned} \Gamma_2 =& P(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n \cup \{X_n^s\}, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \setminus \{X_n^s\} \rangle) \\ & \times \overline{P(X_n^s \text{ omitted})} \times P(X_n^s \text{ delayed}), \end{aligned}$$

$$\begin{aligned} \Gamma_3 =& P(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n \cup \{X_n^s\}, \mathcal{C}_n, \mathcal{Z}_n \setminus \{X_n^s\} \rangle) \\ & \times \overline{P(X_n^s \text{ omitted})} \times \overline{P(X_n^s \text{ delayed})} \times P(X_n^s \text{ corrupted}), \end{aligned}$$

$$\begin{aligned} \Gamma_4 =& P(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n \cup \{X_n^s\}, \mathcal{Z}_n \setminus \{X_n^s\} \rangle) \\ & \times \overline{P(X_n^s \text{ omitted})} \times \overline{P(X_n^s \text{ delayed})} \times \overline{P(X_n^s \text{ corrupted})}, \end{aligned}$$

and X_n^s denotes the message with the smallest ID in $\mathcal{Z}_n$. The probability $P(U_n^y \text{ incorrect} \mid \langle \emptyset, \emptyset, \emptyset, \emptyset, X_n \rangle)$ is also denoted as $P(U_n^y \text{ incorrect})$.

In each step of the recursion, a single message $X_n^s \in \mathcal{Z}_n$ is either **(i)** omitted with probability $P(X_n^y \text{ omitted})$ and inserted into set $\mathcal{O}_n$; **(ii)** not omitted but delayed with probability $\overline{P(X_n^y \text{ omitted})} \times P(X_n^y \text{ delayed})$ and inserted into set $\mathcal{D}_n$; **(iii)** transmitted on time, i.e., neither omitted nor delayed, but is incorrectly computed with probability $\overline{P(X_n^y \text{ omitted})} \times \overline{P(X_n^y \text{ delayed})} \times P(X_n^y \text{ corrupted})$ and inserted into set $\mathcal{I}_n$; or **(iv)** transmitted timely and correctly with probability $\overline{P(X_n^y \text{ omitted})} \times \overline{P(X_n^y \text{ delayed})} \times \overline{P(X_n^y \text{ corrupted})}$, and thus inserted into set $\mathcal{C}_n$. The recursion terminates when all cases have been exhaustively enumerated, i.e., $\mathcal{Z}_n = \emptyset$ and $\mathcal{O}_n \cup \mathcal{D}_n \cup \mathcal{I}_n \cup \mathcal{C}_n = X_n$.

Note that Definition 6.6 could be rephrased alternatively without the use of recursion by simply enumerating all possible cases (i.e., all possible values of $\mathcal{O}_n$, $\mathcal{D}_n$, $\mathcal{I}_n$, $\mathcal{C}_n$, and $\mathcal{Z}_n$), associating with each case a case probability and a conditional probability that U_n^y is incorrect, and then summing up the product of respective case and conditional probabilities. However, the recursive formulation helps in proving that $P(U_n^y \text{ incorrect})$ is monotonic with respect to the exact probabilities. In particular, for each step of the recursion, we only need to prove monotonicity with respect to error probabilities of message X_n^s (and not of other messages in X_n); whereas the recursive call is independent of the error probabilities of message X_n^s, which simplifies the monotonicity proof.

In case of Definition 6.6, however, we show in Appendix A.1 that $P(U_n^y \text{ incorrect})$ is not monotonically increasing in the omission and delay probabilities. In fact, for any message $X_n^s \in X_n$, its monotonicity in $P(X_n^s \text{ omitted})$ and $P(X_n^s \text{ delayed})$ depends on $P(X_n^s \text{ corrupted})$. This is because the overall failure probability could be reduced by simply delaying or omitting a message, if that message is likely to be incorrectly computed and thus has the potential to tilt the voting outcome in favor of an incorrect quorum. In other words, $P(U_n^y \text{ incorrect})$ can be decreased by increasing either $P(X_n^s \text{ delayed})$ or $P(X_n^s \text{ corrupted})$, or both. To get around this problem, we define instead an upper bound on $P(U_n^y \text{ incorrect})$ that compensates for all non-monotonic terms in Definition 6.6 by adding a residual term for the recursive case $\mathcal{Z}_n \neq \emptyset$. A detailed proof of monotonicity for the upper bound is provided in Appendix A.2.

DEFINITION 6.7. An upper bound on the probability that U_n^y is incorrect is given by $Q(U_n^y \text{ incorrect} \mid \langle \emptyset, \emptyset, \emptyset, \emptyset, X_n \rangle)$, where

$$Q\left(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle\right) = \begin{cases} P(\text{SimpleMajority incorrect} \mid \mathcal{I}_n, \mathcal{C}_n) & \mathcal{Z}_n = \emptyset \\ \Gamma'_1 + \Gamma'_2 + \Gamma'_3 + \Gamma'_4 + \Gamma'_5 & \mathcal{Z}_n \neq \emptyset \end{cases},$$

$$\begin{aligned}\Gamma_1' =& Q(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n \cup \{X_n^s\}, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \setminus \{X_n^s\}\rangle) \\ &\times P(X_n^s \text{ omitted}),\end{aligned}$$

$$\begin{aligned}\Gamma_2' =& Q(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n \cup \{X_n^s\}, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \setminus \{X_n^s\}\rangle) \\ &\times \overline{P(X_n^s \text{ omitted})} \times P(X_n^s \text{ delayed}),\end{aligned}$$

$$\begin{aligned}\Gamma_3' =& Q(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n \cup \{X_n^s\}, \mathcal{C}_n, \mathcal{Z}_n \setminus \{X_n^s\}\rangle) \\ &\times \overline{P(X_n^s \text{ omitted})} \times \overline{P(X_n^s \text{ delayed})} \times P(X_n^s \text{ corrupted}),\end{aligned}$$

$$\begin{aligned}\Gamma_4' =& Q(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n \cup \{X_n^s\}, \mathcal{Z}_n \setminus \{X_n^s\}\rangle) \\ &\times \overline{P(X_n^s \text{ omitted})} \times \overline{P(X_n^s \text{ delayed})} \times \overline{P(X_n^s \text{ corrupted})},\end{aligned}$$

$$\begin{aligned}\Gamma_5' =& \; Q(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n \cup \{X_n^s\}, \mathcal{C}_n, \mathcal{Z}_n \setminus \{X_n^s\}\rangle) \\ &\times \left(\begin{array}{l} \overline{P(X_n^s \text{ omitted})} \times P(X_n^s \text{ delayed}) \times P(X_n^s \text{ corrupted}) \\ + P(X_n^s \text{ omitted}) \times P(X_n^s \text{ corrupted}) \end{array} \right),\end{aligned}$$

and X_n^s denotes the message with the smallest ID in $\mathcal{Z}_n$. The probability $Q(U_n^y \text{ incorrect} \mid \langle \emptyset, \emptyset, \emptyset, \emptyset, X_n \rangle)$ is also denoted as $Q(U_n^y \text{ incorrect})$.

Definition 6.7 differs from Definition 6.6 in two ways. First, while terms Γ_1', Γ_2', Γ_3', and Γ_4' in Definition 6.7 are similar to terms Γ_1, Γ_2, Γ_3, and Γ_4 in Definition 6.6, they rely on $Q(U_n^y \text{ incorrect} \mid \ldots)$ instead of $P(U_n^y \text{ incorrect} \mid \ldots)$. Second, case $\mathcal{Z} \neq \emptyset$ in Definition 6.7 is defined using an addition term Γ_5', which, as shown in Appendix A.2, ensures that the upper bound is monotonic in the exact message error probabilities. $Q(U_n^y \text{ incorrect})$ thus yields a monotonic upper bound on the probability that U_n^y is incorrect.

6.3.1.2 *Analyzing whether U_n^y is Omitted*

In this step, we evaluate the probability that the output of controller task C_n^y's voter instance U_n^y is omitted because all its inputs were either delayed or omitted, i.e., the special case in Algorithm 6.1 (Line 7). Similar to Step 1, we state the probability as a recursive expression, relying on the exact probabilities in Definition 6.1-Definition 6.3, as well as on the error status in Definition 6.5, as follows.

DEFINITION 6.8. The probability that U_n^y is omitted is given by $P(U_n^y \text{ omitted} \mid \langle \emptyset, \emptyset, \emptyset, \emptyset, X_n \rangle)$, where

$$P\left(U_n^y \text{ omitted} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle\right) = \begin{cases} \Lambda_1 + \Lambda_2 + \Lambda_3 + \Lambda_4 & \mathcal{Z}_n \neq \emptyset \\ 1 & \mathcal{I}_n \cup \mathcal{C}_n = \emptyset \\ 0 & \mathcal{I}_n \cup \mathcal{C}_n \neq \emptyset \end{cases},$$

$$\Lambda_1 = P(U_n^y \text{ omitted} \mid \langle \mathcal{O}_n \cup \{X_n^s\}, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \setminus \{X_n^s\} \rangle) \times P(X_n^s \text{ omitted}),$$

$$\Lambda_2 = P(U_n^y \text{ omitted} \mid \langle \mathcal{O}_n, \mathcal{D}_n \cup \{X_n^s\}, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \setminus \{X_n^s\} \rangle) \times \overline{P(X_n^s \text{ omitted})} \times P(X_n^s \text{ delayed}),$$

$$\Lambda_3 = P(U_n^y \text{ omitted} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n \cup \{X_n^s\}, \mathcal{C}_n, \mathcal{Z}_n \setminus \{X_n^s\} \rangle) \times \overline{P(X_n^s \text{ omitted})} \times \overline{P(X_n^s \text{ delayed})} \times P(X_n^s \text{ corrupted}),$$

$$\Lambda_4 = P(U_n^y \text{ omitted} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n \cup \{X_n^s\}, \mathcal{Z}_n \setminus \{X_n^s\} \rangle) \times \overline{P(X_n^s \text{ omitted})} \times \overline{P(X_n^s \text{ delayed})} \times \overline{P(X_n^s \text{ corrupted})},$$

and X_n^s denotes the message with the smallest ID in $\mathcal{Z}_n$. The probability $P(U_n^y \text{ omitted} \mid \langle \emptyset, \emptyset, \emptyset, \emptyset, X_n \rangle)$ is also denoted as $P(U_n^y \text{ omitted})$.

Definition 6.8 has the same termination condition as Definition 6.6 and Definition 6.7 in Step 1, i.e., $\mathcal{Z}_n = \emptyset$. However, the evaluation of the probability for the termination condition in Definition 6.8 is different. In particular, when evaluating the probability of message omission, it suffices to check whether the voter has *some* input to work with, in which case the voter output is certainly not omitted. The probability of omission is thus zero when $\mathcal{I}_n \cup \mathcal{C}_n \neq \emptyset$ and one otherwise.

In addition, as a result of this difference in termination condition, Definition 6.8 does not depend on the correctness of inputs used to compute U_n^y or (consecutively) on the simple majority procedure in Algorithm 6.1, but only on the timeliness of these inputs. Hence, Definition 6.8's monotonicity in exact probabilities $P(X_n^s \text{ omitted})$ and $P(X_n^s \text{ delayed})$ does not depend on $P(X_n^s \text{ corrupted})$, unlike Definition 6.6 (see Appendix A.3 for a detailed proof). As a result, addition of a residual probability term, such as Γ_5 in Definition 6.7, is not required in this case to obtain monotonicity.

6.3.2 Actuator Voter Output

In this step, we evaluate the probability that the output V_n of the actuator voter task A_n's voter instance is incorrect or omitted.

Recall from the system model description that the network guarantees atomic broadcast and that the NCS tasks are deterministic. Under these assumptions, if any one correct controller voter instance outputs an incorrect value because of wrong inputs (corrupted sensor values), it implies that all correct controller voter instances output incorrect values as well, since all controller voter instances operate on the same input values. In fact, in such a scenario, the actuator voter instance too is guaranteed to get only incorrect control messages, since all of the control messages will be prepared using the corrupted sensor values.

A similar observation 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 6.1). As explained in Section 6.1, if the host clocks are synchronized, delayed messages can be discarded consistently on all correct replicas (i.e., while maintaining the atomic broadcast property). As a result, if any one controller voter instance does not choose any value because all its inputs are delayed or omitted, then all controller voter instances 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.

In light of these correlations between the sensor inputs to the controller tasks and the final actuation of the control loop, analyzing the output of the actuator voter instance is either straightforward if the input control commands are incorrect, delayed, or omitted due to these problematic inputs begin consistently observed at all PEs, or it requires a recursive decomposition approach similar to that used in Section 6.3.1. Since the former case results in a guaranteed failure, we consider it directly in Step 4 (Section 6.3.3) where we analyze the probability that the final actuation Z_n is faulty or omitted. In the following, we only define the probability that the actuator voter instance output V_n is incorrect or omitted for the latter case.

DEFINITION 6.9. $P(V_n \text{ incorrect})$ denotes the probability that V_n is incorrect, conditioned on the assumption that the sensor inputs to the controller voter instances during this iteration did not result in a corrupted output.

DEFINITION 6.10. $Q(V_n \text{ incorrect})$ denotes an upper bound on the probability that V_n is incorrect, conditioned on the assumption that the sensor inputs to the controller voter instances during this iteration did not result in a corrupted output.

DEFINITION 6.11. $P(V_n \text{ omitted})$ denotes the probability that V_n is omitted, conditioned on the assumption that the sensor inputs to the controller voter instances during this iteration did not result in an omitted output.

Probabilities $P(V_n \text{ incorrect})$, $Q(V_n \text{ incorrect})$ and $P(V_n \text{ omitted})$ are defined using the recursive procedures discussed in Section 6.3.1, respectively, by replacing the set of voter inputs X_n with Y_n (recall from the system model in Section 6.1 that Y_n denotes the set of all inputs to the actuator voter instance during the nth control loop iteration). For monotonicity reasons, same as in Section 6.3.1, the upper bound in Definition 6.10 is introduced since the exact probability in Definition 6.9 is not monotonic. Definition 6.11, on the other hand, is monotonic by itself.

6.3.3 Final Output

In this step, we bound the probability that the final output of the nth control loop iteration, Z_n, is either skipped or incorrect.

We first bound the probability that the actuation during the nth control loop iteration is incorrect, followed by the probability that it is omitted, and finally the joint probability of both events (see Definition 6.12-Definition 6.14, respectively).

DEFINITION 6.12. An upper bound on the probability that the actuation during the nth control loop iteration is incorrect is given by

$$\begin{aligned} Q(Z_n \text{ incorrect}) = & \left(\begin{array}{l} P(Z_n \text{ corrupted}) + Q(U_n^y \text{ incorrect}) \\ + Q(V_n \text{ incorrect}) \end{array} \right) \\ & + \left(\begin{array}{l} P(Z_n \text{ corrupted}) \times Q(U_n^y \text{ incorrect}) \\ \times Q(V_n \text{ incorrect}) \end{array} \right), \end{aligned}$$

for any $U_n^y \in U_n$.

DEFINITION 6.13. An upper bound on the probability that the actuation during the nth control loop iteration is skipped is given by

$$\begin{aligned} Q(Z_n \text{ skipped}) = & \left(\begin{array}{l} P(Z_n \text{ omitted}) + P(U_n^y \text{ omitted}) \\ + P(V_n \text{ omitted}) \end{array} \right) \\ & + \left(\begin{array}{l} P(Z_n \text{ omitted}) \times P(U_n^y \text{ omitted}) \\ \times P(V_n \text{ omitted}) \end{array} \right), \end{aligned}$$

for any $U_n^y \in U_n$.

The upper bound in Definition 6.12 (and in Definition 6.13) is derived by considering the two cases described in Step 3, i.e., whether the sensor inputs result in a consistent corruption (omission, respectively) of the controller voter instance outputs, or not; and then dropping any negative terms for ensuring monotonicity. A detailed proof for both the upper bounds is given in the Appendix A.4.

In Definition 6.14, we compose the probability upper bounds in Definition 6.12 and Definition 6.13 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 probability upper bounds in Definition 6.12 and Definition 6.13 are mutually 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. However, the negative term corresponding to mutual dependence is dropped for the sake of preserving monotonicity.

DEFINITION 6.14. An upper bound on the probability that the n^{th} control loop iteration fails, i.e., the actuation during the n^{th} control loop iteration is either incorrect or skipped, is given by

$$Q(n^{\text{th}} \text{ control loop iteration fails}) = \begin{pmatrix} Q(Z_n \text{ incorrect}) \\ + Q(Z_n \text{ skipped}) \end{pmatrix}.$$

In summary, Definitions 6.6 to 6.14 account for all direct and indirect dependencies between the individual message error events and the final actuation of the controlled plant, and the probability upper bound $Q(n^{\text{th}} \text{ control loop iteration fails})$ automates propagation of the exact message error probabilities 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 Sections 6.3.1 and 6.3.2, since the number of replicas of any task is likely small, i.e., typically under five, the analysis can be quickly performed.

6.4 ANALYSIS INSTANTIATION

As mentioned in the analysis overview (Section 6.2), since the exact error probabilities in Definition 6.1-Definition 6.4 are impossible to obtain, we instantiate the analysis presented in the previous section with upper bounds on these exact probabilities. Given that the analysis is monotonically increasing in the exact probabilities, soundness is guaranteed despite the use of upper bounds.

We next define upper bounds on the individual message error probabilities and an upper bound on the probability that the simple majority procedure in Algorithm 6.1 outputs an incorrect value.

Like in Section 6.2, let m denote a message carrying a sensor value (i.e., one of the messages in X_n), a message carrying a control command (i.e., one of the messages in U_n), or the final actuation command

Z_n that is applied to the physical plant. Recall the description of the fault-induced errors and the modeling of their arrivals from Section 3.3. In particular, recall that $\gamma_{err}(\text{comp})$ denotes the peak rate at which component *comp* experiences errors belonging to class *err*, and that $\mathcal{P}(x, \delta, \gamma_{err}(\text{comp}))$ denotes the probability that x instances of such errors occur in any interval of length δ on component *comp*.

An upper bound on $P(m \text{ omitted})$ depends on whether the host from which message m is transmitted experiences a crash error or not. For instance, suppose that message m's sender task is deployed on host H_m, and that message m 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). If R_m is the maximum time to recover from a crash error on host H_m, and if there is at least one crash error during the interval $[t - R_m,\ t + J)$, message m's arrival may be skipped. Thus,

$$P(m \text{ omitted}) \leqslant \sum_{x>0} \mathcal{P}(x,\ R_m + J,\ \gamma_{crash}(H_m)). \tag{6.1}$$

An upper bound on $P(m \text{ corrupted})$ can be similarly obtained by evaluating the probability that there is at least one incorrect computation error during the exposure interval of message m. Recall from Section 3.3.2 that the *exposure interval* of a message denotes the interval during which it is exposed to and can be potentially corrupted by incorrect computation PE errors. Thus, if $E(m)$ denotes the exposure interval of message m, and if m's sender task is deployed on host H_m,

$$P(m \text{ corrupted}) \leqslant \sum_{x>0} \mathcal{P}(x,\ E(m),\ \gamma_{corrupt}(H_m)). \tag{6.2}$$

While we can use a similar method to upper-bound $P(m \text{ delayed})$, i.e., evaluate the probability that there is at least one retransmission error on the network during message m's transmission window, the resulting upper bound would be extremely pessimistic since real-time workloads are typically provisioned assuming interference from a finite number of such retransmissions. Instead, more accurate network timing analyses could be used, such as the one proposed by Broster et al. [39] in the context of CAN, or our prior work [88] that simultaneously analyses timing properties of multiple message replicas.

Next, we upper-bound the probability that the SimpleMajority($\mathcal{I} \cup \mathcal{C}$) procedure in Algorithm 6.1 outputs an incorrect value, i.e., probability $P(\text{SimpleMajority incorrect} \mid \mathcal{I},\ \mathcal{C})$, given that $\mathcal{C}$ and $\mathcal{I}$ denote the sets of correct and incorrect inputs, respectively. To upper-bound $P(\text{SimpleMajority incorrect} \mid \mathcal{I},\ \mathcal{C})$, we make the worst-case assumption that incorrect inputs in $\mathcal{I}$ are identically faulty. Suppose that $s_0 \in \mathcal{C} \cup \mathcal{I}$ denotes the message in $\mathcal{C} \cup \mathcal{I}$ with the smallest ID. Recall that any ties in quorum size while computing the simple majority (Algorithm 6.1, Line 8) are broken deterministically using message IDs.

Let $n_c = |\mathcal{C}|$ and $n_i = |\mathcal{I}|$. If $n_i > n_c$, the largest-sized quorum belongs to incorrect messages, and the simple majority 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 ($s_0 \in \mathcal{I}$), the simple majority is incorrect; otherwise ($s_0 \in \mathcal{C}$), it is correct. If $n_i < n_c$, the largest-sized quorum belongs to correct messages, and the simple majority is again correct. If $n_i = n_c = 0$, the voter has received no inputs, so the probability of choosing an incorrect output in this case is also 0. Considering all of these cases,

$$P\left(\begin{array}{c}\text{SimpleMajority}\\ \text{incorrect}\end{array}\middle|\ \mathcal{I}, \mathcal{C}\right) \leqslant \begin{cases} 1 & n_i > n_c \\ 1 & n_i = n_c \neq 0 \wedge s_0 \in \mathcal{I} \\ 0 & n_i = n_c \neq 0 \wedge s_0 \in \mathcal{C} \\ 0 & n_i < n_c \\ 0 & n_i = n_c = 0 \end{cases} \quad (6.3)$$

THE IID PROPERTY Since each of the upper bounds defined above is independent of n, the upper bound in Definition 6.14 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 [39, 54]; and 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 Definition 6.14 for any two iterations n_1 and n_2 are mutually independent as well. As a result, when $Q(n^{th}$ 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. The IID property is useful for a long-run analysis of the system across all its iterations, e.g., for evaluating metrics such as MTTF and FIT, which is discussed in Chapter 7.

ANALYSIS INSTANTIATION FOR OTHER NETWORKS The analysis can similarly be instantiated for other types of networks. Instantiation for CAN-like field buses is trivial; only the analysis to upper-bound $P(m$ delayed$)$ must be altered as per the protocol specifications. On the other hand, for point-to-point networks like Ethernet, the analysis to upper-bound $P(m$ delayed$)$ must be updated to take into account the end-to-end delay encountered by message m across every transmission step. Use of an Achal-like system in an Ethernet-based NCS to ensure replica coordination does not affect the analysis presented in this chapter, since the FIT rate of the replica coordination protocol would be separately computed and added to the system-wide SOFR analysis.

6.5 EVALUATION

The objective of the evaluation is to assess the accuracy of the proposed reliability analysis of an NCS iteration. To achieve this objective, we compare the analytically-derived bound on the iteration failure probability (using Definition 6.14) with an estimate of the mean iteration failure probability obtained through simulation.

Since timing analysis of network messages is an integral component of our reliability analysis, we implemented the proposed analysis using the SchedCAT (Schedulability test Collection And Toolkit) library [33]. We extended SchedCAT to support CAN-based FT-SISO control loops, and implemented Broster et al.'s probabilistic response-time analysis [38] for CAN messages as the underlying timing analysis of the network. To ensure correct rounding in floating-point computations involving very small probabilities, 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 [234]. As a baseline, we also implemented a discrete-event simulation of a CAN-based FT-SISO control loop along with CAN's network transmission protocol (see Section 2.1.3.1 for a detailed description).

WORKLOAD AND PARAMETERS We base our experiments on a fault-tolerant version of the CAN-based active suspension workload studied by Anta and Tabuada [8], since it nicely matches our FT-SISO model and since active suspension (see [132] for more details) plays an important role in ensuring the stability of a vehicle. The workload consists of tasks and messages corresponding to four control loops (L_1, L_2, L_3, and L_4), each of which corresponds to the control of four wheels (W_1, W_2, W_3, and W_4) with magnetic suspensions and executes with a time period of 1.75 ms. In addition, the workload consists of two hard real-time messages that report the current in the power line cable and the internal temperature of the coils. Both these messages are critical to the NCS and are transmitted every 4 ms and 10 ms, respectively.

Since we assume that hosts have synchronized clocks, we assumed the presence of clock synchronization messages with a period of 50 ms based on the protocol by Gergeleit and Streich [84]. We also assumed a soft real-time message responsible for logging (which is common in many CPS) with a period of 100 ms. Note that these additions were not part of the workload studied by Anta and Tabuada [8].

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 (recall from Section 2.1.3.1 that each CAN message can carry up to eight bytes of payload). Considering a bus rate of 1 Mbit/s (CAN buses are typically operated at bus rates of 256 kbit/s, 512 kbit/s, 1 Mbit/s, or

4 Mbit/s), the workload resulted in a total bus utilization of 40 %. The clock synchronization message stream had the highest priority (which is required as per Gergeleit and Streich's scheme), followed by the current and temperature monitoring message streams (since these were carrying hard real-time messages without any redundancy), followed by the control message streams, and last, the logging message stream.

The recovery time from a crash was set to $R_h = 1\,\text{s}$ 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 (recall from Section 6.4 that these are necessary to upper-bound the iteration failure probability). The error rates used in each experiment are mentioned along with the experiment descriptions. All error rates are reported as the mean number of errors per ms.

For context, Ferreira et al. [72] and Rufino et al. [188] reported peak transmission error rates range from 10^{-4} in aggressive environments to 10^{-10} in lab conditions, and as per Hazucha and Svensson [97], a 4 Mbit SRAM chip has a fault rate of approximately 10^{-12}. However, the error rates used in the following experiments are relatively higher than realistically expected values as otherwise the simulations would be extremely time-consuming.

EXPERIMENT SETUP Recall from Section 6.3 that:

1. the analysis first upper-bounds the control loop iteration failure probability as a monotonic function of the exact message error probabilities; and

2. since it is impossible to determine the exact message error probabilities, a safe upper bound on the iteration failure probability is then obtained by instantiating the monotonic function from (1) with upper bounds on the exact message error probabilities.

To separately evaluate the pessimism incurred in steps (1) and (2), we used two different simulator versions `Sim-v1` and `Sim-v2`. These are similar to those used in Section 5.5 for evaluating the pessimism incurred in the IC protocol analysis.

In the simple version (`Sim-v1`), for each sensor message (and similarly for each control message), the message error probabilities were known to the simulator. Thus, each time any message 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 (1).

The second version `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

CONFIGURATION	$\gamma_{\text{crash}}(H_i)$	$\gamma_{\text{corrupt}}(H_i)$	$\gamma_{\text{retransmission}}$
A	10^{-4}	10^{-20}	3×10^{-20}
B	10^{-20}	10^{-4}	3×10^{-20}
C	10^{-20}	10^{-20}	3
D	10^{-5}	10^{-5}	3×10^{-1}

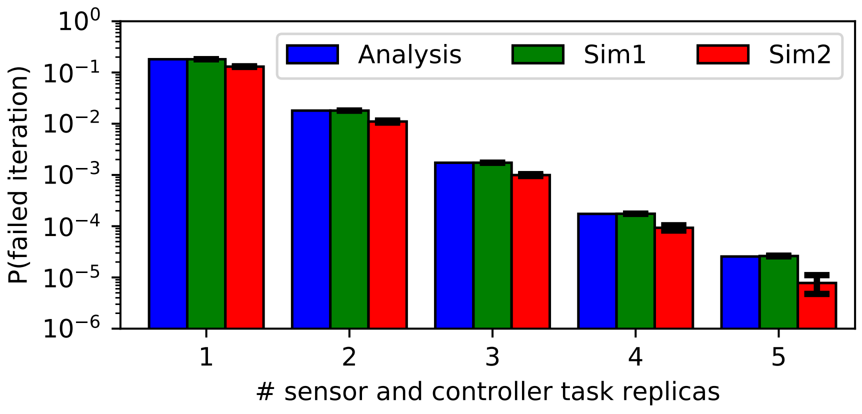
Table 6.2: Error rate configurations used for evaluation. Notations $\gamma_{\text{crash}}(H_i)$, $\gamma_{\text{corrupt}}(H_i)$, and $\gamma_{\text{retransmission}}$, denote the omission error rate on host H_i, the incorrect computation error rate on host H_i, and the retransmission error rate on the CAN bus, respectively. Highlighted error rates indicate non-negligible values.

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 using the Poisson model, e.g., when using the Poisson-based CAN timing analysis [39] to determine bounds on deadline violation probabilities. It also evaluates whether this pessimism significantly impacts the overall iteration failure probability bound.

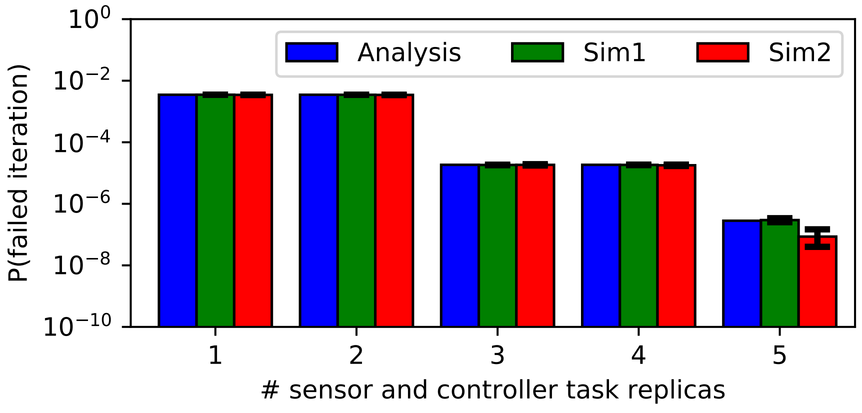
Both `Sim-v1` and `Sim-v2` make the worst-case assumption that any two faulty message copies are identical, as in the analysis. Thus, any pessimism due to this assumption is not evaluated.

We compared the analysis, `Sim-v1`, and `Sim-v2` for four different sets of error rates, which are enumerated in Table 6.2. To understand the effects of individual error types, in each of the first three configurations, one of the three error types was assigned a non-negligible error rate, i.e., $\gamma_{\text{crash}}(H_i) = 10^{-4}$ (Configuration A), $\gamma_{\text{corrupt}}(H_i) = 10^{-4}$ (Configuration B), and $\gamma_{\text{retransmission}} = 3$ (Configuration C), respectively, whereas the other error rates were assigned negligible values. Additionally, in Configuration D, all three error rates were assigned non-negligible values, i.e., $\gamma_{\text{crash}}(H_i) = 10^{-5}$, $\gamma_{\text{corrupt}}(H_i) = 10^{-5}$, and $\gamma_{\text{retransmission}} = 3 \times 10^{-1}$. For each configuration, the number of sensor and controller task replicas of L_1 were varied from one to five. The results are illustrated in Figs. 6.3 and 6.4.

We also compared the failure probabilities for different CAN bus utilizations (by assuming increased message payload sizes) and for different reboot times (100 ms to 2000 ms), with a replication factor of three. For the first experiment with varying bus utilizations, we used Configuration D (where all error types are assigned non-negligible error rates). For the second experiment with different reboot times, we used Configuration A (where crash errors dominate). The results for these experiments are illustrated in Figs. 6.5a and 6.5b, respectively.



(a) Configuration A (crash errors dominate)



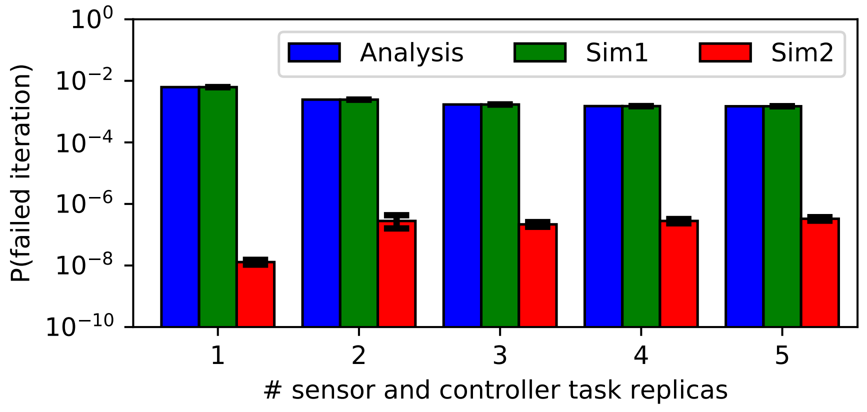
(b) Configuration B (corruption errors dominate)

Figure 6.3: Results for configurations A and B. See Table 6.2 for the corresponding error rates.

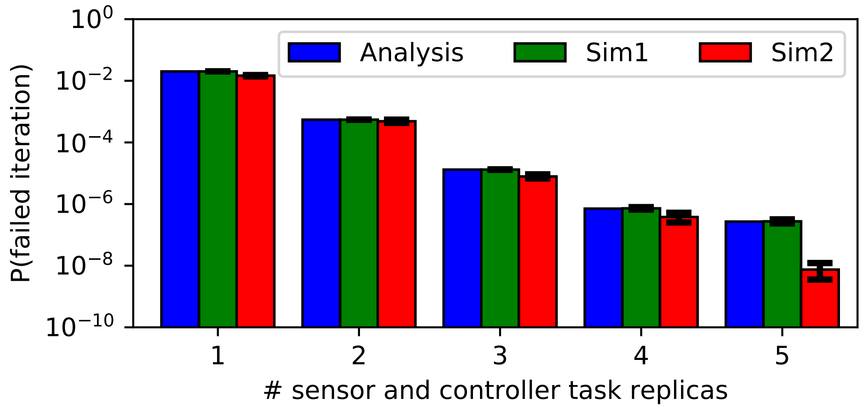
For each experiment, simulations were run for 10,000,000 iterations to compute high-confidence failure probability estimates along with 99 % confidence intervals (which are shown as vertical errors bars).

RESULTS AND 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 (1) to ensure monotonicity of the model with respect to the exact error rates is negligible.

The results shown in Figs. 6.3a, 6.3b and 6.4b 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, as is the case in Fig. 6.4a). 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 the dominant source of failures. This confirms the overall accuracy of



(a) Configuration C (retransmission errors dominate)



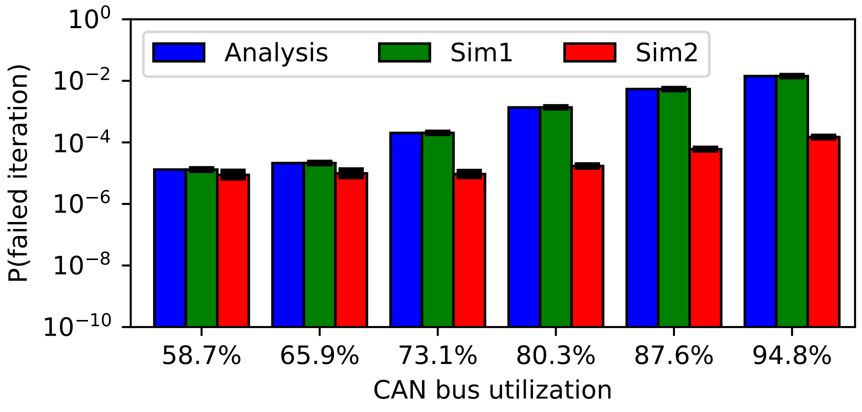
(b) Configuration D (all error types assigned non-negligible rates)

Figure 6.4: Results for configurations C and D. See Table 6.2 for the corresponding error rates.

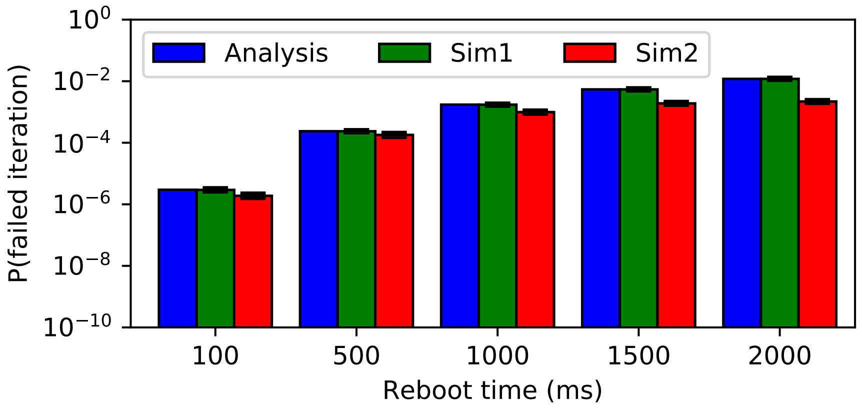
the approach for the intended use cases: the proposed analysis closely tracks and soundly bounds the actual iteration failure probabilities in the presence of crashes, retransmissions, and message corruptions.

However, as is evident from Figs. 6.3a, 6.3b and 6.4b, 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. 6.4a, 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. 6.5a indicates that the gap between `Sim-v2` and the analysis increases with CAN bus utilization. And even in Fig. 6.3a, when the replication factor is increased to five (resulting in high network contention), `Sim-v2` begins to deviate from the analysis. We attribute



(a) Configuration D with varying CAN bus utilization



(b) Configuration A with varying reboot times

Figure 6.5: Variation in the iteration failure probability when the CAN bus utilization and the reboot times are increased. The results are illustrated in insets (a) and (b), respectively. Configurations D and A (see Table 6.2) were used to obtain the results in (a) and (b), respectively. In each case, three sensor and controller task replicas of L_1 were configured.

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, which is rather unlikely in the case of realistic error rates. That is, the extremely high error rates assumed in this experiment for the sake of simulation speed exaggerate the impact of the CAN analysis.

Finally, Fig. 6.5b 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 errors in this scenario (i.e., $\gamma_{\text{crash}}(H_i) = 10^{-4}$ per ms, 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 scales).

7 FROM ITERATION TO SYSTEM FAILURE*

Quantifying the reliability of a CPS involves bounding the probability that a failure occurs in any one step of operation, as well as bounding the probability of failure of the entire system. These are distinct problems because many CPS are designed to continue to operate safely even in the presence of occasional failures. This is especially true for NCS applications, which are routinely designed to be robust to occasional failures.

** This chapter is based on our CERTS 2017 [90] and ECRTS 2019 [89] papers.*

For example, Majumdar et al. [141] describe an NCS where the control system continues using the previous iteration parameters in case the current iteration is dropped. Using networked control techniques [35], they also provide methods to estimate a maximum dropout rate tolerated by a control system without compromising its stability (e.g., they show that an inverted pendulum control system with mass $0.5\,\mathrm{kg}$, length $0.20\,\mathrm{m}$, and sampling time $10\,\mathrm{ms}$ remains asymptotically stable with at least $76.51\,\%$ successful control loop iterations). Recently, Pazzaglia et al. [172] used the intrinsic robustness of well-designed controllers to propose a novel Deadline-Miss-Aware Control (DMAC) strategy, which can be implemented in a real-time task that may miss some deadlines.

In general, prior studies [22, 35, 45, 95, 141, 142, 172] have demonstrated that a control system can be (and typically is) designed to withstand occasionally failing control loop iterations, without compromising its intended service (i.e., the first control loop iteration failure does not denote a full-system failure). We denote such well-designed control systems as *temporally robust*.

In this chapter, we address the problem of bounding the long-run failure probability of periodic systems (NCS applications being a specific example), given a specification of their temporal robustness and given bounds on their per-iteration failure probabilities (which can be derived using the analysis presented in Chapter 6). In particular, we consider the problem of soundly and accurately estimating the Mean Time To Failure (MTTF) (or equivalently, the Failures-In-Time (FIT) rate) of a periodic control system whose temporal robustness is expressed as one or more *weakly-hard constraints*.

As an example, consider the (m, k) constraint, which is one of the simplest forms of weakly-hard constraints. It specifies that a periodic system remains functional as long as at least m iterations in any window of k consecutive iterations are successful. The temporal robustness of the inverted pendulum control system discussed above (i.e., asymptotic stability with at least 76.51 % successful iterations) directly translates to an (m, k) constraint with $m = 77$ and $k = 100$.

In many cases, such a single (m, k) constraint may not be sufficient to satisfy other performance specifications (such as settling time), and must be appended with an additional short-range "liveness" constraint. For example, given a sampling time of $10\,ms$, the inverted pendulum control system would surely crash if it experienced 33 consecutive dropouts. In such cases, the temporal robustness of the control system is better specified using either a harder constraint (e.g., $m = 4$ and $k = 5$ instead of $m = 77$ and $k = 100$) or multiple constraints (e.g., using both $m_1 = 77$ and $k_1 = 100$ as well as $m_2 = 1$ and $k_2 = 4$) [27]. The objective of this chapter is thus to use the temporal robustness property of control systems, specified using one or more generic weakly-hard constraints, for estimating their long-run reliability from the per-iteration failure probabilities.

We start by discussing the limitations of prior work from the reliability modeling literature in estimating MTTF/FIT of periodic systems with weakly-hard constraints, and characteristics expected of in an ideal MTTF/FIT analysis (Section 7.1). We then formalize the MTTF/FIT estimation problem as the expectation of the stopping time of a stochastic process (Section 7.2), and propose three orthogonal analyses, PMC, MART, and SAP, with different trade-offs (Section 7.3). We provide an empirical evaluation of these techniques in terms of their accuracy and numerical precision, their expressiveness for different definitions of weakly-hard constraints, and their space and time complexities, which affect their scalability and applicability in different regions of the space of weakly-hard constraints (Section 7.4).

Finally, using the analyses from this chapter and the previous chapter, we explore how different weakly-hard parameters (e.g., different values of m and k) and different error rates impact the reliability estimates of a temporally robust NCS (Section 7.5). We use an active suspension workload (the same as in Section 6.5) for this exploration.

7.1 PRIOR WORK AND OBJECTIVES

Weakly-hard constraints have been widely studied in the context of firm real-time systems to represent robustness of a time-sensitive task against occasional timing failures [22, 41, 95, 180, 183]. In particular, the focus has been on analyzing task schedulability according to a given weakly-hard (usually (m, k)) constraint [180, 181], design of online schedulers to meet these constraints [22, 41, 95], and co-design approaches to find the schedulable set of (m, k) parameters that maximizes an application's quality of service [45, 114, 209].

Most recently, Pazzaglia et al. [171] introduced state-based representation of the evolution of a control system with respect to deadline misses, and showed the merits of having multiple (m, k) constraints for a control application. Similarly, Kauer et al. [114] derived a bound on the consecutive message drops an architecture can experience, and translated it to a set of (m, k) constraints.

However, none of these works provides a means for bounding a system's MTTF with respect to its weakly-hard specification.

In contrast, in the general reliability literature, there is a long tradition of work on deriving a system's MTTF if the occurrence of failures is described by well-known probability distributions (see [124] for a comprehensive overview). Similarly, the problem of evaluating the reliability of series- or parallel-redundant systems, both with and without repairs, in the context of robustness specifications such as k-out-of-n, consecutive-k-out-of-n, multidimensional consecutive-k-out-of-n, etc. is well understood, e.g., see [174, 192]. However, the available techniques in this domain do not directly apply to the problem at hand. Either the constraints cannot be reduced to these techniques or symbolically integrating the applicable technique over an infinite domain is non-trivial. Further, for multiple weakly-hard specifications, a model-based approach helps to account for dependencies.

Therefore, even given a bound on per-iteration failure probability, soundly characterizing the overall MTTF/FIT rate remains challenging. While simulation-based methods can be used to estimate the MTTF/FIT of weakly-hard periodic systems, they do not yield exact answers—they may even under-approximate the true failure rate—and scale poorly, especially when analyzing low-probability events. In fact, an ideal MTTF/FIT analysis must satisfy three requirements:

- It must be *generic* or *expressive* enough to support complex weakly-hard requirements in order to stand for the needs of larger and more complicated systems.
- Further, it must be *accurate*, ideally, *exact*, to minimize pessimism in the final system reliability.
- Last, but not least, it must be *scalable* with respect to the problem size, since capturing asymptotic properties requires dealing with large problem windows.

To respond to each of these requirements, we propose and compare three approaches for MTTF/FIT analysis: PMC, MART, and SAP.

PMC (Probabilistic Model Checking) models the problem as an expected reward problem in a discrete-time Markov chain, which can be solved using state-of-the-art probabilistic model checkers such as PRISM [125] and STORM [55]. PMC is able to express complex robustness constraints as well as sophisticated system models with state-dependent probabilities of failure, such as in [171].

For the special case of Bernoulli systems, where failure probabilities are independent and identically distributed (IID), martingale theory

APPRAOCH	ACCURACY	SCALABILITY	EXPRESSIVENESS
PMC	Exact	Poor	General sys., all properties
MART	Exact	Poor	IID systems, all properties
SAP	Approx.	Good	IID systems, single (m, k)

Table 7.1: Approaches to MTTF/FIT derivation.

allows for a direct approach that we call MART. It constructs a system of linear equations, whose solution gives the expected time to failure, and is therefore able to leverage powerful linear algebra routines such as the Linear Algebra PACKage (LAPACK) [127] and Basic Linear Algebra Subprograms (BLAS) [16]. Like PMC, MART provides an exact analysis, too, and can support general weakly-hard constraints, but both PMC and MART have limited scalability.

To scale to large window-size constraints, we introduce SAP (Sound Approximation), an empirically-driven, scalable, and yet sound, approach designed to evaluate a *single* (m, k) constraint.

The tradeoffs of the three proposed techniques, which are all sound by construction, are summarized in Table 7.1.

7.2 SYSTEM MODEL

We model the problem of computing a system's MTTF/FIT as the expected stopping time of a stochastic process. To that end, we model a periodic system S abstractly as a stochastic process $(X_n)_{n \geqslant 0}$ evolving in discrete time. We assume that system S is periodic with a period of T time units, i.e., the observation X_n is emitted at time nT. Each random variable X_n is boolean-valued: $X_n = 1$ indicates that S executes correctly in its n^{th} period and $X_n = 0$ indicates S executes incorrectly. An *execution* of system S is a string in $\{0, 1\}^*$ denoting an outcome of the stochastic process $(X_n)_{n \geqslant 0}$. We emphasize that S is *not* just a single, periodic task, but the entire system, divided into logical iterations. For example, one iteration of the system may involve end-to-end execution of a set of real-time tasks and message exchanges, as in the CAN-based NCS with active replication analyzed in Chapter 6, or the NCS over Achal/Ethernet analyzed in Chapter 5.

Failure probabilities in system S can be modeled as in a *Bernoulli system* where each observation X_n is an Independent and Identically Distributed (IID) Bernoulli variable, with $\Pr[X_i = 0] = P_F$ and $\Pr[X_i = 1] = 1 - P_F$. Such a system represents a periodic system where errors occur independently in each iteration, and the probability of error in each iteration is (bounded by) P_F. It can also represent periodic systems where errors in multiple iterations are dependent, but the bound P_F derived for each iteration is independent of the iteration (this is possible if P_F is derived pessimistically assuming the worst-possible error scenario, which is a common approach in the analysis of hard real-time systems, and also used in Chapters 5 and 6).

Alternatively, to capture history-dependence in failures and more accurate iteration-specific error scenarios, the failure probabilities can be modeled more expressively using a *discrete-time labeled Markov chain* [17]. In this case, the system is modeled as a set of states Q and a probabilistic transition function $\Pr(s' \mid s) : Q \times Q \mapsto [0, 1]$ that specifies the probability with which the system transitions from state s at any step n to state s' at step $n + 1$. Each state is labeled with a Boolean variable denoting success (1) or failure (0), and observation X_n is the label of the (random) state at step n.

Next, we formalize *robustness specifications* to capture the intuition that a periodic system, such as a well-designed controller, continues to provide overall acceptable service despite individual iteration failures, as long as there are not "too many" such iteration failures. In particular, we characterize the set of safe executions for which a periodic system is guaranteed to provide its service as a prefix-closed[1] set of executions $\mathcal{R} \subseteq \{0, 1\}^*$. Thus, the intersection of two robustness specifications is again a robustness specification.

We focus on the classic (m, k), $\langle m, k\rangle$, and $\overline{\langle m\rangle}$ weakly-hard robustness specifications, which have been originally proposed in the context of *firm* real-time systems that can tolerate a limited number of deadline misses [21] (see Section 2.1.2.2 for more details).

Formally, an execution $w \in \{0, 1\}^*$ is (m, k) robust if every window of size k has at least m successes, i.e.,

$$\forall u, v, w' : w = uw'v \wedge |w'| = k \Rightarrow \pi_1(w') \geqslant m, \tag{7.1}$$

where $\pi_1(w)$ denotes the number of 1's in w; it is $\langle m, k\rangle$ robust if every window of size k has at least m consecutive successes, i.e.,

$$\forall u, v, w' : w = uw'v \wedge |w'| = k \Rightarrow \exists u', v' : w' = u'1^m v'; \tag{7.2}$$

and $\overline{\langle m\rangle}$ robust if there are never more than m failures in a row, i.e.,

$$\nexists u', v' : w = u'0^{m+1}v'. \tag{7.3}$$

For a given system, one can be interested in several robustness specifications simultaneously, e.g., to express both asymptotic properties (such as "no more than 5% failed iterations") and short-term requirements (such as "no more than two iteration failures in a row"). Thus, for example, we can ask that a system is (m_1, k_1) robust and also $\overline{\langle m_2\rangle}$ robust. This just means that executions of the system satisfy

1 Recall that a set is *prefix-closed* if whenever an execution belongs to the set, all prefixes of the execution also belong to the set.

both the (m_1, k_1) constraint and the $\overline{\langle m_2 \rangle}$ constraint. In general, given a set of robustness specifications, an execution is considered correct if it satisfies all the specifications in the set.

Given a periodic system S and its robustness specification $\mathcal{R}$, we next define its MTTF based on the definition in Section 2.2.2.

Let a *system failure* denote an execution that is not in $\mathcal{R}$. For example, for a system with a robustness specification $(2, 5)$, an execution 010100100 denotes a failure (since the last five iterations consist of only one successful iteration). We assume that system S stops if it encounters a system failure, and therefore to compute the MTTF and FIT we are interested in a failing execution whose proper prefixes (i.e., prefixes excluding the last iteration) satisfy the robustness specification. Accordingly, given a robustness specification $\mathcal{R}$, we define the *stopping time* of system S as a random variable

$$N(S, \mathcal{R}) = \min \left\{ n \geqslant 0 \;\middle|\; \begin{array}{l} X_0 \ldots X_n \notin \mathcal{R} \wedge \\ \forall i < n \; X_0 \ldots X_i \in \mathcal{R} \end{array} \right\}. \tag{7.4}$$

The MTTF is the expectation of the stopping time multiplied by the period T of the system,

$$\text{MTTF} = T \sum_{n=0}^{\infty} n \cdot \Pr[N(S, \mathcal{R}) = n]. \tag{7.5}$$

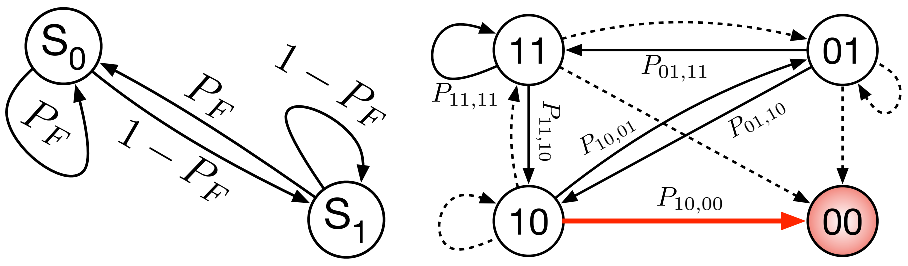
Eq. (7.5) is anologous to Eq. (2.10) in Section 2.2.2, except that the stopping time of system S is defined taking into consideration its robustness specification $\mathcal{R}$. Also recall from Section 2.2.2 that the FIT is simply the inverse of the MTTF, with a human-friendly scale factor, to the effect that the FIT represents the expected number of failures in one billion operating hours. That is, FIT $= 10^9/(\text{MTTF in hours})$.

7.3 PROBABILISTIC ANALYSES

In the following, we propose three approaches for FIT derivation: PMC, MART, and SAP. To explain the techniques in detail, we initially focus on a single (m, k) robustness specification, and discuss the applicability of the respective technique for evaluating a generic set of robustness specifications such as $\{(m_1, k_1), \langle m_2, k_2 \rangle, \overline{\langle m_3 \rangle}\}$ at the end of each section. Wherever a Bernoulli system is considered, P_F is used to denote the probability of a failed iteration, and $P_S = 1 - P_F$.

7.3.1 PMC: Markov Chain Analysis

We provide a method to compute the MTTF by modeling the system as a *labeled discrete-time Markov chain*. Our observation is that computing



(a) Markov chain of a Bernoulli system **(b)** Monitor (Type 1) for $k = 2$

Figure 7.1: PMC approach. In inset (b), $P_{x_1x_2,\,y_1y_2}$ is a shorthand for transition probability $P'(q \mid q')$ where states q and q' have labels $L'(q) = x_1x_2$ and $L'(q') = y_1y_2$, respectively. Transitions with zero probability are marked with dashed arrows. The state labeled 00, which is the only state in $\mathrm{Bad}(1, 2)$, is colored red.

the MTTF reduces to finding the expected total reward in an *absorbing Markov chain*. Our method conceptually works for any *regular* robustness specification (i.e., robustness specifications that can be accepted by a finite automaton), but we focus our discussion on the class of weakly-hard robustness specifications, which we expect to be most widely used in practice, and for concreteness.

Suppose that system S is modeled as a Markov chain $M = (Q, P, L, s_i)$, where Q denotes a finite set of system states, $P : Q \times Q \mapsto [0, 1]$ denotes the transition probability matrix, $L : Q \mapsto \{0, 1\}$ denotes the state labels with 1 and 0 corresponding to *success* and *failure* (respectively), and $s_i \in Q$ denotes the initial state. For example, if S is a Bernoulli system, then M, as illustrated in Fig. 7.1a, consists of states s_0 and s_1 and transition probabilities $P(s_0, s_0) = P(s_1, s_0) = P_F$ and $P(s_0, s_1) = P(s_1, s_1) = 1 - P_F$.

Given the Markov model M and a robustness specification $\mathcal{R} = (m, k)$, we run a *monitor* Markov chain along with M, which is denoted $\mathrm{Monitor}(M, k) = (Q', P', L', q_i)$. The monitor tracks a finite execution history of M of length k to decide whether S has *failed*, i.e., whether there were more than $k - m$ failures in the last k steps. Thus, Q' consists of 2^k states, and each state $q \in Q'$ is labeled with a unique label $L'(q) \in \{0, 1\}^k$, e.g., a label of $1^{k-1}0$ implies that every but the last iteration was successful. Every time M takes a step, the monitor state is updated to reflect the past k steps of M's execution. Thus, the transition probability of $\mathrm{Monitor}(M, k)$ from state q with label w to state q' with label w' is $P'(q, q') = P(s, s')$ if system S can transition from history w to w' by transitioning from state s to s'; otherwise, it is $P'(q, q') = 0$. The initial state $q_i \in Q'$ is labeled 1^k to model absence of any failures during system start.

In addition, recall from Section 7.2 that system S stops as soon as it encounters an execution that does not satisfy (m, k) robustness. To model this aspect, we define $\mathrm{Bad}(m, k)$ as the set of all "bad" states in

Q' and make all these states *absorbing*, i.e., once the monitor enters a state in $\mathrm{Bad}(m, k)$, it does not transition into another state. Formally,

$$\mathrm{Bad}(m, k) = \{(q \mid q \in Q' \wedge L'(q) \notin \mathcal{R}\}. \tag{7.6}$$

As an example, the monitor representation for $\mathcal{R} = (1, 2)$ is illustrated in Fig. 7.1b, with states in $\mathrm{Bad}(2, 3)$ explicitly marked in red.

Given the monitor Markov chain, we reduce the MTTF computation to deriving the expected number of steps until the monitor enters a bad state. For this, assume that each step of the monitor has a reward of 1. We define the *expected number of steps* E as the expected reward until any state in $\mathrm{Bad}(m, k)$ is reached (starting from the initial state $q_i \in Q'$), which can be obtained using probabilistic model checkers such as PRISM [125] and Storm [55]. Thus, if system S has period T, the MTTF of S with respect to robustness specification (m, k) is $T \times E$.

Note that the monitor representation discussed above is independent of m. While the monitor's simple structure makes it trivial to implement, its $O(2^k)$ space complexity can be detrimental in practice. Fortunately, for the common case where $k - m \ll k$, e.g., $(98, 100)$, the monitor representation can be optimized to be much more space efficient. Since the system stops as soon as the (m, k) constraint is violated, we need not keep any executions that have more than $k - m$ failures. In other words, it suffices to store a limited history as a string of length $k - m$, where each element in the string is from $\{1, \ldots, k\} \cup \{\perp\}$, representing the positions along the previous k steps when a failure occurred ($\perp$ is used in case we have seen fewer than $k - m$ failures). Furthermore, we can coalesce all states in $\mathrm{Bad}(m, k)$ into a single "bad" state, resulting in a space complexity of only $O((k+1)^{(k-m)} + 1)$.

As an example, the monitor representation for $\mathcal{R} = (2, 3)$ is illustrated in Fig. 7.2. The optimized monitor representation consists of only five states, whereas otherwise it would have required eight states. Since this monitor representation is more concise, the node labels are not equal to the execution histories, unlike in the simple monitor representation illustrated in Fig. 7.1b, e.g., label '3' indicates an execution history of '110' where the latest iteration has failed.

Similarly, for $m \ll k$, we can optimize the model by storing a history as a string of length m, where each element in the string is from $\{1, \ldots, k\}$. We refer to the three representations, i.e., the default one, the optimized version for $k - m \ll k$, and the optimized version for $m \ll k$, as Type 1, Type 2, and Type 3 models, respectively.[2]

Compared to the aforementioned monitor representations for an $\overline{(m, k)}$ robustness specification, monitor representations for $\langle m, k \rangle$ and $\langle m \rangle$ robustness specifications are both simpler and more efficient.

For $\langle m, k \rangle$ robustness, the monitor needs to keep track of positions corresponding to **(i)** the latest run of 1's of length at least m and **(ii)** the

2 See Appendix C for an encoding of each monitor type in PRISM.

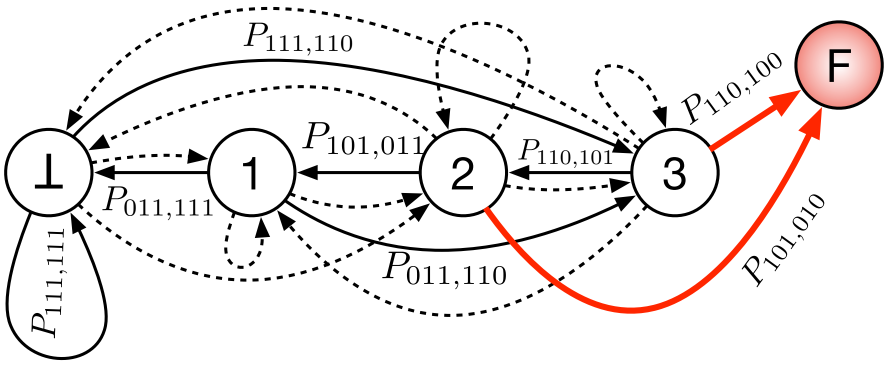


Figure 7.2: Monitor (Type 2) for $(2,3)$. $P_{x_1x_2x_3,\, y_1y_2y_3}$ is a shorthand for $P'(q \mid q')$, where states q and q' correspond to execution histories $x_1x_2x_3$ and $y_1y_2y_3$, respectively. Transitions with zero probability are marked with dashed arrows, and states in $\text{Bad}(2,3)$ are coalesced in to a single state, which is colored in red.

current run of 1's of length at most m. For (i), since the beginning and the end of run can be any element in a window of size k, a string of length two belonging to $\{1 \ldots k\}^2$ is needed, whereas for (ii), since the current run must always include the latest element, a string of length one belonging to $\{1 \ldots m\}$ is sufficient. In both cases, $\perp$ can be used to denote the absence of a run, resulting in a space complexity of $O((k+1)^2 \cdot (m+1))$. For $\overline{\langle m \rangle}$ robustness, the monitor can be simplified even further, since we only need one accumulator to store the current sequence of consecutive 0's, and so the space complexity is $O(m)$.

For a generic robustness specification of the form $\mathcal{R} = \{(m,k), \langle m', k' \rangle, \overline{\langle m'' \rangle}\}$, we run the monitor for each specification in parallel, and set Bad to denote states where *some* monitor is in a bad state.

7.3.2 MART: The Martingale Approach

While the PMC approach allows modeling history-dependent failures, in the special case of Bernoulli systems, there is a direct and elegant approach based on the martingale theory to deriving a linear system of equations whose solution provides the expected stopping time. We summarize this approach for (m,k) robustness next.

The first step is similar to enumerating the "bad" states of the monitor Markov chain in the PMC approach. In particular, we list all *failure strings* over $\{0,1\}^k$ that correspond to a violation of the (m,k) constraint: these are strings of length up to k in which at least $k-m+1$ failures occur. We do this by fixing the last position to be a failure and then choosing all possible combinations of $k-m$ indices from the set $\{1,\ldots,k\}$. There are $O(k^{(k-m)})$ such strings.

In the second step, given an exhaustive list of failure strings, we reduce the problem of computing MTTF to that of computing the *expected waiting time* until one of the failure strings is realized by the system execution. To find the expected waiting time, we use an elegant algorithm from the theory of occurrence patterns in repeated experiments proposed by Li [133]. Li's algorithm translates the failure strings into a set of linear equations, such that solving these linear equations directly yields an expected waiting time for each individual failure string (i.e., until a specific failure string is realized by the system) as well as an expected waiting time until any of the failure strings manifests. To compute the MTTF, we require only the latter.

We summarize Li's algorithm in the following. Let $\Pi = \{\pi_1, \pi_2, \ldots\}$ be the set of failure strings obtained in the first step. Let $|\pi_i|$ denote the length of a string $\pi_i \in \Pi$, and let $\pi_{i,j}$ denote the j^{th} character in string π_i. Key to Li's algorithm is a combinatorial operator '$*$' (Eq. 2.3 in [133]) between any pair of strings π_a and π_b from Π:

$$\begin{aligned} \pi_a * \pi_b = {} & (\delta_{1,1}\delta_{2,2}\ldots\delta_{x,x}) + (\delta_{2,1}\delta_{3,2}\ldots\delta_{x,x-1}) \\ & + \ldots + (\delta_{x-1,1}\delta_{x,2}) + (\delta_{x,1}), \end{aligned} \tag{7.7}$$

where

$$\delta_{i,j} = \begin{cases} \frac{1}{P_F} & \text{if } i \in [1,x],\ j \in [1,y],\ \pi_{a,i} = \pi_{b,j} = 0 \\ \frac{1}{P_S} & \text{if } i \in [1,x],\ j \in [1,y],\ \pi_{a,i} = \pi_{b,j} = 1 \\ 0 & \text{otherwise,} \end{cases}$$

$x = |\pi_a|$, and $y = |\pi_b|$.

Using this operator, the expected waiting time e_0 until any one of the sequence patterns in Π occurs for the first time satisfies the following linear system of $n = |\Pi|$ equations for vector $\langle e_0, e_1, \ldots, e_n \rangle$.

$$\begin{bmatrix} 0 & 1 & 1 & \ldots & 1 \\ -1 & \pi_1 * \pi_1 & \pi_2 * \pi_1 & \ldots & \pi_n * \pi_1 \\ -1 & \pi_1 * \pi_2 & \pi_2 * \pi_2 & \ldots & \pi_n * \pi_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ -1 & \pi_1 * \pi_n & \pi_2 * \pi_n & \ldots & \pi_n * \pi_n \end{bmatrix} \begin{bmatrix} e_0 \\ e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{7.8}$$

Thus, if S has period T, the MTTF is given by $e_o \times T$.

In the following, we show a step-by-step computation of the MTTF for an example periodic system S using the MART approach.

EXAMPLE Consider a system with period 5 ms, iteration failure probability bounded by $P_F = 0.1$, and robustness specification $(2, 3)$, i.e., at most one 0 is allowed in any execution of length three.

The set of all strings over $\{0, 1\}^3$ that violate $(2, 3)$ robustness and end in a failure is given by $\Pi = \{00, 010, 100\}$. Using Eq. (7.7), $\pi_2 * \pi_2$, for example, is computed as follows.

$$\pi_2 * \pi_2 = \delta_{1,1}\delta_{2,2}\delta_{3,3} + \delta_{2,1}\delta_{3,2} + \delta_{3,1}$$

$\{\text{since } \pi_{2,2} \neq \pi_{2,1},\, \delta_{2,1} = 0\}$

$$= \delta_{1,1}\delta_{2,2}\delta_{3,3} + \delta_{3,1}$$

$\{\text{since } \pi_{2,1} = \pi_{2,3} = 0,\, \delta_{1,1} = \delta_{3,3} = 1/P_F = 10\}$

$$= 10 \cdot \delta_{2,2} \cdot 10 + \delta_{3,1}$$

$\{\text{since } \pi_{2,3} = \pi_{2,1} = 0,\, \delta_{3,1} = 1/P_F = 10\}$

$$= 10 \cdot \delta_{2,2} \cdot 10 + 10$$

$\{\text{since } \pi_{2,2} = 1,\, \delta_{2,2} = 1/P_S = 10/9\}$

$$= 10 \cdot \frac{10}{9} \cdot 10 + 10 = \frac{1090}{9}.$$

Other $\pi_a * \pi_b$'s can be similarly computed, resulting in the following system of linear equations:

$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ -1 & 110 & 10 & 110 \\ -1 & 10 & \frac{1090}{9} & 10 \\ -1 & 10 & \frac{100}{9} & \frac{1000}{9} \end{bmatrix} \begin{bmatrix} e_0 \\ e_1 \\ e_2 \\ e_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \tag{7.9}$$

which yields $e_0 = 62.63$ and $\text{MTTF} = e_0 \times 5 = 313.15\,\text{ms}$.

Asymptotically, the aforementioned MART approach has the same complexity as the PMC approach. Solving the expected reward until absorption in a Markov chain to compute the MTTF (as per the PMC approach) also reduces to the problem of solving a system of n linear equations, where n denotes the number of states in the Markov model [17]. However, since the MART approach directly provides us with the final set of linear equations, we can leverage mature linear algebra libraries (such as LAPACK [127] and BLAS [16]), to compute the MTTF in a more scalable way than PMC (see Section 7.4 for details).

In addition, with the MART approach, accounting for a generic set of robustness specifications, such as $\{(m_1, k_1),\, \langle m_2, k_2 \rangle,\, \overline{\langle m_3 \rangle}\}$, is relatively straightforward in comparison to PMC. We need to modify only the first step of MART to obtain an appropriate set of failure strings that corresponds to violation of any of the robustness specifications, which is used as before to instantiate the system of linear equations defined in Eq. (7.8). However, it must be ensured that any two patterns $\pi_a, \pi_b \in \Pi$ do not contain one another [133]. This is possible if, for example, the failure patterns for constraints $(95, 100)$ and $\overline{\langle 3 \rangle}$ are merged. For such cases, the longer pattern is removed from Π, since the shorter pattern occurs first.

7.3.3 SAP: Sound Approximation

We present next an approximate analysis with the objective of scaling it to large values of k. The presented analysis SAP is *sound*, that is, it estimates an approximate value of the MTTF that lower-bounds the MTTF as given by exact analyses PMC and MART. Like MART, SAP can be used only for Bernoulli systems. Unlike PMC and MART though, SAP is applicable only for a single (m, k) robustness constraint; it does not support constraints of the form $\langle m, k \rangle$ or $\overline{\langle m \rangle}$, or combinations thereof.

SAP consists of two key steps. Recall the definition of MTTF from Section 7.2. For brevity, let $g(n) = \Pr[N(S, \mathcal{R}) = n]$. In the first step, we derive a lower bound on $g(n)$, denoted $g_{LB}(n)$. For this, we split the (m, k) robustness specification into three conditions, compute an exact or lower bound on the probability for each of these conditions, and then compute a product of these probabilities. In the second step, as per Eq. (7.5), we integrate $n \cdot g_{LB}(n)$ numerically (but in a sound manner) to strictly lower-bound the MTTF of system S. The two steps are discussed in detail below.

For S to violate the (m, k) specification for the first time during its n^{th} iteration, the following three conditions must hold.

E_1: The n^{th} iteration must fail.

E_2: Exactly $k - m$ iterations must fail out of the $k - 1$ iterations between the $(n - k + 1)^{\text{th}}$ and the $(n - 1)^{\text{th}}$ iteration.

E_3: Fewer than $k - m + 1$ iterations fail out of any k consecutive iterations, among the first $n - 1$ iterations.

Then $g(n) = \Pr(E_1) \times \Pr(E_2) \times \Pr(E_3)$. Now, $\Pr(E_1) = P_F$, and summing over all possible combinations of $k - m$ iteration failures in $k - 1$ consecutive iterations yields $\Pr(E_2) = \binom{k-1}{k-m} P_F^{(k-m)} P_S^{(m-1)}$. However, obtaining the exact value of $\Pr(E_3)$ is challenging.

To tackle this challenge, we use the *a-within-consecutive-b-out-of-c:F* model [124, §11.4] (or *a/Con/b/c:F* in short), proposed originally for a system that consists of c ($c \geqslant a$) linearly ordered components and that fails iff at least a ($a \leqslant b$) components fail among any b consecutive components. Thus, in terms of the (m, k) constraint, for $a = k - m + 1$, $b = k$, and $c = n - 1$, a successful execution of an a/Con/b/c:F system is equivalent to condition E_3, and the *reliability* of an a/Con/b/c:F system, whose approximations have been well studied in the past, yields $\Pr(E_3)$. Since we are interested in a sound approximation, we reuse the reliability lower bound $R_{LB}(a, b, c)$ of the a/Con/b/c:F system as proposed by Sfakianakis et al. [199].[3] Using this reliability

3 $R_{LB}(a, b, c)$ is defined unambiguously for all possible cases in Appendix B.

lower bound and the definitions of $\Pr(E_1)$ and $\Pr(E_2)$, we define a lower bound $g_{LB}(n)$ on $g(n)$ as

$$g_{LB}(n) = \binom{k-1}{k-m} P_F^{(k-m+1)} P_S^{(m-1)} R_{LB}\left(k-m+1, k, n-1\right). \tag{7.10}$$

The next step is to use $g_{LB}(n)$ for lower-bounding the system's MTTF. This requires solving Eq. (7.5), but with $g_{LB}(n)$ in place of $\Pr[N(S,\mathcal{R}) = n]$. Unfortunately, we were not able to obtain a closed-form solution with current symbolic solvers due to the complicated definition of $g_{LB}(n)$. In particular, $g_{LB}(n)$ is defined in terms of $R_{LB}(k-m+1, k, n-1)$, which is a recursive expression with complex definitions of its subproblems, as is explained in detail in Appendix B.

Therefore, similar to numerical integration methods, we adopt an empirical solution for MTTF derivation that is both fast and reasonably accurate. We empirically compute the value of function $g_{LB}(n)$ at finitely many sampling points $d_0, d_1, d_2, \ldots, d_D \in \mathbb{N}$ such that $d_0 = k-m+1$, and $d_0 < d_1 < d_2 < \ldots < d_D$. Using the empirically-determined values $g_{LB}(d_0)$, $g_{LB}(d_1)$, ..., $g_{LB}(d_D)$, we then define a lower bound on the MTTF as follows.[4]

$$\mathrm{MTTF}_{LB} = \sum_{i=0}^{D-1} \Big(d_i T \times g_{LB}(d_{i+1}) \times (d_{i+1} - d_i) \Big) \tag{7.11}$$

Since scalability is the primary motivation for SAP, we choose $D \ll d_D$, so that MTTF_{LB} can be quickly computed from Eq. (7.11). We further choose the sampling points $d_1, \ldots, d_D$ to minimize the amount of pessimism introduced by numerical integration. Another source of inaccuracy is the use of the reliability lower bound $R_{LB}(a, b, c)$ proposed by Sfakianakis et al. [199], which inherently introduces some pessimism. We discuss the choice of sampling points in detail in Section 7.4, and compare SAP with PMC and MART in terms of accuracy.

7.4 EVALUATION

In this section, we discuss implementation choices and challenges, compare the three types of Markov chain models discussed in Section 7.3.1, and then explore the scalability versus accuracy tradeoffs of PMC, MART, and SAP. Since the approximate analysis SAP is not applicable to generic robustness specifications as defined in Section 7.2, and since (m, k) constraints are the limiting factor when it comes to scaling up the analysis, we focus on Bernoulli systems and a single (m, k) constraint in the evaluation. In the end, we revisit the strengths and weaknesses of each approach.

4 Eq. (7.11) is derived in Appendix B.

All experiments were carried out on Intel Xeon E7-8857 v2 machines with 4x12 cores and 1.5TB of memory.

IMPLEMENTATION CHOICES AND CHALLENGES We realized PMC using the state-of-the-art probabilistic model checker PRISM [125].[5] However, configuring PRISM properly to ensure that the estimated results are both accurate and sound is not trivial. PRISM provides many different configuration options that affect the method used for linear equation solving (e.g., Jacobi, Gauss-Seidel, etc.), the model checking engine (MTBDD, Sparse, Hybrid, or Explicit), parameters for precision tuning (i.e., the *epsilon* value and maximum number of iterations for convergence checks during iterative linear solving), and even options to select *exact* (with arbitrary precision) or *parametric* model checking (where some model parameters are not fixed). Choosing the right set of options is thus important because they can significantly affect the estimated MTTF, as we show next.

With the parametric model checking option, PRISM outputs the MTTF as a function of parameter P_F, e.g., the MTTF for $(2, 4)$ is:

$$T \times \frac{P_F^5 - 3P_F^4 + 3P_F^3 - 2P_F^2 - P_F - 1}{P_F^{10} - 4P_F^9 + 6P_F^8 - 5P_F^6 - 3P_F^5 + 4P_F^4 - 3P_F^3}\ .$$

Parametric model checking is thus an ideal choice since it allows for fast reliability analysis across a range of failure probabilities without the need to build and check the model repeatedly. However, as we show later, parametric model checking is also the costliest analysis approach. Thus, for scalability purposes, we also considered both exact and non-exact alternatives to parametric model checking.

We observed that non-exact model checking resulted in significant inaccuracy. For example, Table 7.2 reports the MTTF results for specification $(2, 4)$ obtained with non-exact model checking (using PRISM's Explicit engine) and with exact model checking. The non-exact engine did not converge (first row of the table) for default configuration options. For $P_F = 10^{-10}$, even upon decreasing the epsilon value and increasing the maximum number of iterations, the estimated MTTF is several orders of magnitude off from the exact value, indicating the sensitivity of non-exact model checking to small probabilities. In our evaluation of PMC, we thus worked only with parametric and exact model checking, which we denote as PMC-P and PMC-E, respectively.

The MART approach was implemented in C++ using the *Elemental* library [69], since it uses LAPACK-based routines [127] for solving linear equations, allows for arbitrary precision using the GNU MPFR library [217], and is parallelized using OpenMPI [167]. SAP was implemented in Python using the *mpmath* library [234] for arbitrary

5 See Appendix C for PMC encoded in the PRISM modeling language, and an empirical comparison of PRISM with STORM [55], a more recent probabilistic model checker.

ENGINE	ITERATIONS	EPSILON	MTTF FOR $P_F = 10^{-2}$	MTTF FOR $P_F = 10^{-10}$
Explicit	10^{04}	10^{-06}	–	–
	10^{09}	10^{-06}	3.36×10^{05}	0.23×10^{15}
	10^{09}	10^{-10}	3.41×10^{05}	1.21×10^{17}
Exact	N/A	N/A	3.41×10^{05}	3.33×10^{29}

Table 7.2: MTTF values derived using PRISM engines.

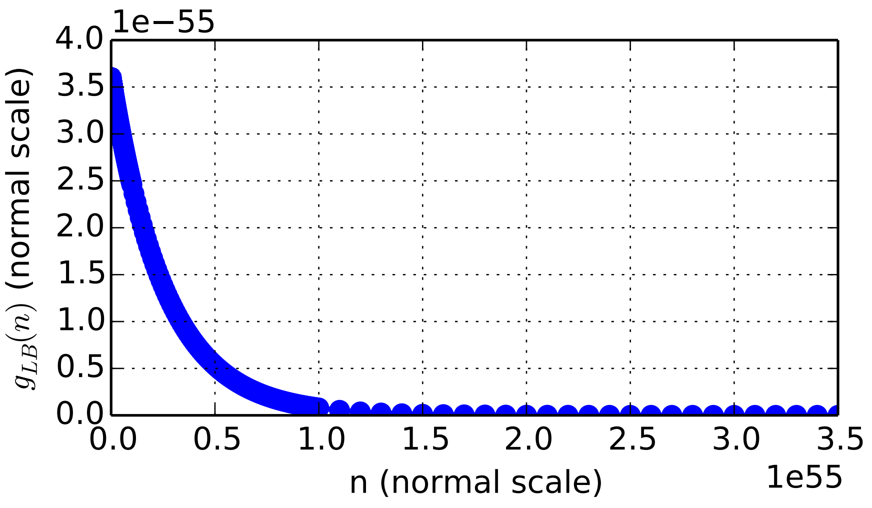
PRECISION	$P_F = y \cdot 10^{-10}$	$P_F = y \cdot 10^{-30}$	$P_F = y \cdot 10^{-50}$
10	-2.20×10^{-00}	-3.96×10^{-01}	-1.42×10^{-00}
20	$+1.81 \times 10^{-04}$	-2.70×10^{-04}	$+3.04 \times 10^{-04}$
30	$+3.39 \times 10^{-07}$	-5.26×10^{-07}	$+1.36 \times 10^{-06}$
40	-2.75×10^{-10}	$+1.20 \times 10^{-09}$	-2.00×10^{-09}
50	-1.89×10^{-14}	$+2.99 \times 10^{-13}$	-4.80×10^{-13}

Table 7.3: % errors in FIT for $\mathcal{R} = (8, 10)$ and $y = 1.234{,}567{,}89$.

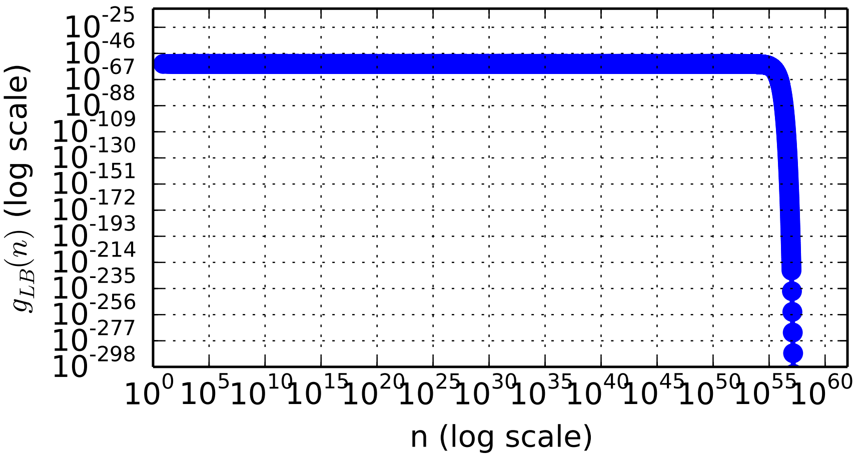
precision. Thus, for MART and SAP, unlike for PMC-E, we could explicitly set the global working *precision*, i.e., the number of decimal digits used to represent the floating point significand.

However, the choice of the global working precision was not obvious. Table 7.3 reports the percentage errors in the estimated FIT when the precision is varied from 10 to 50, with respect to the FIT estimated using a precision of 1000. The results indicate that low precision may result in significant errors if P_F is also small, and sometimes, the results can even be unsafe (i.e., resulting in negative errors). In general, estimating a precision that is safe to use based on the computations involved requires rigorous analysis, e.g., [110]. To be on the safe side, we used a precision of 1000 for MART and SAP, which ensures that any remaining errors are of negligible magnitude.

Finally, when implementing SAP, recall that we need a mechanism to choose an appropriate set of data points $d_0, d_1, d_2, \ldots, d_D$ over which to run the empirical computations. We discuss this mechanism with the help of an example. Let $m = 3$, $k = 10$, and $P_F = 10^{-7}$. In Fig. 7.3, we illustrate $g_{LB}(n)$ given these parameters. Since MTTF_{LB} depends on $g_{LB}(n)$, the key idea is to ensure that points $d_0, d_1, d_2, \ldots,$ d_D are sufficient to trace the shape of function $g_{LB}(n)$, and that the magnitude of $g_{LB}(n)$ is negligible beyond $n = d_D$. The first point d_0, as mentioned before, is set to $(k - m + 1)$. To compute the last point d_D, i.e., the point at which $g_{LB}(n)$ becomes negligible, we observed the logarithm of function $g_{LB}(n)$ for $n \in \{1, 10^1, 10^2, 10^3, \ldots\}$. That is, we plotted the function $g_{LB}(n)$ on a logarithmic scale for both the x-



(a) Normal-scale axes



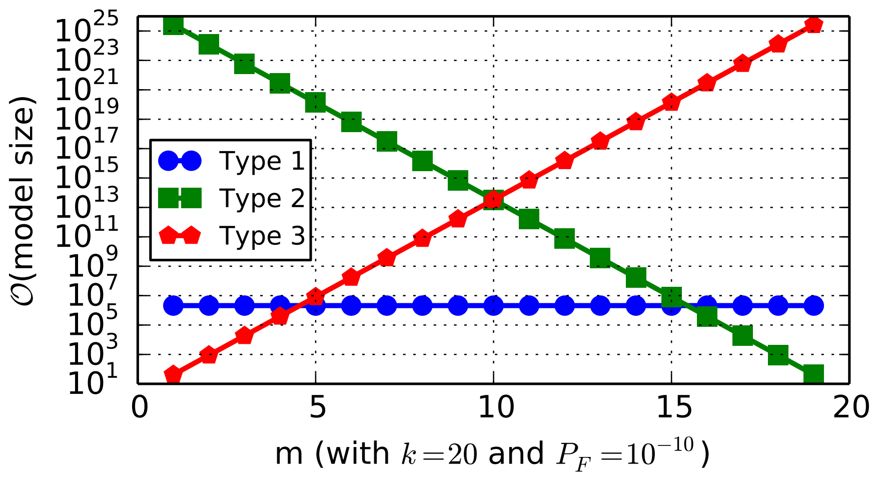
(b) Log-scale axes

Figure 7.3: MTTF estimation using the SAP approach. Insets (a) and (b) illustrate the sampling points $g_{LB}(d_0), g_{LB}(d_1), \ldots, g_{LB}(d_D)$ for $\mathcal{R} = (3, 10)$, $P_F = 10^{-7}$, and $T = 10ms$ in normal scale and log scale, respectively. In this example, $D = 5050$ and $d_D = 9.90 \times 10^{57}$.

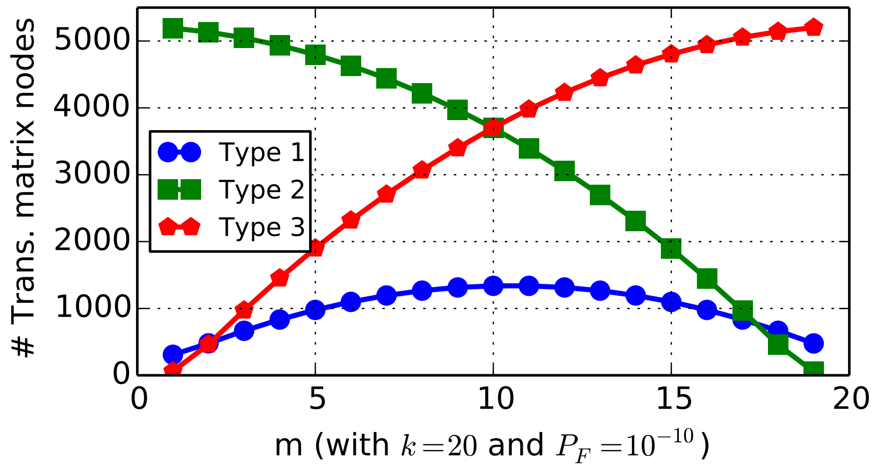
and y-axes as in Fig. 7.3b, and then determined a threshold at which the curve starts falling rapidly (e.g., $d_D \approx 10^{55}$ in Fig. 7.3b).

The intermediate points $d_1, d_2, \ldots, d_{D-1}$ were chosen such that the step size $d_{i+1} - d_i$ between any two consecutive points d_i and d_{i+1} **(i)** is small enough to closely track the function $g_{LB}(n)$, and **(ii)** yet still proportional to the order of magnitude of d_i, to avoid evaluating an exponential number of points. For example, while generating Fig. 7.3, the step size was 1 for $n \in (10, 100]$ and 10^{52} for $n \in (10^{53}, 10^{54}]$.

PMC MODEL TYPES Recall from Section 7.3.1 that we introduced three different types of Markov chain models, Type 1, Type 2, and



(a)

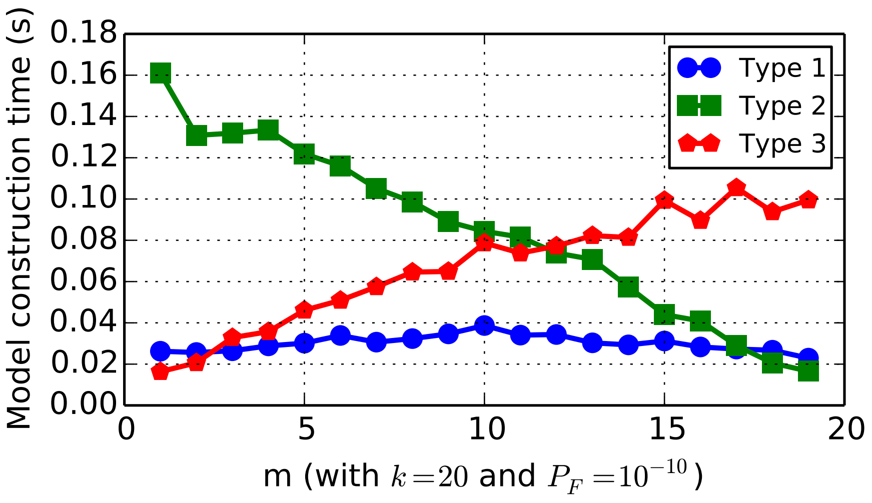


(b)

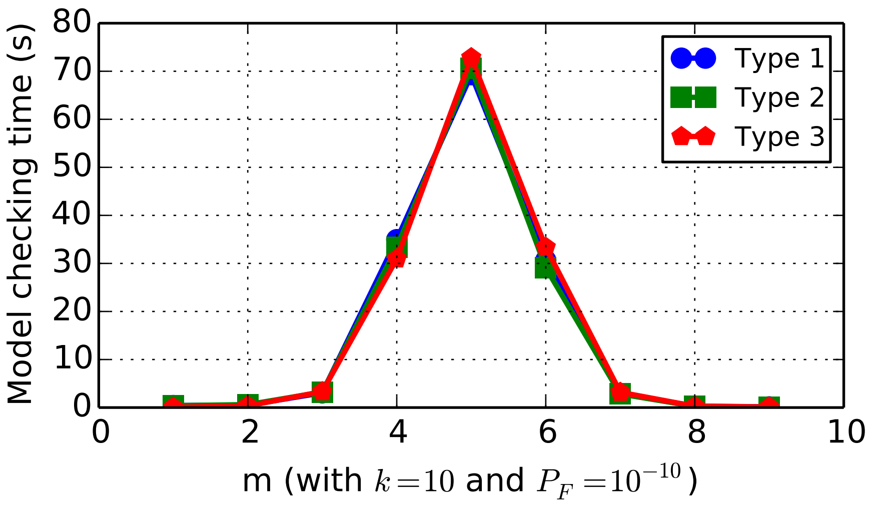
Figure 7.4: Asymptotic model size and the number of nodes in the transition matrix (as reported by PRISM) for the three PMC models.

Type 3, each resulting in a different asymptotic model size. Does the use of one model over the other affect the computation times or even the model building times in practice? To answer this question, we measured the asymptotic model sizes for $k = 20$ and $m \in [1, k-1]$, and compared the measurements with the model size and build time statistics reported by PRISM. We also measured the checking time statistics for $k = 10$ (since model checking for $k = 20$ frequently timed out). We summarize the results obtained for PMC-E in Figs. 7.4 and 7.5.

Fig. 7.4a plots the asymptotic size for each model type, indicating that none of the models is an optimal choice for all parameters. Fig. 7.4b report the number of elements in the transition matrix as reported by PRISM. The number of transition matrix nodes varies with m in the same way as the asymptotic model size, but the absolute



(a)



(b)

Figure 7.5: Model building and solving times for the three PMC models. Unlike in Figs. 7.4a, 7.4b and 7.5a, $k = 10$ was used in Fig. 7.5b since model checking for $k = 20$ frequently timed out.

numbers are less than the asymptotic sizes. This is because PRISM already prunes some states that are unreachable during the build process. Fig. 7.5a and Fig. 7.5b illustrate the time to build and check the models, respectively. The model construction time for each model type is proportional to the respective model size. The model checking time, however, is independent of the model type, since the models are equivalent and result in the same set of linear equations.

In summary, to achieve maximum scalability, it is important to choose a model that requires the minimum time for construction. In the subsequent experiments, we thus use the asymptotic model sizes as a guideline to choose the appropriate model type for an (m, k)

CONFIGURATION	m	P_F	Results
A	$m = \lfloor k/2 \rfloor$	$P_F = 10^{-10}$	Fig. 7.6a
B	$m = \lfloor k/2 \rfloor$	$P_F = 10^{-20}$	Fig. 7.6b
C	$m = k - 2$	$P_F = 10^{-10}$	Fig. 7.7a
D	$m = k - 2$	$P_F = 10^{-20}$	Fig. 7.7b

Table 7.4: (m, k) and P_F configurations used for evaluation, $k \in [2, 20]$

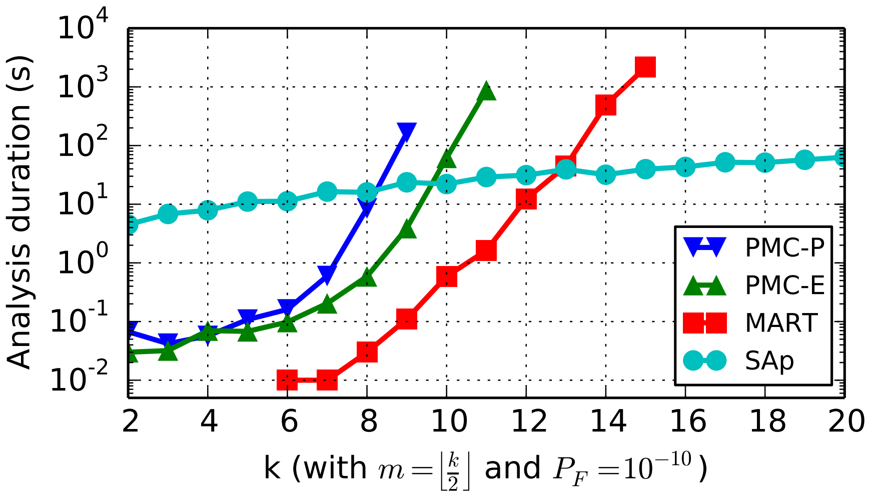
specification. That is, if $k = 20$, based on Fig. 7.4a, we use the Type 3 model if $m \leqslant 4$, the Type 2 model if $m \geqslant 16$, or the Type 1 model.

SCALABILITY VERSUS ACCURACY We start by evaluating the scalability of the analyses PMC-P, PMC-E, MART, and SAP by measuring the analysis duration for each $k \in [2, 20]$, and for four different configurations of m and P_F (see Table 7.4).

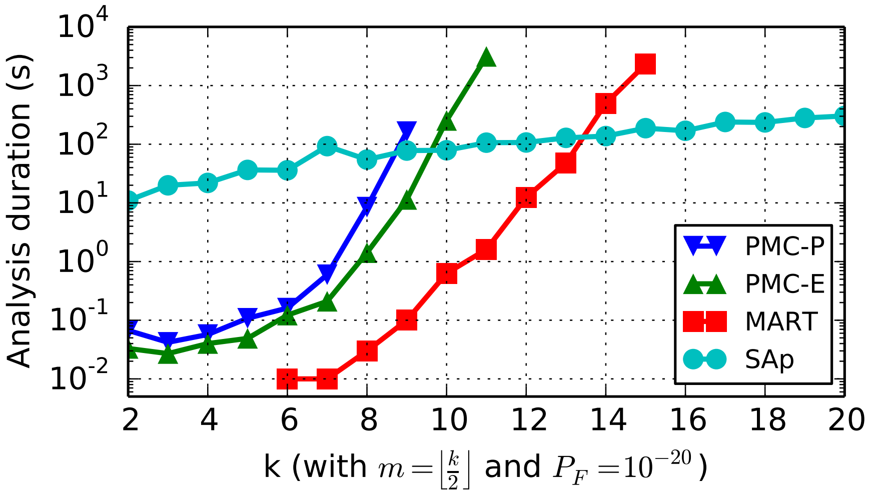
Since evaluating (m, k) requires maximum time if $m = \lfloor k/2 \rfloor$ and minimum time if m is close to either 1 or $k-1$ (see Fig. 7.5b), comparing the results of Configurations A and C (or Configurations B and D) indicate the minimum and maximum scalability that can be achieved by the analyses. In contrast, comparing the results of Configurations A and B (or Configurations C and D) help us to understand the impact, if any, of P_F's value on the analysis scalability. In all cases, a time out of one hour was applied.

First, as evident from Figs. 7.6 and 7.7, and as expected, PMC-P, PMC-E, and MART do not scale well in comparison to SAP. For Configurations A and B, where $m = \lfloor k/2 \rfloor$ (see Fig. 7.6), PMC-P and PMC-E scale only up to $k = 9$ and $k = 11$, respectively. The MART approach performs better and scales up to $k = 15$ for both configurations, mainly because it gives up exactness (but still guarantees soundness owing to its very high precision). In contrast, SAP easily scales up to the maximum value of $k = 20$. Also, notice that while SAP's analysis time grows exponentially in k (the y-axis is log scale), PMC's and MART's analysis times grow super-exponentially. For Configurations C and D, where $m = k - 2$ (see Fig. 7.7), PRISM-based analyses scale better than in the first two configurations because the Type 3 model allows for a concise representation of the (m, k) specification and hence fast building of the model. SAP's scalability also improves significantly in this case because the recursion involved in computing $R_{LB}(k - m + 1,\ k,\ n - 1)$ for the empirical data points is eliminated.

Between Configurations A and B as well as between Configurations C and D, only the failure probability P_F is changed from 10^{-10} to 10^{-20}. As a result, PMC-E takes an order of magnitude more time. This is because lower probabilities require more space for exact representation, and hence more time for computations on these representations. SAP is also affected since the number of data points to be measured is



(a) Configuration A

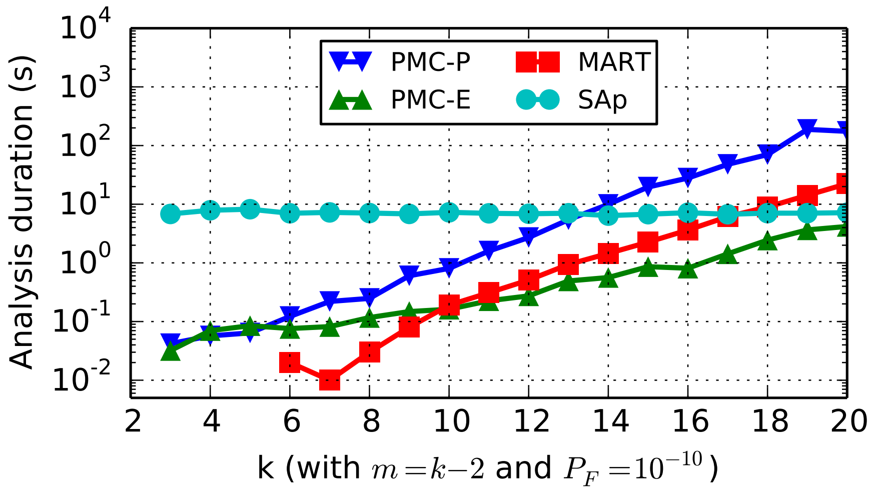


(b) Configuration B

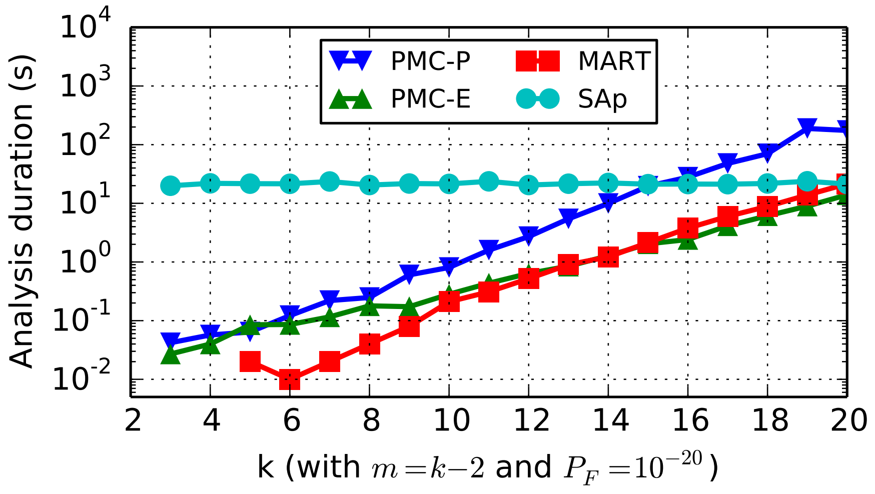
Figure 7.6: Comparing analysis duration for PMC-P, PMC-E, MART, and SAP for Configurations A and B (see Table 7.4). The analysis duration for MART for $k \leqslant 5$ was extremely small and is hence not illustrated.

larger in this case. MART is unaffected because irrespective of P_F, it uses a precision of 1000. Parametric model checking is also unaffected since it is independent of P_F.

To summarize the discussion on analysis scalability, we illustrate in Fig. 7.8 for each $k \in [1, 25]$ and $m \in [2, k-1]$ whether analyses PMC-P, PMC-E, MART, and SAP finished on time, i.e., within a one-hour timeout window. For each cell, P denotes that PMC-P was successful, E denotes that PMC-P timed out but PMC-E was successful, M denotes that both PMC-P and PMC-E timed out but MART was successful, and A denotes that only SAP was successful.



(a) Configuration C

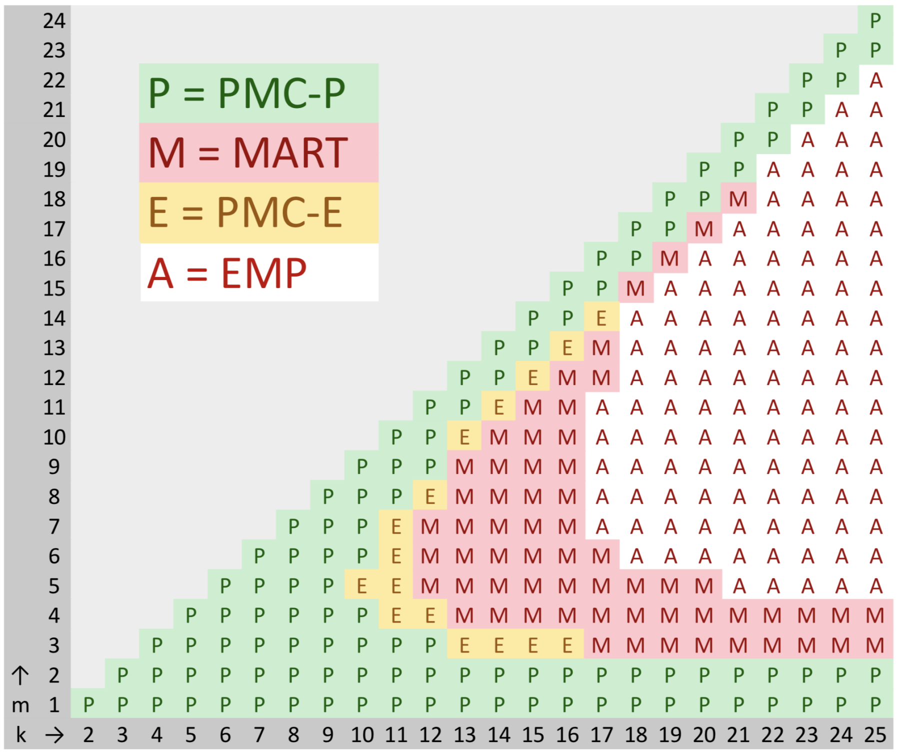


(b) Configuration D

Figure 7.7: Comparing analysis duration for PMC-P, PMC-E, MART, and SAP for Configurations A and B (see Table 7.4). The analysis duration for MART for $k \leqslant 5$ was extremely small and is hence not illustrated. The configuration $k = 2$ was ignored since $(0, 2)$ is not a valid (or rather a trivial) specification.

Clearly, the results indicate that exact analyses can be used only if $k \leqslant 15$, or else if m is either very small or very large relative to k. Thus, for larger values of k, an approximate analysis, such as SAP, is needed, that trades some accuracy for scalability. But is SAP accurate enough to be useful at very large values of k? And is it accurate for small values of k so that the costly exact analyses may not be needed at all? To answer these questions, we evaluate next SAP's accuracy with respect to MART and PMC.

In Fig. 7.9a (similar in structure to Fig. 7.8), we report the percentage error in the MTTF obtained using SAP versus that obtained from


P = PMC-P
M = MART
E = PMC-E
A = EMP

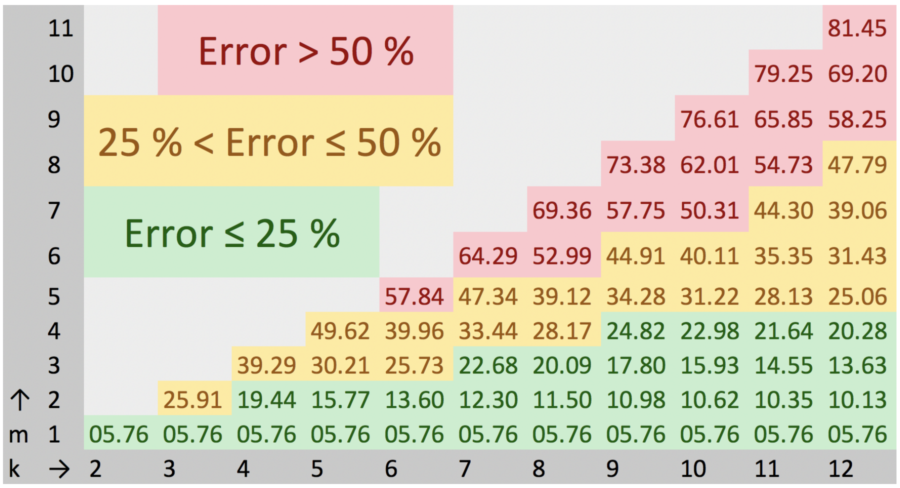
m ↑ / k →	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
24																								P
23																							P	P
22																						P	P	A
21																					P	P	A	A
20																				P	P	A	A	A
19																			P	P	A	A	A	A
18																		P	P	M	A	A	A	A
17																	P	P	M	A	A	A	A	A
16																P	P	M	A	A	A	A	A	A
15															P	P	M	A	A	A	A	A	A	A
14														P	P	E	A	A	A	A	A	A	A	A
13													P	P	E	M	A	A	A	A	A	A	A	A
12												P	P	E	M	M	A	A	A	A	A	A	A	A
11											P	P	E	M	M	A	A	A	A	A	A	A	A	A
10										P	P	E	M	M	M	A	A	A	A	A	A	A	A	A
9									P	P	P	M	M	M	M	A	A	A	A	A	A	A	A	A
8								P	P	P	E	M	M	M	M	A	A	A	A	A	A	A	A	A
7							P	P	P	E	M	M	M	M	M	A	A	A	A	A	A	A	A	A
6						P	P	P	P	E	M	M	M	M	M	M	A	A	A	A	A	A	A	A
5					P	P	P	P	E	E	M	M	M	M	M	M	M	M	M	A	A	A	A	A
4				P	P	P	P	P	P	E	E	M	M	M	M	M	M	M	M	M	M	M	M	M
3			P	P	P	P	P	P	P	P	P	E	E	E	E	M	M	M	M	M	M	M	M	M
2		P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P
1	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P

Figure 7.8: Scalability results for different values of m and k.

either PMC or MART (PMC was preferred, if available) for each $k \in [2, 12]$ and $m \in [1, k-1]$. As expected, SAP always resulted in a lower, pessimistic MTTF than PMC and MART since it is sound by construction. Thus, error signs are not explicitly denoted in the figure.

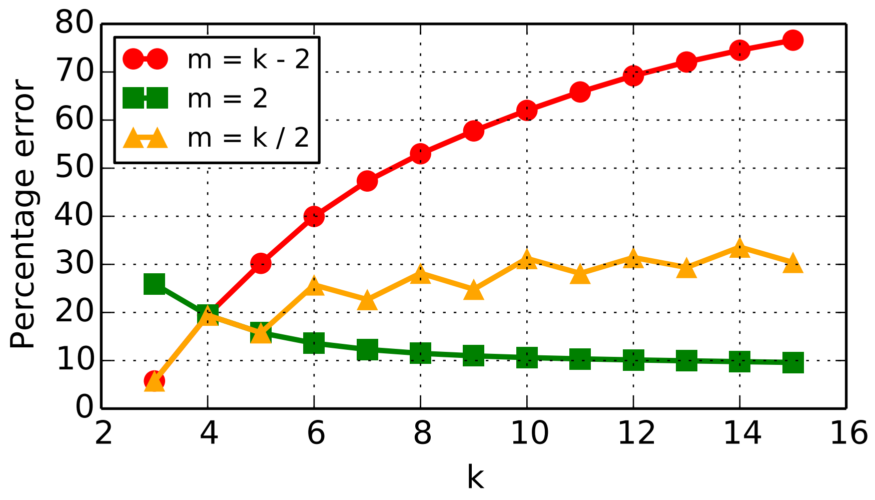
We make the two key observations regarding SAP's accuracy. First, even for small values of k, the relative errors are significant (see the red cells in Fig. 7.9 denoting specifications with relative error greater than 50%). This validates the need for an exact analysis whenever feasible. Second, the relative errors are higher if the ratio m/k is closer to one. To investigate this further, we also plot the percentage errors for $m = k-2$, $m = 2$, and $m = k/2$ with respect to k in Fig. 7.9b. From this figure, we observe that in all evaluated cases, the MTTF estimated with SAP was within an order of magnitude of the exact MTTF. Since in the context of reliability analyses the order of magnitude is typically of prime interest (rather than the exact value), we conclude that SAP is a reasonably accurate alternative for large values of k.

In summary, MART always outperforms both PMC-P and PMC-E, which is not surprising. In fact, for the scenario with IID iteration failure probabilities that we evaluated, MART directly represents the underlying system of linear equations without needing to construct a model. PMC's benefits lie in its ability to express non-IID iteration failure probabilities. SAP on the other hand scales much better than both PMC and MART, at the cost of acceptable, but non-zero pessimism.
]


Error > 50 %
25 % < Error ≤ 50 %
Error ≤ 25 %

m ↑ / k →	2	3	4	5	6	7	8	9	10	11	12
11											81.45
10										79.25	69.20
9									76.61	65.85	58.25
8								73.38	62.01	54.73	47.79
7							69.36	57.75	50.31	44.30	39.06
6						64.29	52.99	44.91	40.11	35.35	31.43
5					57.84	47.34	39.12	34.28	31.22	28.13	25.06
4				49.62	39.96	33.44	28.17	24.82	22.98	21.64	20.28
3			39.29	30.21	25.73	22.68	20.09	17.80	15.93	14.55	13.63
2		25.91	19.44	15.77	13.60	12.30	11.50	10.98	10.62	10.35	10.13
1	05.76	05.76	05.76	05.76	05.76	05.76	05.76	05.76	05.76	05.76	05.76

(a) Summary of SAP's accuracy with respect to MART and PMC



(b) SAP's accuracy trend for $m = 2$, $m = k/2$, and $m = k - 2$

Figure 7.9: Accuracy for different m and k.

PMC, MART, and SAP are useful alternatives for reliability evaluation depending on the value of m and k. PMC and MART are ideal to evaluate *short-range* properties on short window lengths to ensure short-term *safety* properties, e.g., such as "there should not be more than 3 consecutive failures in any window of 10 iterations" [45]. In contrast, SAP can evaluate asymptotic properties that are defined over a large window of events and reflect minimum acceptable longterm *quality-of-service levels*, e.g., such as "at least 90% of actuation commands must be applied on the plant in every 100 iterations" [193].

Although we focused on a binary failure type, i.e., each iteration was categorized either as a successful iteration or a failed iteration, one could also use fine-grained label types for each iteration, such as deadline violation, message loss, miscomputation, and so on. That is, an execution of system S could be modeled as a string in $\{0 \dots \lambda\}^*$, instead of a string in $\{0, 1\}^*$, where λ is the number of failure categories. Both PMC and MART easily extend to such systems.

As mentioned earlier, SAP has limited extensibility in its current form, since our objective when designing SAP was primarily to scale the evaluation of (m, k) specifications that are widely used in practice. However, the same blueprint could be used to safely approximate other types of robustness specifications as well, i.e., by breaking each specification into smaller events, computing the product of respective event probabilities (or a lower bound), and then reusing Eq. (7.11) for MTTF estimation. We leave similar approximate analysis for the other types of robustness constraints as future work.

7.5 CASE STUDY: ACTIVE SUSPENSION

The analyses proposed in the previous chapter and in this chapter can be used together to upper-bound the FIT rate of actively replicated NCS applications with temporal robustness properties. We demonstrate these benefits using a case study of an active suspension workload (the same as that used in Section 6.5). We first demonstrate the ability of our analyses to reveal and quantify non-obvious differences in the reliability of workloads with different weakly-hard requirements, thereby emphasizing the need for temporal robustness-aware reliability analyses. Thereafter, we illustrate the utility of our analysis in a design-space exploration context by comparing FITs of different replication schemes. We rely on (m, k) constraints and FIT rates obtained using the SAP approach in this case study, since we evaluate very large values of k for one of the experiments.

WORKLOAD AND PARAMETERS Like in Section 6.5, we base our experiments on a fault-tolerant version of the CAN-based active suspension workload studied by Anta and Tabuada [8]. The workload consists of tasks and messages corresponding to four control loops (L_1, L_2, L_3, and L_4), each of which corresponds to the control of four wheels (W_1, W_2, W_3, and W_4) with magnetic suspensions, two hard real-time messages that report the current in the power line cable and the internal temperature of the coil, another hard real-time clock synchronization message, and a soft real-time message responsible for logging. The message parameters are summarized in Table 7.5.

FIT FOR DIFFERENT VALUES OF m AND k To evaluate the impact of different (m, k) requirements, we evaluated control loop L_1's FIT while varying the value of parameters m and k for varying numbers of sensor and controller task replicas. In Fig. 7.10a, m and k were varied as follows: $1 \leqslant m \leqslant 5$, and $k = 5$ or $k = 2m$; and in Fig. 7.10b, m and k were varied such that m/k is either 90%, 95%, 99%, or 99.99% (in

MESSAGES	PAYLOAD SIZE	PERIOD
Clock synchronization	1 byte	50 ms
Current monitoring	1 byte	4 ms
Temperature monitoring	1 byte	10 ms
L_1 sensor & control messages	3 bytes	1.75 ms
L_2 sensor & control messages	3 bytes	1.75 ms
L_3 sensor & control messages	3 bytes	1.75 ms
L_4 sensor & control messages	3 bytes	1.75 ms
L_1–L_4 message replicas (if any)	3 bytes	1.75 ms
Logging	8 bytes	100 ms

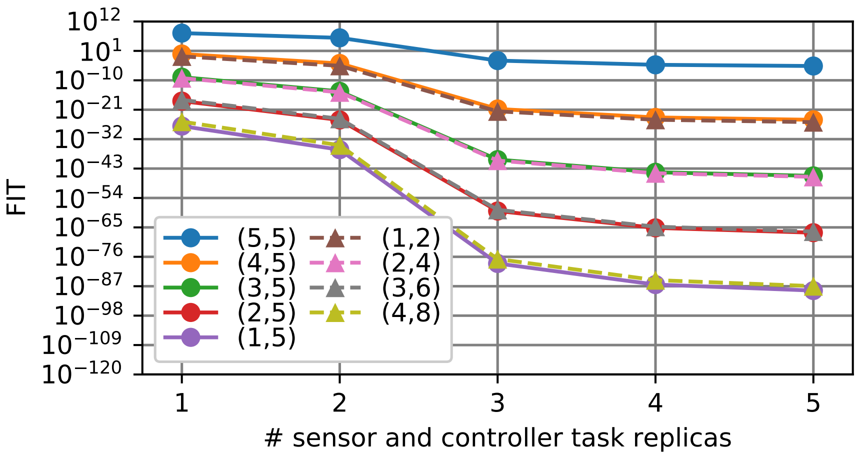
Table 7.5: CAN messages transmitted as part of the active suspension workload. The message list is ordered by priority, with the topmost message being the highest priority message. The highlighted message(s) depend on the replication factor used, and therefore vary with each experiment.

each case, minimizing the values of m and k). In both these cases, L_1's replication factor was varied from 1 to 5.

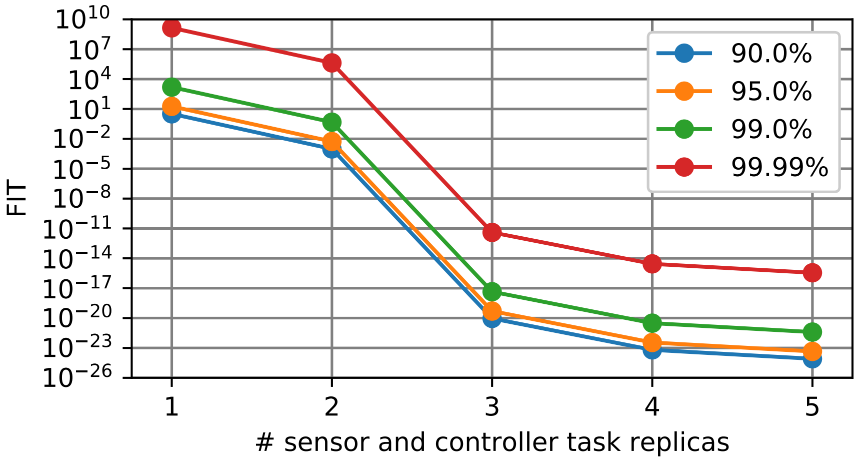
Overall, the experiments confirm that hard specifications—where $m = k$ (such as the $(5, 5)$ configuration in Fig. 7.10a) or where m/k is close to 100 % (such as the 99.99 % configuration in Fig. 7.10b)—yield much higher FIT rates compared to all other specifications, which highlights the need for a temporal robustness-aware reliability analysis. In addition, Fig. 7.10a shows that increasing both m and k while keeping m/k constant reduces the FIT rate, which indicates 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 $(2, 4)$ or the curves of $(3, 6)$ and $(2, 5)$ in Fig. 7.10a overlap.

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, 100,000,000 cars, then as per Mancuso's calculations [147], no more than about one vehicle per day will experience a failure in its active suspension NCS.

We considered the following error rates: $\gamma_{\text{retransmission}} = 10^{-4}$ for the CAN bus, and $\gamma_{\text{crash}}(H_i) = 10^{-8}$ and $\gamma_{\text{corrupt}}(H_i) = 10^{-12}$ for each host H_i (each rate is reported as the mean number of errors per ms).



(a) $1 \leqslant m \leqslant 5$ and $k = 5$ or $k = 2m$



(b) m/k is either 90%, 95%, 99%, or 99.99% (while minimizing m and k)

Figure 7.10: Parameters m and k are varied.

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.

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}. Therefore, to find a replication scheme with a FIT rate under 10, 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

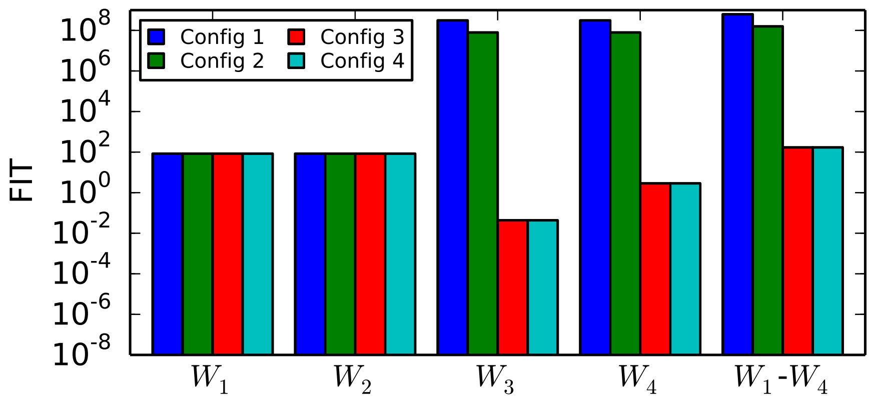
#	W_1	W_2	W_3	W_4	PERIOD	(m, k)	UTIL.
1	2S, 2C	2S, 2C	1S, 1C	1S, 1C	1.75 ms	(9, 10)	59 %
2	2S, 2C	2S, 2C	2S, 1C	2S, 1C	1.75 ms	(9, 10)	68 %
3	2S, 2C	2S, 2C	2S, 2C	2S, 2C	1.75 ms	(9, 10)	77 %
4	4S, 2C	4S, 2C	2S, 2C	2S, 2C	1.75 ms	(9, 10)	96 %
5	2S, 1C	2S, 1C	1S, 1C	1S, 1C	1.25 ms	(3, 5)	68 %
6	2S, 2C	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 7.6: 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.

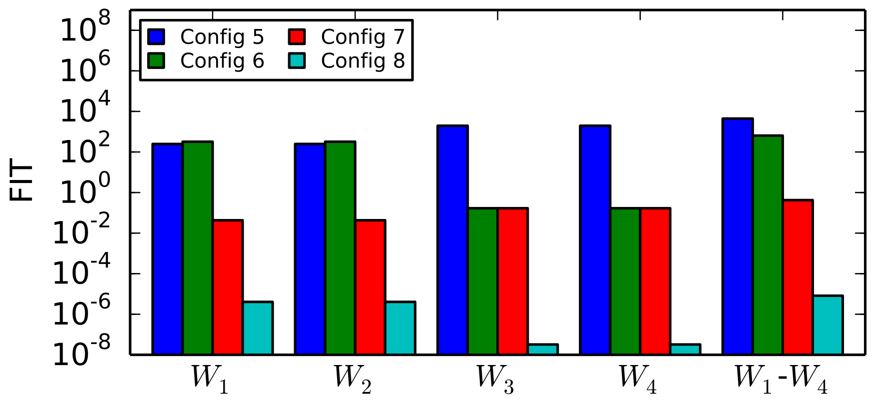
scheme with the minimum number of replicas, as given by Configurations 1–4 in Table 7.6 and Fig. 7.11a (Configurations 5–8 and the corresponding Fig. 7.11b are discussed below).

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 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* [8]? 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 (Configurations 5–8 in Table 7.6 and Fig. 7.11b).



(a) Configurations 1–4



(b) Configurations 5–8

Figure 7.11: Replication factors of the different control loops are varied.

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 [8], whereas case (i) would give up control quality for no appreciable gain.

In summary, 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 and 2, and L_1 and L_2 in Configuration 3), and that the proposed analysis is an effective aid in this process.

Part IV

THE ROAD AHEAD

8 CONCLUSION

This dissertation proposes reliability analyses of actively replicated NCS applications that are deployed on CAN or Ethernet in the presence of errors due to environmentally induced transient faults. In this chapter, we summarize our contributions, discuss open questions and future work, and finally conclude.

8.1 SUMMARY OF RESULTS

The contributions of this dissertation are broadly divided into two parts. The first part deals with tolerating Byzantine errors in CPS using an appropriate BFT protocol and analysing an upper bound on the FIT of the BFT protocol. The second part deals with the FIT analysis of temporally robust NCS applications. The resulting FIT bounds can then be used as inputs for a system-wide FIT analysis.

8.1.1 Byzantine Fault Tolerance

Ethernet-based implementations of NCS can fail due to environmentally induced Byzantine errors. To make such implementations ultra-reliable, we presented in Chapter 4 the design of a hard real-time, BFT IC protocol. Our choice of the IC protocol follows from an extensive survey of prior work on custom processors and networks that were developed for building ultra-reliable avionics systems.

In addition, based on the proposed design, we prototyped a BFT, time-aware key-value service (called Achal) for actively replicated NCS applications. The key-value service case study demonstrated that (i) the hard real-time IC protocol design can be implemented without difficulty on COTS processors and Ethernet-like networks; (ii) it provides a useful primitive to implement atomic broadcast or reliable communication on top of Ethernet despite Byzantine errors, while taking into account the real-time requirements of NCS applications; and that (iii) Achal outperforms similar services implemented using state-of-the art systems like Cassandra and BFT-SMaRt.

Ultra-reliability implies quantifiably negligible failure rates. Using BFT protocols helps reduce the failure rate in the presence of Byzantine errors. The next step is thus to quantify the failure rate, and validate if it is negligible. To this end, we proposed in Chapter 5 a reliability analysis to upper-bound the FIT rate of the presented hard real-time IC protocol. A key contribution is identifying and formalizing the notion of reliability anomalies in a hard real-time setting, and ensuring that our analysis is sound despite such anomalies. We also demonstrated the usefulness of such an analysis with a design-space exploration of the IC protocol and Ethernet network topology parameter space. Our experiments revealed non-trivial reliability trade-offs, such as the significant impact of the number of message exchange rounds and the network topology on the overall reliability.

8.1.2 Networked Control Systems

Actively replicated NCS (e.g., in DMR, TMR, or QMR configurations) can be used to safeguard safety-critical applications against crash and corruption errors. Clock synchronization and atomic broadcast services ensure that the active replicas do not diverge. Nonetheless, the NCS may still fail, say, when a sensor source is faulty or when the replicas experience correlated errors. Such scenarios necessitate an additional FIT analysis of the active replication protocol.

To this end, we presented in Chapter 6 a reliability analysis to upper-bound the failure probability of a single NCS iteration that is implemented on a CAN-based distributed real-time system. The analysis takes into account any correlations that may arise due to the synchronous execution of replicas and the voting semantics used for redundancy suppression. While the CAN protocol implicitly exposes an atomic broadcast layer, our analysis can also be applied to NCSs implemented over a software atomic broadcast layer (such as Achal).

We next presented in Chapter 7 analyses to derive an upper bound on the FIT rate of the NCS as a function of its iteration failure probability, periodicity, and weakly-hard temporal robustness specification. In particular, we presented three different techniques—PMC, MART, and SAP—which are based on probabilistic model checking, martingale theory, and sound approximation, respectively. PMC is expressive and yields exact results; MART also yields exact results but is not as expressive as PMC; SAP does not yield exact results, but is highly scalable in the parameter sizes of the weakly-hard robustness specification.

Our experiments combining the analyses in Chapters 6 and 7 also showed that accounting for an NCS's temporal robustness resulted in vastly more accurate FIT estimates, as opposed to using the conventional approach of computing MTTF using the time to first fault.

8.2 OPEN QUESTIONS AND FUTURE WORK

In the following, we discuss open questions and opportunities for future work regarding development of distributed real-time systems with ultra-reliability guarantees.

8.2.1 Improving the Analysis Accuracy

The proposed analyses currently consider the NCS application running on each node as a black box. In future work, we plan to exploit the program source of NCS applications to improve the modeling accuracy of the reliability analysis framework. In particular, we intend to develop a finer-grained strategy, at the granularity of program variables, to more accurately upper-bound the probability of application-specific message errors. For example, using program analysis techniques, we can trace the propagation of bit flips from registers to program variables; identify program variables that, if corrupted, result in the payload corruption; and then more accurately upper-bound the probability of silent data corruption in the network message payload.

Such program analysis techniques have been previously used to compute error bounds in the approximate computing domain, e.g., [110]. In the reliability domain, similar techniques have been used to replace fault injection (empirical) techniques with faster analyses without much loss in accuracy, e.g., [184]. We thus believe that using fine-grained techniques to improve the existing reliability analysis framework is both feasible and will help design ultra-reliable systems with better resource efficiency.

8.2.2 Reliability Analysis of Other Critical Services

In order to evaluate an upper bound on the system-wide FIT as per the SOFR model, failures in every critical service must be accounted for as part of one of the many constituent FIT analyses. Clock synchronization is one such critical service and an integral assumption in this dissertation. One of the open questions is, thus, how can the FIT rate of a PTP-like software clock synchronization protocol be upper-bounded when the protocol is realized over Ethernet-like COTS networks.

Prior work in this regard analyses upper bounds on the clock skews of deterministic clock synchronization algorithms, and also analyses upper bounds on the *invalidation probabilities* of probabilistic clock synchronization algorithms (i.e., the probability with which a given clock skew bound is exceeded) [157]. Both sets of upper bounds rely on implementation-specific parameters such as the time that a process takes to read another process's local clock (recall background on clock synchronization from Section 2.1.1.2). However, when clock synchronization is implemented in software over contemporary COTS hardware, these implementation-specific parameters may no longer be predictable (as expected by the prior analyses). The unpredictability may render existing provably correct clock synchronization algorithms to be only "intuitively correct," and also significantly increase their minimum achievable clock skew. Hence, in future, we would also like to bring COTS-based implementations of fault-tolerant clock synchronization algorithms into the ultra-reliability fold.

8.2.3 Reliability Analysis of Intelligent NCS

Next-generation CPS will also consist of NCS with Artificial Intelligence (AI) components. For instance, control loops integrated with Deep Neural Networks (DNNs) will be responsible for making critical scene recognition and trajectory planning decisions on the fly. However, certifying the reliability of AI components, specifically DNN implementations, is quite challenging. Unlike conventional software, DNNs are guided by millions of tunable parameters, which means analyzing their entire state space is not feasible. Furthermore, to meet strict end-to-end latency goals (e.g., $100\,\mathrm{ms}$ in autonomous vehicles), DNNs are typically executed on highly parallel accelerator platforms, e.g., Google's TensorFlow [216] and MIT's Eyeriss [70], which have not yet been analyzed from a safety or timeliness perspective. Reliability analysis of safety-critical DNN frameworks is thus an open question.

We are particularly interested in extending our existing reliability analysis framework to explore ultra-reliable designs for NCS that consist of DNN executions. In particular, recent studies have shown that DNNs have intrinsic resilience against transient faults owing to their sparse network structure, but the resilience of different structural units (e.g., layers and neurons) within a single DNN differ significantly. Hence, there is an opportunity to design DNN frameworks that are as resilient as a triple modular redundant system (i.e., with three functionally identical DNNs executing in parallel) but at roughly one-third the cost. That is, conceptually, if the most critical computations inside a DNN can be accurately identified, we can selectively safeguard them through spatial or temporal redundancy mechanisms. Our goal is to design analyses to estimate a minimal spare capacity needed for achieving the desired reliability target (e.g., the number of processing elements in a hardware accelerator to be reserved as spares) and propose mechanisms to efficiently orchestrate critical computations over this spare capacity while maximizing data reuse, so that end-to-end latencies remain under the specified threshold.

8.3 CLOSING REMARKS

The commercial aircraft industry has over the years set very high standards of reliability. Each aircraft design is rigorously tested before deployment; it is engineered to remain functional despite intolerable errors during runtime (such as environmentally-induced hardware faults); and its failure probability is quantified in advance and shown to be negligible for safety certification.

In this dissertation, we take inspiration from reliability engineering practices in the commercial aircraft industry. Our aim is to bring the notion of ultra-reliability—i.e., the practice of ensuring quantifiably negligible residual failure rates—to the next generation of fully-autonomous CPS, including autonomous vehicles, drones, robots, and industrial automation systems. To this end, we have designed analyses to upper bound the failure rates of COTS-based implementations of NCS applications, which are integral to many of these CPS. Over the next few decades, when the use of fully autonomous CPS and their impact on human lives grows substantially, we hope that these analyses will be useful to build CPS that are as trustworthy as airplanes.

Part V

APPENDICES

A MONOTONICITY PROOFS

In this Appendix, we provide monotonicity proofs for the reliability analysis of an NCS iteration provided in Chapter 6.

A.1 NON-MONOTONICITY OF $P(U_n^y$ INCORRECT$)$

Recall the controller output analysis from Section 6.3.1. Definition 6.6 defines a recursive procedure to compute the probability that U_n^y (which denotes the output of controller task C_n^y's voter instance) is incorrect, using the notion of an *error status tuple* (Definition 6.5). In each step of the recursion, a message $X_n^s \in \mathcal{Z}_n$ is selected and placed in one of the four sets ($\mathcal{O}_n$, $\mathcal{D}_n$, $\mathcal{I}_n$, or $\mathcal{C}_n$) in the error status tuple.

In the following, we show that for any tuple $\langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle$, while $P(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle)$ is either independent of, or monotonically increasing in, the exact probability $P(X_n^s \text{ corrupted})$, it is not independent of or monotonically increasing in the exact probabilities $P(X_n^s \text{ delayed})$ and $P(X_n^s \text{ omitted})$. The result implies that probability $P(U_n^y \text{ incorrect} \mid \langle \emptyset, \emptyset, \emptyset, \emptyset, X_n \rangle)$, also denoted as $P(U_n^y \text{ incorrect})$, is subject to reliability anomalies, when computed using upper bounds on the exact message error probabilities.

For brevity of the following proofs, we introduce a shorthand notation to denote the exact error probabilities and the intermediate probabilities used in the definition of $P(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle)$. (Definition 6.6). The shorthand notation is summarized in Table A.1. Based on the new notation, we first restate in Eq. (A.1) below the definition of $P(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle)$.

$$P\left(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle\right) =$$
$$\begin{cases} P_s & \mathcal{Z}_n = \emptyset \\ \left(\begin{array}{l} C_o \cdot P_o \;+\; C_d \cdot \overline{P_o} \cdot P_d \\ + \; C_i \cdot \overline{P_o} \cdot \overline{P_d} \cdot P_i \;+\; C_c \cdot \overline{P_o} \cdot \overline{P_d} \cdot \overline{P_i} \end{array} \right) & \mathcal{Z}_n \neq \emptyset. \end{cases} \tag{A.1}$$

Note that P_o, P_d, and P_i are not defined with respect to *any* message, but with respect to the specific message X_n^s. Thus, probabilities C_o, C_d, C_i, and C_c are independent of P_o, P_d, and P_i. In particular, since $P(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle)$ only depends on message error probabilities of the messages in set $\mathcal{Z}_n$, and since $X_n^s \notin \mathcal{Z}_n$ for each C_o, C_d, C_i, and C_c, they are each independent of P_o, P_d, and P_i.

NOTATION USED IN SECTION 6.3	SHORTHAND
$P(\text{SimpleMajority incorrect} \mid \mathcal{I}_n, \mathcal{C}_n)$	P_s
$P(X_n^s \text{ omitted})$	P_o
$P(X_n^s \text{ delayed})$	P_d
$P(X_n^s \text{ corrupted})$	P_i
$\overline{P(X_n^s \text{ omitted})}$	$\overline{P_o}$
$\overline{P(X_n^s \text{ delayed})}$	$\overline{P_d}$
$\overline{P(X_n^s \text{ corrupted})}$	$\overline{P_i}$
$P(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n \cup \{X_n^s\}, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \setminus \{X_n^s\} \rangle)$	C_o
$P(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n \cup \{X_n^s\}, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \setminus \{X_n^s\} \rangle)$	C_d
$P(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n \cup \{X_n^s\}, \mathcal{C}_n, \mathcal{Z}_n \setminus \{X_n^s\} \rangle)$	C_i
$P(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n \cup \{X_n^s\}, \mathcal{Z}_n \setminus \{X_n^s\} \rangle)$	C_c

Table A.1: Shorthand notation for the analysis in Section 6.3.

Likewise, P_s, too, is independent of P_o, P_d, and P_i. This is because P_o, P_d, and P_i only determine the probability with which a message is inserted into $\mathcal{I}_n$ or $\mathcal{C}_n$, whereas in the computation of $P_s = P(\text{SimpleMajority incorrect} \mid \mathcal{I}_n, \mathcal{C}_n)$, $\mathcal{I}_n$ and $\mathcal{C}_n$ are given.

In addition, note that C_o, C_d, C_i, and C_c differ only in terms of whether message X_n^s is inserted from set $\mathcal{Z}_n$ into set $\mathcal{O}_n$, $\mathcal{D}_n$, $\mathcal{I}_n$, or $\mathcal{C}_n$, respectively. In the first two cases, X_n^s is either omitted or delayed and hence it does not participate in voting. Thus, the probability that U_n^y is incorrect is the same for these cases, i.e., $C_d = C_o$. In the third case, X_n^s is neither omitted nor delayed but corrupted, and hence it participates in voting with a faulty value. Thus, the probability that U_n^y is incorrect can only increase in comparison with the first two cases, i.e., $C_i \geqslant C_d = C_o$. In contrast, in the forth case, X_n^s is neither omitted, delayed, nor corrupted, and hence it participates in voting with a correct value. Thus, the probability that U_n^y is incorrect can only decrease in comparison with the first two cases, i.e., $C_d = C_o \geqslant C_c$. In summary, C_o, C_d, C_i, and C_c are related in the following way.

$$C_i \geqslant C_d = C_o \geqslant C_c. \tag{A.2}$$

Next, we prove the main results in Theorem A.1 to Theorem A.3.

THEOREM A.1. $P(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle)$ is either independent of or monotonically increasing in P_i.

Proof. If $\mathcal{Z}_n = \emptyset$, then $P(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle) = P_s$, and P_s is independent of P_i. If $\mathcal{Z}_n \neq \emptyset$, then from Eq. (A.1),

$$P(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle)$$
$$= \begin{pmatrix} C_o \cdot P_o \ + \ C_d \cdot \overline{P_o} \cdot P_d \ + \ C_i \cdot \overline{P_o} \cdot \overline{P_d} \cdot P_i \\ + \, C_c \cdot \overline{P_o} \cdot \overline{P_d} \cdot \overline{P_i} \end{pmatrix}$$

{since $C_o = C_d$ (from Eq. (A.2)), replacing C_d with C_o}

$$= \begin{pmatrix} C_o \cdot P_o \ + \ C_o \cdot \overline{P_o} \cdot P_d \ + \ C_i \cdot \overline{P_o} \cdot \overline{P_d} \cdot P_i \\ + \, C_c \cdot \overline{P_o} \cdot \overline{P_d} \cdot \overline{P_i} \end{pmatrix}$$

{replacing $\overline{P_i}$ with $1 - P_i$, and simplifying}

$$= \begin{pmatrix} C_o \cdot P_o \ + \ C_o \cdot \overline{P_o} \cdot P_d \ + \ C_i \cdot \overline{P_o} \cdot \overline{P_d} \cdot P_i \\ + \, C_c \cdot \overline{P_o} \cdot \overline{P_d} \ - \ C_c \cdot \overline{P_o} \cdot \overline{P_d} \cdot P_i \end{pmatrix}$$

{letting $K_1 = C_o \cdot P_o \ + \ C_o \cdot \overline{P_o} \cdot P_d \ + \ C_c \cdot \overline{P_o} \cdot \overline{P_d}$}

$$= K_1 \ + \ C_i \cdot \overline{P_o} \cdot \overline{P_d} \cdot P_i \ - \ C_c \cdot \overline{P_o} \cdot \overline{P_d} \cdot P_i$$

{letting $K_2 = (C_i \ - \ C_c) \cdot \overline{P_o} \cdot \overline{P_d}$}

$$= K_1 \ + \ K_2 \cdot P_i. \tag{A.3}$$

In Eq. (A.3), by definition, K_1 and K_2 are independent of P_i. $K_1 \geqslant 0$, because it is defined as a sum of all positive terms (each term is a product of probabilities). Also, $K_2 \geqslant 0$, since $C_i \geqslant C_c$ (from Eq. (A.2)). Thus, $K_1 + K_2 \cdot P_i$ is monotonically increasing in P_i. □

While $P(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle)$ is always either independent of or monotonically increasing in P_i, this is not the case for P_d. In particular, we show that under specific conditions, $P(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle)$ decreases if P_d is increased.

THEOREM A.2. If $\mathcal{Z}_n \neq \emptyset$ and $(C_o - C_c) - (C_i - C_c) \cdot P_i < 0$, then $P(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle)$ decreases with increasing P_d.

Proof. If $\mathcal{Z}_n \neq \emptyset$, then from Eq. (A.1),

$$P(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle)$$
$$= \begin{pmatrix} C_o \cdot P_o \ + \ C_d \cdot \overline{P_o} \cdot P_d \ + \ C_i \cdot \overline{P_o} \cdot \overline{P_d} \cdot P_i \\ + \, C_c \cdot \overline{P_o} \cdot \overline{P_d} \cdot \overline{P_i} \end{pmatrix}$$

{since $C_o = C_d$ (from Eq. (A.2)), replacing C_d with C_o}

$$= \left(\begin{array}{l} C_o \cdot P_o \ + \ C_o \cdot \overline{P_o} \cdot P_d \ + \ C_i \cdot \overline{P_o} \cdot \overline{P_d} \cdot P_i \\ + \ C_c \cdot \overline{P_o} \cdot \overline{P_d} \cdot \overline{P_i} \end{array} \right)$$

{replacing $\overline{P_d}$ with $1 - P_d$, and simplifying}

$$= \left(\begin{array}{l} C_o \cdot P_o \ + \ C_o \cdot \overline{P_o} \cdot P_d \ + \ C_i \cdot \overline{P_o} \cdot P_i \ - \ C_i \cdot \overline{P_o} \cdot P_d \cdot P_i \\ + \ C_c \cdot \overline{P_o} \cdot \overline{P_i} \ - \ C_c \cdot \overline{P_o} \cdot P_d \cdot \overline{P_i} \end{array} \right)$$

{letting $K_1 = C_o \cdot P_o \ + \ C_i \cdot \overline{P_o} \cdot P_i \ + \ C_c \cdot \overline{P_o} \cdot \overline{P_i}$}

$$= K_1 \ + \ C_o \cdot \overline{P_o} \cdot P_d \ - \ C_i \cdot \overline{P_o} \cdot P_d \cdot P_i \ - \ C_c \cdot \overline{P_o} \cdot P_d \cdot \overline{P_i}$$

{replacing $\overline{P_i}$ with $1 - P_i$, and simplifying}

$$= \left(\begin{array}{l} K_1 \ + \ C_o \cdot \overline{P_o} \cdot P_d \ - \ C_i \cdot \overline{P_o} \cdot P_d \cdot P_i \\ - \ C_c \cdot \overline{P_o} \cdot P_d \ + \ C_c \cdot \overline{P_o} \cdot P_d \cdot P_i \end{array} \right)$$

{taking out $\overline{P_o} \cdot P_d$ as a common factor, and upon further simplifying}

$$= K_1 \ + \ ((C_o - C_c) \ - \ (C_i \ - \ C_c) \cdot P_i) \cdot \overline{P_o} \cdot P_d$$

{letting $K_2 = (C_o - C_c) \ - \ (C_i \ - \ C_c) \cdot P_i$}

$$= K_1 \ + \ K_2 \cdot \overline{P_o} \cdot P_d. \tag{A.4}$$

In Eq. (A.4), by definition, K_1 and K_2 are independent of P_d. Hence, since $K_2 = (C_o - C_c) - (C_i - C_c) \cdot P_i < 0$ (from the premise), the probability $K_1 \ + \ K_2 \cdot \overline{P_o} \cdot P_d$ decreases with increasing P_d. □

Like Theorem A.3, we show next that under specific conditions, $P(U^y_n \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle)$ decreases if P_o is increased.

THEOREM A.3. If $\mathcal{Z}_n \neq \emptyset$ and $(C_o - C_c) - (C_i - C_c) \cdot P_i < 0$, then $P(U^y_n \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle)$ decreases with increasing P_o.

Proof. If $\mathcal{Z}_n \neq \emptyset$, from Eq. (A.1),

$$P(U^y_n \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle)$$
$$= \left(\begin{array}{l} C_o \cdot P_o \ + \ C_d \cdot \overline{P_o} \cdot P_d \ + \ C_i \cdot \overline{P_o} \cdot \overline{P_d} \cdot P_i \\ + \ C_c \cdot \overline{P_o} \cdot \overline{P_d} \cdot \overline{P_i} \end{array} \right)$$

{since $C_o = C_d$ (from Eq. (A.2)), replacing C_d with C_o}

$$= \left(\begin{array}{l} C_o \cdot P_o \; + \; C_o \cdot \overline{P_o} \cdot P_d \; + \; C_i \cdot \overline{P_o} \cdot \overline{P_d} \cdot P_i \\ + \; C_c \cdot \overline{P_o} \cdot \overline{P_d} \cdot \overline{P_i} \end{array} \right)$$

{replacing $\overline{P_o}$ with $1 - P_o$, and simplifying}

$$= \left(\begin{array}{l} C_o \cdot P_o \; + \; C_o \cdot P_d \; - \; C_o \cdot P_o \cdot P_d \; + \; C_i \cdot \overline{P_d} \cdot P_i \\ - \; C_i \cdot P_o \cdot \overline{P_d} \cdot P_i \; + \; C_c \cdot \overline{P_d} \cdot \overline{P_i} \; - \; C_c \cdot P_o \cdot \overline{P_d} \cdot \overline{P_i} \end{array} \right)$$

{letting $K_1 = C_o \cdot P_d \; + \; C_i \cdot \overline{P_d} \cdot P_i \; + \; C_c \cdot \overline{P_d} \cdot \overline{P_i}$}

$$= K_1 \; + \; C_o \cdot P_o \; - \; C_o \cdot P_o \cdot P_d \; - \; C_i \cdot P_o \cdot \overline{P_d} \cdot P_i \; - \; C_c \cdot P_o \cdot \overline{P_d} \cdot \overline{P_i}$$

{taking out P_o as a common factor}

$$= K_1 \; + \; P_o \cdot (C_o \; - \; C_o \cdot P_d \; - \; C_i \cdot \overline{P_d} \cdot P_i \; - \; C_c \cdot \overline{P_d} \cdot \overline{P_i})$$

{replacing $C_o \; - \; C_o \cdot P_d$ with $C_o \cdot \overline{P_d}$, and taking out $\overline{P_d}$ as a common factor}

$$= K_1 \; + \; (C_o \; - \; C_i \cdot P_i \; - \; C_c \cdot \overline{P_i}) \cdot P_o \cdot \overline{P_d}$$

{replacing $\overline{P_i}$ with $1 - P_i$, and upon further simplifying}

$$= K_1 \; + \; ((C_o \; - \; C_c) \; - \; (C_i \; - \; C_c) \cdot P_i) \cdot P_o \cdot \overline{P_d}$$

{letting $K_2 = (C_o \; - \; C_c) \; - \; (C_i \; - \; C_c) \cdot P_i$}

$$= K_1 \; + \; K_2 \cdot P_o \cdot \overline{P_d}. \tag{A.5}$$

In Eq. (A.5), by definition, K_1 and K_2 are independent of P_o. Hence, since $K_2 = (C_o - C_c) - (C_i - C_c) \cdot P_i < 0$ (from the premise), the probability $K_1 \; + \; K_2 \cdot P_o \cdot \overline{P_d}$ decreases with increasing P_o. □

A.2 MONOTONICITY OF $Q(U^y_n$ INCORRECT)

The exact probability that U^y_n is incorrect, as defined in Definition 6.6, may decrease with increasing message error probabilities (as we proved in the previous section). Hence, for safety reasons, i.e., to avoid reliability anomalies, we also defined in Section 6.3.1 an upper bound on the probability that U^y_n is incorrect (Definition 6.7). We prove below that this probability is independent of or monotonic in all the message error probabilities.

NOTATIONS USED IN SECTION 6.3	SHORTHAND
$Q(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n \cup \{X_n^s\}, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \setminus \{X_n^s\}\rangle)$	C_o'
$Q(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n \cup \{X_n^s\}, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \setminus \{X_n^s\}\rangle)$	C_d'
$Q(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n \cup \{X_n^s\}, \mathcal{C}_n, \mathcal{Z}_n \setminus \{X_n^s\}\rangle)$	C_i'
$Q(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n \cup \{X_n^s\}, \mathcal{Z}_n \setminus \{X_n^s\}\rangle)$	C_c'

Table A.2: Extensions to the shorthand notation for the analysis in Section 6.3.

In particular, we show that for any tuple $\langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle$, upper bound $Q(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle)$ is independent of or monotonic in the exact probabilities $P(X_n^s \text{ corrupted})$, $P(X_n^s \text{ delayed})$ and $P(X_n^s \text{ omitted})$. The result implies that probability $Q(U_n^y \text{ incorrect} \mid \langle \emptyset, \emptyset, \emptyset, \emptyset, X_n \rangle)$, also denoted as $Q(U_n^y \text{ incorrect})$, is can be safely computed using upper bounds on the message error probabilities, i.e., without the possibility of any reliability anomalies. Monotonicity with respect to the exact probability $P_s = P(\text{SimpleMajority incorrect} \mid \mathcal{I}_n, \mathcal{C}_n)$ trivially holds since $Q(U_n^y \text{ incorrect})$ is only defined in terms of P_s and not in terms of $\overline{P_s}$.

Once again, for the brevity of the following proofs, we extend the shorthand notation introduced in Table A.1 with shorthand for the new intermediate probability upper bounds that are used in the definition of $Q(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle)$ (Definition 6.7). See Table A.2 for the extensions. Using the extended shorthand notation, we first restate the definition of $Q(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle)$.

$$Q\left(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle\right) =$$
$$\begin{cases} P_s & \mathcal{Z}_n = \emptyset \\ \left(\begin{array}{l} C_o' \cdot P_o \;+\; C_d' \cdot \overline{P_o} \cdot P_d \\ +\, C_i' \cdot \overline{P_o} \cdot \overline{P_d} \cdot P_i \;+\; C_c' \cdot \overline{P_o} \cdot \overline{P_d} \cdot \overline{P_i} \\ +\, C_i' \cdot \overline{P_o} \cdot P_d \cdot P_i \;+\; C_i' \cdot P_o \cdot P_i \end{array} \right) & \mathcal{Z}_n \neq \emptyset. \end{cases} \tag{A.6}$$

Also recall from Appendix A.1 that P_s is independent of P_o, P_d, and P_i. In addition, similar to C_o, C_d, C_i, and C_c in Appendix A.1, C_o', C_d', C_i', and C_c', too, are independent of P_o, P_d, and P_i, as well as

$$C_i' \geqslant C_d' = C_o' \geqslant C_c'. \tag{A.7}$$

The monotonicity proofs (Theorem A.4-Theorem A.6) follow.

THEOREM A.4. $Q(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle)$ is either independent of or monotonically increasing in P_i.

Proof. If $\mathcal{Z}_n = \emptyset$, then $Q(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle) = P_s$, and P_s is independent of P_i. If $\mathcal{Z}_n \neq \emptyset$, then from Eq. (A.1),

$$Q(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle)$$
$$= \left(\begin{array}{l} C'_o \cdot P_o \;+\; C'_d \cdot \overline{P_o} \cdot P_d \;+\; C'_i \cdot \overline{P_o} \cdot \overline{P_d} \cdot P_i \\ +\; C'_c \cdot \overline{P_o} \cdot \overline{P_d} \cdot \overline{P_i} \;+\; C'_i \cdot \overline{P_o} \cdot P_d \cdot P_i \;+\; C'_i \cdot P_o \cdot P_i \end{array} \right)$$

{since $C'_o = C'_d$ (from Eq. (A.7)), replacing C'_d with C'_o}

$$= \left(\begin{array}{l} C'_o \cdot P_o \;+\; C'_o \cdot \overline{P_o} \cdot P_d \;+\; C'_i \cdot \overline{P_o} \cdot \overline{P_d} \cdot P_i \\ +\; C'_c \cdot \overline{P_o} \cdot \overline{P_d} \cdot \overline{P_i} \;+\; C'_i \cdot \overline{P_o} \cdot P_d \cdot P_i \;+\; C'_i \cdot P_o \cdot P_i \end{array} \right)$$

{replacing $\overline{P_i}$ with $1 - P_i$, and simplifying}

$$= \left(\begin{array}{l} C'_o \cdot P_o \;+\; C'_o \cdot \overline{P_o} \cdot P_d \;+\; C'_i \cdot \overline{P_o} \cdot \overline{P_d} \cdot P_i \\ +\; C'_c \cdot \overline{P_o} \cdot \overline{P_d} \;-\; C'_c \cdot \overline{P_o} \cdot \overline{P_d} \cdot P_i \;+\; C'_i \cdot \overline{P_o} \cdot P_d \cdot P_i \\ +\; C'_i \cdot P_o \cdot P_i \end{array} \right)$$

{letting $K_1 = C'_o \cdot P_o \;+\; C'_o \cdot \overline{P_o} \cdot P_d \;+\; C'_c \cdot \overline{P_o} \cdot \overline{P_d}$}

$$= \left(\begin{array}{l} K_1 \;+\; C'_i \cdot \overline{P_o} \cdot \overline{P_d} \cdot P_i \;-\; C'_c \cdot \overline{P_o} \cdot \overline{P_d} \cdot P_i \\ +\; C'_i \cdot \overline{P_o} \cdot P_d \cdot P_i \;+\; C'_i \cdot P_o \cdot P_i \end{array} \right)$$

{letting $K_2 = (C'_i \;-\; C'_c) \cdot \overline{P_o} \cdot \overline{P_d}$}

$$= K_1 \;+\; K_2 \cdot P_i \;+\; C'_i \cdot \overline{P_o} \cdot P_d \cdot P_i \;+\; C'_i \cdot P_o \cdot P_i$$

{letting $K_3 = C'_i \cdot \overline{P_o} \cdot P_d \;+\; C'_i \cdot P_o$}

$$= K_1 \;+\; K_2 \cdot P_i \;+\; K_3 \cdot P_i. \tag{A.8}$$

In Eq. (A.8), by definition, K_1, K_2, and K_3 are independent of P_i. $K_1 \geqslant 0$ and $K_3 \geqslant 0$, since both are defined as a sum of all positive terms (each term is a product of probabilities). Also, $K_2 \geqslant 0$, since $C'_i \geqslant C'_c$ (from Eq. (A.7)). Thus, $K_1 + K_2 \cdot P_i + K_3 \cdot P_i$ is monotonically increasing in P_i. □

THEOREM A.5. $Q(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle)$ is either independent of or monotonically increasing in P_d.

Proof. If $\mathcal{Z}_n = \emptyset$, then $Q(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle) = P_s$, and P_s is independent of P_d. If $\mathcal{Z}_n \neq \emptyset$, then from Eq. (A.1),

$$Q(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle)$$
$$= \begin{pmatrix} C'_o \cdot P_o \;+\; C'_d \cdot \overline{P_o} \cdot P_d \;+\; C'_i \cdot \overline{P_o} \cdot \overline{P_d} \cdot P_i \\ +\; C'_c \cdot \overline{P_o} \cdot \overline{P_d} \cdot \overline{P_i} \;+\; C'_i \cdot \overline{P_o} \cdot P_d \cdot P_i \;+\; C'_i \cdot P_o \cdot P_i \end{pmatrix}$$

{since $C'_o = C'_d$ (from Eq. (A.7)), replacing C'_d with C'_o}

$$= \begin{pmatrix} C'_o \cdot P_o \;+\; C'_o \cdot \overline{P_o} \cdot P_d \;+\; C'_i \cdot \overline{P_o} \cdot \overline{P_d} \cdot P_i \\ +\; C'_c \cdot \overline{P_o} \cdot \overline{P_d} \cdot \overline{P_i} \;+\; C'_i \cdot \overline{P_o} \cdot P_d \cdot P_i \;+\; C'_i \cdot P_o \cdot P_i \end{pmatrix}$$

{replacing $\overline{P_d}$ with $1 - P_d$, and simplifying}

$$= \begin{pmatrix} C'_o \cdot P_o \;+\; C'_o \cdot \overline{P_o} \cdot P_d \;+\; C'_i \cdot \overline{P_o} \cdot P_i \;-\; C'_i \cdot \overline{P_o} \cdot P_d \cdot P_i \\ +\; C'_c \cdot \overline{P_o} \cdot \overline{P_i} \;-\; C'_c \cdot \overline{P_o} \cdot P_d \cdot \overline{P_i} \;+\; C'_i \cdot \overline{P_o} \cdot P_d \cdot P_i \\ +\; C'_i \cdot P_o \cdot P_i \end{pmatrix}$$

{cancelling $C'_i \cdot \overline{P_o} \cdot P_d \cdot P_i$}

$$= \begin{pmatrix} C'_o \cdot P_o \;+\; C'_o \cdot \overline{P_o} \cdot P_d \;+\; C'_i \cdot \overline{P_o} \cdot P_i \;+\; C'_c \cdot \overline{P_o} \cdot \overline{P_i} \\ -\; C'_c \cdot \overline{P_o} \cdot P_d \cdot \overline{P_i} \;+\; C'_i \cdot P_o \cdot P_i \end{pmatrix}$$

{letting $K_1 = C'_o \cdot P_o \;+\; C'_i \cdot \overline{P_o} \cdot P_i \;+\; C'_c \cdot \overline{P_o} \cdot \overline{P_i} \;+\; C'_i \cdot P_o \cdot P_i$}

$$= K_1 \;+\; C'_o \cdot \overline{P_o} \cdot P_d \;-\; C'_c \cdot \overline{P_o} \cdot P_d \cdot \overline{P_i}$$

{letting $K_2 = C'_o \cdot \overline{P_o} - C'_c \cdot \overline{P_o} \cdot \overline{P_i} = (C'_o - C'_c \cdot \overline{P_i}) \cdot \overline{P_o}$}

$$= K_1 \;+\; K_2 \cdot P_d. \tag{A.9}$$

In Eq. (A.9), by definition, K_1 and K_2 are independent of P_d. $K_1 \geqslant 0$, because it is defined as a sum of all positive terms (each term is a product of probabilities). Also, $K_2 \geqslant 0$, since $C'_o \geqslant C'_c$ (from Eq. (A.7)), which in turn implies $C'_o \geqslant \overline{P_i} \cdot C'_c$ ($\overline{P_i}$ being a probability). Thus, $K_1 \;+\; K_2 \cdot P_d$ is monotonically increasing in P_d. □

THEOREM A.6. $Q(U_n^y \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle)$ is neither independent of nor monotonically increasing in P_o.

Proof. If $\mathcal{Z}_n = \emptyset$, then $Q(U^y_n \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle) = P_s$, and P_s is independent of P_d. If $\mathcal{Z}_n \neq \emptyset$, then from Eq. (A.1),

$$Q(U^y_n \text{ incorrect} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle)$$

$$= \begin{pmatrix} C'_o \cdot P_o \; + \; C'_d \cdot \overline{P_o} \cdot P_d \; + \; C'_i \cdot \overline{P_o} \cdot \overline{P_d} \cdot P_i \\ + \, C'_c \cdot \overline{P_o} \cdot \overline{P_d} \cdot \overline{P_i} \; + \; C'_i \cdot \overline{P_o} \cdot P_d \cdot P_i \; + \; C'_i \cdot P_o \cdot P_i \end{pmatrix}$$

{since $C'_o = C'_d$ (from Eq. (A.7)), replacing C'_d with C'_o}

$$= \begin{pmatrix} C'_o \cdot P_o \; + \; C'_o \cdot \overline{P_o} \cdot P_d \; + \; C'_i \cdot \overline{P_o} \cdot \overline{P_d} \cdot P_i \\ + \, C'_c \cdot \overline{P_o} \cdot \overline{P_d} \cdot \overline{P_i} \; + \; C'_i \cdot \overline{P_o} \cdot P_d \cdot P_i \; + \; C'_i \cdot P_o \cdot P_i \end{pmatrix}$$

{replacing $\overline{P_d}$ with $1 - P_d$, and simplifying}

$$= \begin{pmatrix} C'_o \cdot P_o \; + \; C'_o \cdot \overline{P_o} \cdot P_d \; + \; C'_i \cdot \overline{P_o} \cdot P_i \; - \; C'_i \cdot \overline{P_o} \cdot P_d \cdot P_i \\ + \, C'_c \cdot \overline{P_o} \cdot \overline{P_i} \; - \; C'_c \cdot \overline{P_o} \cdot P_d \cdot \overline{P_i} \; + \; C'_i \cdot \overline{P_o} \cdot P_d \cdot P_i \\ + \, C'_i \cdot P_o \cdot P_i \end{pmatrix}$$

{cancelling $C'_i \cdot \overline{P_o} \cdot P_d \cdot P_i$}

$$= \begin{pmatrix} C'_o \cdot P_o \; + \; C'_o \cdot \overline{P_o} \cdot P_d \; + \; C'_i \cdot \overline{P_o} \cdot P_i + \; C'_c \cdot \overline{P_o} \cdot \overline{P_i} \\ - \, C'_c \cdot \overline{P_o} \cdot P_d \cdot \overline{P_i} \; + \; C'_i \cdot P_o \cdot P_i \end{pmatrix}$$

{replacing $\overline{P_o}$ with $1 - P_o$, and simplifying}

$$= \begin{pmatrix} C'_o \cdot P_o \; + \; C'_o \cdot P_d \; - \; C'_o \cdot P_o \cdot P_d \; + \; C'_i \cdot P_i \\ - \, C'_i \cdot P_o \cdot P_i \; + \; C'_c \cdot \overline{P_i} \; - \; C'_c \cdot P_o \cdot \overline{P_i} \; - \; C'_c \cdot P_d \cdot \overline{P_i} \\ + \, C'_c \cdot P_o \cdot P_d \cdot \overline{P_i} \; + \; C'_i \cdot P_o \cdot P_i \end{pmatrix}$$

{cancelling $C'_i \cdot P_o \cdot P_i$}

$$= \begin{pmatrix} C'_o \cdot P_o \; + \; C'_o \cdot P_d \; - \; C'_o \cdot P_o \cdot P_d \; + \; C'_i \cdot P_i \; + \; C'_c \cdot \overline{P_i} \\ - \, C'_c \cdot P_o \cdot \overline{P_i} \; - \; C'_c \cdot P_d \cdot \overline{P_i} \; + \; C'_c \cdot P_o \cdot P_d \cdot \overline{P_i} \end{pmatrix}$$

{letting $K_1 = C'_o \cdot P_d \; + \; C'_i \cdot P_i \; + \; C'_c \cdot \overline{P_i} \; - \; C'_c \cdot P_d \cdot \overline{P_i}$}

$$= \begin{pmatrix} K_1 \; + \; C'_o \cdot P_o \; - \; C'_o \cdot P_o \cdot P_d \; - \; C'_c \cdot P_o \cdot \overline{P_i} \\ + \, C'_c \cdot P_o \cdot P_d \cdot \overline{P_i} \end{pmatrix}$$

{taking out P_o as a common factor}

$$= K_1 \; + \; P_o \cdot (C'_o \; - \; C'_o \cdot P_d \; - \; C'_c \cdot \overline{P_i} \; + \; C'_c \cdot P_d \cdot \overline{P_i})$$

NOTATIONS USED IN SECTION 6.3	SHORTHAND
$P(U_n^y \text{ omitted} \mid \langle \mathcal{O}_n \cup \{X_n^s\}, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \setminus \{X_n^s\}\rangle)$	R_o
$P(U_n^y \text{ omitted} \mid \langle \mathcal{O}_n, \mathcal{D}_n \cup \{X_n^s\}, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \setminus \{X_n^s\}\rangle)$	R_d
$P(U_n^y \text{ omitted} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n \cup \{X_n^s\}, \mathcal{C}_n, \mathcal{Z}_n \setminus \{X_n^s\}\rangle)$	R_i
$P(U_n^y \text{ omitted} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n \cup \{X_n^s\}, \mathcal{Z}_n \setminus \{X_n^s\}\rangle)$	R_c

Table A.3: Extensions to the shorthand notation for the analysis in Section 6.3.

{replacing $C'_o - C'_o \cdot P_d$ with $C'_o \cdot \overline{P_d}$}

$$= K_1 + P_o \cdot (C'_o \cdot \overline{P_d} - C'_c \cdot \overline{P_i} + C'_c \cdot P_d \cdot \overline{P_i})$$

{replacing $(-C'_c \cdot \overline{P_i} + C'_c \cdot P_d \cdot \overline{P_i})$ with $(-C'_c \cdot \overline{P_d} \cdot \overline{P_i})$}

$$= K_1 + P_o \cdot (C'_o \cdot \overline{P_d} - C'_c \cdot \overline{P_d} \cdot \overline{P_i})$$

{letting $K_2 = (C'_o - C'_c \cdot \overline{P_i}) \cdot \overline{P_d}$}

$$= K_1 + K_2 \cdot P_o. \tag{A.10}$$

In Eq. (A.10), by definition, K_1 and K_2 are independent of P_o. $K_1 \geqslant 0$, because it is defined as a sum of all positive terms (each term is a product of probabilities). Also, $K_2 \geqslant 0$, since $C'_o \geqslant C'_c$ (from Eq. (A.7)), which in turn implies $C'_o \geqslant \overline{P_i} \cdot C'_c$ ($\overline{P_i}$ being a probability). Thus, $K_1 + K_2 \cdot P_o$ is monotonically increasing in P_o. □

A.3 MONOTONICITY OF $P(U_n^y$ OMITTED$)$

In this section, we show that the exact probability with which U_n^y is omitted, as defined in Definition 6.8, is either independent of or monotonic in all message error probabilities.

In particular, we show that for any tuple $\langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle$, probability $P(U_n^y \text{ omitted} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle)$ is independent of or monotonic in the exact probabilities $P(X_n^s$ corrupted$)$, $P(X_n^s$ delayed$)$ and $P(X_n^s$ omitted$)$. The result implies that $P(U_n^y \text{ omitted} \mid \langle \emptyset, \emptyset, \emptyset, \emptyset, X_n \rangle)$, also denoted as $P(U_n^y$ omitted$)$, can be safely computed using upper bounds on the exact message error probabilities, without the possibility of reliability anomalies.

For brevity of the following proofs, like in the previous section, we extend the shorthand notation introduced in Table A.1 with shorthand for the new intermediate probabilities that are used in the definition of $P(U_n^y \text{ omitted} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle)$ (Definition 6.8). See Table A.3 for the extensions. Using the extended shorthand notation, we first restate the definition of $P(U_n^y \text{ omitted} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle)$.

$$P\,(U_n^y \text{ omitted} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle) = \begin{cases} \begin{pmatrix} R_o \cdot P_o \; + \; R_d \cdot \overline{P_o} \cdot P_d \\ + \, R_i \cdot \overline{P_o} \cdot \overline{P_d} \cdot P_i \\ + \, R_c \cdot \overline{P_o} \cdot \overline{P_d} \cdot \overline{P_i} \end{pmatrix} & \mathcal{Z}_n \neq \emptyset \\ 1 & \mathcal{I}_n \cup \mathcal{C}_n = \emptyset \\ 0 & \mathcal{I}_n \cup \mathcal{C}_n \neq \emptyset. \end{cases} \tag{A.11}$$

Note that probabilities R_o, R_d, R_i, and R_c differ only in terms of whether message X_n^s is inserted from set $\mathcal{Z}_n$ into set $\mathcal{O}_n$, $\mathcal{D}_n$, $\mathcal{I}_n$, or $\mathcal{C}_n$, respectively. In the first two cases, X_n^s is either omitted or delayed and hence it does not participate in voting. Thus, the probability that U_n^y is omitted is the same for these cases, i.e., $R_d = R_o$. In contrast, in the last two cases, X_n^s is neither omitted nor delayed and hence it is guaranteed to participate in voting. Thus, the probability that U_n^y is omitted is zero for these cases, irrespective of whether message X_n^s is incorrectly computed or not, i.e., $R_i = R_c = 0$. In summary,

$$R_o = R_d \geqslant R_i = R_c = 0. \tag{A.12}$$

Next, we prove the monotonicity result.

THEOREM A.7. $P(U_n^y \text{ omitted} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle)$ is either independent of or increasing in P_o, P_d, and P_i.

Proof. If $\mathcal{Z}_n = \emptyset$, then $P(U_n^y \text{ omitted} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle)$ is either 0 or 1, and thus independent of P_o, P_d, and P_i. If $\mathcal{Z}_n \neq \emptyset$, then from Eq. (A.11),

$$\begin{aligned} & P(U_n^y \text{ omitted} \mid \langle \mathcal{O}_n, \mathcal{D}_n, \mathcal{I}_n, \mathcal{C}_n, \mathcal{Z}_n \rangle) \\ & = R_o \cdot P_o \; + \; R_d \cdot \overline{P_o} \cdot P_d \; + \; R_i \cdot \overline{P_o} \cdot \overline{P_d} \cdot P_i \; + \; R_c \cdot \overline{P_o} \cdot \overline{P_d} \cdot \overline{P_i} \end{aligned}$$

{since $R_i = R_c = 0$ (from Eq. (A.12))}

$$= R_o \cdot P_o \; + \; R_d \cdot \overline{P_o} \cdot P_d \tag{A.13}$$

{replacing $\overline{P_o}$ with $1 - P_o$, and simplifying}

$$= R_o \cdot P_o \; + \; R_d \cdot P_d \; - \; R_d \cdot P_o \cdot P_d$$

{since $R_o = R_d$ (from Eq. (A.12))}

$$= R_o \cdot P_o \; + \; R_o \cdot P_d \; - \; R_o \cdot P_o \cdot P_d$$

{replacing $1 - P_d$ with $\overline{P_d}$, and simplifying}

$$= R_o \cdot P_o \cdot \overline{P_d} + R_o \cdot P_d. \tag{A.14}$$

In Eq. (A.13), $R_o \cdot P_o + R_d \cdot \overline{P_o} \cdot P_d$ is independent of P_i, as well as monotonically increasing in P_d, since all other terms are positive. Similarly, in Eq. (A.14), $R_o \cdot P_o \cdot \overline{P_d} + R_o \cdot P_d$ is monotonically increasing in P_o since all other terms are positive. □

A.4 ANALYSIS OF FINAL OUTPUT F_n

In this section, we provide proofs for the upper bounds on the probability that the actuation during the n^{th} control loop iteration is incorrect and the probability that it is omitted. The upper bound were defined earlier in Definitions 6.12 and 6.13, respectively, in Section 6.3.3.

THEOREM A.8. An upper bound on the probability that the actuation during the n^{th} control loop iteration is incorrect is given by

$$Q(Z_n \text{ incorrect}) = \\ \begin{pmatrix} P(Z_n \text{ corrupted}) \\ + Q(U_n^y \text{ incorrect}) \\ + Q(V_n \text{ incorrect}) \end{pmatrix} + \begin{pmatrix} P(Z_n \text{ corrupted}) \\ \times Q(U_n^y \text{ incorrect}) \\ \times Q(V_n \text{ incorrect}) \end{pmatrix},$$

for any $U_n^y \in U_n$.

Proof. We consider two cases based on whether the sensor inputs to any controller voter instance during the n^{th} control loop iteration results in corruption of the controller voter outputs (case 1) or not (case 2). From Definition 6.6, the probability that case 1 occurs is

$$\phi_{case1} = P(U_n^y \text{ incorrect}). \tag{A.15}$$

For this case, since the sensor inputs to controller voter instance in controller task C_n^y results in corruption of its output, voter instances in all controller tasks choose an incorrect output, too. Thus, all control commands transmitted are incorrect, and it is guaranteed that the actuation during the n^{th} control loop iteration is incorrect. Hence, the conditional probability in this case is

$$\phi_{cond1} = 1. \tag{A.16}$$

The probability that case 2 occurs is

$$\phi_{case2} = 1 - \phi_{case1} = 1 - P(U_n^y \text{ incorrect}). \tag{A.17}$$

For this case, the conditional probability that the actuation during the n^{th} control loop iteration is incorrect depends on two sources:

(a) actuator voter instance output V_n is incorrect, and

(b) actuator host is affected by incorrect computation errors.

From Definition 6.9, the probability for case (a) is

$$\phi_{case2a} = P(V_n \text{ incorrect}), \tag{A.18}$$

and from Definition 6.3, the probability for case (b) is

$$\phi_{case2b} = P(Z_n \text{ corrupted}). \tag{A.19}$$

Using theorem $P(A_1 \cup A_2) = P(A_1) + P(A_2) - P(A_1) \cdot P(A_2)$, the conditional probability for case 2 is

$$\phi_{cond2} = \phi_{case2a} + \phi_{case2b} - \phi_{case2a}\, \phi_{case2b}. \tag{A.20}$$

By the law of total probability, the probability that the actuation during the n^{th} control loop iteration is incorrect is given by

$$P(Z_n \text{ incorrect})$$
$$= \phi_{case1}\, \phi_{cond1} + \phi_{case2}\, \phi_{cond2}.$$

{simplifying using Eq. (A.15)-Eq. (A.20)}

$$= \left(\begin{array}{l} P(U_n^y \text{ incorrect}) \\ + \left(\begin{array}{l} (1 - P(U_n^y \text{ incorrect})) \\ \times \left(\begin{array}{l} P(V_n \text{ incorrect}) + P(Z_n \text{ corrupted}) \\ -P(V_n \text{ incorrect}) \times P(Z_n \text{ corrupted}) \end{array} \right) \end{array} \right) \end{array} \right)$$

{simplifying, and dropping all negative terms to obtain an upper bound}

$$\leqslant \left(\begin{array}{l} \left(\begin{array}{l} P(U_n^y \text{ incorrect}) + P(V_n \text{ incorrect}) \\ + P(Z_n \text{ corrupted}) \end{array} \right) \\ + \left(\begin{array}{l} P(U_n^y \text{ incorrect})) \times P(V_n \text{ incorrect}) \\ \times P(Z_n \text{ corrupted}) \end{array} \right) \end{array} \right)$$

{replacing exact probabilities with monotonic upper bounds}

$$\leqslant \left(\begin{array}{l} \left(\begin{array}{l} Q(U_n^y \text{ incorrect}) + Q(V_n \text{ incorrect}) \\ + P(Z_n \text{ corrupted}) \end{array} \right) \\ + \left(\begin{array}{l} Q(U_n^y \text{ incorrect}) \times Q(V_n \text{ incorrect}) \\ \times P(Z_n \text{ corrupted}) \end{array} \right) \end{array} \right). \qquad \square$$

Theorem A.8 is similar to Theorem A.9, except for the last step of the proof, where exact probabilities are replaced with the respective monotonic upper bounds.

THEOREM A.9. An upper bound on the probability that the actuation during the n^{th} control loop iteration is skipped is given by

$$Q(Z_n \text{ skipped}) = \left(\begin{array}{l} P(Z_n \text{ omitted}) + Q(U_n^y \text{ omitted}) \\ + Q(V_n \text{ omitted}) \end{array} \right) + \left(\begin{array}{l} P(Z_n \text{ omitted}) \times Q(U_n^y \text{ omitted}) \\ \times Q(V_n \text{ omitted}) \end{array} \right),$$

for any $U_n^y \in U_n$.

Proof. We consider two cases based on whether the sensor inputs to any controller voter instance during the n^{th} control loop iteration results in omission of the controller voter outputs (case 1) or not (case 2). From Definition 6.8, the probability that case 1 occurs is

$$\phi_{\text{case1}} = P(U_n^y \text{ omitted}). \tag{A.21}$$

For this case, since the delayed/omitted sensor inputs to controller voter instance in controller task C_n^y results in omission of its output, voter instances in all controller tasks omit their outputs, too (see Section 6.3.1.2 for details). Thus, none of the control commands are prepared, and it is guaranteed that the actuation during the n^{th} control loop iteration is skipped. Hence, the conditional probability here is

$$\phi_{\text{cond1}} = 1. \tag{A.22}$$

The probability that case 2 occurs is

$$\phi_{\text{case2}} = 1 - \phi_{\text{case1}} = 1 - P(U_n^y \text{ omitted}). \tag{A.23}$$

For this case, the conditional probability that the actuation during the n^{th} control loop iteration is skipped depends on two sources:

(a) actuator voter instance output V_n is omitted, and

(b) actuator host is affected by message omission errors.

From Definition 6.11, the probability for case (a) is

$$\phi_{\text{case2a}} = P(V_n \text{ omitted}), \tag{A.24}$$

and from Definition 6.1, the probability for case (b) is

$$\phi_{\text{case2b}} = P(Z_n \text{ omitted}). \tag{A.25}$$

Using theorem $P(A_1 \cup A_2) = P(A_1) + P(A_2) - P(A_1) \cdot P(A_2)$, the conditional probability for case 2 is

$$\phi_{\text{cond2}} = \phi_{\text{case2a}} + \phi_{\text{case2b}} - \phi_{\text{case2a}}\, \phi_{\text{case2b}}. \tag{A.26}$$

By the law of total probability, the probability that the actuation during the n^{th} control loop iteration is skipped is given by

$$\begin{aligned} &P(Z_n \text{ skipped}) \\ &= \phi_{\text{case1}}\, \phi_{\text{cond1}} + \phi_{\text{case2}}\, \phi_{\text{cond2}}. \end{aligned}$$

{simplifying using Eq. (A.21)-Eq. (A.26)}

$$= \left(\begin{array}{l} P(U_n^y \text{ omitted}) \\ + \left(\begin{array}{l} (1 - P(U_n^y \text{ omitted})) \\ \times \left(\begin{array}{l} P(V_n \text{ omitted}) + P(Z_n \text{ omitted}) \\ -P(V_n \text{ omitted}) \times P(Z_n \text{ omitted}) \end{array} \right) \end{array} \right) \end{array} \right)$$

{simplifying, and dropping all negative terms for an upper bound}

$$\leqslant \left(\begin{array}{l} \left(\begin{array}{l} P(U_n^y \text{ omitted}) + P(V_n \text{ omitted}) \\ + P(Z_n \text{ omitted}) \end{array} \right) \\ + \left(\begin{array}{l} P(U_n^y \text{ omitted})) \times P(V_n \text{ omitted}) \\ \times P(Z_n \text{ omitted}) \end{array} \right) \end{array} \right). \qquad \square$$

B SAP PROOFS

In Section 7.3.3, we introduced SAP, a sound approximation approach based on numerical analysis to estimate a lower bound on the MTTF of periodic systems with weakly-hard constraints. The approach rested on prior results from the reliability modeling literature, particularly the reliability lower bound of the *a-within-consecutive-b-out-of-c:F* system model (or *a/Con/b/c:F* in short).[1]

In the following, we first provide a primer on the a/Con/b/c:F system model along with an unambiguous definition of its reliability lower bound, since prior works do not explicitly enumerate all its corner cases. We also provide a proof of monotonicity of this reliability lower bound. Using this monotonicity result, we then derive the MTTF lower bound defined in Eq. (7.11) as part of the SAP approach.

B.1 THE A/CON/B/C:F SYSTEM MODEL

An a/Con/b/c:F system [124, Section 11.4] consists of c linearly or cyclically ordered components. The system fails if there are a ($a \leqslant b$) or more failed components among any consecutive b ($b \leqslant c$) components. This model can be used, for example, in quality control of a manufacturing process, where b items manufactured consecutively are randomly selected for a quality check, and if at least a of these are defective, the manufacturing process is required to be readjusted. In this dissertation, since we apply the model to the problem of MTTF estimation of periodic systems, we are interested in the linear model.

We are particularly interested in the *reliability* of the a/Con/b/c:F system with IID components, i.e., the probability that the system does not fail given an IID failure probability P_F for each of the c components. This problem has been thoroughly studied in the past [143–145, 162, 168, 169, 177, 178, 199] for different flavors of the system model and resulting in exact as well as approximate solutions (see [124, Section 11.4] for a comprehensive summary). However, since we use the reliability definition to define $g_{LB}(n)$, a lower bound on $g(n) = \Pr[N(\mathcal{S}, \mathcal{R}) = n]$ (with $a = k - m + 1$, $b = k$, $c = n - 1$, and $\mathcal{R} = (m, k)$), and since $n \cdot g_{LB}(n)$ needs to be soundly integrated as

1 While the a-within-consecutive-b-out-of-c:F system model is typically denoted as k-within-consecutive-m-out-of-n:F system in the reliability modeling literature, we choose to replace k, m, and n with a, b, and c (respectively) in order to disambiguate the system model notation from the notation corresponding to the weakly-hard robustness specifications defined in Section 7.2.

per Eq. (7.5) to estimate a lower bound on the MTTF of a periodic system with (m, k) robustness, we need a reliability definition (exact or a lower bound) of the linear a/Con/b/c:F system model that can be either:

- symbolically integrated with respect to n, or
- computed quickly for multiple (thousands of) and very large values of n (up to $n = 10^{50}$) for numerical integration.

Since we were not able obtain a reliability definition satisfying the first requirement, the SAP approach relies on the second alternative, and specifically on the results of Sfakianakis et al. [199].

B.1.1 Reliability of an a/Con/b/c:F System

Consider a linear a/Con/b/c:F system with IID components, each of which fails with probability P_F, and let $P_S = 1 - P_F$. Let $R(a, b, c)$ denote the exact reliability of the system. We use the results of Sfakianakis et al. [199] to derive a lower bound on $R(a, b, c)$, denoted $R_{LB}(a, b, c)$, for large values of c. Sfakianakis et al.'s analysis breaks the problem into smaller subproblems for which exact analyses are available and that can be computed quickly. However, neither Sfakianakis et al. [199] nor any prior work explicitly enumerates the reliability definitions for an exhaustive set of subproblems, i.e., which covers all possibles values of parameters a, b, and c. Therefore, we provide an unambiguous definition of the reliability lower bound $R_{LB}(a, b, c)$ that draws from Sfakianakis et al.'s analysis for large values of c and from other prior works for some special cases and smaller values of c. Note that in many cases, there are multiple ways to define $R_{LB}(a, b, c)$, in which case we prefer a definition that can be quickly computed. We summarize our definition of $R_{LB}(a, b, c)$ in Table B.1, and provide a reliability definition for each case in Table B.1 next.

Case 1 is trivial: if $a = 0$, the system is always unreliable, thus,

$$R_1(a, b, c) = 0. \tag{B.1}$$

Similarly, Case 2 is also trivial: if $a = 1$, the system is reliable only if none of the c components fail, thus,

$$R_2(a, b, c) = (P_S)^c. \tag{B.2}$$

For the special case when $a = 2$, Naus [162] and Sfakianakis et al. [199] provide an exact reliability definition (or see Equation 11.9 and

#	CASE DESCRIPTION	DEFINITION	TYPE
1	$a = 0$	$R_1(a,b,c)$	Exact
2	$a = 1$	$R_2(a,b,c)$	Exact
3	$a = 2 \wedge c \leqslant 4b$	$R_3(a,b,c)$	Exact
4	$a = 2 \wedge c > 4b$	$R_4(a,b,c)$	Lower Bound
5	$a > 2 \wedge c \leqslant 2b \wedge a = b$	$R_5(a,b,c)$	Exact
6	$a > 2 \wedge c \leqslant 2b \wedge a \neq b \wedge c \leqslant b$	$R_6(a,b,c)$	Exact
7	$a > 2 \wedge c \leqslant 2b \wedge a \neq b \wedge c > b$	$R_7(a,b,c)$	Exact
8	$a > 2 \wedge c > 2b$	$R_8(a,b,c)$	Lower Bound

Table B.1: Reliability lower bound of a linear a/Con/b/c:F system with IID components. TYPE indicates whether the reliability definition is an exact value or a lower bound on the exact value.

11.10 in [124, Section 11.4.1]). If c is small, this exact definition can be quickly computed. Thus, we define

$$R_3(a,b,c) = \sum_{i=0}^{\left\lfloor \frac{c+b-1}{b} \right\rfloor} \binom{c-(i-1)(b-1)}{i} (P_F)^i (P_S)^{c-i}. \tag{B.3}$$

If c is large, though, we do not use an exact reliability but rely on the reliability lower bound proposed by Sfakianakis et al. [199] (see Equation 11.16 in [124, Section 11.4.1] for an explanation of this lower bound).[2] In particular, for $a = 2$ and $c > 4b$ (Case 4), this lower bound reduces to the following definition,

$$R_4(a,b,c) = R_3(a,b,b+t-1)(R_3(a,b,b+3))^u, \tag{B.4}$$
$$\text{where } \ t = (c-b+1) \bmod 4 \ \text{ and } \ u = \left\lfloor \frac{c-b+1}{4} \right\rfloor.$$

For the general case $a > 2$, we consider four sub-cases. First, we consider the special case $a = b$, for which the a/Con/b/c:F system reduces to a simpler *Con/a/c:F* system [124, Chapter 9]. In particular,

2 Notice that while we are interested in a reliability lower bound, we point to Equation 11.16 in [124, Section 11.4.1] that refers to an upper bound. This mismatch is due to slight inconsistency in how the textbook chapter [124, Section 11.4.1] adopts the result from the original paper by Sfakianakis et al. [199]. Notations L and U in Table I in [199] denote lower and upper bounds (respectively) on the *failure rate* of the system. Equation 11.16 in [124, Section 11.4.1] uses the same notation. Thus, UB_a in Equation 11.16 in [124, Section 11.4.1] actually refers to an upper bound on the system failure probability, and not an upper bound on the system reliability (although the text in the chapter may seem contradictory). Since we require a lower bound on the system reliability, and since system reliability is one minus its failure rate, we use $1 - UB_a$, where UB_a is defined as in Equation 11.16 in [124, Section 11.4.1].

for $a = b$ and $c \leqslant 2b$ (Case 5), we define an exact reliability using the following closed-form expression [124, Section 9.1.1, Equation 9.20]:

$$R_5(a, b, c) = \begin{cases} 1 & 0 \leqslant c < a \\ 1 - (P_F)^a - (c - a)(P_F)^a(P_S) & a \leqslant c \leqslant 2a. \end{cases} \tag{B.5}$$

Next, we consider the special case where $a \neq b$ but $c \leqslant b$ (Case 6). In this case, the number of working components in the system follows the binomial distribution with parameters c and P_F. Thus, as per Equation 7.2 in [124, Section 7.1.1], the exact reliability in this case is

$$R_6(a, b, c) = \sum_{i=c-a+1}^{c} \binom{c}{i} (P_S)^i (P_F)^{c-i}. \tag{B.6}$$

Case 7 is the last special case where $a \neq b$ and $b < c \leqslant 2b$. In this case, Sfakianakis et al. [199]'s analysis provides an exact reliability using the aforementioned cases as subproblems (see Equation 11.14 in [124, Section 11.4.1] for details). Their recursive definition is given below.

$$R_7(a, b, c) = \sum_{i=0}^{a-1} \binom{b-s}{i} (P_F)^i (P_S)^{b-s-i} M(a', s, 2s), \tag{B.7}$$

$$\text{where} \;\; s = c - b \;\; \text{and} \;\; a' = a - i$$

$$\text{and} \;\; M(a', s, 2s) = \begin{cases} 1 & a' > s \\ R_2(a', s, 2s) & a' = 1 \\ R_3(a', s, 2s) & a' = 2 \\ R_5(a', s, 2s) & a' > 2 \land a' = s \\ R_7(a', s, 2s) & a' > 2 \land a' \neq s. \end{cases}$$

Finally, we consider the most general case where $a > 2$ and $c > 2b$ (Case 8). For this case, we once again rely on the reliability lower bound proposed by Sfakianakis et al. [199], which we also used for Case 4. However, the lower bound cannot be further simplified as in Case 4, and relies on Cases 5-7, which we denote together as $R_{5\text{-}7}(a, b, c)$.

$$R_8(a, b, c) = R_{5\text{-}7}(a, b, b + t - 1) \times (R_{5\text{-}7}(a, b, b + 3))^u, \tag{B.8}$$

$$\text{where} \;\; t = (c - b + 1) \bmod 4 \;\; \text{and} \;\; u = \left\lfloor \frac{c - b + 1}{4} \right\rfloor$$

$$\text{and} \;\; R_{5\text{-}7}(a, b, c) = \begin{cases} R_5(a, b, c) & a > 2 \land a = b \\ R_6(a, b, c) & a > 2 \land a \neq b \land c \leqslant b \\ R_7(a, b, c) & a > 2 \land a \neq b \land c > b. \end{cases} \tag{B.9}$$

Cases 1 to 8 are mutually exclusive and exhaustive, i.e., they cover all possible values of parameters a, b, and c. Therefore, a generic lower bound $R_{LB}(a, b, c)$ is defined by combining all of these cases.

B.1.2 Monotonicity of Reliability Lower Bound

The derivation of the MTTF lower bound (which is provided in Appendix B.2 next) depends on the property that the reliability lower bound $R_{LB}(a, b, c)$ decreases with increasing c. This property trivially holds for cases $a = 0$ and $a = 1$, as seen from the definitions of $R_1(a, b, c)$ (Eq. (B.1)) and $R_2(a, b, c)$ (Eq. (B.2)). However, proving the property for cases $a > 2$ and $a = 2$ is not trivial and discussed explicitly in the following.

The definition of $R_{LB}(a, b, c)$ for $a > 2$ is split into multiple cases (Cases 5-8 in Table B.1). In fact, because of the recursive definitions for Cases 7 and 8, the definition of $R_{LB}(a, b, c)$ for $a > 2$ actually depends on the remaining cases as well. This recursive dependence makes it hard to prove that $R_{LB}(a, b, c)$ decreases with increasing c.

Instead, we prove a weaker property: we show that if $R_{LB}(a, b, c)$ decreases with increasing c for small values of c (i.e., for $c \leqslant 2b$), then $R_{LB}(a, b, c)$ also decreases with increasing c for larger values of c (i.e., for $c > 2b$). Since b is typically relatively small (recall from Section 7.3.3 that $b = k$), the premise can be easily checked for specific values of a, b, c and P_F through exhaustive enumeration.

Note that in all theorems below, we consider $a \leqslant b \leqslant c$ (recall the a/Con/b/c:F system model).

THEOREM B.1. For $a > 2$, if $R_{LB}(a, b, c)$ is monotonically decreasing for $c \in \{a, \dots, 2b + 1\}$, then $R_{LB}(a, b, c)$ is also monotonically decreasing for $c \geqslant 2b + 1$, i.e.,

$$\begin{aligned} &\text{if} && \forall c \leqslant 2b : \ R_{LB}(a, b, c) \geqslant R_{LB}(a, b, c+1), \\ &\text{then} && \forall c > 2b : \ R_{LB}(a, b, c) \geqslant R_{LB}(a, b, c+1). \end{aligned} \tag{B.10}$$

PROOF. Let

$$\Omega = \frac{R_{LB}(a, b, c)}{R_{LB}(a, b, c+1)}. \tag{B.11}$$

We prove that $\Omega \geqslant 1$ when $c > 2b$.

Since $a > 2$ and $c > 2b$, both terms $R_{LB}(a, b, c)$ and $R_{LB}(a, b, c + 1)$ in Eq. (B.11) are resolved using Case 8 in Table B.1. Thus, from $R_8(a, b, c)$'s definition in Eq. (B.8), and letting $x = c - b + 1$,

$$\Omega = \frac{R_{5\text{-}7}(a, b, b + (x \bmod 4) - 1)(R_{5\text{-}7}(a, b, b+3))^{\lfloor \frac{x}{4} \rfloor}}{R_{5\text{-}7}(a, b, b + ((x+1) \bmod 4) - 1)(R_{5\text{-}7}(a, b, b+3))^{\lfloor \frac{x+1}{4} \rfloor}}. \tag{B.12}$$

To simplify Eq. (B.12), we consider two separate cases based on whether $x \bmod 4 = 3$ or $x \bmod 4 < 3$.

Case A ($x \bmod 4 = 3$): Since $x \bmod 4 = 3$ implies that $(x+1) \bmod 4 = 0$ and $\lfloor \frac{x+1}{4} \rfloor = \lfloor \frac{x}{4} \rfloor + 1$, Eq. (B.12) is simplified as follows.

$$\Omega = \frac{R_{5\text{-}7}(a,b,b+2)(R_{5\text{-}7}(a,b,b+3))^{\lfloor \frac{x}{4} \rfloor}}{R_{5\text{-}7}(a,b,b-1)(R_{5\text{-}7}(a,b,b+3))^{\lfloor \frac{x}{4} \rfloor + 1}}$$

{dividing numerator and denominator by $(R_{5\text{-}7}(a,b,b+3))^{\lfloor \frac{x}{4} \rfloor}$}

$$\Omega = \frac{R_{5\text{-}7}(a,b,b+2)}{R_{5\text{-}7}(a,b,b-1)R(a,b,b+3)}$$

{since $R_{5\text{-}7}(a,b,b-1) \leqslant 1$ (being a probability)}

$$\Omega \geqslant \frac{R_{5\text{-}7}(a,b,b+2)}{R_{5\text{-}7}(a,b,b+3)}. \tag{B.13}$$

Case B ($x \bmod 4 < 3$): Since $x \bmod 4 < 3$ implies that $\lfloor \frac{x+1}{4} \rfloor = \lfloor \frac{x}{4} \rfloor$ and $(x+1) \bmod 4 = 1 + x \bmod 4$, Eq. (B.12) can be simplified as

$$\Omega = \frac{R_{5\text{-}7}(a,b,b+(x \bmod 4)-1)(R_{5\text{-}7}(a,b,b+3))^{\lfloor \frac{x}{4} \rfloor}}{R_{5\text{-}7}(a,b,b+(x \bmod 4))(R_{5\text{-}7}(a,b,b+3))^{\lfloor \frac{x}{4} \rfloor}}$$

{dividing numerator and denominator by $(R(a,b,b+3))^{\lfloor \frac{x}{4} \rfloor}$}

$$\Omega = \frac{R_{5\text{-}7}(a,b,b+(x \bmod 4)-1)}{R_{5\text{-}7}(a,b,b+(x \bmod 4))}. \tag{B.14}$$

Next, we unify the two cases. Since $a \leqslant b$,

$$2 < a \implies 2 < b \implies 2 + b < 2b. \tag{B.15}$$

Also, for Case B in particular,

$$x \bmod 4 < 3 \implies b + (x \bmod 4) - 1 < b + 2$$

{from Eq. (B.15)}

$$\implies b + (x \bmod 4) - 1 < 2b \tag{B.16}$$

Using Eq. (B.15) and Eq. (B.16), the constraints on Ω in both the cases, i.e. Eq. (B.13) and Eq. (B.14), can be unified as

$$\Omega \geqslant \frac{R_{5\text{-}7}(a,b,c')}{R_{5\text{-}7}(a,b,c'+1)}, \tag{B.17}$$

where $c' < 2b$ ($c' = b + 2$ in case of Eq. (B.13), and $c' = b + (x \bmod 4) - 1$ in case of Eq. (B.14)). Now we simply need to show that the RHS in Eq. (B.17) is greater than or equal to 1.

Since $c' < 2b$, from the premise in Eq. (B.10),

$$R_{LB}(a, b, c') \geqslant R_{LB}(a, b, c' + 1). \tag{B.18}$$

In addition, since $a > 2$ and $c' < 2b$, we define $R_{LB}(a, b, c')$ using Cases 5-7 in Table B.1. Thus, from Eq. (B.9), $R_{LB}(a, b, c') = R_{5\text{-}7}(a, b, c')$ (which combines Cases 5-7). Similarly, $a > 2$ and $c' + 1 \leqslant 2b$, and thus $R_{LB}(a, b, c' + 1) = R_{5\text{-}7}(a, b, c' + 1)$. Substituting these definitions of $R_{LB}(a, b, c')$ and $R_{LB}(a, b, c' + 1)$ in Eq. (B.18),

$$R_{5\text{-}7}(a, b, c') \geqslant R_{5\text{-}7}(a, b, c' + 1)$$

{upon rearranging, and from Eq. (B.17)}

$$\Omega \geqslant \frac{R_{5\text{-}7}(a, b, c')}{R_{5\text{-}7}(a, b, c' + 1)} \geqslant 1. \tag{B.19}$$

□

We adopt a similar approach for $a = 2$ as well. That is, we once again prove a weaker monotonicity property: we show that if $R_{LB}(a, b, c)$ decreases with increasing c for small values of c (i.e., for $c \leqslant 4b$), then $R_{LB}(a, b, c)$ also decreases with increasing c for larger values of c (in this case, for $c > 4b$). This is because the reliability lower bound for $a = 2$ is defined using Cases 3 and 4, and definitions of both $R_3(a, b, c)$ and $R_4(a, b, c)$ make it non-trivial to establish monotonicity.

Since $R_4(a, b, c)$'s definition is similar in structure to $R_8(a, b, c)$'s definition (both use Sfakianakis et al.'s analysis), the proof structure of the following theorem is same as that of Theorem B.1.

THEOREM B.2. For $a = 2$, if $R_{LB}(a, b, c)$ is monotonically decreasing for $c \in \{a, \ldots, 4b + 1\}$, then $R_{LB}(a, b, c)$ is also monotonically decreasing for $c \geqslant 4b + 1$, i.e.,

$$\begin{aligned} &\text{if} && \forall c \leqslant 4b : \; R_{LB}(a, b, c) \geqslant R_{LB}(a, b, c + 1), \\ &\text{then} && \forall c > 4b : \; R_{LB}(a, b, c) \geqslant R_{LB}(a, b, c + 1). \end{aligned} \tag{B.20}$$

PROOF. Let

$$\Omega = \frac{R_{LB}(a, b, c)}{R_{LB}(a, b, c + 1)}. \tag{B.21}$$

We prove that $\Omega \geqslant 1$ when $c > 4b$.

Since $a = 2$ and $c > 4b$, both terms $R_{LB}(a, b, c)$ and $R_{LB}(a, b, c + 1)$ in Eq. (B.21) are resolved using Case 4 in Table B.1. Thus, from $R_4(a, b, c)$'s definition in Eq. (B.4), and letting $x = c - b + 1$,

$$\Omega = \frac{R_3(a, b, b + (x \bmod 4) - 1)(R_3(a, b, b + 3))^{\lfloor \frac{x}{4} \rfloor}}{R_3(a, b, b + ((x + 1) \bmod 4) - 1)(R_3(a, b, b + 3))^{\lfloor \frac{x+1}{4} \rfloor}}. \tag{B.22}$$

To simplify Eq. (B.22), we consider two separate cases based on whether $x \bmod 4 = 3$ or $x \bmod 4 < 3$.

Case A ($x \bmod 4 = 3$): Since $x \bmod 4 = 3$ implies that $(x + 1) \bmod 4 = 0$ and $\lfloor \frac{x+1}{4} \rfloor = \lfloor \frac{x}{4} \rfloor + 1$, Eq. (B.22) simplifies as follows.

$$\Omega = \frac{R_3(a, b, b + 2)(R_3(a, b, b + 3))^{\lfloor \frac{x}{4} \rfloor}}{R_3(a, b, b - 1)(R_3(a, b, b + 3))^{\lfloor \frac{x}{4} \rfloor + 1}}$$

{dividing numerator and denominator by $(R_3(a, b, b + 3))^{\lfloor \frac{x}{4} \rfloor}$}

$$\Omega = \frac{R_3(a, b, b + 2)}{R_3(a, b, b - 1)R_3(a, b, b + 3)}$$

{since $R_3(a, b, b - 1) \leqslant 1$ (being a probability)}

$$\Omega \geqslant \frac{R_3(a, b, b + 2)}{R_3(a, b, b + 3)}. \tag{B.23}$$

Case B ($x \bmod 4 < 3$): Since $x \bmod 4 < 3$ implies that $\lfloor \frac{x+1}{4} \rfloor = \lfloor \frac{x}{4} \rfloor$ and $(x + 1) \bmod 4 = 1 + x \bmod 4$, Eq. (B.22) can be simplified as

$$\Omega = \frac{R_3(a, b, b + (x \bmod 4) - 1)(R_3(a, b, b + 3))^{\lfloor \frac{x}{4} \rfloor}}{R_3(a, b, b + (x \bmod 4)(R_3(a, b, b + 3))^{\lfloor \frac{x}{4} \rfloor}}$$

{dividing numerator and denominator by $(R(a, b, b + 3))^{\lfloor \frac{x}{4} \rfloor}$}

$$\Omega = \frac{R_3(a, b, b + (x \bmod 4) - 1)}{R_3(a, b, b + (x \bmod 4))}. \tag{B.24}$$

Next, we unify the two cases. Since $a \leqslant b$,

$$2 = a \implies 2 \leqslant b \implies 2 + b \leqslant 2b < 4b. \tag{B.25}$$

Also, for Case B in particular,

$$x \bmod 4 < 3 \implies b + (x \bmod 4) - 1 < b + 2$$

{from Eq. (B.25)}

$$\implies b + (x \bmod 4) - 1 < 4b. \tag{B.26}$$

Using Eq. (B.25) and Eq. (B.26), the constraints on Ω in both the cases, i.e., Eq. (B.23) and Eq. (B.24), can be unified as

$$\Omega \geqslant \frac{R_3(a, b, c')}{R_3(a, b, c' + 1)}, \tag{B.27}$$

where $c' < 4b$ ($c' = b + 2$ in case of Eq. (B.23), and $c' = b + (x \bmod 4) - 1$ in case of Eq. (B.24)). Now we simply need to show that the RHS in Eq. (B.27) is greater than or equal to 1.

Since $c' < 4b$, from the *if* condition in Eq. (B.20),

$$R_{LB}(a, b, c') \geqslant R_{LB}(a, b, c' + 1). \tag{B.28}$$

In addition, since $a = 2$ and $c' < 4b$, from Table B.1, $R_{LB}(a, b, c')$ is defined using Case 3. Thus, from Eq. (B.3), $R_{LB}(a, b, c') = R_3(a, b, c')$. Similarly, since $c' < 4b$ also implies that $c' + 1 \leqslant 4b$, $R_{LB}(a, b, c' + 1) = R_3(a, b, c' + 1)$. Substituting these definitions of $R_{LB}(a, b, c')$ and $R_{LB}(a, b, c' + 1)$ in Eq. (B.28),

$$R_3(a, b, c') \geqslant R_3(a, b, c' + 1)$$

{upon rearranging, and from Eq. (B.27)}

$$\Omega \geqslant \frac{R_3(a, b, c')}{R_3(a, b, c' + 1)} \geqslant 1. \tag{B.29}$$

$\square$

In the next section, while deriving the MTTF lower bound, we assume that $R_{LB}(a, b, c)$ decreases with increasing c. When applying the proposed analysis SAP (e.g., in the evaluation results presented in Section 7.4), for every use of $R_{LB}(a, b, c)$, we check that the premise of Theorem B.1 or Theorem B.2 (depending on the value of a) holds in order to justify the monotonicity assumption.

B.2 DERIVATION OF THE MTTF LOWER BOUND

In the following, we derive MTTF_{LB} defined in Eq. (7.11) as part of the SAP approach (Section 7.3.3). Recall the definition of $g_{LB}(n)$ from Eq. (7.10). The MTTF lower bound derivation depends on the property that $g_{LB}(n)$ decreases with increasing n. Since all the terms except $R_{LB}(k - m + 1, k, n - 1)$ in the definition of $g_{LB}(n)$ are independent of n, $g_{LB}(n)$ decreases with increasing n if $R_{LB}(k-m+1,k,n-1)$ decreases with increasing n, which we proved in the previous section.

THEOREM B.3. A lower bound on the MTTF of system S with period T and robustness specification (m,k) is given by:

$$\text{MTTF}_{LB} = \sum_{i=0}^{D-1} \Big(d_i T \times g_{LB}(d_{i+1}) \times (d_{i+1} - d_i) \Big). \tag{B.30}$$

Proof. From Eq. (7.5), MTTF is defined as

$$\text{MTTF} = T \sum_{n=0}^{\infty} n \cdot \Pr[N(S,\mathcal{R}) = n]. \tag{B.31}$$

Since $g_{LB} \leqslant g(n) = \Pr[N(S,\mathcal{R}) = n]$ (recall from Section 7.3.3), we lower-bound MTTF as

$$\text{MTTF} \geqslant T \sum_{n=0}^{\infty} n \times g_{LB}(n). \tag{B.32}$$

Next, we split the summation range $(0,\infty)$ in Eq. (B.32) into a finite number of subintervals $(0,d_0]$, $(d_0,\, d_1], \ldots,\, (d_{D-1},\, d_D]$, $(d_D,\, \infty)$. Further, since all terms under the summation are non-negative, and since we are interested in a lower bound, we drop the summation terms corresponding to subintervals $(0,d_0]$ and $(d_D,\, \infty)$. Thus,

$$\text{MTTF} \geqslant T \sum_{i=0}^{D-1} \sum_{n=d_i}^{d_{i+1}} n \times g_{LB}(n). \tag{B.33}$$

Now, since $g_{LB}(n)$ is decreasing with increasing n, for each interval $(d_i,\, d_{i+1}]$, we replace $g_{LB}(n)$ with $g_{LB}(d_{i+1})$, which is a constant with respect to n. This replacement yields the desired lower bound.

$$\text{MTTF} \geqslant T \sum_{i=0}^{D-1} g_{LB}(d_{i+1}) \times \sum_{n=d_i}^{d_{i+1}} n$$

{using sum of arithmetic progression}

$$\text{MTTF} \geqslant T \sum_{i=0}^{D-1} g_{LB}(d_{i+1}) \times \frac{(d_{i+1} - d_i + 1)(d_i + d_{i+1})}{2}$$

{since $d_{i+1} - d_i + 1 > d_{i+1} - d_i$ and $d_i + d_{i+1} > 2d_i$}

$$\text{MTTF} \geqslant T \sum_{i=0}^{D-1} g_{LB}(d_{i+1}) \times (d_{i+1} - d_i) \times (d_i). \tag{B.34}$$

□

C IMPLEMENTING PMC IN PRISM

In Section 7.3.1, we introduced PMC, an approach based on Markov chain analysis to estimate the exact MTTF of periodic systems with weakly-hard constraints. As per PMC, the system is modeled as a labeled discrete-time Markov chain M. For the (m, k) weakly-hard constraint, for example, another monitor Markov chain $\mathrm{Monitor}(M, k)$ (classified as a Type-1 monitor) runs alongside M. Each step of the monitor is assumed to have a reward of 1. The set of all states in $\mathrm{Monitor}(M, k)$ that violate the (m, k) constraints are denoted $\mathrm{Bad}(m, k)$ and are made *absorbing*. The MTTF of the system is then given by $T \times E$, where T denotes the system period, and E denotes the expected reward until any state in $\mathrm{Bad}(m, k)$ is reached starting from an initial state.

In this chapter, we explain how the Type-1 monitor representation $\mathrm{Monitor}(M, k)$ and its optimized variants (Type 2 and Type 3) can be encoded in the PRISM language, and given an encoded model, how the expected reward E is computed. In the end, we report on comparison of PRISM's performance with that of Storm, which is a more recent probabilistic model checker.

C.1 EXAMPLE

PRISM accepts discrete-time Markov chains described using a state-based modeling language based on the *reactive modules* formalism of Alur and Henzinger [4]. For an example robustness specification of $(5, 10)$, we illustrate implementations of the corresponding Type-1, Type-2, and Type-3 monitors in the PRISM language in Listing C.1, Listing C.2, and Listing C.3, respectively. These implementations are explained in brief in the following.

C.1.1 Type-1 Monitor

The keyword `dtmc` indicates to PRISM that the following model should be interpreted as a *discrete time Markov chain*. Constant `q` denotes the iteration failure probability bound P_F. Each boolean variable `si` keeps track of whether the i^{th} most recent iteration of the system was successful (`true`) or not (`false`). The formula `num_failures_k_1` thus computes the number of failed iterations among the last $k - 1$ consecutive iterations. It is used to decide whether a new iteration results in the violation of the (m, k) robustness specification. That is, if the new

Listing C.1: Type-1 PRISM model for $(5, 10)$

```
dtmc

const int m = 5;
const int k = 10;
const double q = 1e-10;
formula p = 1.0 - q;

formula num_failures_k_1 = (s1?0:1) + (s2?0:1) + (s3?0:1) +
                           (s4?0:1) + (s5?0:1) + (s6?0:1) +
                           (s7?0:1) + (s8?0:1) + (s9?0:1);

formula failure_allowed = (num_failures_k_1 < k-m);

module reliability_analysis

s1  : bool init true;
s2  : bool init true;
s3  : bool init true;
s4  : bool init true;
s5  : bool init true;
s6  : bool init true;
s7  : bool init true;
s8  : bool init true;
s9  : bool init true;
s10 : bool init true;

safe : bool init true;

[] true -> p: (safe'=safe)
              & (s1'=true)  & (s2'=s1) & (s3'=s2) & (s4'=s3)
              & (s5'=s4)    & (s6'=s5) & (s7'=s6) & (s8'=s7)
              & (s9'=s8)    & (s10'=s9)
         + q: (safe'=safe & failure_allowed)
              & (s1'=false) & (s2'=s1) & (s3'=s2) & (s4'=s3)
              & (s5'=s4)    & (s6'=s5) & (s7'=s6) & (s8'=s7)
              & (s9'=s8)    & (s10'=s9);

endmodule

rewards "steps"
    true : 1;
endrewards
```

iteration is not successful, and if the last $k-1$ consecutive iterations already consisted of $k-m$ or more failures (`num_failures_k_1` $\geqslant k-m$), less than m iterations are successful in the last k consecutive iterations and hence the (m,k) specification is violated. Alternatively, if `num_failures_k_1` $< k-m$ (defined using formula `failure_allowed`), even if the new iteration is not successful, the (m,k) specification is not violated.

A single step of the monitor corresponds to an execution of one iterations of the system. The command from Line 29 to Line 36 updates the global state at the end of each step. Note that `si'` denotes the updated state of variable `si`. Since the i^{th} latest iteration *before* the step corresponds to the $i+1^{st}$ latest iteration *after* the step, for each $2 \leqslant j = i+1 \leqslant k$, variable `sj` is updated to `si` (e.g., `s4'=s3`).

If the latest iteration is successful, which happens with probability `p`, variable `s1` is updated to `true`; otherwise, with probability `q`, variable `s1` is updated to `false`. In the latter case, the command also check if the (m,k) specification is violated, and updates the `safe` variable accordingly. Note that variable `safe` is mainly used to simply property specification (described below). The *reward* structure in the end of the listing (Line 40 to Line 42) associates a reward of one with each step, which is then used to compute the MTTF.

Once the model file is built, it can be queried with temporal logic queries, which must be specified in PRISM's property specification language. In particular, to compute the MTTF, we query the model with the reward-based property `R=? [ F safe=false ]`, which is essentially asking PRISM the following question: "what is the expected reward accumulated (denoted `R`) until (i.e., operator `F`) the safety property is violated (`safe=false`)?" To answer the question, PRISM's engine performs a reachability analysis over the model state space. Since we associate one reward per step, PRISM in this case returns the expected number of steps to failure, and the result can then be multiplied with the system's time period T to obtain the MTTF. Alternatively, one could simply associate a reward of T with each step.

C.1.2 Type-2 Monitor

Listing C.2 illustrates the optimized monitor representation of Type 2. In this case, only $k-m=5$ variables (`s1`-`s5`), instead of $k=10$ variables, are needed to capture the global state, i.e., the status of last k consecutive iterations. However, each variable `si` denotes the position of a failed iteration (among the last k consecutive iterations) and thus takes up to $k+1$ different values (0 is a sentinel value that indicates that variable `si` does not point to a failed iteration). In addition, the global state update (Line 20 to Line 27) ensures that each variable `si` points to a unique failed iteration (i.e., $\nexists i,j \,:\, 1 \leqslant i,j \leqslant k-m \wedge \texttt{si} =$

Listing C.2: Type-2 PRISM model for $(5, 10)$

```
dtmc

const int m = 5;
const int k = 10;
const double q = 1e-10;
formula p = 1.0 - q;

formula failure_allowed = ((s1<2)|(s2<2)|(s3<2)|(s4<2)|(s5<2));

module reliability_analysis

s1 : [0..k] init 0;
s2 : [0..k] init 0;
s3 : [0..k] init 0;
s4 : [0..k] init 0;
s5 : [0..k] init 0;

safe : bool init true;

[] true -> p: (safe'=safe)
              & (s1'=((s1>0)?(s1-1):0)) & (s2'=((s2>0)?(s2-1):0))
              & (s3'=((s3>0)?(s3-1):0)) & (s4'=((s4>0)?(s4-1):0))
              & (s5'=((s5>0)?(s5-1):0))
         + q: (safe'=safe & failure_allowed)
              & (s1'=k)                   & (s2'=((s1>0)?(s1-1):0))
              & (s3'=((s2>0)?(s2-1):0)) & (s4'=((s3>0)?(s3-1):0))
              & (s5'=((s4>0)?(s4-1):0));

endmodule

rewards "steps"
        true : 1;
endrewards
```

Listing C.3: Type-3 PRISM model for $(5, 10)$

```
dtmc

const int m = 5;
const int k = 10;
const double q = 1e-10;
formula p = 1.0 - q;

formula failure_allowed = ((s1>1)&(s2>1)&(s3>1)&(s4>1)&(s5>1));

module reliability_analysis

s1 : [0..k] init (k-0);
s2 : [0..k] init (k-1);
s3 : [0..k] init (k-2);
s4 : [0..k] init (k-3);
s5 : [0..k] init (k-4);

safe : bool init true;

[] true -> p: (safe'=safe)
              & (s1'=k)
              & (s2'=((s1>1)?(s1-1):0)) & (s3'=((s2>1)?(s2-1):0))
              & (s4'=((s3>1)?(s3-1):0)) & (s5'=((s4>1)?(s4-1):0))
         + q: (safe'=safe & failure_allowed)
              & (s1'=((s1>1)?(s1-1):0)) & (s2'=((s2>1)?(s2-1):0))
              & (s3'=((s3>1)?(s3-1):0)) & (s4'=((s4>1)?(s4-1):0))
              & (s5'=((s5>1)?(s5-1):0));

endmodule

rewards "steps"
        true : 1;
endrewards
```

`sj` $\neq 0$). In particular, the variable `si` always points to the i^{th} most recent failed iteration.

Since (m, k) robustness allows for up to $k - m$ failed iterations, a new failed iteration violates safety only if there are already $k - m$ failed iterations among the last $k - 1$ consecutive iterations (i.e., for each $1 \leqslant i \leqslant k - m$, variables `si` $\geqslant 2$). The status of the (k^{th}) oldest iteration does not matter after the new iteration is executed, since it does not affect the (m, k) robustness specification anymore. In other words, the system remains safe despite the new iteration being not successful if, for some $1 \leqslant i \leqslant k - m$, variable `si` < 2 (i.e., `failure_allowed`).

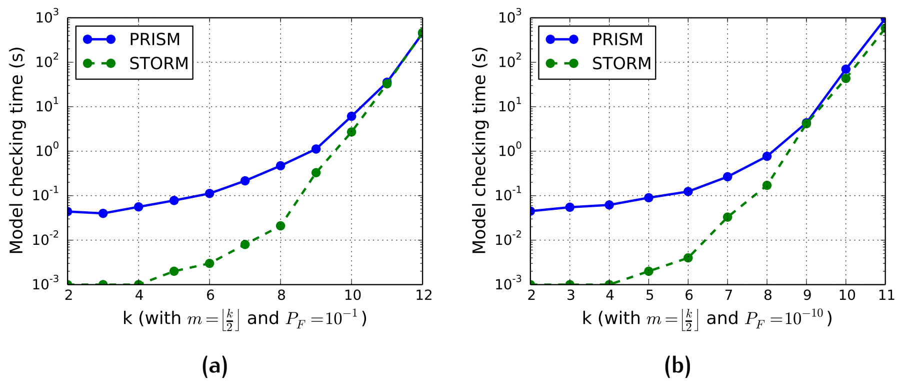


Figure C.1: Exact model checking in PRISM and STORM.

C.1.3 Type-3 Monitor

Listing C.3 illustrates the Type-3 monitor representation. It is similar to the Type-2 representation, except that state variables in this case track the successful iterations instead of the failed iterations (since Type 2 is optimized for $m \ll k$ and not $k - m \ll k$). In particular, the representation uses $m = 5$ variables (`s1`-`s5`), where each variable `si` denotes the position of the i^{th} most recent successful iteration. A failed iteration does not violate the (m, k) specification as long as there are at least m successful iterations among the last $k - 1$ consecutive iterations, i.e., for each $1 \leqslant i \leqslant m$, variable `si` > 1 (denoted as formula `failure_allowed` in the listing). Once again, the status of the oldest iteration does not matter after the new iteration is executed, since it does not affect (m, k) robustness anymore.

C.2 PRISM VERSUS STORM

While PRISM is a state-of-the-art probabilistic model checker and has been widely used for probabilistic analyses, STORM provides another alternative. In fact, Dehnert et al. [55] reported that for *exact* model checking (which is needed for numerical precision), STORM performs better than PRISM by up to three orders of magnitude. However, when we compared their performance in the context of our MTTF estimation problem, we observed that both the tools have similar performance for $k > 10$ and $k > 8$ when $P_F = 10^{-1}$ and $P_F = 10^{-10}$, respectively (see Fig. C.1). Hence, we favored PRISM over STORM owing to its better tool support.

BIBLIOGRAPHY

[1] *32-bit TriCore™ AURIX™ - TC2xx - Infineon Technologies*. URL: https://infineon.com/cms/en/product/microcontroller/32-bit-tricore-microcontroller/32-bit-tricore-aurix-tc2xx/.

[2] M. Abd-El-Malek, G. R. Ganger, G. R. Goodson, M. K. Reiter, and J. J. Wylie. "Fault-scalable Byzantine fault-tolerant services." In: *ACM SIGOPS Operating Systems Review* 39.5 (2005), p. 59. ISSN: 01635980. DOI: 10.1145/1095809.1095817. URL: http://portal.acm.org/citation.cfm?doid=1095809.1095817.

[3] *Adaptive Cruise Control*. URL: https://www.nxp.com/docs/en/product-selector-guide/SG2025.pdf.

[4] R. Alur and T. Henzinger. "Reactive modules." In: *11th IEEE Symposium on Logic in Computer Science (LICS 1996)*. New Brunswick, NJ, USA, pp. 207–218. ISBN: 978-0-8186-7463-1. DOI: 10.1109/LICS.1996.561320. URL: http://ieeexplore.ieee.org/document/561320/.

[5] H. Alzer. "On some inequalities for the incomplete gamma function." In: *Mathematics of Computation* 66.218 (1997). ISSN: 0025-5718. DOI: 10.1090/S0025-5718-97-00814-4. URL: http://www.ams.org/journal-getitem?pii=S0025-5718-97-00814-4.

[6] Y. Amir, B. Coan, J. Kirsch, and J. Lane. "Byzantine replication under attack." In: *IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN 2008)*, pp. 197–206. ISBN: 978-1-4244-2397-2. DOI: 10.1109/DSN.2008.4630088. URL: http://ieeexplore.ieee.org/document/4630088/.

[7] Y. Amir, B. Coan, J. Kirsch, and J. Lane. "Prime: Byzantine Replication under Attack." In: *IEEE Transactions on Dependable and Secure Computing* 8.4 (2011), pp. 564–577. ISSN: 1545-5971. DOI: 10.1109/TDSC.2010.70. URL: http://ieeexplore.ieee.org/document/5654509/.

[8] A. Anta and P. Tabuada. "On the Benefits of Relaxing the Periodicity Assumption for Networked Control Systems over CAN." In: *30th IEEE Real-Time Systems Symposium (RTSS 2009)*. Washington DC, USA, pp. 3–12. ISBN: 978-0-7695-3875-4. DOI: 10.1109/RTSS.2009.39. URL: http://ieeexplore.ieee.org/document/5369357/.

[9] *Apache Cassandra Documentation v4.0*. URL: https://cassandra.apache.org/doc/latest/.

[10] M. Appel, A. Gujarati, and B. B. Brandenburg. "A Byzantine Fault-Tolerant Key-Value Store for Safety-Critical Distributed Real-Time Systems." In: *2nd Workshop on the Security and Dependability of Critical Embedded Real-Time Systems (CERTS 2017)*. URL: https://certs2017.uni.lu/wp-content/uploads/sites/39/2017/11/certs_2017-proceedings.pdf.

[11] R. B. Ash. *Basic probability theory*. Mineola, N.Y: Dover Publications, 2008. ISBN: 978-0-486-46628-6.

[12] A. Atallah, G. Bany Hamad, and O. Ait Mohamed. "Reliability-Aware Routing of AVB Streams in TSN Networks." In: *Recent Trends and Future Technology in Applied Intelligence (2018)*. Vol. 10868. Springer International Publishing, pp. 697–708. ISBN: 978-3-319-92057-3 978-3-319-92058-0. DOI: 10.1007/978-3-319-92058-0_67. URL: http://link.springer.com/10.1007/978-3-319-92058-0_67.

[13] P.-L. Aublin, S. B. Mokhtar, and V. Quema. "RBFT: Redundant Byzantine Fault Tolerance." In: *IEEE 33rd International Conference on Distributed Computing Systems (ICDCS 2013)*, pp. 297–306. ISBN: 978-0-7695-5000-8. DOI: 10.1109/ICDCS.2013.53. URL: http://ieeexplore.ieee.org/document/6681599/.

[14] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings. "Hard Real-Time Scheduling: The Deadline-Monotonic Approach." In: *IFAC Proceedings Volumes* 24.2 (1991), pp. 127–132. ISSN: 1474-6670. DOI: 10.1016/S1474-6670(17)51283-5. URL: http://www.sciencedirect.com/science/article/pii/S1474667017512835.

[15] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. "Basic concepts and taxonomy of dependable and secure computing." In: *IEEE Transactions on Dependable and Secure Computing* 1.1 (2004), pp. 11–33. ISSN: 1545-5971. DOI: 10.1109/TDSC.2004.2. URL: http://ieeexplore.ieee.org/document/1335465/.

[16] *BLAS (Basic Linear Algebra Subprograms)*. URL: http://www.netlib.org/blas/.

[17] C. Baier and J.-P. Katoen. *Principles of model checking*. Cambridge, Mass: The MIT Press, 2008. ISBN: 978-0-262-02649-9.

[18] S. S. Banerjee, S. Jha, J. Cyriac, Z. T. Kalbarczyk, and R. K. Iyer. "Hands Off the Wheel in Autonomous Vehicles?: A Systems Perspective on over a Million Miles of Field Data." In: *48th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2018)*. Luxembourg City, pp. 586–597. ISBN: 978-1-5386-5596-2. DOI: 10.1109/DSN.2018.00066. URL: https://ieeexplore.ieee.org/document/8416518/.

[19] M. Barborak, A. Dahbura, and M. Malek. "The consensus problem in fault-tolerant computing." In: *ACM Computing Surveys* 25.2 (1993), pp. 171–220. ISSN: 03600300. DOI: 10.1145/152610.152612. URL: http://portal.acm.org/citation.cfm?doid=152610.152612.

[20] M. Ben-Or. "Another advantage of free choice (Extended Abstract): Completely asynchronous agreement protocols." In: *2nd ACM Symposium on Principles of Distributed Computing (PODC 1983)*. Montreal, Quebec, Canada, pp. 27–30. ISBN: 978-0-89791-110-8. DOI: 10.1145/800221.806707. URL: http://portal.acm.org/citation.cfm?doid=800221.806707.

[21] G. Bernat, A. Burns, and A. Liamosi. "Weakly hard real-time systems." In: *IEEE Transactions on Computers* 50.4 (2001), pp. 308–321. ISSN: 00189340. DOI: 10.1109/12.919277. URL: http://ieeexplore.ieee.org/document/919277/.

[22] G. Bernat and A. Burns. "Combining (/sub m//sup n/)-hard deadlines and dual priority scheduling." In: *18th IEEE Real-Time Systems Symposium (RTSS 1997)*. San Francisco, CA, USA, pp. 46–57. ISBN: 978-0-8186-8268-1. DOI: 10.1109/REAL.1997.641268. URL: http://ieeexplore.ieee.org/document/641268/.

[23] P. A. Bernstein. "Sequoia: a fault-tolerant tightly coupled multiprocessor for transaction processing." In: *Computer* 21.2 (1988), pp. 37–45. DOI: 10.1109/2.17.

[24] M. Bertogna and M. Cirinei. "Response-Time Analysis for Globally Scheduled Symmetric Multiprocessor Platforms." In: *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, pp. 149–160. DOI: 10.1109/RTSS.2007.31.

[25] A. Bessani, J. Sousa, and E. E. Alchieri. "State Machine Replication for the Masses with BFT-SMART." In: *44th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2014)*, pp. 355–362. ISBN: 978-1-4799-2233-8. DOI: 10.1109/DSN.2014.43. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6903593.

[26] A. Birolini. *Reliability engineering: theory and practice*. 8th edition. New York, NY: Springer Berlin Heidelberg, 2017. ISBN: 978-3-662-54208-8.

[27] R. Blind and F. Allgower. "Towards Networked Control Systems with guaranteed stability: Using weakly hard real-time constraints to model the loss process." In: *54th IEEE Conference on Decision and Control (CDC 2015)*. Osaka, pp. 7510–7515. ISBN: 978-1-4799-7886-1. DOI: 10.1109/CDC.2015.7403405. URL: http://ieeexplore.ieee.org/document/7403405/.

[28] G. Boole and J. Slater. *The mathematical analysis of logic: being an essay towards a calculus of deductive reasoning*. Repr. from the 1847 ed. Key texts. Bristol: Thoemmes, 1998. ISBN: 978-1-85506-583-3.

[29] F. Borran and A. Schiper. "A Leader-Free Byzantine Consensus Algorithm." In: *11th International Conference on Distributed Computing and Networking (ICDCN 2010)*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 67–78. ISBN: 978-3-642-11322-2. DOI: 10.1007/978-3-642-11322-2_11. URL: https://link.springer.com/chapter/10.1007%2F978-3-642-11322-2_11.

[30] R. C. Bose and D. K. Ray-Chaudhuri. "On a class of error correcting binary group codes." In: *Information and Control* 3.1 (1960), pp. 68–79. ISSN: 0019-9958. DOI: 10.1016/S0019-9958(60)90287-4. URL: http://www.sciencedirect.com/science/article/pii/S0019995860902874.

[31] B. B. Brandenburg. "Scheduling and Locking in Multiprocessor Real-time Operating Systems." PhD Thesis. Chapel Hill, NC, USA: University of North Carolina at Chapel Hill, 2011.

[32] B. B. Brandenburg and M. Gul. "Global Scheduling Not Required: Simple, Near-Optimal Multiprocessor Real-Time Scheduling with Semi-Partitioned Reservations." In: *IEEE Real-Time Systems Symposium (RTSS 2016)*. Porto, Portugal, pp. 99–110. ISBN: 978-1-5090-5303-2. DOI: 10.1109/RTSS.2016.019. URL: http://ieeexplore.ieee.org/document/7809847/.

[33] B. B. Brandenburg. *The Schedulability Test Collection And Toolkit*. URL: https://www.mpi-sws.org/~bbb/projects/schedcat.

[34] B. B. Brandenburg. *Liu and Layland and Linux: A Blueprint for "Proper" Real-Time Tasks*. 2020. URL: https://sigbed.org/2020/09/05/liu-and-layland-and-linux-a-blueprint-for-proper-real-time-tasks/.

[35] M. S. Branicky, S. M. Phillips, and Wei Zhang. "Scheduling and feedback co-design for networked control systems." In: *41st IEEE Conference on Decision and Control (CDC 2002)*. Vol. 2. Las Vegas, NV, USA, pp. 1211–1217. ISBN: 978-0-7803-7516-1. DOI: 10.1109/CDC.2002.1184679. URL: http://ieeexplore.ieee.org/document/1184679/.

[36] N. Braud-Santoni, R. Guerraoui, and F. Huc. "Fast byzantine agreement." In: *ACM Symposium on Principles of Distributed Computing (PODC 2013)*. Montr&#233;al, Qu&#233;bec, Canada, p. 57. ISBN: 978-1-4503-2065-8. DOI: 10.1145/2484239.2484243. URL: http://dl.acm.org/citation.cfm?doid=2484239.2484243.

[37] I. Broster and A. Burns. “Timely use of the CAN protocol in critical hard real-time systems with faults.” In: *13th Euromicro Conference on Real-Time Systems (ECRTS 2001)*. Delft, Netherlands, pp. 95–102. ISBN: 978-0-7695-1221-1. DOI: 10.1109/EMRTS.2001.934009. URL: http://ieeexplore.ieee.org/document/934009/.

[38] I. Broster, A. Burns, and G. Rodriguez-Navas. “Probabilistic analysis of CAN with faults.” In: *23rd IEEE Real-Time Systems Symposium (RTSS 2002)*. Austin, TX, USA, pp. 269–278. ISBN: 978-0-7695-1851-0. DOI: 10.1109/REAL.2002.1181581. URL: http://ieeexplore.ieee.org/document/1181581/.

[39] I. Broster, A. Burns, and G. Rodriguez-Navas. “Timing Analysis of Real-Time Communication Under Electromagnetic Interference.” In: *Real-Time Systems* 30.1-2 (2005), pp. 55–81. ISSN: 0922-6443, 1573-1383. DOI: 10.1007/s11241-005-0504-z. URL: http://link.springer.com/10.1007/s11241-005-0504-z.

[40] *CAN specification version 2.0*. 1991.

[41] M. Caccamo and G. Buttazzo. “Exploiting skips in periodic tasks for enhancing aperiodic responsiveness.” In: *18th IEEE Real-Time Systems Symposium (RTSS 1997)*. San Francisco, CA, USA, pp. 330–339. ISBN: 978-0-8186-8268-1. DOI: 10.1109/REAL.1997.641294. URL: http://ieeexplore.ieee.org/document/641294/.

[42] *Care-O-bot 4 User Manual*. 2018. URL: https://wiki.ros.org/Robots/cob4/manual.

[43] M. Castro and B. Liskov. “Practical Byzantine Fault Tolerance.” In: *3rd USENIX Symposium on Operating Systems Design and Implementation (OSDI 1999)*. event-place: New Orleans, Louisiana, USA. Berkeley, CA, USA, pp. 173–186. ISBN: 978-1-880446-39-3. URL: http://dl.acm.org/citation.cfm?id=296806.296824.

[44] J.-J. Chen et al. “Many suspensions, many problems: a review of self-suspending tasks in real-time systems.” In: *Real-Time Systems* (2018). ISSN: 1573-1383. DOI: 10.1007/s11241-018-9316-9. URL: https://doi.org/10.1007/s11241-018-9316-9.

[45] H. S. Chwa, K. G. Shin, and J. Lee. “Closing the Gap Between Stability and Schedulability: A New Task Model for Cyber-Physical Systems.” In: *24th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2018)*. Porto, pp. 327–337. ISBN: 978-1-5386-5295-4. DOI: 10.1109/RTAS.2018.00040. URL: https://ieeexplore.ieee.org/document/8430094/.

[46] A. Clement, E. L. Wong, L. Alvisi, M. Dahlin, and M. Marchetti. “Making Byzantine Fault Tolerant Systems Tolerate Byzantine Faults.” In: *6th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2009)*. Vol. 9. Boston, MA, USA,

pp. 153–168. URL: https://www.usenix.org/legacy/events/nsdi09/tech/full_papers/clement/clement.pdf.

[47] *Clemson Vehicular Electronics Laboratory: Airbag Deployment Systems*. URL: https://cecas.clemson.edu/cvel/auto/systems/airbag_deployment.html.

[48] M. Correia, N. F. Neves, and P. Veríssimo. "From Consensus to Atomic Broadcast: Time-Free Byzantine-Resistant Protocols without Signatures." In: *The Computer Journal* 49.1 (2006), pp. 82–96. ISSN: 1460-2067, 0010-4620. DOI: 10.1093/comjnl/bxh145. URL: http://academic.oup.com/comjnl/article/49/1/82/419030/From-Consensus-to-Atomic-Broadcast-TimeFree.

[49] M. Correia, G. S. Veronese, N. F. Neves, and P. Verissimo. "Byzantine consensus in asynchronous message-passing systems: a survey." In: *International Journal of Critical Computer-Based Systems* 2.2 (2011), p. 141. ISSN: 1757-8779, 1757-8787. DOI: 10.1504/IJCCBS.2011.041257. URL: http://www.inderscience.com/link.php?id=41257.

[50] G. F. Coulouris, J. Dollimore, T. Kindberg, and G. Blair. *Distributed systems: concepts and design*. 5th ed. Boston: Addison-Wesley, 2012. ISBN: 978-0-13-214301-1.

[51] J. Cowling, D. Myers, B. Liskov, R. Rodrigues, and L. Shrira. "HQ Replication: A Hybrid Quorum Protocol for Byzantine Fault Tolerance." In: *7th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2006)*. event-place: Seattle, Washington. Berkeley, CA, USA, pp. 177–190. ISBN: 978-1-931971-47-8. URL: http://dl.acm.org/citation.cfm?id=1298455.1298473.

[52] R. I. Davis and A. Burns. "Robust priority assignment for messages on Controller Area Network (CAN)." In: *Real-Time Systems* 41.2 (2009), pp. 152–180. ISSN: 1573-1383. DOI: 10.1007/s11241-008-9065-2. URL: https://doi.org/10.1007/s11241-008-9065-2.

[53] R. I. Davis and A. Burns. "A survey of hard real-time scheduling for multiprocessor systems." In: *ACM Computing Surveys* 43.4 (2011), pp. 1–44. ISSN: 03600300. DOI: 10.1145/1978802.1978814. URL: http://dl.acm.org/citation.cfm?doid=1978802.1978814.

[54] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien. "Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised." In: *Real-Time Systems* 35.3 (2007), pp. 239–272. ISSN: 0922-6443, 1573-1383. DOI: 10.1007/s11241-007-9012-7. URL: http://link.springer.com/10.1007/s11241-007-9012-7.

[55] C. Dehnert, S. Junges, J.-P. Katoen, and M. Volk. "A Storm is Coming: A Modern Probabilistic Model Checker." In: *29th International Conference on Computer Aided Verification (CAV 2017)*. Springer International Publishing, pp. 592–600. ISBN: 978-3-319-63390-9.

[56] T. J. Dell. *A White Paper on the Benefits of Chipkill-Correct ECC for PC Server Main Memory*. 1997.

[57] Y. Deswarte, K. Kanoun, and J.-C. Laprie. "Diversity against accidental and deliberate faults." In: *IEEE Conference on Computer Security, Dependability, and Assurance: From Needs to Solutions (Cat. No.98EX358) (CSDA 1998)*, pp. 171–181. ISBN: 978-0-7695-0337-0. DOI: 10.1109/CSDA.1998.798364. URL: http://ieeexplore.ieee.org/document/798364/.

[58] M. Di Natale, H. Zeng, P. Giusto, and A. Ghosal. *Understanding and Using the Controller Area Network Communication Protocol*. New York, NY: Springer New York, 2012. ISBN: 978-1-4614-0313-5 978-1-4614-0314-2. DOI: 10.1007/978-1-4614-0314-2. URL: http://link.springer.com/10.1007/978-1-4614-0314-2.

[59] J. Diemer, D. Thiele, and R. Ernst. "Formal worst-case timing analysis of Ethernet topologies with strict-priority and AVB switching." In: *7th IEEE International Symposium on Industrial Embedded Systems (SIES 2012)*. Karlsruhe, Germany, pp. 1–10. ISBN: 978-1-4673-2684-1 978-1-4673-2685-8 978-1-4673-2683-4. DOI: 10.1109/SIES.2012.6356564. URL: http://ieeexplore.ieee.org/document/6356564/.

[60] J. Diemer, J. Rox, and R. Ernst. "Modeling of Ethernet AVB Networks for Worst-Case Timing Analysis." In: *IFAC Proceedings Volumes* 45.2 (2012), pp. 848–853. ISSN: 14746670. DOI: 10.3182/20120215-3-AT-3016.00150. URL: https://linkinghub.elsevier.com/retrieve/pii/S1474667016307832.

[61] S. Distefano and Liudong Xing. "A new approach to modeling the system reliability: dynamic reliability block diagrams." In: *IEEE Reliability and Maintainability Symposium (RAMS 2006)*. Newport Beach, CA, USA, pp. 189–195. ISBN: 978-1-4244-0007-2. DOI: 10.1109/RAMS.2006.1677373. URL: http://ieeexplore.ieee.org/document/1677373/.

[62] T. Distler, C. Cachin, and R. Kapitza. "Resource-Efficient Byzantine Fault Tolerance." In: *IEEE Transactions on Computers* 65.9 (2016), pp. 2807–2819. ISSN: 0018-9340. DOI: 10.1109/TC.2015.2495213. URL: http://ieeexplore.ieee.org/document/7307998/.

[63] D. Dolev, C. Dwork, and L. Stockmeyer. "On the minimal synchronism needed for distributed consensus." In: *24th IEEE Symposium on Foundations of Computer Science (SFCS 1983)*. Tucson, AZ, USA, pp. 393–402. ISBN: 978-0-8186-0508-6. DOI: 10.

1109/SFCS.1983.41. URL: http://ieeexplore.ieee.org/document/4568103/.

[64] K. Driscoll, B. Hall, H. Sivencrona, and P. Zumsteg. "Byzantine Fault Tolerance, from Theory to Reality." In: *Computer Safety, Reliability, and Security*. Vol. 2788. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 235–248. ISBN: 978-3-540-20126-7 978-3-540-39878-3. DOI: 10.1007/978-3-540-39878-3_19. URL: http://link.springer.com/10.1007/978-3-540-39878-3_19.

[65] C. Drăgoi, T. A. Henzinger, H. Veith, J. Widder, and D. Zufferey. "A Logic-Based Framework for Verifying Consensus Algorithms." In: *Verification, Model Checking, and Abstract Interpretation*. Vol. 8318. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 161–181. ISBN: 978-3-642-54012-7 978-3-642-54013-4. DOI: 10.1007/978-3-642-54013-4_10. URL: http://link.springer.com/10.1007/978-3-642-54013-4_10.

[66] J. B. Dugan and R. Van Buren. "Reliability evaluation of fly-by-wire computer systems." In: *Journal of Systems and Software* 25.1 (1994), pp. 109–120. ISSN: 01641212. DOI: 10.1016/0164-1212(94)90061-2. URL: http://linkinghub.elsevier.com/retrieve/pii/0164121294900612.

[67] C. Dwork, N. Lynch, and L. Stockmeyer. "Consensus in the presence of partial synchrony." In: *Journal of the ACM* 35.2 (1988), pp. 288–323. ISSN: 00045411. DOI: 10.1145/42282.42283. URL: http://portal.acm.org/citation.cfm?doid=42282.42283.

[68] J. C. Eidson. *Measurement, control, and communication using IEEE 1588*. Advances in industrial control. London: Springer, 2006. ISBN: 978-1-84628-250-8 978-1-84628-251-5.

[69] *Elemental: distributed-memory dense and sparse-direct linear algebra and optimization — Elemental*. URL: http://libelemental.org/.

[70] *Eyeriss Project*. URL: http://eyeriss.mit.edu/.

[71] D. Faggioli, F. Checconi, M. Trimarchi, and C. Scordino. "An EDF scheduling class for the Linux kernel." In: *11th Real-Time Linux Workshop (RTLWS 2009)*, p. 8.

[72] J. Ferreira, A. Oliveira, P. Fonseca, and J. Fonseca. "An Experiment to Assess Bit Error Rate in CAN." In: *3rd International Workshop of Real-Time Networks (RTN 2004)*, pp. 15–18.

[73] M. J. Fischer and N. A. Lynch. "A lower bound for the time to assure interactive consistency." In: *Information Processing Letters* 14.4 (1982), pp. 183–186. ISSN: 00200190. DOI: 10.1016/0020-0190(82)90033-3. URL: https://linkinghub.elsevier.com/retrieve/pii/0020019082900333.

[74] M. J. Fischer, N. A. Lynch, and M. S. Paterson. "Impossibility of distributed consensus with one faulty process." In: *Journal of the ACM* 32.2 (1985), pp. 374–382. ISSN: 00045411. DOI: 10.1145/3149.214121. URL: http://portal.acm.org/citation.cfm?doid=3149.214121.

[75] D. Fontanelli, D. Macii, P. Wolfrum, D. Obradovic, and G. Steindl. "A clock state estimator for PTP time synchronization in harsh environmental conditions." In: *IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication (ISPCS 2011)*. Munich, Germany, pp. 99–104. ISBN: 978-1-61284-893-8. DOI: 10.1109/ISPCS.2011.6070142. URL: http://ieeexplore.ieee.org/document/6070142/.

[76] E. Foxlin. "Inertial head-tracker sensor fusion by a complementary separate-bias Kalman filter." In: *IEEE Virtual Reality Annual International Symposium (VRAIS 1996)*. Santa Clara, CA, USA, pp. 185–194. ISBN: 978-0-8186-7296-5. DOI: 10.1109/VRAIS.1996.490527. URL: http://ieeexplore.ieee.org/document/490527/.

[77] L. Franks. "Carrier and Bit Synchronization in Data Communication - A Tutorial Review." In: *IEEE Transactions on Communications* 28.8 (1980), pp. 1107–1121. DOI: 10.1109/TCOM.1980.1094775.

[78] R. Friedman and R. Licher. "Hardening Cassandra Against Byzantine Failures." In: Leibniz International Proceedings in Informatics (LIPIcs) 95 (). Ed. by J. Aspnes, A. Bessani, P. Felber, and J. Leitão, 27:1–27:20. ISSN: 1868-8969. DOI: 10.4230/LIPIcs.OPODIS.2017.27. URL: http://drops.dagstuhl.de/opus/volltexte/2018/8642.

[79] H. C. Gabler and J. Hinch. "Evaluation of Advanced Air Bag Deployment Algorithm Performance using Event Data Recorders." In: *Annals of Advances in Automotive Medicine / Annual Scientific Conference* 52 (2008), pp. 175–184. ISSN: 1943-2461. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3256779/.

[80] G. M. Garner, F. Feng, K. D. Hollander, H. Jeong, B. Kim, Byoung-Joon Lee, Tae-Chul Jung, and J. Joung. "IEEE 802.1 AVB and Its Application in Carrier-Grade Ethernet [Standards Topics]." In: *IEEE Communications Magazine* 45.12 (2007), pp. 126–134. DOI: 10.1109/MCOM.2007.4395377.

[81] M. Gaukler, A. Michalka, P. Ulbrich, and T. Klaus. "A New Perspective on Quality Evaluation for Control Systems with Stochastic Timing." In: *21st ACM International Conference on Hybrid Systems: Computation and Control (HSCC 2018)*. Porto, Portugal, pp. 91–100. ISBN: 978-1-4503-5642-8. DOI: 10.1145/3178126.3178134. URL: http://dl.acm.org/citation.cfm?doid=3178126.3178134.

[82] W. Gautschi. "Some Elementary Inequalities Relating to the Gamma and Incomplete Gamma Function." In: *Journal of Mathematics and Physics* 38.1-4 (1959), pp. 77–81. ISSN: 00971421. DOI: 10.1002/sapm195938177. URL: http://doi.wiley.com/10.1002/sapm195938177.

[83] L. George, N. Rivierre, and M. Spuri. *Preemptive and Non-Preemptive Real-Time Uniprocessor Scheduling*. Tech. rep. 1996. URL: https://hal.inria.fr/inria-00073732.

[84] M. Gergeleit and H. Streich. "Implementing a Distributed High-Resolution Real-Time Clock using the CAN-Bus." In: *1st International CAN Conference (iCC 1994)*. Mainz, Germany, p. 7.

[85] A. E. P. Goodloe. *Monitoring Distributed Real-Time Systems: A Survey and Future Directions*. Tech. rep. 2010. URL: https://ntrs.nasa.gov/search.jsp?R=20100027427.

[86] M. Gottscho, C. Schoeny, L. Dolecek, and P. Gupta. "Software-Defined Error-Correcting Codes." In: *46th IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W 2016)*. Toulouse, France, pp. 276–282. ISBN: 978-1-5090-3688-2. DOI: 10.1109/DSN-W.2016.67. URL: http://ieeexplore.ieee.org/document/7575399/.

[87] R. Guerraoui and A. Schiper. "Fault-tolerance by replication in distributed systems." In: *Reliable Software Technologies — Ada-Europe '96*. Vol. 1088. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 38–57. ISBN: 978-3-540-61317-6 978-3-540-68457-2. DOI: 10.1007/BFb0013477. URL: http://link.springer.com/10.1007/BFb0013477.

[88] A. Gujarati and B. B. Brandenburg. "When Is CAN the Weakest Link? A Bound on Failures-in-Time in CAN-Based Real-Time Systems." In: *36th IEEE Real-Time Systems Symposium (RTSS 2015)*. San Antonio, Texas, pp. 249–260. ISBN: 978-1-4673-9507-6. DOI: 10.1109/RTSS.2015.31. URL: http://ieeexplore.ieee.org/document/7383582/.

[89] A. Gujarati, M. Nasri, R. Majumdar, and B. B. Brandenburg. "From Iteration to System Failure: Characterizing the FITness of Periodic Weakly-Hard Systems." In: *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*. Vol. 133. Leibniz International Proceedings in Informatics (LIPIcs). Stuttgart, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 9:1–9:23. ISBN: 978-3-95977-110-8. DOI: 10.4230/lipics.ecrts.2019.9. URL: http://drops.dagstuhl.de/opus/volltexte/2019/10746/.

[90] A. Gujarati, M. Nasri, and B. B. Brandenburg. "Lower-Bounding the MTTF for Systems with (m,k) Constraints and IID Iteration Failure Probabilities." In: *2nd Workshop on the Security*

and Dependability of Critical Embedded Real-Time Systems (CERTS 2017). URL: https://certs2017.uni.lu/wp-content/uploads/sites/39/2017/11/certs_2017-proceedings.pdf.

[91] A. Gujarati, M. Nasri, and B. B. Brandenburg. "Quantifying the Resiliency of Fail-Operational Real-Time Networked Control Systems." In: *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*. Vol. 106. Leibniz International Proceedings in Informatics (LIPIcs). Barcelona, Spain: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 16:1–16:24. ISBN: 978-3-95977-075-0. DOI: 10.4230/lipics.ecrts.2018.16. URL: http://drops.dagstuhl.de/opus/volltexte/2018/8988/.

[92] A. Gujarati, S. Bozhko, and B. B. Brandenburg. "Real-Time Replica Consistency over Ethernet with Reliability Bounds." In: *26th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2020)*. Sydney, Australia, pp. 376–389. ISBN: 978-1-72815-499-2. DOI: 10.1109/RTAS48715.2020.00012. URL: https://ieeexplore.ieee.org/document/9113102/.

[93] A. Gujarati, M. Appel, and B. B. Brandenburg. "Achal: Building Highly Reliable Networked Control Systems." In: *15th ACM SIGBED International Conference on Embedded Software Companion (EMSOFT 2019)*. New York, New York: ACM Press, 2019, pp. 1–2. ISBN: 978-1-4503-6924-4. DOI: 10.1145/3349568.3351545. URL: http://dl.acm.org/citation.cfm?doid=3349568.3351545.

[94] Hagbae Kim and K. Shin. "Modeling of externally-induced/common-cause faults in fault-tolerant systems." In: *13th AIAA/IEEE Digital Avionics Systems Conference (DASC 1994)*. Phoenix, AZ, USA, pp. 402–407. ISBN: 978-0-7803-2425-1. DOI: 10.1109/DASC.1994.369450. URL: http://ieeexplore.ieee.org/document/369450/.

[95] M. Hamdaoui and P. Ramanathan. "A dynamic priority assignment technique for streams with (m, k)-firm deadlines." In: *IEEE Transactions on Computers* 44.12 (1995), pp. 1443–1451. ISSN: 00189340. DOI: 10.1109/12.477249. URL: http://ieeexplore.ieee.org/document/477249/.

[96] K. Hashimoto, T. Tsuchiya, and T. Kikuno. "Effective Scheduling of Duplicated Tasks for Fault Tolerance in Multiprocessor Systems." In: *IEICE TRANSACTIONS on Information and Systems* E85-D.3 (2002), pp. 525–534. ISSN: , 0916-8532. URL: http://search.ieice.org/bin/summary.php?id=e85-d_3_525&category=D&year=2002&lang=E&abst=.

[97] P. Hazucha and C. Svensson. "Impact of CMOS technology scaling on the atmospheric neutron soft error rate." In: *IEEE Transactions on Nuclear Science* 47.6 (2000), pp. 2586–2594. ISSN: 00189499. DOI: 10.1109/23.903813. URL: http://ieeexplore.ieee.org/document/903813/.

[98] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst. "System level performance analysis – the SymTA/S approach." In: *IEE Proceedings - Computers and Digital Techniques* 152.2 (2005), p. 148. ISSN: 13502387. DOI: 10.1049/ip-cdt:20045088. URL: https://digital-library.theiet.org/content/journals/10.1049/ip-cdt_20045088.

[99] T. Henzinger, B. Horowitz, and C. Kirsch. "Giotto: a time-triggered language for embedded programming." In: *Proceedings of the IEEE* 91.1 (2003), pp. 84–99. ISSN: 0018-9219. DOI: 10.1109/JPROC.2002.805825. URL: http://ieeexplore.ieee.org/document/1173196/.

[100] A. Hopkins, T. Smith, and J. Lala. "FTMP—A highly reliable fault-tolerant multiprocess for aircraft." In: *Proceedings of the IEEE* 66.10 (1978), pp. 1221–1239. ISSN: 0018-9219. DOI: 10.1109/PROC.1978.11113. URL: http://ieeexplore.ieee.org/document/1455382/.

[101] A. L. Hopkins. "A New Standard for Information Processing Systems for Manned Space Flight." In: *IFAC Proceedings Volumes* 3.1 (1970), pp. 223–229. ISSN: 14746670. DOI: 10.1016/S1474-6670(17)68779-2. URL: http://linkinghub.elsevier.com/retrieve/pii/S1474667017687792.

[102] K. Hoyme and K. Driscoll. "SAFEbus (for avionics)." In: *IEEE Aerospace and Electronic Systems Magazine* 8.3 (1993), pp. 34–39. ISSN: 0885-8985. DOI: 10.1109/62.199819. URL: http://ieeexplore.ieee.org/document/199819/.

[103] M. Y. Hsiao. "A Class of Optimal Minimum Odd-weight-column SEC-DED Codes." In: *IBM Journal of Research and Development* 14.4 (1970), pp. 395–401. DOI: 10.1147/rd.144.0395.

[104] *IEEE 802.1: 802.1Qat - Stream Reservation Protocol*. URL: http://www.ieee802.org/1/pages/802.1at.html.

[105] *IEEE 802.1: 802.1Qav - Forwarding and Queuing Enhancements for Time-Sensitive Streams*. 2018. URL: http://www.ieee802.org/1/pages/802.1av.html.

[106] *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*. Tech. rep. IEEE. DOI: 10.1109/IEEESTD.2008.4579760. URL: http://ieeexplore.ieee.org/document/4579760/.

[107] *ISO 11519-3:1994*. URL: http://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/01/94/19471.html.

[108] *Industrial communication networks – Fieldbus specifications – Part 6-2: Application layer protocol specification – Type 2 elements*. Tech. rep. IEC 61158-6-2:2019. International Electrotechnical Commission, 2019. URL: https://webstore.iec.ch/publication/65168.

[109] R. Isermann, R. Schwarz, and S. Stölzl. "Fault-Tolerant Drive-by-Wire Systems – Concepts and Realizations –." In: *IFAC Proceedings Volumes* 33.11 (2000), pp. 1–15. ISSN: 14746670. DOI: 10.1016/S1474-6670(17)37335-4. URL: https://linkinghub.elsevier.com/retrieve/pii/S1474667017373354.

[110] A. Izycheva and E. Darulova. "On Sound Relative Error Bounds for Floating-point Arithmetic." In: *17th Conference on Formal Methods in Computer-Aided Design (FMCAD 2017)*. Austin, TX, pp. 15–22. ISBN: 978-0-9835678-7-5. URL: http://dl.acm.org/citation.cfm?id=3168451.3168462.

[111] J. Kaiser and M. A. Livani. "Achieving Fault-Tolerant Ordered Broadcasts in CAN." In: *3rd European Dependable Computing Conference (EDCC 1999)*. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 351–363. ISBN: 978-3-540-48254-3. DOI: 10.1007/3-540-48254-7_24.

[112] KapDae Ahn, Jong Kim, and SungJe Hong. "Fault-tolerant real-time scheduling using passive replicas." In: *IEEE Pacific Rim International Symposium on Fault-Tolerant Systems (PRFTS 1997)*, pp. 98–103. DOI: 10.1109/PRFTS.1997.640132.

[113] T. Karnik and P. Hazucha. "Characterization of soft errors caused by single event upsets in CMOS processes." In: *IEEE Transactions on Dependable and Secure Computing* 1.2 (2004), pp. 128–143. ISSN: 1545-5971. DOI: 10.1109/TDSC.2004.14. URL: http://ieeexplore.ieee.org/document/1350778/.

[114] M. Kauer, D. Soudbakhsh, D. Goswami, S. Chakraborty, and A. M. Annaswamy. "Fault-tolerant Control Synthesis and Verification of Distributed Embedded Systems." In: *EDAA Conference on Design, Automation & Test in Europe (DATE 2014)*. 3001 Leuven, Belgium, Belgium, 56:1–56:6. ISBN: 978-3-9815370-2-4. URL: http://dl.acm.org/citation.cfm?id=2616606.2616675.

[115] R. Keichafer, C. Walter, A. Finn, and P. Thambidurai. "The MAFT architecture for distributed fault tolerance." In: *IEEE Transactions on Computers* 37.4 (1988), pp. 398–404. ISSN: 00189340. DOI: 10.1109/12.2183. URL: http://ieeexplore.ieee.org/document/2183/.

[116] K. Kihlstrom, L. Moser, and P. Melliar-Smith. "The SecureRing protocols for securing group communication." In: *31st IEEE Hawaii International Conference on System Sciences (HICSS 1998)*. Vol. 3. Kohala Coast, HI, USA, pp. 317–326. ISBN: 978-0-8186-

8255-1. DOI: 10.1109/HICSS.1998.656294. URL: http://ieeexplore.ieee.org/document/656294/.

[117] C. M. Kirsch and A. Sokolova. "The Logical Execution Time Paradigm." In: *Advances in Real-Time Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 103–120. ISBN: 978-3-642-24348-6 978-3-642-24349-3. DOI: 10.1007/978-3-642-24349-3_5. URL: http://link.springer.com/10.1007/978-3-642-24349-3_5.

[118] P. Koopman. "32-bit cyclic redundancy codes for Internet applications." In: *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2002)*. Washington, DC, USA, pp. 459–468. ISBN: 978-0-7695-1597-7. DOI: 10.1109/DSN.2002.1028931. URL: http://ieeexplore.ieee.org/document/1028931/.

[119] H. Kopetz, H. Kantz, G. Grunsteidl, P. Puschner, and J. Reisinger. "Tolerating transient faults in MARS." In: *20th International Symposium on Fault-Tolerant Computing (FTCS 1990)*, pp. 466–473. DOI: 10.1109/FTCS.1990.89384.

[120] H. Kopetz, A. Damm, C. Koza, M. Mulazzani, W. Schwabl, C. Senft, and R. Zainlinger. "Distributed fault-tolerant real-time systems: the Mars approach." In: *IEEE Micro* 9.1 (1989), pp. 25–40. ISSN: 0272-1732. DOI: 10.1109/40.16792. URL: http://ieeexplore.ieee.org/document/16792/.

[121] H. Kopetz. *Real-Time Systems*. Real-Time Systems Series. Boston, MA: Springer US, 2011. ISBN: 978-1-4419-8236-0 978-1-4419-8237-7. DOI: 10.1007/978-1-4419-8237-7. URL: http://link.springer.com/10.1007/978-1-4419-8237-7.

[122] H. Kopetz, G. Bauer, and S. Poledna. "Tolerating Arbitrary Node Failures in the Time-Triggered Architecture." In: *SAE 2001 World Congress*. DOI: 10.4271/2001-01-0677. URL: https://www.sae.org/content/2001-01-0677/.

[123] R. Kotla, A. Clement, E. Wong, L. Alvisi, and M. Dahlin. "Zyzzyva: speculative Byzantine fault tolerance." In: *Communications of the ACM* 51.11 (2008), p. 86. ISSN: 00010782. DOI: 10.1145/1400214.1400236. URL: http://portal.acm.org/citation.cfm?doid=1400214.1400236.

[124] W. Kuo and M. J. Zuo. *Optimal reliability modeling: principles and applications*. Hoboken, N.J: John Wiley & Sons, 2003. ISBN: 978-0-471-39761-8.

[125] M. Kwiatkowska, G. Norman, and D. Parker. "PRISM 4.0: Verification of Probabilistic Real-Time Systems." In: *23rd International Conference on Computer Aided Verification (CAV 2011)*. Vol. 6806. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 585–591. ISBN: 978-3-642-22109-5 978-3-642-22110-1. DOI: 10.1007/

978-3-642-22110-1_47. URL: http://link.springer.com/10.1007/978-3-642-22110-1_47.

[126] M. Kwiatkowska, G. Norman, and D. Parker. "Controller dependability analysis by probabilistic model checking." In: *Control Engineering Practice* 15.11 (2007), pp. 1427–1434. ISSN: 09670661. DOI: 10.1016/j.conengprac.2006.07.003. URL: https://linkinghub.elsevier.com/retrieve/pii/S0967066106001262.

[127] *LAPACK – Linear Algebra PACKage*. URL: http://www.netlib.org/lapack/.

[128] J. Lala, R. Harper, K. Jaskowiak, G. Rosch, L. Alger, and A. Schor. "Advanced Information Processing System (AIPS)-based fault tolerant avionics architecture for launch vehicles." In: *9th IEEE/AIAA/NASA Conference on Digital Avionics Systems (DASC 1990)*, pp. 125–132. DOI: 10.1109/DASC.1990.111274. URL: http://ieeexplore.ieee.org/document/111274/.

[129] L. Lamport, R. Shostak, and M. Pease. "The Byzantine Generals Problem." In: *ACM Transactions on Programming Languages and Systems* 4.3 (1982), pp. 382–401. ISSN: 01640925. DOI: 10.1145/357172.357176. URL: http://portal.acm.org/citation.cfm?doid=357172.357176.

[130] D. Lavo, T. Larrabee, and B. Chess. "Beyond the byzantine generals: unexpected behavior and bridging fault diagnosis." In: *International Test Conference (TEST 1996)*. Washington, DC, USA, pp. 611–619. ISBN: 978-0-7803-3541-7. DOI: 10.1109/TEST.1996.557118. URL: http://ieeexplore.ieee.org/document/557118/.

[131] W. Lawrenz, ed. *CAN System Engineering*. London: Springer London, 2013. ISBN: 978-1-4471-5612-3 978-1-4471-5613-0. DOI: 10.1007/978-1-4471-5613-0. URL: http://link.springer.com/10.1007/978-1-4471-5613-0.

[132] H. Li. "Robust Control Design for Vehicle Active Suspension Systems with Uncertainty." PhD thesis. Portsmouth: University of Portsmouth, 2012. URL: https://core.ac.uk/download/pdf/40012843.pdf.

[133] S.-Y. R. Li. "A Martingale Approach to the Study of Occurrence of Sequence Patterns in Repeated Experiments." In: *The Annals of Probability* 8.6 (1980), pp. 1171–1176. ISSN: 0091-1798. URL: https://www.jstor.org/stable/2243018.

[134] X. Li, S. V. Adve, P. Bose, and J. A. Rivers. "Architecture-Level Soft Error Analysis: Examining the Limits of Common Assumptions." In: *37th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2007)*. Edinburgh, UK, pp. 266–275. ISBN: 978-0-7695-2855-7. DOI: 10.1109/DSN.2007.15. URL: http://ieeexplore.ieee.org/document/4272978/.

[135] S. Lin and D. J. Costello. *Error control coding: fundamentals and applications*. 2nd ed. Upper Saddle River, N.J: Pearson-Prentice Hall, 2004. ISBN: 978-0-13-042672-7 978-0-13-017973-9.

[136] B. Littlewood and L. Strigini. "Redundancy and Diversity in Security." In: *9th European Symposium on Research in Computer Security (ESORICS 2004)*. Vol. 3193. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 423–438. ISBN: 978-3-540-22987-2 978-3-540-30108-0. DOI: 10.1007/978-3-540-30108-0_26. URL: http://link.springer.com/10.1007/978-3-540-30108-0_26.

[137] C. L. Liu and J. W. Layland. "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment." In: *Journal of the ACM* 20.1 (1973), pp. 46–61. ISSN: 00045411. DOI: 10.1145/321738.321743. URL: http://portal.acm.org/citation.cfm?doid=321738.321743.

[138] Lui Sha, R. Rajkumar, and S. S. Sathaye. "Generalized rate-monotonic scheduling theory: a framework for developing real-time systems." In: *Proceedings of the IEEE* 82.1 (1994), pp. 68–82. DOI: 10.1109/5.259427.

[139] M. Lüdtke. "The service robot Care-O-bot 4." In: *CAN Newsletter* 1 (2016), pp. 36–39. URL: https://can-newsletter.org/uploads/media/raw/fa0136c44242f5e4befa4594b97bf8cb.pdf.

[140] R. Maier, G. Bauer, G. Stoger, and S. Poledna. "Time-triggered architecture: a consistent computing platform." In: *IEEE Micro* 22.4 (2002), pp. 36–45. ISSN: 0272-1732. DOI: 10.1109/MM.2002.1028474. URL: http://ieeexplore.ieee.org/document/1028474/.

[141] R. Majumdar, I. Saha, and M. Zamani. "Performance-aware scheduler synthesis for control systems." In: *9th ACM International Conference on Embedded Software (EMSOFT 2011)*. Taipei, Taiwan, p. 299. ISBN: 978-1-4503-0714-7. DOI: 10.1145/2038642.2038689. URL: http://dl.acm.org/citation.cfm?doid=2038642.2038689.

[142] R. Majumdar, I. Saha, and M. Zamani. "Synthesis of Minimal-error Control Software." In: *10th ACM International Conference on Embedded Software (EMSOFT 2012)*. EMSOFT '12. New York, NY, USA, pp. 123–132. ISBN: 978-1-4503-1425-1. DOI: 10.1145/2380356.2380380. URL: http://doi.acm.org/10.1145/2380356.2380380.

[143] F. Makri and Z. Psillakis. "Bounds for reliability of k-within two-dimensional consecutive-r-out-of-n failure systems." In: *Microelectronics Reliability* 36.3 (1996), pp. 341–345. ISSN: 00262714. DOI: 10.1016/0026-2714(95)00102-6. URL: http://linkinghub.elsevier.com/retrieve/pii/0026271495001026.

[144] J. Malinowski and W. Preuss. "A recursive algorithm evaluating the exact reliability of a consecutive k-within-m-out-of-n:F system." In: *Microelectronics Reliability* 35.12 (1995), pp. 1461–1465. ISSN: 00262714. DOI: 10.1016/0026-2714(95)91271-V. URL: http://linkinghub.elsevier.com/retrieve/pii/002627149591271V.

[145] J. Malinowski and W. Preuss. "A recursive algorithm evaluating the exact reliability of a circular consecutive k-within-m-out-of-n:F system." In: *Microelectronics Reliability* 36.10 (1996), pp. 1389–1394. ISSN: 00262714. DOI: 10.1016/0026-2714(96)00015-7. URL: http://linkinghub.elsevier.com/retrieve/pii/0026271496000157.

[146] D. Malkhi and M. Reiter. "Byzantine quorum systems." In: *Distributed Computing* 11.4 (1998), pp. 203–213. ISSN: 01782770. DOI: 10.1007/s004460050050. URL: http://link.springer.com/10.1007/s004460050050.

[147] R. Mancuso. "Next-generation safety-critical systems on multi-core platforms." PhD thesis. University of Illinois at Urbana-Champaign, 2017. URL: http://hdl.handle.net/2142/97399.

[148] K. Matheus and T. Königseder. *Automotive Ethernet*. Cambridge: Cambridge University Press, 2015. ISBN: 978-1-107-05728-9.

[149] F. Mathur. "On Reliability Modeling and Analysis of Ultrareliable Fault-Tolerant Digital Systems." In: *IEEE Transactions on Computers* C-20.11 (1971), pp. 1376–1382. ISSN: 0018-9340. DOI: 10.1109/T-C.1971.223142. URL: http://ieeexplore.ieee.org/document/1671735/.

[150] P. McKenney. *A realtime preemption overview [LWN.net]*. 2005. URL: https://old.lwn.net/Articles/146861/.

[151] P. Miner, M. Malekpour, and W. Torres. "A conceptual design for a Reliable Optical Bus (ROBUS)." In: *21st IEEE Digital Avionics Systems Conference (DASC 2002)*. Vol. 2. Irvine, CA, USA, pp. 13D3–1–13D3–11. ISBN: 978-0-7803-7367-9. DOI: 10.1109/DASC.2002.1053014. URL: http://ieeexplore.ieee.org/document/1053014/.

[152] M. Modarres, M. Kaminskiy, and V. Krivtsov. *Reliability engineering and risk analysis*. Quality and reliability 55. New York: Marcel Dekker, 1999. ISBN: 978-0-8247-2000-1.

[153] A. K. Mok. *Fundamental Design Problens of Distributed Systems for the Hard-Real-Ttime Environment*. Tech. rep. Cambridge, MA, USA: Massachusetts Institute of Technology, 1983.

[154] S. Mukherjee, C. Weaver, J. Emer, S. Reinhardt, and T. Austin. "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor." In: *36th IEEE/ACM International Symposium on Microarchitecture (MICRO 2003)*. San Diego, CA, USA, pp. 29–40. ISBN: 978-0-7695-2043-8. DOI: 10.1109/MICRO.2003.1253181. URL: http://ieeexplore.ieee.org/document/1253181/.

[155] N. Murphy. "Watchdog Timers." In: *Embedded Systems Programming* (2000), pp. 112–124. URL: https://www.embedded.com/design/debug-and-optimization/4402288/Watchdog-Timers.

[156] N. Murphy and M. Barr. "Watchdog Timers." In: *Embedded Systems Programming* (2001), pp. 79–80. URL: https://www.embedded.com/electronics-blogs/beginner-s-corner/4023849/Introduction-to-Watchdog-Timers.

[157] NASA Technical Reports Server (NTRS). *NASA Technical Reports Server (NTRS) 19880011510: A survey of provably correct fault-tolerant clock synchronization techniques*. 1988. URL: http://archive.org/details/NASA_NTRS_Archive_19880011510.

[158] NEC. *NEC V60 CPU Manual*. 1986. URL: http://archive.org/details/NEC_V60pgmRef.

[159] M. Nahas, M. Short, and M. J. Pont. "The impact of bit stuffing on the real-time performance of a distributed control system." In: *10th International CAN Conference (iCC 2005)*.

[160] N. Nakka, G. P. Saggese, Z. Kalbarczyk, and R. K. Iyer. "An Architectural Framework for Detecting Process Hangs/Crashes." In: *5th European Dependable Computing Conference (EDCC 2005)*. Vol. 3463. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 103–121. ISBN: 978-3-540-25723-3 978-3-540-32019-7. DOI: 10.1007/11408901_8. URL: http://link.springer.com/10.1007/11408901_8.

[161] A. Nasrallah, A. S. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. ElBakoury. "Ultra-Low Latency (ULL) Networks: The IEEE TSN and IETF DetNet Standards and Related 5G ULL Research." In: *IEEE Communications Surveys & Tutorials* 21.1 (2019), pp. 88–145. ISSN: 1553-877X, 2373-745X. DOI: 10.1109/COMST.2018.2869350. URL: https://ieeexplore.ieee.org/document/8458130/.

[162] J. I. Naus. "The Teacher's Corner: An Extension of the Birthday Problem." In: *The American Statistician* 22.1 (1968), pp. 27–29. ISSN: 0003-1305, 1537-2731. DOI: 10.1080/00031305.1968.10480438. URL: http://www.tandfonline.com/doi/abs/10.1080/00031305.1968.10480438.

[163] N. Navet, Y.-Q. Song, and F. Simonot. "Worst-case deadline failure probability in real-time applications distributed over controller area network." In: *Journal of Systems Architecture* 46.7 (2000), pp. 607–617. ISSN: 13837621. DOI: 10.1016/S1383-7621(99)00016-8. URL: http://linkinghub.elsevier.com/retrieve/pii/S1383762199000168.

[164] E. Neuman. "Inequalities and Bounds for the Incomplete Gamma Function." In: *Results in Mathematics* 63.3-4 (2013), pp. 1209–1214. ISSN: 1422-6383, 1420-9012. DOI: 10.1007/s00025-012-0263-9. URL: http://link.springer.com/10.1007/s00025-012-0263-9.

[165] R. R. Obelheiro, A. N. Bessani, L. C. Lung, and M. Correia. *How Practical Are Intrusion-Tolerant Distributed Systems?* Technical Report DI-FCUL-TR-06-15. Department of Informatics, University of Lisbon, 2006. URL: http://hdl.handle.net/10451/14093.

[166] D. Ongaro and J. Ousterhout. "In Search of an Understandable Consensus Algorithm." In: *USENIX Annual Technical Conference (ATC 2014)*, pp. 305–319. ISBN: 978-1-931971-10-2. URL: https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro.

[167] *Open MPI: Open Source High Performance Computing*. URL: https://www.open-mpi.org/.

[168] S. Papastavridis and M. Koutras. "Bounds for reliability of consecutive k-within-m-out-of-n:F systems." In: *IEEE Transactions on Reliability* 42.1 (1993), pp. 156–160. ISSN: 00189529. DOI: 10.1109/24.210288. URL: http://ieeexplore.ieee.org/document/210288/.

[169] S. G. Papastavridis and M. Sfakianakis. "Optimal-arrangement and importance of the components in a consecutive-k-out-of-r-from-n:F system." In: *IEEE Transactions on Reliability* 40.3 (1991), pp. 277–279. ISSN: 00189529. DOI: 10.1109/24.85439. URL: http://ieeexplore.ieee.org/document/85439/.

[170] A. Patra, A. Choudhury, and C. P. Rangan. "Asynchronous Byzantine Agreement with optimal resilience." In: *Distributed Computing* 27.2 (2014), pp. 111–146. ISSN: 0178-2770, 1432-0452. DOI: 10.1007/s00446-013-0200-5. URL: http://link.springer.com/10.1007/s00446-013-0200-5.

[171] P. Pazzaglia, L. Pannocchi, A. Biondi, and M. D. Natale. "Beyond the Weakly Hard Model: Measuring the Performance Cost of Deadline Misses." In: *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*. Vol. 106. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 10:1–10:22. ISBN:

978-3-95977-075-0. DOI: 10.4230/LIPIcs.ECRTS.2018.10. URL: http://drops.dagstuhl.de/opus/volltexte/2018/8993.

[172] P. Pazzaglia, C. Mandrioli, M. Maggio, and A. Cervin. "DMAC: Deadline-Miss-Aware Control." In: Leibniz International Proceedings in Informatics (LIPIcs) 133 (). Ed. by S. Quinton, 1:1–1:24. ISSN: 1868-8969. DOI: 10.4230/LIPIcs.ECRTS.2019.1. URL: http://drops.dagstuhl.de/opus/volltexte/2019/10738.

[173] M. Pease, R. Shostak, and L. Lamport. "Reaching Agreement in the Presence of Faults." In: *Journal of the ACM* 27.2 (1980), pp. 228–234. ISSN: 00045411. DOI: 10.1145/322186.322188. URL: http://portal.acm.org/citation.cfm?doid=322186.322188.

[174] H. Pham. "Optimal design of k-out-of-n redundant systems." In: *Microelectronics Reliability* 32.1 (1992), pp. 119–126. ISSN: 0026-2714. DOI: 10.1016/0026-2714(92)90091-X. URL: http://www.sciencedirect.com/science/article/pii/002627149290091X.

[175] L. M. Pinho and F. Vasques. "Improved fault tolerant broadcasts in CAN." In: *8th International Conference on Emerging Technologies and Factory Automation. Proceedings (Cat. No.01TH8597) (ETFA 2001)*, 305–313 vol.1. DOI: 10.1109/ETFA.2001.996383.

[176] S. Poledna. *Fault-tolerant real-time systems: the problem of replica determinism*. The Kluwer international series in engineering and computer science ; Real-time systems SECS 345. Boston: Kluwer Academic Publishers, 1996. ISBN: 978-0-7923-9657-4.

[177] E. Possan and J. J. d. O. Andrade. "Markov Chains and reliability analysis for reinforced concrete structure service life." In: *Materials Research* 17.3 (2014), pp. 593–602. ISSN: 1980-5373, 1516-1439. DOI: 10.1590/S1516-14392014005000074. URL: http://www.scielo.br/scielo.php?script=sci_arttext&pid=S1516-14392014000300009&lng=en&tlng=en.

[178] W. Preuss. "On the reliability of generalized consecutive systems." In: *Nonlinear Analysis: Theory, Methods & Applications* 30.8 (1997), pp. 5425–5429. ISSN: 0362546X. DOI: 10.1016/S0362-546X(96)00114-9. URL: http://linkinghub.elsevier.com/retrieve/pii/S0362546X96001149.

[179] S. Punnekkat, H. Hansson, and C. Norstrom. "Response time analysis under errors for CAN." In: *6th IEEE Real-Time Technology and Applications Symposium (RTAS 2000)*, pp. 258–265. ISBN: 978-0-7695-0713-2. DOI: 10.1109/RTTAS.2000.852470. URL: http://ieeexplore.ieee.org/document/852470/.

[180] G. Quan and X. Hu. "Enhanced fixed-priority scheduling with (m,k)-firm guarantee." In: *Proceedings 21st IEEE Real-Time Systems Symposium (RTSS 2000)*, pp. 79–88. DOI: 10.1109/REAL.2000.895998.

[181] S. Quinton and R. Ernst. “Generalized Weakly-Hard Constraints.” In: *Leveraging Applications of Formal Methods, Verification and Validation. Applications and Case Studies (ISoLA 2012)*. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 96–110. ISBN: 978-3-642-34032-1.

[182] M. O. Rabin. “Randomized byzantine generals.” In: *24th IEEE Symposium on Foundations of Computer Science (SFCS 1983)*. Tucson, AZ, USA, pp. 403–409. ISBN: 978-0-8186-0508-6. DOI: 10.1109/SFCS.1983.48. URL: http://ieeexplore.ieee.org/document/4568104/.

[183] P. Ramanathan. “Overload management in real-time control applications using (m, k)-firm guarantee.” In: *IEEE Transactions on Parallel and Distributed Systems* 10.6 (1999), pp. 549–559. ISSN: 1045-9219. DOI: 10.1109/71.774906.

[184] L. Rashid, K. Pattabiraman, and S. Gopalakrishnan. “Modeling the Propagation of Intermittent Hardware Faults in Programs.” In: *16th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC 2010)*. Tokyo, Japan, pp. 19–26. ISBN: 978-1-4244-8975-6. DOI: 10.1109/PRDC.2010.52. URL: http://ieeexplore.ieee.org/document/5703223/.

[185] *Raspberry Pi 3 Model B+*. 2018. URL: https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/.

[186] M. K. Reiter. “The Rampart toolkit for building high-integrity services.” In: *Theory and Practice in Distributed Systems*. Vol. 938. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 99–110. ISBN: 978-3-540-60042-8 978-3-540-49409-6. DOI: 10.1007/3-540-60042-6_7. URL: http://link.springer.com/10.1007/3-540-60042-6_7.

[187] *Robots/cob4/manual/modules - ROS Wiki*. 2016. URL: https://wiki.ros.org/Robots/cob4/manual/modules.

[188] J. Rufino, P. Verissimo, G. Arroz, C. Almeida, and L. Rodrigues. “Fault-tolerant broadcasts in CAN.” In: *28th IEEE International Symposium on Fault-Tolerant Computing (Cat. No.98CB36224) (FTCS 1998)*. Munich, Germany, pp. 150–159. ISBN: 978-0-8186-8470-8. DOI: 10.1109/FTCS.1998.689464. URL: http://ieeexplore.ieee.org/document/689464/.

[189] E. Ruijters and M. Stoelinga. “Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools.” In: *Computer Science Review* 15-16 (2015), pp. 29–62. ISSN: 15740137. DOI: 10.1016/j.cosrev.2015.03.001. URL: http://linkinghub.elsevier.com/retrieve/pii/S1574013715000027.

[190] S-18 Aircraft and Sys Dev and Safety Assessment Committee. *Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment*. Tech. rep. SAE International. DOI: 10.4271/ARP4761. URL: https://www.sae.org/content/arp4761.

[191] G. Saggese, N. Wang, Z. Kalbarczyk, S. Patel, and R. Iyer. "An Experimental Study of Soft Errors in Microprocessors." In: *IEEE Micro* 25.6 (2005), pp. 30–39. ISSN: 0272-1732. DOI: 10.1109/MM.2005.104. URL: http://ieeexplore.ieee.org/document/1566554/.

[192] R. K. Sah. "An explicit closed-form formula for profit-maximizing k-out-of-n systems subject to two kinds of failures." In: *Microelectronics Reliability* 30.6 (1990), pp. 1123–1130. ISSN: 0026-2714. DOI: 10.1016/0026-2714(90)90291-T. URL: http://www.sciencedirect.com/science/article/pii/002627149090291T.

[193] I. Saha, S. Baruah, and R. Majumdar. "Dynamic Scheduling for Networked Control Systems." In: *18th ACM International Conference on Hybrid Systems: Computation and Control (HSCC 2015)*. New York, NY, USA, pp. 98–107. ISBN: 978-1-4503-3433-4. DOI: 10.1145/2728606.2728636. URL: http://doi.acm.org/10.1145/2728606.2728636.

[194] J. Santic. *Watchdog Timer Techniques*. URL: https://johnsantic.com/comp/wdt.html.

[195] F. B. Schneider. "Implementing fault-tolerant services using the state machine approach: a tutorial." In: *ACM Computing Surveys* 22.4 (1990), pp. 299–319. ISSN: 03600300. DOI: 10.1145/98163.98167. URL: http://portal.acm.org/citation.cfm?doid=98163.98167.

[196] S. Schuster, P. Ulbrich, I. Stilkerich, C. Dietrich, and W. SchröDer-Preikschat. "Demystifying Soft-Error Mitigation by Control-Flow Checking – A New Perspective on its Effectiveness." In: *ACM Transactions on Embedded Computing Systems* 16.5s (2017), pp. 1–19. ISSN: 15399087. DOI: 10.1145/3126503. URL: http://dl.acm.org/citation.cfm?doid=3145508.3126503.

[197] *ScyllaDB | The Real-Time Big Data Database*. URL: https://www.scylladb.com/.

[198] M. Sebastian, P. Axer, and R. Ernst. "Utilizing Hidden Markov Models for Formal Reliability Analysis of Real-Time Communication Systems with Errors." In: *17th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC 2011)*. Pasadena, CA, USA, pp. 79–88. ISBN: 978-1-4577-2005-5 978-0-7695-4590-5. DOI: 10.1109/PRDC.2011.19. URL: http://ieeexplore.ieee.org/document/6133069/.

[199] M. Sfakianakis, S. G. Kounias, and A. E. Hillaris. "Reliability of a consecutive k-out-of-r-from-n:F system." In: *IEEE Transactions on Reliability* 41.3 (1992), pp. 442–447. ISSN: 0018-9529. DOI: 10.1109/24.159817.

[200] L. Sha, M. H. Klein, and J. B. Goodenough. "Rate Monotonic Analysis for Real-Time Systems." In: *Foundations of Real-Time Computing: Scheduling and Resource Management*. The Springer International Series in Engineering and Computer Science. Boston, MA: Springer US, 1991, pp. 129–155. ISBN: 978-1-4615-3956-8. DOI: 10.1007/978-1-4615-3956-8_5. URL: https://doi.org/10.1007/978-1-4615-3956-8_5.

[201] L. Sha, T. Abdelzaher, K.-E. årzén, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. K. Mok. "Real Time Scheduling Theory: A Historical Perspective." In: *Real-Time Systems* 28.2/3 (2004), pp. 101–155. ISSN: 0922-6443. DOI: 10.1023/B:TIME.0000045315.61234.1e. URL: http://link.springer.com/10.1023/B:TIME.0000045315.61234.1e.

[202] P. Sinha. "Architectural design and reliability analysis of a fail-operational brake-by-wire system from ISO 26262 perspectives." In: *Reliability Engineering & System Safety* 96.10 (2011), pp. 1349–1359. ISSN: 09518320. DOI: 10.1016/j.ress.2011.03.013. URL: http://linkinghub.elsevier.com/retrieve/pii/S095183201100041X.

[203] C. Slayman. "Cache and memory error detection, correction, and reduction techniques for terrestrial servers and workstations." In: *IEEE Transactions on Device and Materials Reliability* 5.3 (2005), pp. 397–404. ISSN: 1530-4388. DOI: 10.1109/TDMR.2005.856487. URL: http://ieeexplore.ieee.org/document/1545899/.

[204] F. Smirnov, M. Glaß, F. Reimann, and J. Teich. "Formal reliability analysis of switched ethernet automotive networks under transient transmission errors." In: *53rd Design Automation Conference (DAC 2016)*. Austin, Texas: ACM Press, pp. 1–6. ISBN: 978-1-4503-4236-0. DOI: 10.1145/2897937.2898026. URL: http://dl.acm.org/citation.cfm?doid=2897937.2898026.

[205] T. Smith and J. Yelverton. "Processor architectures for fault tolerant avionic systems." In: *10th IEEE/AIAA Digital Avionics Systems Conference (DASC 1991)*. Los Angeles, CA, USA, pp. 213–219. DOI: 10.1109/DASC.1991.177169. URL: http://ieeexplore.ieee.org/document/177169/.

[206] *Soft real-time systems: predictability vs. efficiency*. Series in computer science. New York: Springer, 2005. ISBN: 978-0-387-23701-5.

[207] A. K. Somani and M. Bagha. "Meshkin A Fault Tolerant Computer Architecture with Distributed Fault Detection and Reconfiguration." In: *Fehlertolerierende Rechensysteme / Fault-tolerant Computing Systems*. Vol. 214. Berlin, Heidelberg: Springer Berlin Heidelberg, 1989, pp. 197–208. ISBN: 978-3-540-51565-4 978-3-642-75002-1. DOI: 10.1007/978-3-642-75002-1_16. URL: http://www.springerlink.com/index/10.1007/978-3-642-75002-1_16.

[208] J. Song, J. Wittrock, and G. Parmer. "Predictable, Efficient System-Level Fault Tolerance in C^3." In: *34th IEEE Real-Time Systems Symposium (RTSS 2013)*. Vancouver, BC, Canada, pp. 21–32. ISBN: 978-1-4799-2006-8. DOI: 10.1109/RTSS.2013.11. URL: http://ieeexplore.ieee.org/document/6728858/.

[209] D. Soudbakhsh, L. T. X. Phan, O. Sokolsky, I. Lee, and A. Annaswamy. "Co-design of Control and Platform with Dropped Signals." In: *4th ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS 2013)*. New York, NY, USA, pp. 129–140. ISBN: 978-1-4503-1996-6. DOI: 10.1145/2502524.2502542. URL: http://doi.acm.org/10.1145/2502524.2502542.

[210] B. Srinivasan, S. Pather, R. Hill, F. Ansari, and D. Niehaus. "A firm real-time system implementation using commercial off-the-shelf hardware and free software." In: *4th IEEE Real-Time Technology and Applications Symposium (Cat. No.98TB100245) (RTAS 1998)*, pp. 112–119. DOI: 10.1109/RTTAS.1998.683194.

[211] J. Srinivasan, S. Adve, P. Bose, and J. Rivers. "Lifetime Reliability: Toward an Architectural Solution." In: *IEEE Micro* 25.3 (2005), pp. 70–80. ISSN: 0272-1732. DOI: 10.1109/MM.2005.54. URL: http://ieeexplore.ieee.org/document/1463187/.

[212] M. Stamatelatos and H. Dezfuli. *Probabilistic Risk Assessment Procedures Guide for NASA Managers and Practitioners*. Technical Report NASA/SP-2011-3421. 2011. URL: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20120001369.pdf.

[213] D. H. Stamatis. *Failure mode and effect analysis: FMEA from theory to execution*. 2nd ed., rev. and expanded. Milwaukee, Wisc: ASQ Quality Press, 2003. ISBN: 978-0-87389-598-9.

[214] S. Stanley. *MTBF, MTTR, MTTF & FIT Explanation of Terms*. URL: http://www.bb-elec.com/Learning-Center/All-White-Papers/Fiber/MTBF,-MTTR,-MTTF,-FIT-Explanation-of-Terms/MTBF-MTTR-MTTF-FIT-10262012-pdf.pdf.

[215] M. D. J. Teener, A. N. Fredette, C. Boiger, P. Klein, C. Gunther, D. Olsen, and K. Stanton. "Heterogeneous Networks for Audio and Video: Using IEEE 802.1 Audio Video Bridging." In: *Proceedings of the IEEE* 101.11 (2013), pp. 2339–2354. DOI: 10.1109/JPROC.2013.2275160.

[216] *TensorFlow*. URL: https://www.tensorflow.org/.

[217] *The GNU MPFR Library*. URL: https://www.mpfr.org/.

[218] H. A. Thompson. "Transputer-based fault tolerance in safety-critical systems." In: *Microprocessors and Microsystems* 15.5 (1991), pp. 243–248. ISSN: 01419331. DOI: 10.1016/0141-9331(91)90065-N. URL: https://linkinghub.elsevier.com/retrieve/pii/014193319190065N.

[219] *Timing analysis solutions*. 2018. URL: https://auto.luxoft.com/uth/timing-analysis-tools/.

[220] K. Tindell, A. Burns, and A. Wellings. "Calculating controller area network (can) message response times." In: *Control Engineering Practice* 3.8 (1995), pp. 1163–1169. ISSN: 09670661. DOI: 10.1016/0967-0661(95)00112-8. URL: http://linkinghub.elsevier.com/retrieve/pii/0967066195001128.

[221] K. Tindell and A. Burns. "Guaranteeing Message Latencies On Control Network (CAN)." In: *1st International CAN Conference (iCC 1994)*, pp. 1–2.

[222] K. S. Trivedi. *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2016. ISBN: 978-1-119-28542-7 978-1-119-28544-1. DOI: 10.1002/9781119285441. URL: http://doi.wiley.com/10.1002/9781119285441.

[223] G. S. Veronese, M. Correia, A. N. Bessani, and L. C. Lung. "Spin One's Wheels? Byzantine Fault Tolerance with a Spinning Primary." In: *28th IEEE International Symposium on Reliable Distributed Systems (SRDS 2009)*, pp. 135–144. ISBN: 978-0-7695-3826-6. DOI: 10.1109/SRDS.2009.36. URL: http://ieeexplore.ieee.org/document/5283369/.

[224] W. E. Vesely, F. F. Goldberg, N. H. Roberts, and D. F. Haasl. *NRC: Fault Tree Handbook (NUREG-0492)*. 1981. URL: https://www.nrc.gov/reading-rm/doc-collections/nuregs/staff/sr0492/.

[225] N. Wang, J. Quek, T. Rafacz, and S. Patel. "Characterizing the effects of transient faults on a high-performance processor pipeline." In: *IEEE International Conference on Dependable Systems and Networks (DSN 2004)*. Florence, Italy, pp. 61–70. ISBN: 978-0-7695-2052-0. DOI: 10.1109/DSN.2004.1311877. URL: http://ieeexplore.ieee.org/document/1311877/.

[226] S. Webber and J. Beirne. "The Stratus architecture." In: *21st International Symposium on Fault-Tolerant Computing (FTCS 1991)*, pp. 79–85. DOI: 10.1109/FTCS.1991.146637.

[227] *Welcome — pyCPA current documentation*. URL: https://pycpa.readthedocs.io/en/latest/.

[228] J. Wensley, L. Lamport, J. Goldberg, M. Green, K. Levitt, P. Melliar-Smith, R. Shostak, and C. Weinstock. "SIFT: Design and analysis of a fault-tolerant computer for aircraft control." In: *Proceedings of the IEEE* 66.10 (1978), pp. 1240–1255. ISSN: 0018-9219. DOI: 10.1109/PROC.1978.11114. URL: http://ieeexplore.ieee.org/document/1455383/.

[229] C. Whitby-Strevens. "The transputer." In: *ACM SIGARCH Computer Architecture News* 13.3 (1985), pp. 292–300. ISSN: 01635964. DOI: 10.1145/327070.327269. URL: http://portal.acm.org/citation.cfm?doid=327070.327269.

[230] Wikimedia Commons. *File:CAN-Bus-frame in base format without stuffbits.svg — Wikimedia Commons, the free media repository*. 2017. URL: https://commons.wikimedia.org/w/index.php?title=File:CAN-Bus-frame_in_base_format_without_stuffbits.svg&oldid=232916576.

[231] T. Wolf and A. Strohmeier. "Fault Tolerance by Transparent Replication for Distributed Ada 95." In: *Reliable Software Technologies — Ada-Europe' 99*. Vol. 1622. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 412–424. ISBN: 978-3-540-66093-4 978-3-540-48753-1. DOI: 10.1007/3-540-48753-0_35. URL: http://link.springer.com/10.1007/3-540-48753-0_35.

[232] F. Yang and R. Saleh. "Simulation and analysis of transient faults in digital circuits." In: *IEEE Journal of Solid-State Circuits* 27.3 (1992), pp. 258–264. ISSN: 00189200. DOI: 10.1109/4.121546. URL: http://ieeexplore.ieee.org/document/121546/.

[233] *aiT Worst-Case Execution Time Analyzers*. 2018. URL: https://www.absint.com/ait/index.htm.

[234] *mpmath - Python library for arbitrary-precision floating-point arithmetic*. URL: http://mpmath.org/.

[235] *sched(7) - Linux manual page*. URL: http://man7.org/linux/man-pages/man7/sched.7.html.

DECLARATION

The dissertation is my own work, all sources have been named, and the dissertation (either in part or in full) has not been handed in as part of any other examination procedures.

Kaiserslautern, October 2020

Arpan Gujarati