TECHNISCHE UNIVERSITÄT MÜNCHEN

Lehrstuhl für Realzeit-Computersysteme

System-Level Diagnoses of Safety, Security and Reliability in Automotive Electrical/Electronic (E/E) Architectures

Peter Paul Waszecki

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs (Dr.-Ing.)

genehmigten Dissertation.

Die Dissertation wurde am 19.04.2016 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 23.12.2016 angenommen.

Acknowledgements

This thesis is the outcome of almost five years of research work in the Embedded Systems Group at TUM CREATE in Singapore and the Institute for Real-Time Computer Systems in Munich. It would not have been possible without the professional and personal support of many wonderful people.

First of all, I would like to express my deepest gratitude to Prof. Samarjit Chakraborty for his excellent supervision throughout all these years. Without his initial encouragement I would not have commenced this journey and without his tireless guidance I would not have finished it. I also want to thank Prof. Jian-Jia Chen for his acceptance to act as co-examiner and reviewer of this thesis as well as Prof. Gerhard Rigoll for heading the examination committee.

I am particularly greatly indebted to Martin Lukasiewycz for being a competent, patient and fair-minded mentor. His expertise and knowledge as well as his high quality standards were invaluable for my research and had a significant impact on this thesis.

My warmest thanks go to all my team-mates, co-authors, colleagues and friends I had the opportunity to meet and work with at TUMCREATE and also at RCS in Munich. The countless fruitful and motivating meetings and interesting discussions helped my work in many ways, brightened up the sometimes stressful research and paper writing sessions and made that time much more enjoyable.

Finally and mostly, I wish to thank my parents and my brother for their moral support. They believed in me and encouraged me unconditionally in times of success and struggle.

Abstract

This thesis deals with the diagnoses of safety, security and reliability in modern Automotive Electrical/Electronic (E/E) Architectures. In this context, it introduces three novel approaches for the determination of undesired behavior in distributed embedded systems, specifically the diagnoses of intermittent faults, permanent faults and security attacks.

The design and development of cars is driven by several factors which mainly comprise the increasing customer demands (e.g., comfort and fuel efficiency), the economic aspects (e.g., manufacturing processes) and the legal requirements (e.g., emission and safety standards). Over the past three decades, the technological progress in this development has turned the predominantly mechanical structures of vehicular architectures into electronics- and software-controlled Cyber-Physical Systems (CPSs). Today, these automotive E/E architectures consist of a multitude of control devices, so-called Electronic Control Units (ECUs), sensors and actuators which are communicating via different gateway-connected buses, and thus, provide the necessary hardware platform for hundreds of software functions and applications, ranging from simple electric window lifts over fuel injection control to Advanced Driver Assistance System (ADAS).

One of the major challenges for such complex and distributed architectures is to guarantee a continuous safe and reliable operation at system level, especially in case of faults or other unexpected disruptions. Appropriate diagnosis strategies, which are naturally situated at the beginning of every fault tolerance approach, are crucial and must take both safety-criticality and real-time requirements into account. At the same time, being subjected to a competitive automotive market, they must consider cost efficiency. Current diagnosis methods mostly rely on special hardware components, dedicated functions and exclusive diagnostic messages transmitted to a monitoring controller. However, this disjoint approach may lead to disadvantages such as single points-of-failure, extended diagnosis times and high implementation costs. Working often solely at component level, existing methods rarely consider the benefits a system-level perspective can create for both the diagnosis strategy and the overall reliability of an automotive E/E architecture.

To mitigate the aforementioned drawbacks and address the corresponding challenges, this thesis proposes a novel methodology for system-level diagnoses in distributed architectures. Compared to state-of-the art methods, it uses a holistic, decentralized and non-intrusive approach by analyzing a system specification at design time and using this knowledge to utilize intrinsic system properties, such as message streams and functional tests, for diagnosis at runtime. In the scope of this thesis, the above-mentioned methodology is applied to develop solutions for the following problems: (1) the diagnosis of intermittent faults based on the analysis of existing plausibility test outcomes; (2) the diagnosis of permanent faults based on current

transmission states of messages; (3) the diagnosis of security attacks considering an undesired behavior resulting from an unauthorized intrusion and traffic manipulation.

While transient faults in Integrated Circuit (IC) devices occur rarely and are considered less hazardous, intermittent faults turn up in bursts and can cause longer lasting errors and system disruptions. The objective of the proposed *intermittent fault diagnosis* is to identify an unexpected increase of faults on a resource by performing a stochastic-based evaluation of distributed plausibility tests. For this purpose, the use of a special fault expectation matrix is applied to four specifically developed detection methods which are then compared and evaluated in terms of runtime and diagnosability. With regard to the transition from federated to integrated automotive E/E architectures, the proposed techniques are suitable for multi- and many-core processors. The feasibility and efficiency of the intermittent fault diagnosis as well as its scalability is demonstrated based on a large set of automotive synthetic test cases.

When left untreated, intermittent faults can lead to permanent faults which usually are irreversible and result in a sustained error state of a resource. The objective of the proposed *permanent fault diagnosis* is to identify a faulty resource by analyzing the current states of monitored messages streams (i.e., received or omitted messages) considering system-wide data dependencies. In the scope of this thesis, special diagnosis functions for all potential fault scenarios are developed with the goal to be distributed among appropriate observing resources. The presented framework additionally includes an optimization stage determining trade-offs between diagnosis times and the number of monitored message streams. A test case evaluation as well as a case study demonstrate the functionality of the developed approach and illustrate, among other things, possible reductions of diagnosis times by up to 50 % and more. Furthermore, a schedule synthesis approach is proposed which can enhance the application of the permanent fault diagnosis in time-triggered systems. It aims at a modification of the system schedule in order to increase the diagnosability and considerably decrease the diagnosis time without compromising the previously specified constraints.

Besides fault diagnosis, the concept of monitoring existing message streams can be used to identify manipulated traffic patterns and, hence, detecting security threats in the distributed system. The objective of the proposed *security attack diagnosis* is to identify a potential intrusion into the system by verifying the compliance of each message stream with its predefined communication characteristic. In this regard, particular system parameters from a given architecture specification are used to define so-called arrival curves which are compact representations of the upper and lower bound of the number of messages arriving within a specific time interval. These curves are used to parametrize the corresponding detection algorithm, which is then automatically distributed across the in-vehicle network, taking into account user specified redundancy- and tolerance-levels. The feasibility, efficiency and scalability of this method are proven for current automotive architectures together with an estimation of its computational overhead.

In summary, the main goal of this thesis is the introduction of novel system-level diagnosis strategies for safety and security in automotive E/E architectures which might extend and partly replace existing methods. In a broader sense, the proposed general methodology, where additional but reasonable design time effort is rewarded with lightweight, efficient and fast diagnosis methods, shall help to enhance the dependability of existing and future in-vehicle networks. Beyond that this work might inspire the creation of new automotive solutions in which safety, security and reliability are integral aspects at system level rather than merely extending the functionality of particular components.

Kurzfassung (German Abstract)

Die vorliegende Doktorarbeit behandelt Diagnosen der Betriebssicherheit, Informationssicherheit und Zuverlässigkeit in modernen elektrischen und elektronischen (E/E) Fahrzeugarchitekturen. In diesem Zusammenhang werden drei neuartige Ansätze für die Bestimmung eines unerwünschten Verhaltens in verteilten Systemen vorgestellt, und zwar die Diagnosen intermittierender Fehler, permanenter Fehler sowie Sicherheitsattacken.

Der Entwurf und die Entwicklung von Fahrzeugen wird hauptsächlich durch drei Faktoren bestimmt: steigende Kundenanforderungen (z.B. Komfort und Verbrauch), wirtschaftliche Aspekte (z.B. Produktionsprozesse) und gesetzliche Vorschriften (z.B. Emissions- und Sicherheitsstandards). Die vergangenen drei Jahrzehnte technologischen Fortschritts in der Automobilentwicklung wandelten einen vorwiegend mechanischen Aufbau von E/E Fahrzeugarchitekturen in computergesteuerte cyber-physische Systeme (CPS). Heutzutage bestehen diese Fahrzeugarchitekturen aus einer Vielzahl an Steuergeräten, so genannten ECUs, Sensoren und Aktuatoren, welche über verschiedene Busse und Gateways miteinander kommunizieren und damit die notwendige Hardware-Plattform für hunderte Anwendungen bereitstellen. Dabei reichen letztere vom einfachen Fensterheber über die Kraftstoffeinspritzung bis hin zu gesamten Fahrerassistenzsystemen.

Eine der größten Herausforderungen für solche komplexen und verteilten Architekturen ist die Sicherstellung eines durchgehend sicheren und zuverlässigen Betriebs auf Systemebene, insbesondere dann, wenn Fehler und unerwartete Störungen auftreten. Angemessene Diagnoseansätze, welche naturgemäß am Anfang jeder Fehlertoleranzstrategie stehen, sind von großer Bedeutung und müssen sowohl die hohe Sicherheitsrelevanz als auch die Echtzeitanforderungen der untersuchten Systeme berücksichtigen. Zugleich erfordert ein wettbewerbsorientierter Automobilmarkt, auch die Kosteneffizienz mit einzuberechnen. Derzeitige Diagnosemethoden nutzen meistens spezielle Hardware-Komponenten, dedizierte Funktionen und eigene Diagnosenachrichten, die zu einer zentralen Überwachungseinheit übermittelt werden. Solch ein unzusammenhängender Ansatz hat jedoch auch Nachteile, wie zum Beispiel alleinstehende Fehlerstellen, lange Detektionszeiten und hohe Implementierungskosten. Da sie oft auf Bauteileebene angesiedelt sind, berücksichtigen bestehende Methoden selten die Vorteile einer systemweiten Betrachtung sowohl für Diagnosestrategien als auch für die gesamte Zuverlässigkeit von E/E Fahrzeugarchitekturen.

Um die obengenannten Probleme zu adressieren und die zugehörigen Herausforderungen zu erforschen, stellt diese Doktorarbeit eine neue Methodologie für die Diagnose von verteilten Architekturen auf Systemebene vor. Verglichen mit aktuellen Techniken, wird ein holistischer, dezentraler und eingriffsarmer Ansatz genutzt, bei dem das System zur Entwurfszeit analysiert wird, um während der Laufzeit spezifische Systemeigenschaften (z.B. Nachrichtenströme und

Funktionstests) für die Diagnose zu nutzen. Im Rahmen dieser Arbeit wird diese Methodologie auf die folgenden Problemstellungen angewendet: (1) Diagnose intermittierender Fehler, basierend auf der Auswertung bestehender Plausibilitätsprüfungen; (2) Diagnose permanenter Fehler, basierend auf dem beobachteten Zustand übertragener Nachrichten; (3) Diagnose von Sicherheitsverletzungen, bei denen durch einem unautorisierten Systemeingriff die Nachrichtenkommunikation manipuliert wird.

Während transiente Fehler in IC-Bausteinen relativ selten auftreten und generell nicht als besonders riskant betrachtet werden, erscheinen intermittierende Fehler in einer kurzzeitig großen Anhäufung und können länger anhaltende Störungen verursachen. Das Ziel der *Diagnose intermittierender Fehler* ist es, einen unerwarteten Fehleranstieg auf einer Ressource anhand stochastischer Auswertung verteilter Plausibilitätstest zu identifizieren. Hierzu wird eine spezielle Fehler-Erwartungsmatrix auf vier entwickelte Detektionsmethoden angewendet, welche später bezüglich ihrer Laufzeit und Diagnosefähigkeit verglichen und ausgewertet werden. Um die Entwicklung moderner integrierter Fahrzeugarchitekturen zu unterstützen, sind die vorgestellten Techniken auch für Mehrkern-Systeme geeignet. Die Machbarkeit und Effizienz der Diagnose intermittierender Fehler sowie ihre Skalierbarkeit wird anhand einer großen Zahl synthetischer Testfälle dargestellt.

Wenn sie unentdeckt bleiben, können sich intermittierende Fehler zu permanenten Fehlern entwickeln, die eine Ressource nachhaltig schädigen. Das Ziel der *Diagnose permanenter Fehler* ist es, eine fehlerhafte Ressource anhand der Zustände bestimmter Nachrichtenströme zu identifizieren, wobei Datenabhängigkeiten des Gesamtsystems berücksichtigt werden. Im Rahmen der Arbeit, werden spezielle Diagnosefunktionen generiert, welche später für alle potentiellen Fehlerfälle auf beobachtende Ressourcen verteilt werden. Das hierzu entwickelte Software-Framework beinhaltet eine Optimierungsphase in welcher die Diagnosezeiten und die Anzahl der überwachten Nachrichtenströme gegeneinander abgestimmt werden können. Die Evaluierung von Testfällen und eine Fallstudie belegen die Funktionalität der entwickelten Methode und, unter anderem, eine Reduzierung der Diagnosezeit um bis zu 50 % und mehr. Darüber hinaus, wird auch ein Verfahren zur Ablaufplan-Synthese vorgestellt, der die Diagnose permanenter Fehler in zeitgesteuerten Systemen verbessern kann. Das Ziel ist es, einen bestehenden System-Ablaufplan so zu verändern, dass die Diagnostizierbarkeit verbessert und die Diagnosezeiten verkürzt werden, ohne die vorher festgelegten Ablaufanforderungen zu verletzen.

Abgesehen von der Fehlerdiagnose, kann eine Nachrichtenstromüberwachung dazu genutzt werden, manipulierte Verkehrsmuster zu erkennen und damit unerlaubte Eingriffe in eine E/E Fahrzeugarchitektur zu detektieren. Das Ziel der *Diagnose von Sicherheitsverletzungen* ist es, ein mögliches Eindringen in das System zu identifizieren, indem die Nachrichtenströme auf die Einhaltung ihrer vorgegebenen Kommunikationscharakteristik geprüft werden. Hierzu werden bestimmte Systemparameter einer gegebenen Architektur für die Berechnung sogenannter Ankunftskurven verwendet, welche eine obere und untere Schranke für die Anzahl ankommender Nachrichten innerhalb eines bestimmten Zeitintervalls definieren. Diese Kurven werden zur Parametrisierung des Detektionsalgorithmus verwendet, welcher automatisch auf die verfügbaren Ressourcen verteilt wird und dabei spezielle Redundanz- und Toleranzniveaus berücksichtigt. Die Machbarkeit, Effizienz und Skalierbarkeit der vorgestellten Methode werden für moderne Fahrzeugarchitekturen untersucht.

Zusammengefasst ist das wichtigste Ziel dieser Doktorarbeit, neuartige Diagnosemethoden in E/E Fahrzeugarchitekturen vorzustellen, welche bestehende Ansetze ergänzen oder zum Teil auch ersetzen können. Im weiteren Sinne soll die vorgestellte Methodologie, bei der ein

zusätzlicher aber vertretbarer Aufwand während der Entwicklungszeit durch leichtgewichtige, effiziente und schnelle Diagnosemethoden belohnt wird, die gesamte Verlässlichkeit heutiger Fahrzeugarchitekturen verbessern. Darüber hinaus könnte die Arbeit die Entwicklung neuer Lösungskonzepte für verteilte Systeme in Automobilen anregen, bei denen Betriebssicherheit, Informationssicherheit und Zuverlässigkeit integrale Bestandteile auf Systemebene darstellen, und nicht lediglich die Funktionalität bestimmter Komponenten verbessern.

Contents

CHAPTER 1

Introduction

Automotive Electrical/Electronic (E/E) Architectures are highly distributed and complex systems consisting of multiple networked subsystems and dozens of control devices, namely Electronic Control Units (ECUs). However, the high number of ECUs together with the decreasing geometries of the corresponding Integrated Circuit (IC) components makes these architectures more and more susceptible for both internal and external faults. Additionally, the increasing interconnectedness of cars to the outside world raises serious concerns about the security of in-vehicle networks. The thesis at hand proposes decentralized and non-intrusive strategies for the diagnosis of intermittent faults, permanent faults and security attacks in modern automotive E/E architectures.

This chapter shall provide an introduction to automotive E/E architectures and their components, system-level diagnosis, as well as the corresponding challenges, which underlie the main research part of this work. Section 1.1 introduces the regarded topic on a very general level and, thereby, gives an overall motivation for this thesis. Section 1.2 provides background information about the structure, functionality and the design process of automotive E/E architectures which constitute the main target for the presented diagnosis methods. A general introduction into system-level diagnosis and its challenges is given in Section 1.3. Finally, Section 1.4 summarizes the main contributions of this thesis and Section 1.5 depicts its structure and the associated publications.

1.1 Motivation

Since the advent of the large-scale production of vehicles more than a century ago, over decades, the innovation in the automotive industry was characterized by advancements in mechanical engineering, e.g., better materials, new designs, improved manufacturing technology. And even though the first electric vehicles have seen a short boom at the beginning of the 20th century, they had soon been displaced by the Internal Combustion Engine (ICE) cars. Afterwards, for a long period electrical parts were limited mainly to illumination, the engine starter and windshield wipers, while electronics were at best included in the transistor radio. Nevertheless, over time, the innovation process also captured the electrics in the car. As a consequence, both the mechanical and electrical components could represent complicated devices by themselves. However, their interconnectedness and, thus, the overall automotive system architecture was in its structure quite robust and in its complexity relatively manageable. Technical faults, for example, engine overheating or a leakage of the break fluid, had often to be diagnosed by the driver who was monitoring and interpreting the temperature and pressure gouges that were directly indicating the condition of a device.

All of this started to change in the late 1970's, when microprocessor-equipped *engine control units* were introduced in order to regulate the fuel injection and ignition of an ICE and, hence, increase its efficiency [22]. Over the following decades, IC-based control devices, called ECUs[1], occupied more and more automotive domains, beginning with the powertrain and chassis (e.g., transmission control, power steering), over safety (e.g., Anti-lock Braking System (ABS), airbags), to modern infotainment systems (e.g., navigation, telematics). The result of this technological development are the *Automotive E/E Archtiectures* which represent in-vehicle networks that use multiple different bus systems and nowadays can consist of more than 100 ECUs [20]. Eventually, the importance of automotive electronics becomes most evident when looking at its costs and the extent of its innovation. While 30 years ago electronic embedded systems accounted for merely 1 % of the overall production costs of a car, this figure rose to 20 % in 2005 and today might reach up to 40 % [133, 67]. Within the same period of time, it is assumed that 90 % of all automotive innovations are directly or indirectly based on electronics [1] as well as the corresponding ECU-software, whose amount is growing exponentially [15].

The paradigm shift described above, namely the evolution from a mostly mechanical architecture to electronics- and software-controlled Cyber-Physical Systems (CPSs), might be also regarded as a shift from the driver to the car when it comes to the question of immediate vehicle control. For instance, Advanced Driver Assistance Systems (ADASs) enhance or entirely automate particular operations during driving and, by doing so, noticeably increase the comfort and especially the safety of the driver, passengers and other road participants. In modern luxury cars, these systems can range from Adaptive Cruise Control, which adjusts the speed of the vehicle to that of the cars in front, over an Intelligent Parking Assist System, to the Lane

[1]Although the acronym ECU predominantly stands for an electronic control unit in general, some literature still uses it specifically when relating to engine control units.

Departure Warning System. Most of the ADAS and other automotive systems require sophisticated signal processing and control algorithms which are often distributed over multiple ECUs and strongly depend on data communicated frequently (often in millisecond range) and in real-time between different sensors, processing units and actuators. Considering the time-critical maneuvers, actions and decisions initiated by the ECUs and their transmitted messages, every unexpected disruption of the system may become a safety risk.

As a consequence, an early detection of both *faults* and potential malicious *intrusions* in automotive E/E architectures becomes crucial and inevitable as only a reliable *safety* and *security* diagnosis, respectively, can allow the application of appropriate countermeasures in order to guarantee a safe and predictable vehicle behavior. However, this is also the point where today's automotive industry, driven by its competitiveness, high manufacturing numbers and component costs improvements, starts to call for solutions which are both cost efficient and easily implementable [97].

In the context of this work, the strict requirements for safety, security and, in a broader sense, reliability in automotive E/E architectures are considered, taking into account, inter alia, the specific technical and economic circumstances mentioned above. More precisely, this thesis proposes novel strategies for the distributed diagnosis of intermittent faults, permanent faults and security attacks. To fulfill the demands of the automotive industry, a special focus is put on high efficiency as well as minimal intrusiveness (e.g., by using existing communication data and resources). A detailed description of the contributions of this thesis can be found in Section 1.4.

1.2 Background: Automotive E/E Architectures

In principle, the concepts introduced in this thesis are applicable to data-dependent distributed electronic systems, of which in-vehicle networks are only one representative, albeit a prominent one. Given the impact and relevance of cars in social and economical aspects, the proposed diagnosis methods mainly target automotive E/E architectures and, thus, specifically consider the associated requirements, such as real-time constraints or predefined schedules. In the car industry, the term *E/E architectures* comprises all electrical and electronic components, their topology and interconnectedness, as well as the corresponding wiring harness [37]. A schematic illustration of an automotive E/E architecture is shown in Figure 1.1, where the small boxes represent ECUs that are connected to a central gateway via different buses, such as Controller Area Network (CAN) or FlexRay. The depicted automotive architecture is derived from the boardnet topology presented in [65] and represents a typical, but not exclusive, up-to-date network-based in-vehicle topology.

Usually the different parts of E/E architectures are classified according to their functionality, general application area, and other constraints like safety, performance and Quality-of-Service (QoS). Some of these domains, for instance, *car body* or *Human-Machine Inteface (HMI)*, are dealing with passenger-related, often less safety-critical functions, such as climate control, lighting or mirror adjustment. Other domains, for example, *powertrain* and *chassis*, mainly

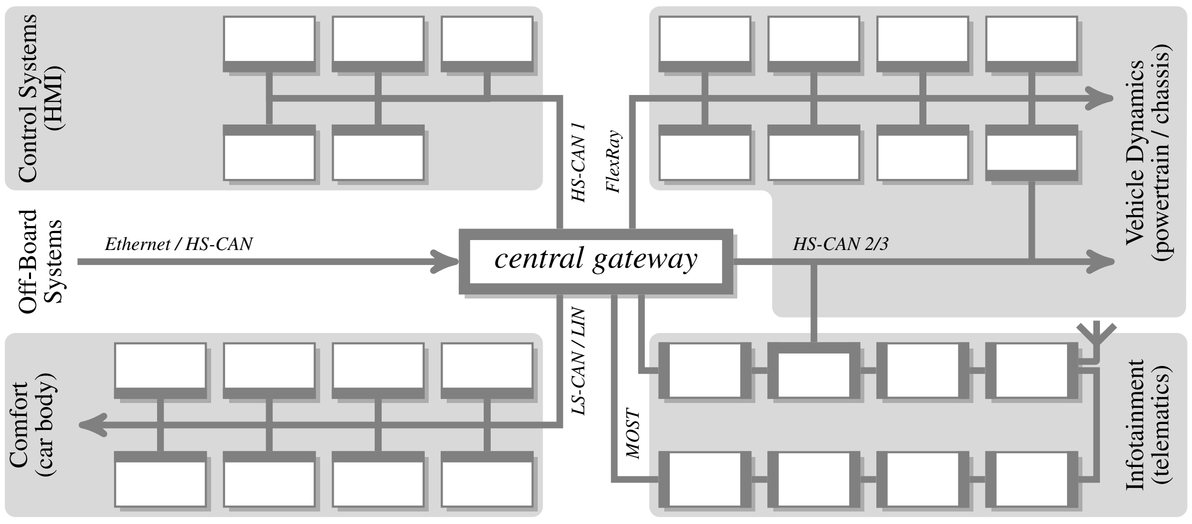


Figure 1.1: *Example for a typical modern automotive E/E architecture divided into different domains (shaded background areas). The architecture uses multiple buses to connect ECUs with each other as well as to a central gateway. Additionally, inter-domain communication is provided via dedicated links. The illustrated architectrue is inspired by the boardnet topology in [65].*

cover safety-critical and real-time dependent functions, such as engine and transmission control or X-by-wire systems, which are affecting the vehicle dynamics. Finally, the *telematics* domain contains diverse functions related to information exchange, navigation and multimedia which might not be as critical regarding safety but, due to (wireless) communication links with the outside word, possess a higher potential for security attacks. The single domains as well as their boundaries are not strictly specified and may vary among different car manufactures.

As the architecture in Figure 1.1 indicates, often the ECUs from one particular domain (highlighted by gray background areas) are also connected to an own bus, especially to facilitate the higher data-dependency of the corresponding domain functions. Nevertheless, there is a considerable and growing amount of data which is transmitted between the domains, e.g., considering the emerging ADASs relying on a number of different senor inputs and with high computational demands [81]. Although the general intra-domain communication is mainly enabled by one or more automotive gateways, dedicated links between the different domains might be used for selected data (compare cross-link between the telematics and the powertrain/chassis domain in Figure 1.1). Lastly, the off-board system communication for device-programming, maintenance or mass data exchange (e.g., navigation maps) is realized via CAN and Ethernet interfaces connected directly to the central gateway.

Clearly, the different demands and requirements in E/E architecture design not only led to a high distribution of hardware and software but also made these systems very heterogeneous, where the ECUs vary, to a greater or lesser extent, in their structure and content. In this context, an expectable trend towards ECU consolidation, which basically means that the functionality of several ECUs is supported by a single ECU, could lead to more homogeneous networks in the future [28] (see also outlook in Chapter 5). However, regardless of the level of heterogeneity, system-level approaches, such as the diagnosis methods proposed in this work, require the development and utilization of suitable automotive E/E architecture *models*. For this, an understanding of the underlying components and their different functionalities and interactions is inevitable. While the abstraction and system models of automotive E/E architectures will be discussed in detail in later chapters, the technical aspects of their main computation and communication components, namely ECUs and buses, shall be outlined below.

1.2.1 Computation

As their name already suggests, ECUs are used to control and regulate electronic systems. In a broader sense, they receive analogue or digital signals from sensors, evaluate and analyze them, compute appropriate control values (e.g., with the help of a control algorithm) and send the latter ones as new signals to the actuators. The specific application area of automotive ECUs makes high demands not only on the reliability of the software functionality but also on the hardware components and casing, as they are exposed to high external stresses and strains, such as temperature changes, strong vibrations and moisture. Furthermore, various aspects of Electromagnetic Compatibility (EMC) must be considered to guarantee both a minimal emission and a maximal immunity of electromagnetic interferences which can particularly cause transient faults [136].

1.2.1.1 ECU Hardware

Although the range of application for ECUs can be very large, they usually resemble each other in their basic structure. Figure 1.2 illustrates a general schematic diagram of a typical automotive ECU with its main interfaces to the bus, sensors and actuators. The fundamental component of an ECU is the *microcontroller* which mainly encompasses the Central Processing Unit (CPU), a memory for variable data (i.e., Random Access Memory (RAM)), different memories for program and permanent data (e.g., Read-Only Memory (ROM), Flash, Electrically Erasable Programmable ROM (EEPROM)), a clock generator and input/output (I/O) functionality. Special automotive microcontrollers often contain elements for signal acquisition, Analog/Digital Converters (ADCs), communication interfaces for automotive buses and extended safety features, such as a lockstep core which runs the same code in parallel to detect computational errors [53]. Regarding their parameters, however, the current automotive microcontrollers show significant differences. As listed in Figure 1.2, they can range from a small 8-bit controller with only a few kilobytes of main and data memory (e.g. in simple car body applications like

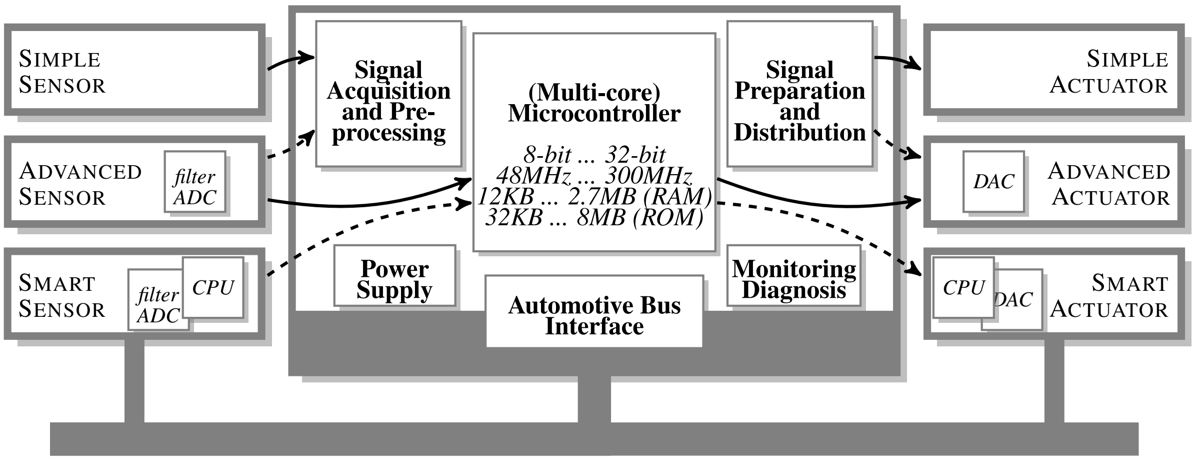


Figure 1.2: *Schematic illustration of a general ECU setup with its main components and peripherals. In the near future, particular ECU functionality could shift towards smart and advanced sensors and actuators even resulting in the removal of particular ECUs. While the shown components are typical for most ECUs, the microcontroller parameters can vary greatly depending on the application area. Here, solid arrows represent necessary interfaces while dashed arrows indicate possible connections.*

electric window lifts) to powerful 32-bit processors with multiple cores and up to 2.7 MB RAM and 8 MB ROM[2].

Other elements which many ECUs have in common are an own power supply module to transform the on-board voltage to the required component supply voltages as well as dedicated communication interfaces for the on-board-diagnosis and the corresponding automotive buses (e.g., CAN or FlexRay transceiver). The latter ones can be also integrated in the microcontroller. In cases, where raw sensor data is collected, the incoming signals must be preprocessed before they can be forwarded to the CPU. This can include noise filtering, signal level adjustments as well as A/D conversion. In contrast to the industrial automation domain, simple raw data sensors are still widely used in the automotive area, in spite of their disadvantages such as exposure to interference signals or the exclusive connection to a single device [12]. Here, novel advanced and smart sensors can transmit already preprocessed and digitized data and, hence, shift their functionality more toward that of an ECU. Together with smart actuators and the potential use of buses instead of dedicated connections, this might lead to the situation where the integrated architectures discussed in [28], although still highly distributed, would now be consisting of only a few powerful ECUs communicating with many intelligent sensors and actuators. Such a development suggests that system-level approaches for highly distributed architectures, such as

[2]The example parameter data belong to state-of-the-art products from automotive suppliers and can be found in [140, 52, 141].

the diagnoses presented in this work, will still be relevant even if the number of classic ECUs will decrease.

Finally, a substantial ECU property is its *monitoring and diagnosis* functionality, which also emphasizes their safety-critical application area. It can be as simple as a watchdog waiting for a regular control signal and as complex as an additional controller evaluating fault statistics and ensuring a controlled behavior of the ECU in case of faults [12]. As fault detection capabilities of single E/E architecture components are also relevant for this thesis, they will be discussed in more detail in the following chapters.

1.2.1.2 ECU Software

While simple vehicle tasks (e.g., the above-mentioned window lift) may still be implemented at register-level directly on an ECU microcontroller, most of the applications use the abstraction of special embedded operating systems or middlewares. Furthermore, an important requirement for automotive systems is the hard real-time capability of control functions, basically meaning that particular tasks must be strictly finished before their deadline expires. Consequently, these constraints must also be considered in the corresponding ECU software, namely in form of so-called Real-Time Operating Systems (RTOSs). Today, there exist a number of suppliers for automotive RTOSs [30, 29], which to some extent also include multi-core functionality [145]. In this context, the two platforms OSEK/VDX and Automotive Open System Architecture (AUTOSAR) are regarded as de-facto standards for automotive embedded software development. They are briefly described below.

OSEK/VDX[3], is widely applied and provides an operating system description [112] which can be used by ECU suppliers for their own specific implementation, together with additional standardized modules for the network management (OSEK NM) and communication (OSEK COM) [12]. The RTOS is based on a traditional preemptive task model but the standard also contains a description for a time-triggered Operating System (OS), OSEKtime, supporting static cycling scheduling [111]. Regarding system reliability, a fault-tolerant communication layer is offered as an extension to OSEKtime [110]. Its standard defines multiple services for interprocess communication, such as message replication and filtering, external clock synchronization and transparent task distribution, which can be used, for example, with the FlexRay bus [72]. However, due to its proximity to OSEKtime, it does not consider event-triggered protocols such as the popular CAN bus.

The AUTOSAR initiative [5], which includes several major automotive manufacturers and suppliers, derives the operating system and communication layer from the OSEK standard but goes beyond a sole OS standardization. The objective of AUTOSAR is to expand the functionality and reduce the complexity of automotive E/E architectures by providing a model-based software structure and component-based development. Standardized layers and interfaces al-

[3]OSEK/VDX stands for "Offene Systeme und deren Schnittstellen für die Elektronik in Kraftfahrzeugen / Vehicle Distributed Executive" (German for: "Open Systems and Their Interfaces for the Electronics in Motor Vehicles").

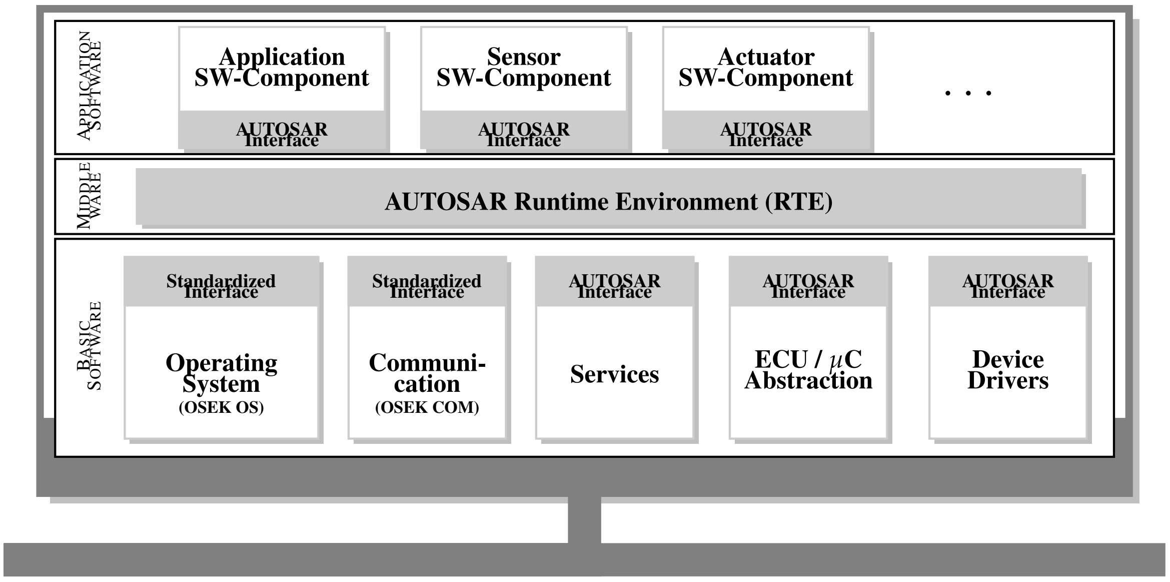


Figure 1.3: *Simplified illustration of an AUTOSAR software architecture on an ECU. Basically, the RTE abstracts the mainly hardware-specific basic software componets from the application software allowing a device-independent development of interconnected function components. A detailed technical overview of AUTOSAR can be found in [6].*

low a separation of an open subsystems integration from proprietary applications, but at the same time, require the ECU hardware to support the AUTOSAR architecture [37]. Figure 1.3 illustrates the three main AUTOSAR layers and its modules as they would be implemented on an ECU. Here, the Application Software layer mainly contains components for the functional behavior of applications, for instance as an ABS, but can also describe physical properties of sensors and actuators and, thus, decouple them from the hardware [116]. The Middleware layer consists of a Runtime Environment (RTE) which "vertically" provides the necessary interface to the Basis Software layer and "horizontally" enables the communication between software components. Using the concept of a virtual function bus (VFB), the RTE hides the lower lying hardware-specific components and facilitates a data exchange between software components on both intra- and inter-ECU level. Finally, the Basic Software layer provides the standardized modules for the actual RTOS, communication standards (e.g., CAN or FlexRay), modules for the hardware periphery, drivers and other services, such as the memory management. At this level, many standards have been adopted from OSEK/VDX [116]. Regarding safety and security, AUTOSAR offers mechanisms such as memory partitioning, end-to-end communication protection and cryptographic services since its release 4.0 [63]. Due to their lightweight and non-intrusive nature, the diagnosis strategies presented in this work could be integrated into an AUTOSAR-based system in the future.

1.2.2 Communication

The methods proposed in this theses are at system-level and, hence, strongly rely on the communication in an automotive E/E architecture as a whole. Although in general these methods are independent of specific buses, in some cases their actual implementation might need to consider the underlying bus protocols. The domain membership of the most prominent automotive bus systems has already been illustrated in Figure 1.1. In the following, they shall be introduced in more detail associated with one of three main application areas.

1.2.2.1 Real-Time

High-speed systems and real-time applications that are used, for instance, in the powertrain and chassis domains require a high frequency and a low latency for the transmitted signals and messages. Here, one of the first bus systems that was integrated in cars was CAN [56]. Its event-triggered approach using Carrier Sence Multiple Access with Bitwise Arbitration (CSMA/BA) allows a message with higher priority (or lower message ID) to win the arbitration phase and, hence, to be transmitted without interruption. To provide a sufficient bandwidth for real-time control applications, the (High-Speed) CAN standard supports transmission rates up to 1 Mbit/s. Although due to cable length restrictions and EMC effects that require expensive shielding, in practice 500 kbit/s are seldom exceeded [79]. Even though it is one of the most important and predominant bus protocols (not least because of legacy reasons), CAN nevertheless suffers from limitations such as a low bandwidth, lack of time-triggered transmission, insufficient redundant bus arrangement or unreliable group communication [130]. To mitigate the first two limitations, the extensions CAN with Flexible Data-Rate (CAN FD) and Time-Triggered CAN (TTCAN) have been introduced. The recently released CAN FD specification allows a higher throughput by increasing the original 8 bytes payload of a message frame to 64 bytes and by octuplicating the data rate of this section. Solutions to overcome compatibility issues with legacy CAN controllers have been proposed as well [18]. The goal of TTCAN, on the other hand, is to increase the predictablity and current utilization of the CAN bus. Here, the latter one can be set as low as $20 - 30$ %, in order to prevent contention and error frames [80]. Although it guarantees a deterministic data transmission, the biggest drawback of TTCAN is that it cannot provide a higher bandwidth which is essential for the increasing demands of the powertrain and chassis domains [154].

A considerably higher bandwidth of up to 10 Mbit/s is offered by FlexRay [58], a standard especially developed to support a broad range of hard real-time systems with different communication requirements [124]. For this purpose, FlexRay integrates both time-triggered and event-triggered functionality together with reliability enhancing mechanisms into one protocol. More precisely, FlexRay communication cycles of predefined length are divided into a mandatory static segment and an optional dynamic segment for the time-triggered and event-triggered communication, respectively. Fault tolerance is enabled by the use of two channels, one of which is applied for redundant transmission, as well as an optional bus guardian allowing to send messages only during its assigned time slots [125]. Considering the predominance of CAN as well as the long specification process of FlexRay, its use in series development is still rare. Nevertheless, the market already sees first cars using this protocol [115, 65].

1.2.2.2 Non Real-Time

Especially in the car body domain many functions, such as rain sensors or comfort electronics, are not safety-critical and, hence, do not require a real-time communication. Two common buses for such low-criticality applications and aiming a reduction of the wiring complexity are used, namely Low-Speed CAN and Local Interconnect Network (LIN) [59]. The former is a fault-tolerant version of the original CAN standard which includes options for fault detection and recovery on the physical-layer. For instance, a differential mode transceiver is used which switches to single-wire in case of an error condition [123]. Low-Speed CAN supports data rates between 40 kbit/s and 125 kbit/s, and hence, provides a considerably lower bandwidth than High-Speed CAN.

However, for simple data transmission and the integration of intelligent sensors and actuators CAN may still be a too expensive solution for the car manufacturers. For this purpose, the LIN standard has been introduced, offering data rates from 1 kbit/s to 20 kbit/s. The low implementation costs of LIN are achieved through a simple single-wired transmission standard and the support of the Universal Asynchronous Receiver/Transmitter (UART) interface which is available on the majority of all commercial microcontrollers. LIN follows a time-triggered approach with a master node controlling up to 16 slave nodes and guaranteed latency times, making it a predictable protocol. Moreover, it is designed as sub-bus where CAN-attached ECUs often act as gateways [105]. This, however, often leads to hierarchical network structures and, thus, more complex E/E architectures. Since version 2.0, the standard is extended, among other things, by additional frame types for sporadic and time-triggered data transmission as well as plug-and-play functionality, which also requires more costly controllers [154].

1.2.2.3 High Throughput

Multimedia and infotainment systems need a high data throughput for audio and video transmission, but do not require strict guarantees on latency and reliability as it is the case for typical control applications. Here, a de-facto standard for the automotive industry is the Media Oriented Systems Transport (MOST) protocol [98]. Within a predominant ring topology, the corresponding physical layer supports optical and electrical connectors and offers three different bit rates of circa 25, 50 and 150 Mbit/s defined in the protocol versions MOST25, MOST50 and MOST150, respectively. MOST provides synchronous channels for continuous streaming of audio and video data as well as an asynchronous channel which is packet-oriented and usually used for sporadic transfers at a high data rate [37]. While the bus access on a synchronous channel is realized by Time Division Multiple Access (TDMA), for the asynchronous channel CSMA is used making MOST both a time-triggered and an event-triggered protocol [95].

Furthermore, given its high data-rate beyond 100 Mbit/s and, not least, its predominance in computer networks, Ethernet gradually finds its way into automotive E/E architectures. For instance, MOST150 integrates an Ethernet channel and, thus, can provide a direct interface for customers' devices, such as laptops and smartphones [154]. Currently, the use of Ethernet in automobiles is mostly limited to the communication with off-board systems for ECU programming or for component diagnostics, as illustrated in Figure 1.1. The reasons are, on the one hand, the high hardware costs for switches and shielded cables and, on the other hand, the non-deterministic communication paradigm making Ethernet unsuitable for real-time applications. However, it is expectable that in the near future Ethernet in cars will expand beyond diagnostics and off-board communication and will be more and more used in the infotainment domain, ADAS or even as a backbone network for the entire automotive E/E architecture [61]. Here, approaches to facilitate the integration of Automotive Ethernet are emerging which not only decrease the wiring costs but also provide lower latencies and real-time capabilities. A prominent example is the OPEN Alliance (One-Pair Ether-Net) Special Interest Group (SIG) [109] which integrates the BroadR-Reach specification. The aim is to simplify the standard Ethernet cable with multiple wires to a single unshielded twisted-pair cable [14]. Finally, solutions providing highly reliable audio and video transmission based on Time-Sensitive Networking (TSN) or Audio Video Bridging (AVB) have been introduced especially for the use in ADASs [38].

1.2.3 Automotive Systems Design

Usually, specific vehicle components like ECUs or bus systems are designed and provided by suppliers, such as Continental, Bosch, or Delphi, rather than by Original Equipment Manufacturers (OEMs) (i.e., the car makers) themselves. These suppliers, on their part, rely on chip manufacturers, for example, Freescale or Infineon. In this context, the main task of the OEMs is the specification and engineering of requirements. OEMs are also responsible for the integration of ECUs and bus systems into their specific car models and for performing extensive testing of the whole set-up. In a nutshell, the different roles of manufacturers and suppliers in the automotive industry are distributed among three groups. The first and closest to the end product is the OEM whose tasks are the requirements specification and engineering, the integration of systems and functions as well as the validation of an entire architecture [146]. The second group consists of the Tier 1 suppliers, who are responsible for the development of ECUs and corresponding application software. Finally, the Tier 2 suppliers deliver the necessary hardware (e.g., microcontrollers and memory chips) and possibly basic software, such as firmwares or operating systems.

1.2.3.1 Development Process

In the supply chain, besides OEMs, particularly the Tier 1 suppliers are involved in the system-level design of E/E architectures. This process normally embraces different development phases which can be classified into five categories, as shown and described in Table 1.1. For the design

Table 1.1: *List of five design phases together with the corresponding descriptions. The first two columns contain the name and a short description for each design phase. The color code in the third column facilitates the assignment of the phases within the V-model in Figure 1.4.*

Design phase	*Description*	*Color*
System Modeling	The first phase covers the E/E architecture development, beginning with the definition of system requirements, followed by the modeling of the entire system and, partly, also encompassing the ECU and network modeling.	🟦
Network Modeling	The second phase focuses on the design and configuration of the network architecture, its components, and the associated data communication.	🟩
Synthesis	In the third phase the actual system functionality is implemented and should support the application design by a model-based environment. In order to deploy the application code onto real hardware, efficient code generators supporting a broad variety of microcontroller architectures for different ECUs are used.	🟨
Verification/ Validation	The fourth phase guarantees that all requirements are fulfilled and, thus, ensures the error-free operation for both the single components and the entire system architecture.	🟧
Management	In the fifth phase an acceptance validation of the entire system is performed. In contrast to the verification/validation phase, here the emphasis is more on legal and safety issues such as compliance with the automotive functional safety standards, such as ISO 26262.	🟥

and development within these phases, there exist numerous tools which have been studied, for instance, in [150].

The design phases can be transfered to a V-model [51] which is an international development standard used in software and systems design to describe the consecutive steps in a development life cycle. Named after its V-formed shape, the model depicts the chronological evolution of a system beginning with its general specification and modeling on the left branch down to the implementation and up again to test, verification, and validation on the right branch. In this respect, the amount of system detail increases on both branches towards the bottom of the V-model. Figure 1.4 illustrates a V-model describing the development process of an automotive system, where each color represents a particular design phases in Table 1.1. As the different color shades indicate, this allocation is fuzzy and the design phases might exceed the boundaries of single V-model steps. For larger and more complex projects, the V-models allows the insertion of connections between development steps at the same level (e.g., *system architecture design* and *system testing*). These connections (illustrated by horizontal arrows) represent an early-stage verification of the three lower V-model levels and the early-stage validation for the topmost level. Although there exist multiple possible representations of the V-model, which may differ in the number and naming of the used steps, they all follow a similar development cycle to the one presented here.

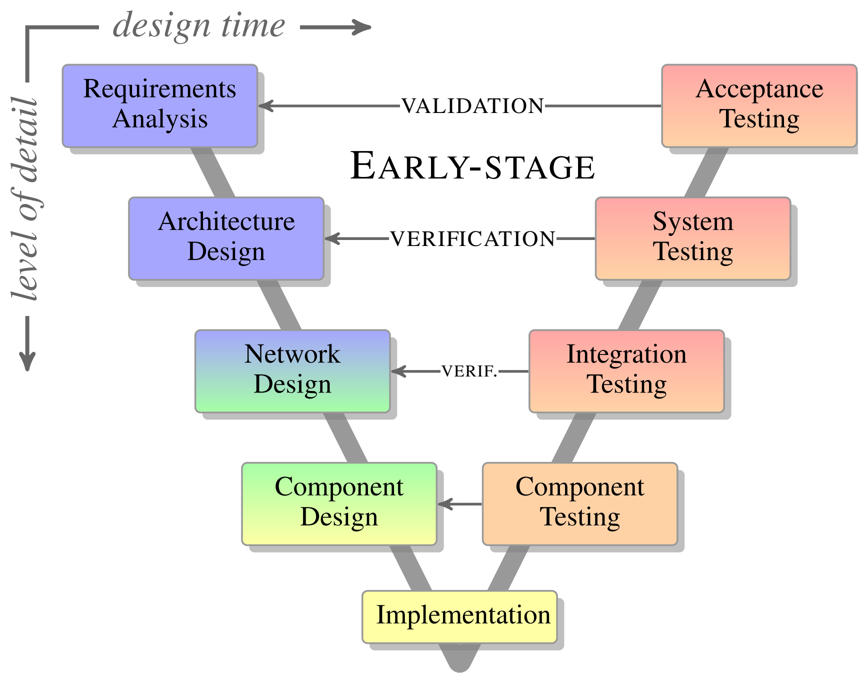


Figure 1.4: *System development life cycle within the V-model illustrating both the temporal progress and the depth of considered system details. The nine design steps are covering five design phases where the latter are highlighted by their colors according to Table 1.1. Horizontal arrows represent an early-stage testing.*

1.2.3.2 Consideration of Automotive Safety

The design of an automotive E/E architecture is a highly complex task with many interdependencies. In order to ensure an efficient and faultless development flow, it is important to use design tools which can exchange information between each other. This means, that relevant model-data from one design phase must be passed on to the next phase, and so on. Such a data exchange requires suitable interfaces and standards and ideally enables the tools to be combined within a tool-chain [150].

A consistent development flow is especially important in the area of automotive safety. Given the aforementioned growing number of electronics in cars and with it the increased risk for faults and system failures, there arises a strong need for special safety standards. Ideally, these standards should cover the entire development process, the production phase and the final operation of the car. In this context, an important step has been done recently, when the functional safety standard IEC 61508 [55] has been adapted for automotive E/E systems leading to the new standard ISO 26262 [57]. It is intended to support the functional safety of road vehicles during the entire automotive lifecycle by providing an approach for the determination of specific risk classes, so-called Automotive Safety Integrity Levels (ASILs). For instance, following the V-model, ISO 26262 Parts 4, 5 and 6 deal with the product development on system-level, hardware-level, and software-level, respectively. Depending on the particular ASIL, they not only propose different strategies to fulfill the safety requirements but also allow the use of own methods.

ISO 26262 is becoming an important requirement for safety-critical automotive ICs. For the corresponding design process, the introduction of these formal safety standards imposes higher costs and longer development times and both usually increase with the level of integrity requirements, where ASIL A is the lowest and ASIL D the highest. However, ASILs are not only defined for single ICs but also for larger subsystems and electronic modules, e.g., for steer-by-wire or brake pedals [99]. This means, that by using additional system-level diagnosis strategies such as the ones introduced in this thesis, the overall ASIL of a subsystem could be increased without the necessity of using IC components with a higher ASIL. As a consequence, an overall improvement of safety standards can be applied at lower costs.

1.3 System-Level Diagnosis and Its Challenges

Aside from the complexity of individual devices in automotive E/E architectures, such as ECUs or smart sensors and actuators, Section 1.2 particularly illustrates the high level of interconnectedness between them. While it is already a difficult task to guarantee the correct, safe and reliable functionality of these single components individually, it becomes much more demanding when taking into account real-time constrained distributed control applications where interdependent functions operate on different ECUs and exchange data.

This section gives an overview of existing diagnosis concepts in distributed systems and presents a definition for system-level diagnosis as it is used in this thesis. Furthermore, it highlights the related difficulties and challenges which are finally leading to the contributions of this thesis, discussed in Section 1.4.

1.3.1 Diagnosis in Distributed Systems

According to the Oxford Concise Medical Dictionary [94], the term *diagnosis* is defined as follows:

> *"[Diagnosis is] the process of determining the nature of a disorder by considering the patient's signs and symptoms, medical background, and - when necessary - results of laboratory tests [...]."*

In the context of this work, the automobile or, more precisely, its E/E architecture, can be regarded as the *patient*. At the same time, the *disorder* and *symptoms* might be interpreted as any unexpected safety-critical events, such as faults, and their detectable effects, such as errors and failures, respectively. Here, according to the IEEE standard glossary of software engineering terminology [54], a *fault* is a defect in a hardware device or component, an *error* is the difference between an observed value and a specified or theoretically correct value and, eventually, a *failure* is the inability of a system or component to perform its required functions within the specified performance requirements.

In the automotive industry, a safety analysis is conducted during system design in order to estimate the types and probabilities of possible faults as well as to evaluate their effects [77]. However, due to the complexity of E/E architectures and especially their interaction with the environment, it is practically impossible to predict all potential faults and to foresee their propagation and manifestation as errors and failures. Moreover, upcoming security threats (i.e., attacks, intrusions and manipulations of the system) increase this uncertainty as they open an additional source of system errors and faults. Obviously, a runtime diagnosis considering potential unexpected system behaviors is inevitable.

One possible solution could be provided by traditional *model-based diagnosis*, which does not require additional costly inspection hardware. As discussed, for example, in [60], this generally assumes that a model of the regarded physical system is executed at runtime in parallel to the actual control system. To detect faults, tests are performed that check if the deviation between the predicted outputs and the real control outputs are within predefined threshold values. Although model-based diagnosis can be used for fault detection in cars on component- or function-level [107, 96, 50], it is computationally very expensive and can hardly be applied on system-level. However, a fast, reliable and robust diagnosis of the entire system is crucial in order to guarantee automotive safety and security and offer adequate mitigation strategies. It is therefore inevitable to go beyond the fault detection of single components, which often only determines that some fault occurred.

A classic example for *system-level diagnosis* in distributed systems and a basis for many papers in the area of fault tolerance, is the PMC model which is based on the work in [114] and named after its three authors Preparata, Metze and Chien. It describes a graph representation of a distributed system where a fault-free node has the ability to correctly test another node (or groups of nodes) for proper or erroneous functionality. Collecting and evaluating the outcomes provided by the test units allows a system-wide identification of a group of faulty nodes in order to isolate them from the functioning ones [92]. The PMC model is one of the first approaches describing how a distributed system might diagnose itself in order to be aware of the health state of the single components [7]. To this effect, it helped to reduce or even eliminate the cost- and resource-intensive n-Modular Redundancy (NMR), where n processing units perform identical tasks and majority votes are used to mask faults. Limitations of the PMC model, such as a centralized analysis allowing a single point-of-failure or the consideration of only permanent faults, have been discussed and extensions have been proposed, for instance, in [84, 113, 93].

Another well known concept for system-level diagnosis is the isolation of faulty nodes in a distributed system, known as the Byzantine fault tolerance [76]. Unlike the PMC model, it can handle malicious faults manifesting themselves in a different way to different observers, but, by merely masking the faults, it lacks the ability of identifying an unhealthy node. However, in

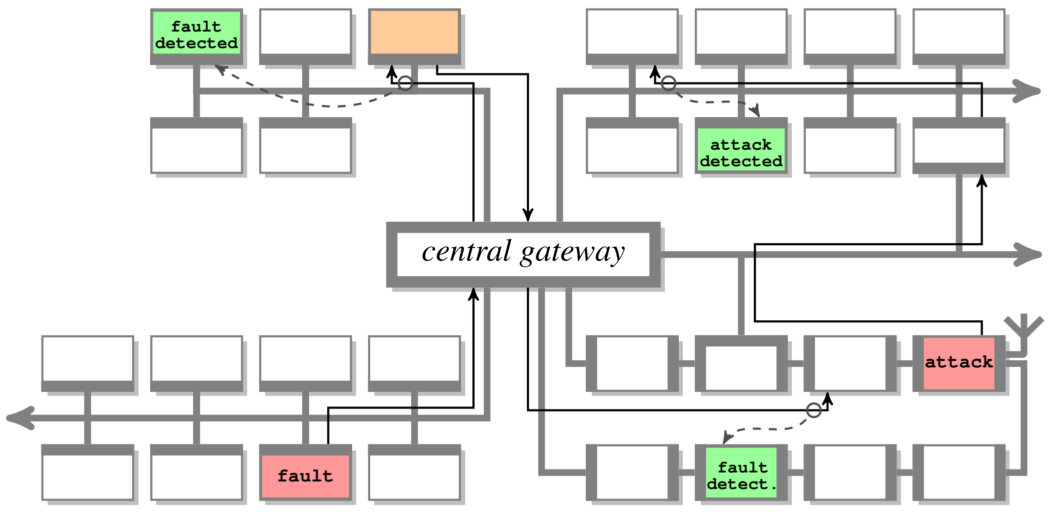


Figure 1.5: *Illustration of the general approach towards a system-level diagnosis in an automotive E/E architecture. Ideally, all ECUs have the ability to monitor other system resources at network-level in order to diagnose both faults and security attacks. Here, black arrows represent existing message-based communication and gray dashed arrows indicate a potential diagnosis ability of other ECUs.*

the heterogeneous automotive E/E architectures, where each ECU usually has a fundamentally different function, the knowledge about the locations and types of faults is essential in order to apply the appropriate countermeasures, for instance, restarting of particular tasks on a different ECU or a controlled degradation of a function or system.

Eventually, based on the discussion above and paraphrasing the original meaning of diagnosis presented before, the definition for system-level diagnosis as it is used in this work can be formulated as follows:

> *"Automotive System-Level Diagnosis is the process of determining the location, cause and kind of an unexpected and unwanted behavior in a distributed system (i.e., automotive E/E architecture) by considering the system design and specifications, the consequential detectable effects on the computation and communication, and - when necessary - the results of integrated plausibility and functional tests."*

Further detailed discussions on related diagnosis methods as well as other relevant work covering the particular diagnosis approaches presented in this thesis are located in the corresponding chapters, in Sections 2.2, 3.2 and 4.2, respectively.

1.3.2 Challenges for the System-Level Diagnosis

On the one hand, theoretical concepts, such as the PMC model, can be a help and inspiration for designing novel diagnosis methods for automotive E/E architectures. On the other hand, as already mentioned above, they also indicate the difficulties to overcome and challenges which must be dealt with in the context of automotive system-level diagnosis. Some of these challenges, which contain the four aspects *distribution*, *timeliness*, *coverage* and *efficiency*, are discribed below. To facilitate the following discussion, the overall idea for an automotive system-level diagnosis is illustrated in Figure 1.5. The depicted architecture corresponds to the in-vehicle network in Figure 1.1 which is inspired by an up-to-date boardnet.

1.3.2.1 Distribution

One of the main challenges of the presented work arises from the physical distribution of ECUs and their functionality in automotive E/E architectures. In contrast to a diagnosis on component-level where the monitoring and detection methods are limited to a particular device, a system-level diagnosis requires the distributed resources in the system to be aware of one another. That characteristic is visualized in Figure 1.5, where particular resources (green ECUs) have the ability to detect faults or security attacks on other resources (red ECUs). Obviously, for an on-line diagnosis on system-level it is necessary to monitor the existing traffic data or specifically injected diagnostic data in one way or another. For instance, one possibility to indicate if the red ECU attached to the bottom left bus is faulty might be, to observe its message stream at the immediate destination, i.e., the orange ECUs, and check for any inconsistencies, such as omitted messages. Generally, this is a valid diagnosis concept which has been used in different variations in the automotive domain, for example, in [4, 127, 64]. On the downside, however, it usually restricts the choice of monitoring resources to nodes at the corresponding messages stream destinations or, at best, to nodes attached to the same communication bus.

Here, a more powerful and beneficial diagnosis approach would be one, that allows an observing resource to be independent from a direct (physical) communication link to the observed resources. On the one hand, this approach can remove single points-of-failure by assigning multiple monitoring resources for particular safety-critical devices. This is demonstrated in Figure 1.5, where two green ECUs (on the top left and bottom right) can diagnose the faulty ECU on the bottom left. On the other hand, it gives the system designer greater flexibility in distributing particular detection tasks, for instance, to resources with lower utilization or higher computational abilities. However, such diagnosis strategies presume a deep knowledge of the entire system, particularly including architectural links between resources and buses, data dependencies of application tasks as well as their corresponding mappings to the hardware. The same applies to off-line approaches, where diagnosis data is collected at runtime but the main system-level evaluation happens when the system is not in (a safety-critical) operation. Consequently, for both on-line and off-line diagnosis strategies, a thorough analysis of a given system specification and its graph-based representation is inevitable.

1.3.2.2 Timeliness

A fast and, especially, timely execution of safety-critical tasks in automotive applications is of utmost importance, e.g., in an air-bag controller. For the corresponding real-time system which must guarantee that critical deadlines are met, there exist two communication principles, *event-triggered* and *time-triggered*. While in an event-triggered systems the occurrence of a specific event initiates a process activity, in a time-triggered system activities are started at predefined points in time [68]. As already discussed in Section 1.2.2, both principles are used in automotive bus protocols and both have their advantages and disadvantages which must be considered for system-level diagnosis methods. For instance, although time-triggered systems exhibit a higher latency than event-triggered systems, the synchronization with a global clock usually removes jitters making them more predictable. On the other hand, event-triggered systems perform better in a hard real-time environment, as they can quickly react to asynchronous internal or external events [2]. As a consequence, while purely time-triggered approaches are more and more considered for the automotive system design [118, 120], event-triggered protocols, such as CAN, will continue to play a major role for automotive E/E architectures in the near future.

In this context, it is important for novel message-based diagnosis strategies to be independent from the underlying protocol and, hence, preferably support both a time-triggered and an event-triggered communication. At the same time, the necessary observation and detection times for each given diagnosis type must be as short as possible to allow sufficient time for the application of countermeasures. For this purpose, a proper choice of the fault distribution model and observation times (e.g., for intermittent faults) as well as the use of suitable communication models (e.g., for permanent faults and security attacks) are crucial.

1.3.2.3 Coverage

An essential aspect for a reliable diagnosis strategy is the consideration of all fault and attack scenarios that are basically detectable with the help of a particular diagnosis approach. This means, that a method does not necessarily need to be capable of detecting all possible faults, but rather should correctly diagnose the specified and considered fault scenarios. Here, the latter might include intermittent faults inside a microcontroller, a group of permanently faulty ECUs as well as entire bus failures. Consequently, it must be assured first, that potential fault scenarios are properly described and, second, that the corresponding diagnosis approaches are able to detect all of them and, thus, cover the entire system. In cases where this coverage is not given, the diagnosis method should be extended appropriately in order to increase the number of diagnosable resources, e.g., through the insertion of additional diagnosis messages.

In the scope of this work, the main focus lies in the diagnosis of faults and attacks, on particular E/E architecture components. For the in-vehilce network in Figure 1.5, for example, that would include all ECUs, the central gateway and each automotive bus.

1.3.2.4 Efficiency

Today's automotive industry is strongly driven by competition and choses solutions which are cost-efficient and easily implementable [97]. Due to automotive mass production, a minimal application code size, fast development processes and even short program execution times are pursued, for which a reuse of software is increasingly applied during system design [70]. The costs become an even more crucial topic when new hardware components are introduced.

Consequently, it is important to consider the cost-efficiency and implementation factors already early during the design of novel diagnosis strategies. Ideally, the proposed approaches should exploit as much as possible of the available resources both on hardware and software level, before implementing new explicit diagnosis components (e.g. specific messages or monitors). In this context, the intrusion into the existing system architecture and additional communication should be kept minimal in order to facilitate the implementation of the appropriate diagnosis method.

1.4 Thesis Contributions

The previous discussions concerning automotive E/E architectures, the system-level diagnosis and the corresponding challenges serve as the basis and an important motivation for the work at hand. In this section, the main contributions of this thesis shall be summarized, particularly highlighting the methods and strategies of the corresponding diagnosis approaches.

As a matter of principle, the presented system-level diagnoses for automotive E/E architectures use a holistic view on the distributed architecture and are following a decentralized and non-intrusive paradigm[4], as already indicated in Section 1.3.2 and in Figure 1.5. More precisely, the proposed methods use specific knowledge acquired at design time, for instance, architecture topologies, message routings and communication timings, in order to provide reliable diagnosis strategies for particular fault types as well as security attacks. At runtime, specific and, for the most part, already integrated system properties, such as message streams and plausibility tests, are utilized. The abandonment of explicit diagnostic hardware and the exploitation of the available system functionality offers a high reliability and efficiency and distinguishes this work from most state-of-the-art diagnosis approaches.

The three main contributions stated below are listed in the same order as the subsequent thesis chapters covering them.

(1) The growing number of IC devices and their shrinking component sizes in modern automotive E/E architectures lead to an increased susceptibility of intermittent faults. We propose an *intermittent fault diagnosis*, in order to detect these faults early and prevent them from causing long lasting system damages, such as a complete failure of an ECU. The introduced

[4]In this thesis, *non-intrusiveness* indicates both a complete renunciation and a merely minimal use of explicit diagnostic components (i.e., special messages or hardware). This circumstance is sometimes also referred to as *implicitness*.

approach evaluates outcomes of existing plausibility tests and, on that basis, identifies system components for which the monitored fault rate exceeds the expected fault rate of transient faults. In contrast to a trivial method with a fault detector on each monitored resource, the proposed approach uses the design-time knowledge about mappings and dependencies of distributed tasks. As a consequence, the number of necessary plausibility tests can be reduced.

Within the proposed diagnosis framework, first, a special expectation matrix is defined, which assigns statistically estimated failure rates of all possible plausibility tests to all system resources. Furthermore, this expectation matrix can be used to analyze the overall diagnosability of the architecture at design time in the first place. When the car is in operation, the plausibility test outcomes are monitored and collected for a predefined period of time (observation time). Because a faults on one resource can also influence test results on other resources, the evaluation of test vectors is non-trivial and will be usually performed off-line. We investigate four different methods for this evaluation, and thus, for the diagnosis of intermittent faults in distributed systems. The first two methods are vector-based and make use of the linear dependencies within the system model. The remaining two methods, on the other hand, are exploiting the statistical properties of the fault model and are implemented with the help of an Integer Linear Programming (ILP) approach. Experiments with a number of automotive test cases demonstrate that while the vector-based methods are generally faster, the ILP methods achieve better results in terms of correctness and show less false negative outcomes. As a consequence, aside from a reliable fault diagnosis, our approach offers the system designer a flexible tool for selecting and implementing the most suitable of the four detection methods with respect to the regarded architecture.

Additionally, we propose an extension of the intermittent fault diagnosis towards many-core systems which are becoming more and more relevant for automotive E/E architectures. Because future many-core systems are expected to consist of hundreds or even thousands of cores, the corresponding experimental results are especially focusing on the scalability of the proposed diagnosis methods.

(2) Since intermittent faults often occur in bursts, in the most cases an affected resource will still be operational afterwards, allowing the application of particular precautionary measures before a more severe failure can happen. This is usually not the case for permanent faults which are preventing the resource from performing any useful functionality. We propose a *permanent fault diagnosis* where the affected resource is detected instantly on the basis of so-called diagnosis functions and can be identified on one or more other resources. By offering a very short and deterministic diagnosis time, our approach keeps the hazardous downtime minimal and supports, inter alia, the application of degradation strategies where crucial system functions must be quickly adopted by other suitable resources.

Overall, the proposed diagnosis framework consists of two stages. In the first stage, a graph-based search algorithm is applied to a given system specification in order to determine possible faulty resources, so-called fault scenarios, as well as the corresponding message streams affected by a potential permanent fault. Based on this, for each fault scenario and each observing resource unique diagnosis functions are generated. These consist of special Boolean representations of omitted and received messages indicating a particular fault scenario. Implemented on the respective observing resources, the diagnosis functions enable a fast detection of permanent faults located anywhere in the architecture, namely, on network-level. The second stage of the framework provides an ILP-based optimization of the generated diagnosis functions aiming at a trade-off analysis between the diagnosis times and the amount of monitored message streams. A number of automotive test cases and a detailed case study give evidence of the overall feasibility and efficiency of our method and demonstrate that the optimization stage is capable of reducing the initial diagnosis time in particular cases by more than half. Additionally, the experimental results comprise the outcome of an implementation of the permanent fault diagnosis on a research platform for distributed automotive systems. Here, the evaluation of real hardware utilization illustrates the practical usefulness of the proposed approach.

In the context of permanent fault diagnosis, we furthermore propose a *diagnosis-aware schedule synthesis* for time-triggered systems. It improves our approach in terms of the required diagnosis times and increases the number of diagnosable resources. For this purpose, first, a given system schedule is modified by adapting the transmission times of existing messages in the network in order to obtain an optimal distribution for their subsequent monitoring. Second, the available bandwidth of the communication channel can be used to insert lightweight diagnostic tasks and messages for fault scenarios which cannot be naturally diagnosed, for instance, because they are located at the end of a transmission chain and lack outgoing message streams.

(3) Novel automotive functionality, such as Car-to-Car (C2C) and Car-to-Infrastructure (C2I) communication or Over-the-Air (OTA) software updates, use wireless interfaces to the outside world and make cars more and more vulnerable to security attacks. In this regard, an exposed and hacked E/E architecture component can increase the overall safety risk for road participants to the same extent as a faulty component. Many common forms of attacks (e.g., message flooding or Denial of Service (DoS)) manifest themselves in altered traffic patterns with unforeseeable consequences for the highly timing-critical automotive control functions. We propose a *security attack diagnosis* to identify a potential intrusion into the system quickly by verifying the compliance of message streams with a predefined communication characteristic.

Within the proposed security diagnosis framework, initially, a given specification of the system is analyzed in terms of its communication parameters as well as its application and architecture structure. The goal is to obtain the necessary information, such as message timings and task mappings, for the subsequent diagnosis process. The second part of the framework describes the actual diagnosis algorithm for the decentralized and lightweight detection of message-based attacks. Furthermore, it defines an optimization-based and redundancy-aware method to efficiently distribute the diagnosis functions among suitable monitoring resources. By using so-called message arrival curves (i.e., representations of the upper and lower bounds for the number of incoming message stream events in an arbitrary time interval), the diagnosis algorithm is easily adaptable for different patterns of message streams and, hence, facilitates the distribution of the single diagnosis tasks. In our approach, this distribution is automated using an ILP and allows to set specific redundancy and tolerance values. As a consequence, the system designer does not have to manually select each monitoring resource or assign the corresponding message streams. Instead, the proposed framework provides sufficient flexibility for a proper configuration of the distribution in terms of redundancy as well as allocation of the monitoring tasks. The experimental results show, that by using a light-weight detection method and an optimization-based task distribution, our security approach guarantees a full coverage and timeliness of the diagnosis and offers a good scalability while imposing a low additional overhead.

1.5 Organization and Publications

This section outlines the remaining chapters of the thesis and contains a list of the corresponding publications.

1.5.1 Organization of the Thesis

After the general introduction presented in this chapter, the thesis continues with the actual description of the conducted research. Chapter 2 presents the proposed intermittent fault diagnosis. The general approach, whose results have been published in [147], is explained in Sections 2.1 to 2.5. The extension towards many-core systems is discussed in Sections 2.6 and 2.6.3 and appeared in [148]. Chapter 3 presents the proposed permanent fault diagnosis and can be roughly divided into two parts. First, Sections 3.1 to 3.5 discuss the main diagnosis approach, its methodology and the corresponding experimental results which have been published in [149]. Second, a diagnosis-aware system design approach for the permanent fault diagnosis is presented in Section 3.6. The corresponding results appeared in [151]. Chapter 4 presents the proposed diagnosis of security attacks. The corresponding results of this approach are still under submission at the time of writing this thesis. The final Chapter 5 concludes the thesis, discusses possible future work and gives a brief outlook.

All publications associated with this thesis are listed below.

1.5.2 Corresponding Publications

The main contributions presented in Chapter 2 appeared in the following two publications.

(1) Peter Waszecki, Matthias Kauer, Martin Lukasiewycz, Samarjit Chakraborty: *Implicit Intermittent Fault Detection in Distributed Systems*. In Proceedings of the 19th Asia and South Pacific Design Automation Conference (ASP-DAC), 2014.

(2) Peter Waszecki, Martin Lukasiewycz, Samarjit Chakraborty: *Multi-Objective Diagnosis of Non-Permanent Faults in Many-Core Systems*. In Workshop Proceedings of the 27th International Conference on Architecture of Computing Systems (ARCS), 2014.

Parts of the contributions presented in Chapter 3 appeared in the following two publications.

(3) Peter Waszecki, Martin Lukasiewycz, Samarjit Chakraborty: *Decentralized Diagnosis of Permanent Faults in Automotive E/E Architectures*. In Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), 2015.

(4) Peter Waszecki, Florian Sagstetter, Martin Lukasiewycz, Samarjit Chakraborty: *Diagnosis-Aware System Design for Automotive E/E Architectures*. In Proceedings of the International Symposium on Integrated Circuits (ISIC), 2014.

The main contributions presented in Chapter 4 have been submitted as the following publication and are currently under review.

(5) Peter Waszecki, Philipp Mundhenk, Sebastian Steinhorst, Martin Lukasiewycz, Samarjit Chakraborty: *Diagnosing Altered Traffic Patterns for Security in Automotive E/E Architectures*. Submitted to IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. (*under review*)

The publications stated below address the general topic of automotive E/E architectures, the corresponding design process as well as task scheduling and synthesis. They are therefore closely related to this thesis.

(6) Florian Sagstetter, Peter Waszecki, Sebastian Steinhorst, Martin Lukasiewycz, Samarjit Chakraborty: *Multi-Schedule Synthesis for Variant Management in Automotive Time-Triggered Systems*. In IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2015.

(7) Florian Sagstetter, Sidharta Andalam, Peter Waszecki, Martin Lukasiewycz, Hauke Staehle, Samarjit Chakraborty, Alois Knoll: *Schedule Integration Framework for Time-Triggered Automotive Architectures*. In Proceedings of the 51st Design Automation Conference (DAC), 2014.

(8) Peter Waszecki, Martin Lukasiewycz, Alejandro Masrur, Samarjit Chakraborty: *How to Engineer Tool-Chains for Automotive E/E Architectures?*. In ACM SIGBED Review, 2013.

(9) Martin Lukasiewycz, Sebastian Steinhorst, Sidharta Andalam, Florian Sagstetter, Peter Waszecki, Wanli Chang, Matthias Kauer, Philipp Mundhenk, Suhaib A. Fahmy, Shreejith Shanker, Samarjit Chakraborty: *System Architecture and Software Design for Electric Vehicles*. In Proceedings of the 50th Design Automation Conference (DAC), 2013.

(10) Martin Lukasiewycz, Sebastian Steinhorst, Florian Sagstetter, Wanli Chang, Peter Waszecki, Matthias Kauer, Samarjit Chakraborty: *Cyber-Physical Systems Design for Electric Vehicles*. In Proceedings of the 15th Euromicro Conference on Digital System Design (DSD), 2012.

CHAPTER 2

Diagnosis of Intermittent Faults

This chapter introduces a strategy for an *intermittent fault diagnosis* in a distributed system which follows the basic assumptions of non-intrusiveness and decentralization introduced in Chapter 1. More precisely, we investigate an approach to identify components, such as ECUs in an automotive E/E architecture, where the occurrence of observed non-permanent faults exceed the amount of expected transient faults caused by environmental phenomena. Our approach indicates the presence of intermittent faults which usually occur due to stressed resources and are often a precursor of the more severe permanent faults.

The proposed fault diagnosis enables an early use of precautionary measures, namely, before a potential affected resource in a distributed system fails permanently. For this, the diagnosis framework analyzes and evaluates four different methods designed to implicitly detect intermittent faults by considering both the distribution of application tasks and their data-dependencies. The methods are using the outcomes of existing plausibility tests which are stored in special expectation matrices so that explicit tests, which would lead to additional costs and resource load, are not required. Moreover, the proposed methods can considerably reduce the number of necessary plausibility tests compared to the conservative solution with one test per resource. The analysis and evaluation of experimental results give evidence of the feasibility of our approach and show a comparison of the implemented methods in terms of runtime and detection rate.

Chapter outline. Chapter 2 is divided in seven sections. In Section 2.1, the topic is introduced with an illustrative example and the particular contribution is described. A detailed discussion of existing related work is presented in Section 2.2. In Section 2.3 the intermittent fault diagnosis approach is explained in detail, including the fault model, the expectation matrix and the diagnosability analysis. A formal definition of four different diagnosis methods is given in Section 2.4 and the corresponding evaluation with respect to automotive E/E architectures is presented in Section 2.5. An extension of the intermittent fault diagnosis towards many-core systems is discussed in Section 2.6 where the appropriate experimental results are presented in Section 2.6.3. Finally, Section 2.7 concludes this chapter and briefly outlines the future work.

2.1 Introduction

This section gives an introduction into the intermittent fault diagnosis. After a general motivation part, it demonstrates a simple illustrative example of the diagnosis approach and points out the specific contributions.

2.1.1 Motivation

Today, the reliability of embedded systems and the associated safety aspects are of high relevance in many domains with strict real-time requirements such as in avionics and automotive. Also for non-safety critical applications, for example, in consumer electronics, efficient fault detection and fault tolerance mechanisms are important to fulfill the customers' quality expectations. At the same time, the ever-growing Very Large Scale Integration (VLSI) processes with shrinking geometries and decreasing power supply voltages result in devices which are increasingly susceptible to transient faults and, hence, might have a negative impact on the system reliability [23]. In distributed systems, the failure of a single component can influence the behavior of a multitude of applications. It is therefore of utmost importance to indicate the risk of potential errors and failures of components before they actually occur. This allows to apply precautionary measures, which can vary from graceful degradation to a replacement of the affected component.

For such an early detection[1], an increased number of non-permanent faults is a suitable indicator to determine stressed components. Assuming that a defective hardware causes the occurrence and accumulation of so-called *intermittent* faults which lead to (possibly distributed) errors, then a potential imminent failure of a specific resource could be projected by analyzing the results of appropriate plausibility tests running within regular tasks or as discrete applications. A major objective of the proposed approach is to perform such a detection implicitly in order to keep the additional costs and resource utilization low. In contrast to an explicit method which would require additional test tasks for each component, the proposed fault diagnosis relies on existing plausibility tests which are part of the distributed applications.

[1]Usually, the term *detection* relates to the identification of the presence of a fault, whereas *diagnosis* describes the location and identification of its cause. However, in some cases this distinction may be ambiguous and the terms can be used interchangeably.

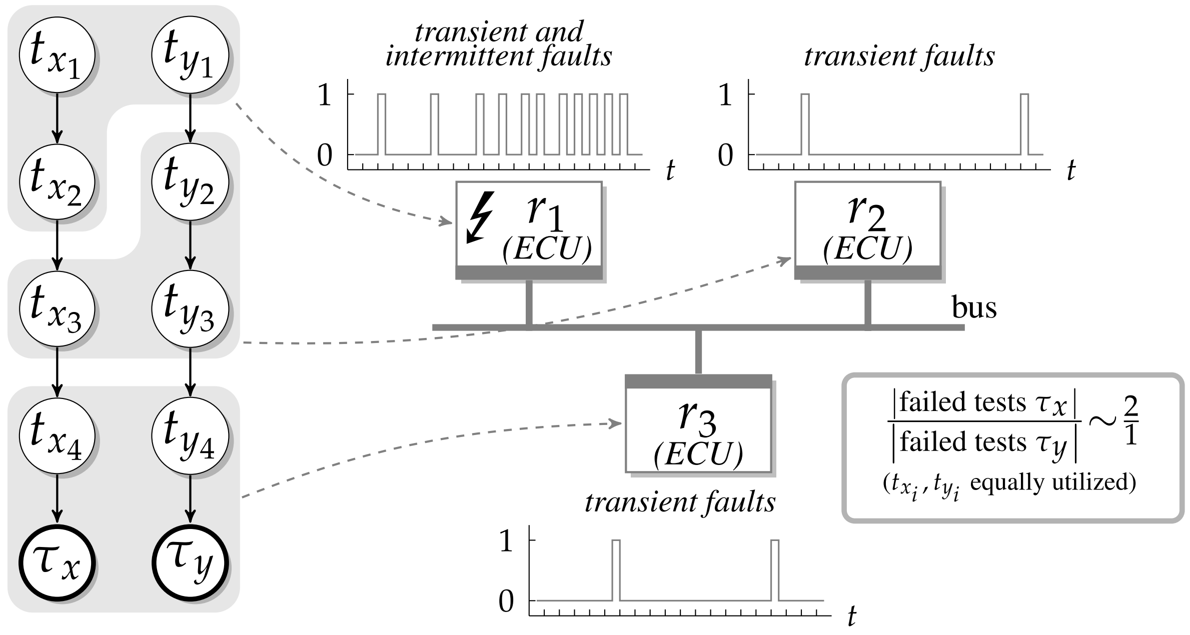


Figure 2.1: *Simplified example for the the intermittent fault diagnosis of a faulty ECU in an automotive E/E architecture. The application tasks t_{x_1}, t_{x_2}, t_{y_1} are mapped to ECU r_1 and t_{x_3}, t_{y_2}, t_{y_3} are mapped to r_2. The remaining t_{x_4} and t_{y_4} as well as the plausibility test tasks[2] τ_x and τ_y are assigned to r_3. Given an equal task utilization, the increased intermittent fault rate on r_1 results in a test failure ratio of 2:1 between τ_x and τ_y.*

In automotive E/E architectures, these plausibility tests are used, for instance, to ensure that the control values an ECU is dealing with are correct. However, due to the architectural constraints of the resources, it is often only possible to check the consistency of test results at specific points, e.g., on particular ECUs. For this purpose, we take the distribution of applications, the runtimes of tasks, and their data-dependencies into account to implicitly determine the component with an increased number of intermittent faults.

2.1.2 Illustrative Example

To better illustrate the diagnosis approach, a motivating example is shown in Figure 2.1. In the left part of the figure an application with two functions, f_x and f_y, is shown, which, for instance, could represent distributed control algorithms. Each function consists of four application tasks t_{x_i/y_i} and two test tasks $\tau_{x/y}$ meant to perform a plausibility check at the end of the function execution. The interconnection between the tasks in each function indicates a strong depend-

[2]Although they are using a different identifier, a *test* τ is regarded and treated equivalent to a task t. Therefore, the terms *test* and *test task* are interchangeable.

ability meaning that any error in one task will be propagated to the corresponding test task. All eight tasks are mapped to the three ECUs in the right part of the figure, as indicated by the gray background areas. Now, depending on the particular assignment and utilization of the tasks, a higher rate of intermittent faults on a resource can be detected by analyzing the failure ratio of the test tasks. Given an equal utilization among all application tasks and a consistent error propagation towards the test tasks, a faulty ECU r_1 will cause a failure ratio between τ_x and τ_y of 2:1, since it is running two tasks from the f_x-task chain and only one from the f_y-task chain. Correspondingly, a test failure ratio of 1:2 would indicate a faulty r_2 and a test failure ratio of 1:1 a faulty r_3.

2.1.3 Contributions

The overall purpose of the diagnosis presented in this chapter is to identify intermittent faults early before a permanent fault occurs in order to apply appropriate precaution measures. Furthermore, an important aspect is the implicit nature of the approach which reduces an additional implementation and testing overhead. The main contributions of the proposed work comprise four different methods for an implicit and decentralized intermittent fault diagnosis in distributed systems. The methods use an *expectation matrix* which comprises the expected failure rates for all plausibility tests for resources in a functioning (i.e., intermittent fault-free) system. The first two methods are based on the analysis of linear dependencies within the system model, while the other two methods use an ILP for an optimization-based approach. Based on the evaluation of a number of synthetic test cases[3], we show the feasibility of the proposed fault diagnosis and compare the different methods in terms of their runtime and detectability.

Basically, we consider the automotive diagnosis to happen on the level of E/E architectures, i.e., we assume ECUs, sensors, and actuators as the main and crucial components. However, multi- and many-core Systems-on-Chip (SoC) are becoming more and more important for the automotive industry and due to their use in safety-critical systems they will be subjected to higher standards regarding the fault diagnosis and fault tolerance. As a matter of principle, the proposed methodology is also applicable at other levels of granularity where, for instance, in case of a multi-core SoCs, the basic components would be processor cores, buses, switches, or memories. In Sec. 2.6 the extension of the intermittent fault diagnosis to many-core systems is discussed. More precisely, the four introduced diagnosis methods are applied to architectures with several hundred cores and evaluated regarding the correct diagnosis rate and runtime. While for E/E architectures the number of basic components stays in a double-digit or lower three-digit range, other domains might easily reach ranges of 10^4 or even 10^5, which makes the scalability of our approach an important objective.

[3]In the scope of this chapter, *test*, *test task* or *plausibility test* refer to the functional tests as part of the system to be analyzed, whereas *test case* and *test run* refer to the different system specifications used for the experimental evaluation in Sections 2.5 and 2.6.3.

2.2 Related Work

In the regarded distributed systems a resource can execute one or multiple periodic tasks or, generally speaking, manipulate data. At the same time, these tasks can have data-dependencies across different resources and use messages or shared memory for communication. Allocating and scheduling these tasks on system resources is a crucial part during system design. It often uses a so-called Y-chart approach, where the applications and architectures are consolidated through task mappings and message routings to get a complete system specification before a subsequent schedule synthesis is performed. This process flow is elaborately discussed, for instance, in [85]. It is assumed, that the system specifications used for the proposed approach are generated based on such a principle such that the correct system information can be properly utilized during the fault diagnosis.

Regarding fault tolerance in distributed systems, the common strategies include task remapping and rescheduling, respectively, to allow a continuing execution of safety-critical functions on other resources in case of errors and failures. For example, in [153], a fault-tolerant scheduling is presented, where a backup copy of a task on a faulty resource will be executed on a healthy one and the scheduler excludes faulty nodes. Furthermore, task redundancy can be used, to either detect a fault using task duplication or resolve a fault using task triplication, as shown in [24]. When intact resources do not have enough free capacity to take over all tasks of the affected resources, a graceful degradation mechanism is necessary as described, for example, in [35] and [31]. A comprehensive overview of state-of-the art techniques used for fault tolerance strategies for real-time distributed systems is presented in [34].

The methods proposed here are set at the level of fault diagnosis and, thus, before the fault tolerance mechanisms come into action. For this reason, they are based on a given task distribution and an existing schedule but are not restricted to a particular method to eliminate the faulty state. In this context, there exists a number of approaches that are dealing with the reliability and fault diagnosis in distributed systems. In [132], a method for the detection of transient faults is proposed where the execution flow of a preemptive RTOS is monitored in order to identify a scheduling misbehavior. Although, the introduced technique is passive and does not interfere with the existing execution flow in the embedded system, it is a hardware-based solution requiring a special guardian core. By contrast, [42] presents multiple on-line self-testing policies for dynamic applications in multi-processors which affect the scheduling and allocation of applications but not the underlying hardware. The work concludes that periodically applied tests can show a good trade-off between performance and fault detection probability if it considers idle states of the processors, the test history as well as task priorities.

Approaches, which address the fault diagnosis problem on a system design level are presented, for instance, in [152] and [142]. The work in [152] proposes a design methodology where safety-critical tasks are selectively duplicated in order to detect soft errors caused by transient faults. As the allocation and scheduling considers idle computation times and mutual exclusiveness of tasks, the method does not increase the existing resource utilization. However, due to the required schedule knowledge, it cannot be easily applied for event-triggered systems. Similarly, the work in [142] addresses soft errors with the help of a high-level synthesis approach. Here, first a reliability characterization of specific system components, such as adders and multipliers, is applied. Then, within a Data Flow Graph (DFG) representation of the system the overall reliability is maximized by selecting the most suitable resources on condition that specific area and performance constraints are met. However, for the latter objectives the selection process may be suboptimal. As a remedy, [36] proposes a multi-objective design space exploration which considers not only reliability but also all other design objectives simultaneously.

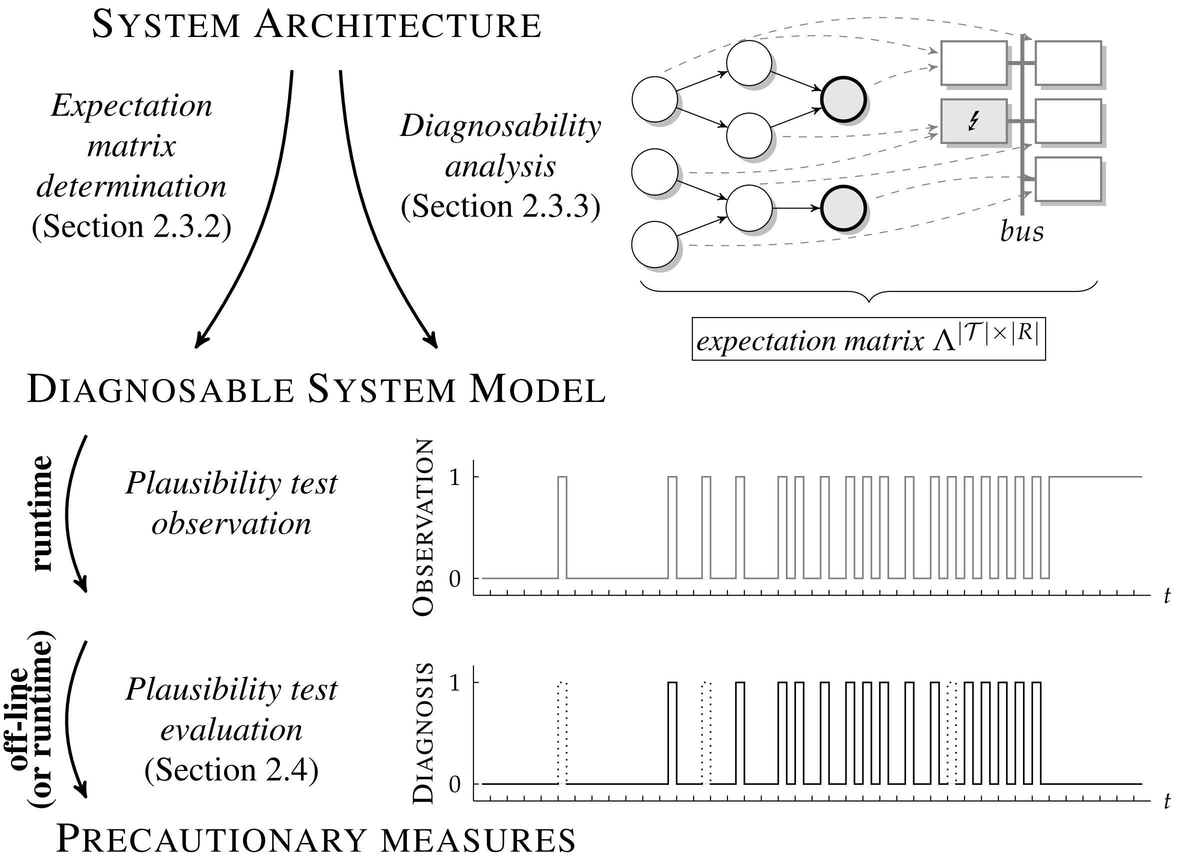
Compared to the approaches above, the intermittent fault diagnosis proposed in this thesis specifically considers non-intrusiveness and does not require any changes to the existing system specification. Moreover, by taking into account that non-permanent faults might finally lead to permanent faults and, in the worst case, system failures, it stresses the importance of a fast and reliable diagnosis in a safety-critical real-time environment.

Many-core systems. For the application of our approach to many-core systems, it is assumed that one core can have multiple periodic tasks assigned to it but it can only execute one task at a time. Nevertheless, this does not restrict the distribution of applications where tasks have data-dependencies across different cores and use, for instance, a Network-on-Chip (NoC) or shared memory for communication. Consequently, also here an efficient mapping between tasks and cores is an important aspect to satisfy high performance and safety requirements.

A survey and categorization of mapping methodologies for multi- and many-core systems is presented in [134], where the analysis differentiates between design-time and runtime optimization methods. In [62], static task mapping for embedded many-core SoC is using an ILP-based and a greedy algorithm-based approach, respectively, in order to find the optimal number of cores for each task. To increase the reliability for many-core system, one can use similar fault tolerance strategies as for the general distributed systems discussed above, such as re-mapping of tasks. Some are presented in [3] and [25] for the use of NoC-based Multiprocessor System-on-Chips (MPSoCs). In [48], a lifetime reliability estimation of homogeneous many-core systems is proposed, which analyzes different configurations and redundancy schemes. However, that paper mainly regards system faults that manifest themselves as permanent faults while our work analyzes and compares test failure rates in order to implicitly detect cores in a many-core system tending to fail due to an increase of intermittent faults.

2.3 Diagnosis Description

This section gives a basic description of the intermittent fault diagnosis and the corresponding framework which is depicted in Figure 2.2. It schematically illustrates how a diagnosable system model is extracted from the system architecture and then used for a runtime observation and an off-line diagnosis in order to locate a faulty resource.



SYSTEM ARCHITECTURE
Expectation matrix determination (Section 2.3.2)
Diagnosability analysis (Section 2.3.3)
bus
expectation matrix $\Lambda^{|\mathcal{T}|\times|R|}$
DIAGNOSABLE SYSTEM MODEL
runtime
Plausibility test observation
OBSERVATION
1
0
t
off-line (or runtime)
Plausibility test evaluation (Section 2.4)
DIAGNOSIS
1
0
t
PRECAUTIONARY MEASURES

Figure 2.2: *Basic framework structure of the intermittent fault diagnosis. At design time, a given system specification is used to determine the expectation matrix as well as its general diagnosability. The plausibility test outcomes are observed at runtime and usually evaluated off-line in order to pinpoint a potential faulty resource. The subsequent precautiory measures are not part of this work.*

2.3.1 Fault Model

Within the considered system model, we assume that a resource can be affected by both permanent faults and non-permanent faults which can be further divided into transient and intermittent faults. Although the latter two fault types only temporarily affect the system, intermittent faults are more severe than transient faults as they last much longer and can occur in bursts [27]. Usually, permanent faults, which often result in enduring physical changes of the affected hardware, are often preceded by an increased number of intermittent faults which themselves are caused by malfunctioning hardware and occur with a high arbitrary frequency, see [23]. In contrast to intermittent faults, transient faults are caused by temporary environmental phenomena, like cosmic rays, Electromagnetic Interference (EMI), Electrostatic Discharge (ESD) or radiation from lightning.

Within the diagnosis process, the presence of intermittent faults is recognized with the help of plausibility tests which are evaluating the outcome of particular periodic tasks. In this context, according to the hypothesis of the Resilience Articulation Point (RAP), all faults originating from a physical phenomenon, if not masked, will manifest themselves as a single- or multi-bit flip [41]. It is assumed that if the errors caused by bit flips are not masked, they are propagated between data-dependent tasks and finally result in plausibility test failures, with the probabilistic fault distribution remaining unaltered. Thus, a failed plausibility test indicates an error in its executing test task or a preceding task, originally caused by a transient or intermittent fault on a corresponding resource.

The occurrence of non-permanent faults is assumed to happen independently of the preceding faults and with a known average rate. This circumstance makes the Poisson distribution a good choice to model the fault probability. Nevertheless, the presented approach is flexible and expandable, so that it can adopt other probabilistic error models if necessary or required.

In order to illustrate the intermittent fault diagnosis principle, Figure 2.3 depicts how fault rates with different expected values are reflected in the occurrence of non-permanent faults over time and how it can be used to distinguish transient and intermittent faults. Here, each peak in the two upper graphs reflects one single fault while the lower curves represent two distinct Poisson distributions. During regular operation a resources will be subject to transient faults. However, with advancing operation time the occurrence rate of intermittent faults can increase, e.g., due to aging effects of the hardware. In the example in Figure 2.3 a possible *observed* increase of non-permanent faults over time finally results in a permanent fault. Using a known fault rate for the expected transient faults (e.g., $\lambda_P = 3$) allows to split the observation into a transient and an intermittent part, with the latter becoming more and more predominant over time as illustrated by a higher expected value of the Poisson distribution λ_P. This illustration also demonstrates, that it is important to consider the transient faults in our diagnosis approach as a kind of "noise", as it helps to avoid possible false positive fault detections.

As already mentioned, we analyze a number of plausibility tests at the end of particular task chains and compare the expected and observed results of several tests. This allows us to diagnose resources affected by intermittent faults. However, due to architectural and cost

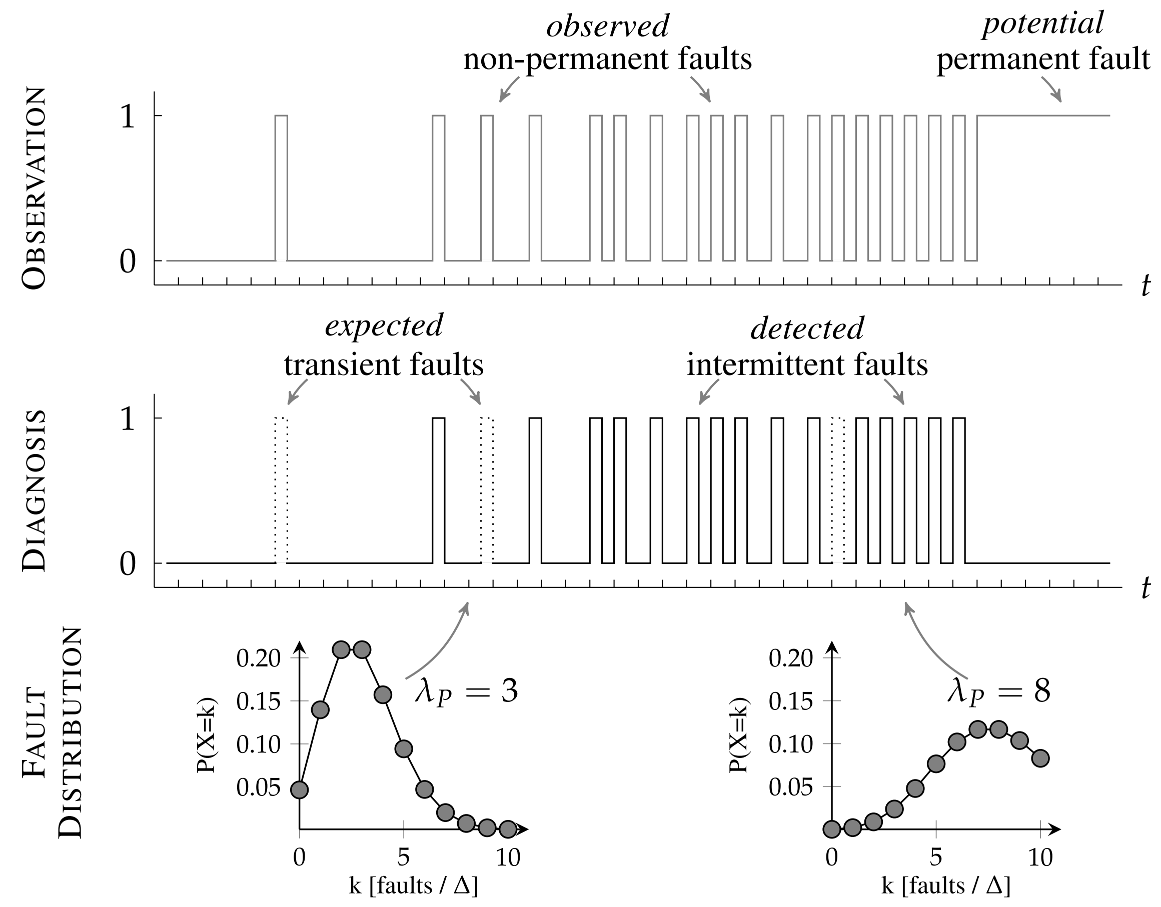


Figure 2.3: *Illustrative example for the occurrence of of observed and detected non-permanent faults on a resource. In the beginning the operation is affected by transient faults only. Then, gradually more and more intermittent faults occur which finally results in a permanent fault. The probabilistic distribution of the intermittent faults depicts an early ($\lambda_P = 3$) and a late ($\lambda_P = 8$) stage of the fault period.*

reasons not every resource can be equipped with specific plausibility tests to detect intermittent faults. Hence, the presented implicit diagnosis is not only designed to use existing plausibility tests which are part of the regular system operation but also considers the data-dependencies of their preceding tasks, hence, enabling a distributed fault detection.

Diagnosis time remarks. As a matter of principle, the fault diagnosis methods presented in this work are designed to being applied either off-line or during low system utilization. This is, on the one hand, due to the necessary observation time during which the plausibility test outcomes are collected and stored. On the other hand, some of the diagnosis methods may be computationally too expensive to be performed during system runtime. As demonstrated in [23], the degradation of a resource due to a gradually increasing occurrence of intermittent faults can last for days or even weeks before a potential permanent fault turns up. As a consequence, intermittent fault approaches do not necessarily have to satisfy strict runtime requirements, but due to possible large system sizes, their timing and computational costs have to be within a reasonable range with a good scalability.

2.3.2 Expectation Matrix

Locating a potentially faulty resource with the help of a simple quantitative analysis of plausibility test failures might not be successful. On the one hand, as already discussed above, to properly diagnose intermittent faults the unavoidable transient faults have to be considered as well. On the other hand, a distributed system can consist of more resources than available test tasks, as shown in the motivating example in Figure 2.1, with three resources but only two tests. As a consequence, a potentially faulty resource not only needs to be associated with test failure rates of its *own* plausibility tests but also of all other tests in the distributed system which, due to task dependencies, may be influenced by faults on that particular resource. For this, the presented fault diagnosis uses a so-called *expectation matrix* which connects all system resources with all plausibility tests and assigns them the corresponding expected test failure rates.

In the following, the determination of the expectation matrix and the test failure rates (i.e., the matrix entires) shall be explained. The parameters, sets and function listed below serve as basis to formally define the expectation matrix and its use for the diagnosis methods in Section 2.4.

$\tau \in \mathcal{T}$	plausibility test task from the set of all available plausibility tests $\mathcal{T}$
$r \in R$	system resource from the set of all resources R to be considered for fault diagnosis
$t \in T$	tasks from the set of all tasks T to be considered for fault diagnosis
$T_r \subseteq T$	subset of tasks executed on a particular resource $r \in R$
$T_\tau^{pred} \subseteq T$	subset of data-dependent predecessor tasks of a test task τ

$e_t \in \mathbb{R}$	specified execution time of a task $t \in T$
$h_t \in \mathbb{R}$	specified execution period of a task $t \in T$
$\lambda_P^r \in \mathbb{R}$	expected value of a Poisson-distrubuted fault rate on resource $r \in R$
$\lambda : \mathcal{T} \times R \to \mathbb{R}$	function describing the frequency of a test $\tau \in \mathcal{T}$ failing due to faults in $r \in R$

We regard the fault rate (i.e., number of faults per time interval) for a specific resource as being subject to the Poisson distribution with a known average value (see Figure 2.3). This is defined in Equation (2.1), where the fault rate of a resource r follows an independent random variable X_r of the Poisson distribution with an expected value λ_P^r, where e represents the Euler's number and k is a positive integer value.

$$\text{Poisson}(X_r = k) = \frac{(\lambda_P^r)^k \cdot e^{-\lambda_P^r}}{k!}, \text{ with } k \in \mathbb{N}_0^+ \text{ and } \lambda_P^r > 0 \tag{2.1}$$

The expected value is strongly dependent on the resource's susceptibility to transient faults and can vary among the different sorts of resources and components within a distributed system. As transient faults can be influenced by many factors, such as radiation or electromagnetic interference, the determination of the expected values is not trivial and must be investigated experimentally or gleaned from the manufacturer's hardware description. In the scope of this work λ_P^r shall be assumed to be known.

Thereby, for each test $\tau \in \mathcal{T}$ and each resource $r \in R$ we can calculate a $\lambda(\tau, r)$ which represents the number of expected test failures per time interval caused by the respective resource. $\lambda(\tau, r)$ is defined in Equation (2.2), with e_t describing the average execution time of a periodic task $t \in T_r$ and h_τ describing its period. Here, T_τ^{pred} represents the set of predecessor tasks of a test τ within a task tree or function.

$$\lambda(\tau, r) = \sum_{t \in T_r \cap T_\tau^{pred}} \frac{e_t}{h_t} \cdot \lambda_P^r \tag{2.2}$$

As the equation indicates, $\lambda(\tau, r)$ proportionally includes the effect of any task executed on the corresponding resource in which an error would lead to a failure of the test τ. That can be the test task itself, or any of its predecessors. In the case that a resource has no influence on a particular test the corresponding failure rate is null ($\lambda(\tau, r)$ = 0). The allocation of each test to the resources which can cause its failure leads to a test expectation matrix Λ which comprises all $\lambda(\tau, r)$ values as its elements. This is formally defined in Equation (2.3).
$\forall \tau \in \mathcal{T}, r \in R :$

$$\Lambda = \big(\lambda(\tau, r)\big)_{\tau, r} \tag{2.3}$$

Here, each row of the matrix contains the resulting expected test failure rates of one single test caused by each resource. Correspondingly, each column of the matrix is made of the failure rates of all tests a single resource can influence. Again, as already mentioned above, the test tasks do not necessarily have to reside on the resources causing their failure. To better illustrate this principle, Figure 2.4 shows an example system specification with the corresponding expectation matrix.

The specification comprises a process graph G_P defining the application which consists of two functions f_x and f_y with three interdependent tasks $t_{x/y}$ and one test task $\tau_{x/y}$ each. The architecture graph G_R connects five resources r_1 - r_5 to a central bus and the mapping E_M associates each task with one resource. The expectation matrix depicted in the lower part of the figure represents the expected failure rates (i.e., transient faults under normal operation) for the two tests depending on the regarded resource. In the context of this work, we consider only a single faulty resource at a time, which is a common assumption during automotive architecture design [77]. This limitation is also briefly discussed in Section 2.7. Now, assuming that resource r_3 is affected by intermittent faults, the observation of the test failure rates will deviate from the original expected values in the third column (see shaded area) enable an implicit fault diagnosis as described in Section 2.4.

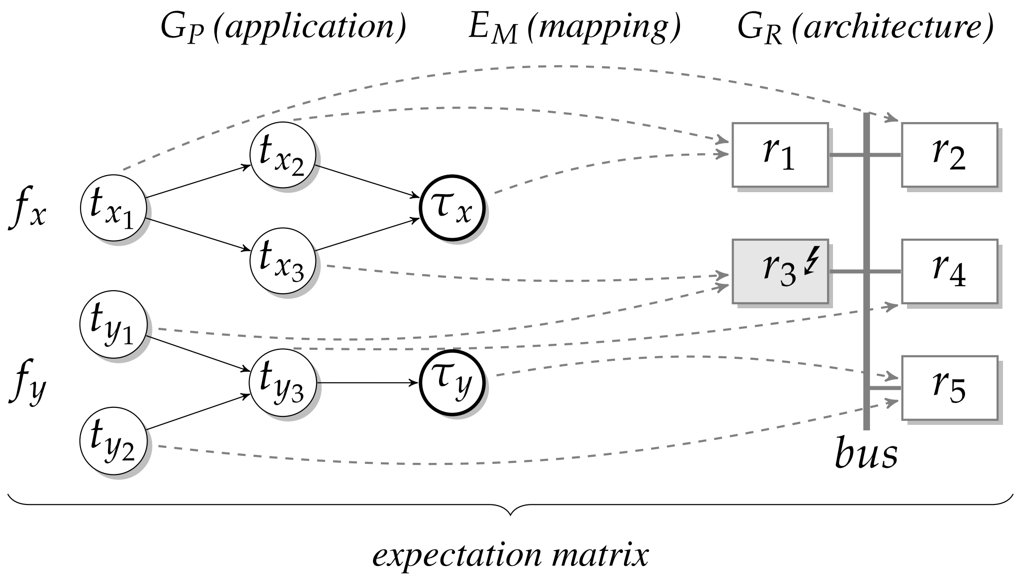
2.3.3 Diagnosability Analysis

An important prerequisite for an efficient and successful detection of faulty resources is an optimal placement of tests in terms of both a minimal system utilization resulting from their execution and an unambiguous decision whether and which resource shows an increased fault rate. Currently, the presented diagnosis follows an implicit approach where the observed tests are regarded to be part of the specified system. Consequently, in this paragraph only the latter aspect shall be discussed, namely, how to ensure that the given distribution of test tasks leads to an unambiguous identification of the affected resource.

Given the presumption that only one resource can fail within a considered time interval, a mutual comparison of the tests associated with specific resources enables a conclusion about the system's diagnosability, i.e., its ability to diagnose a faulty resource. For this, the so-called *cosine similarity CosSim* is used which compares, to which extent two vectors are varying among themselves. More precisely, the cosine similarity represents the cosine of the angle θ between two vectors and can be derived from the Euclidean dot product of these vectors, as defined in Equation (2.4).

$$CosSim_{r_i,r_j} = cos(\theta) = \frac{\mathbf{v}_{r_i} \cdot \mathbf{v}_{r_j}}{\|\mathbf{v}_{r_i}\| \cdot \|\mathbf{v}_{r_j}\|} \Rightarrow \theta = \angle(\mathbf{v}_{r_i}, \mathbf{v}_{r_j}) \tag{2.4}$$

As a consequence, the *CosSim* between two arbitrary column vectors in the Λ matrix, $\mathbf{v}_{r_i}$ and $\mathbf{v}_{r_j}$, will result in a number proportionally approximating 1, the smaller the angle θ between these two vectors is and, hence, the more similar these two vectors are. The more the value deviates from 1 the greater the angle between two vectors and, hence, the clearer the difference between a potentially faulty resource and a healthy one. Therefore, for two opposite vectors



$$\Lambda^{|\mathcal{T}|\times|R|} = \begin{pmatrix} \lambda(\tau_x, r_1) & \lambda(\tau_x, r_2) & \lambda(\tau_x, r_3) & 0 & 0 \\ 0 & 0 & \lambda(\tau_y, r_3) & \lambda(\tau_y, r_4) & \lambda(\tau_y, r_5) \end{pmatrix}$$

Figure 2.4: *Illustration of a graph-based systems specification and the corresponding expectation matrix. It consists of an application G_P with two functions (f_x, f_y), an architecture G_R with five resources (r_1 - r_5) and the mapping E_M between tasks and resources ($\dashrightarrow$). The expectation matrix derived from the specification highlights the affected $\lambda(\tau, r)$ values of the faulty resource r_3.*

the $CosSim$ will result in -1. Equation (2.5) illustrates this correlation, where, due to the inaccuracy induced by the expected transient faults, a threshold value ϵ_{ana} is used to define the maximum acceptable deviation of the cosines from 1.

$\forall i, j \in \{1, \cdots, |R|\}$:

$$1 - CosSim_{r_i,r_j} \begin{cases} \leq \epsilon_{ana} \Rightarrow r_i \approx r_j \text{ ("weak diagnosability")} \\ > \epsilon_{ana} \Rightarrow r_i \neq r_j \text{ ("strong diagnosability")} \end{cases} \tag{2.5}$$

Combining the outcome of a cosine similarity analysis allows us to define the diagnosability of a system specification as follows:

> *"The allocation of plausibility tests to resources is called fully diagnosable, if the resulting $CosSim$ between each to vectors $\mathbf{v}_{r_i}, \mathbf{v}_{r_j}$ in the expectation matrix Λ exceeds a predefined threshold value."*

A trivial representation of a fully diagnosable system would be an architecture where each single resource contains one plausibility test without considering any data-dependencies with predecessor tasks. Such a configuration would results in a diagonal expectation matrix $\Lambda = diag(\lambda(r_1, \tau_1), \lambda(r_2, \tau_2), \cdots, \lambda(r_{|R|}, \tau_{|\mathcal{T}|}))$ with all vectors being orthogonal to each other and, thus, leading to a minimal cosine similarity. One of the goals of this work is to beat that reference solution by regarding a reduced number of required tests.

2.4 Methodology

This section presents formal definitions and descriptions of four different methods for the evaluation of plausibility test outcomes leading to an implicit and decentralized diagnosis of intermittent faults in distributed systems.

2.4.1 Diagnosis Approaches

The presented fault diagnosis is designed to indicate resources in a distributed system experiencing an increased occurrence of intermittent faults. These resources can be the ECUs in an automotive E/E architecture, but also the single cores in a many-core processor, as will be discussed in Section 2.6. As a matter of principle, the fault diagnosis problem can be abstracted to an analysis of the quantitative relation between observed and expected test failures. Thus, based on the expectation matrix Λ introduced in Section 2.3.2, four different approaches towards the intermittent fault diagnosis shall be described and evaluated with respect to their correctness and performance. These approaches are summarized in Table 2.1 listing both the diagnosis methods as well as the computational principle underlying their implementations. In addition to the the parameters introduced in Section 2.3, the following parameters will be used for the definitions of the diagnosis methods.

Table 2.1: *List of the four proposed methods for the intermittent fault diagnosis, additionally specifying the general diagnosis principle.*

Identifier	*Diagnosis method*	*Diagnosis principle*
Method I	Cosine Similarity	Vector-based
Method II	Singular Value Decomposition	Vector-based
Method III	Confidence Interval	ILP / Probabilistic
Method IV	Pearson's χ^2-Test	ILP / Probabilistic

$\Delta_\tau \in \mathbb{R}$	time interval for a test failure observation
$O_\tau \in \mathbb{R}$	observed failures of a test $\tau \in \mathcal{T}$in Δ_τ
$E_\tau \in \mathbb{R}$	expected failures of a test $\tau \in \mathcal{T}$ in Δ_τ
$\epsilon_{cos} \in \mathbb{R}$	deviation threshold for the determination of the cosine similarity
$\epsilon_{svd} \in \mathbb{R}$	deviation threshold for the determination of the Singular Value Decomposition (SVD)
$\lambda_{lo,O_\tau} \in \mathbb{R}$	lower test failure limit of the confidence interval for an observation O_τ
$\lambda_{hi,O_\tau} \in \mathbb{R}$	upper test failure limit of the confidence interval for an observation O_τ
$\mathbf{x}_r \in \mathbb{R}$	stress variable which weights the expected test failure rate $\lambda(\tau, r)$
$\mathbf{y}_r \in \{0, 1\}$	switch variable for the optimization methods
$\delta_r \in \mathbb{R}$	decision threshold for the switch variable $\mathbf{y}_r$

The sets O_τ and E_τ are representing the observed and the expected number of test failures $\tau \in \mathcal{T}$, respectively, that occur in the observing time interval Δ_τ. Basically, it can be assumed that at least one resource $r \in R$ is faulty when the number of observed failures of a test is significantly higher than the number of its expected failures caused by the transient faults on all system resources within a specific time interval Δ_τ, This is defined in Equation (2.6), where the summation is done over all system resources as those not contributing to a particular test τ have a zero value.

$$O_\tau \gg \Delta_\tau \cdot \sum_{r \in R} \lambda(\tau, r) \tag{2.6}$$

Fault rate remarks. The magnitude of this inequality (2.6) is clearly depending on how often faults on a resource will cause a particular test to fail as well as the observation time Δ_τ for this test. In this thesis, for the evaluation of synthetic test cases in Section 2.5, a stress factor of approximately 10^3 between healthy and unhealthy resources is used. This static range has been chosen in order to evaluate and verify the general suitability and effectiveness of the four detection methods. However, it is likely that in reality, the rate of intermittent faults caused, e.g., by aging, would increase over time and also occur in bursts. This behavior is not specifically considered in the experimental results.

2.4.2 Method I - Cosine Similarity

The diagnosis Methods I and II are based on a vector analysis for which they mainly use the columns from the expectation matrix Λ. As the name already indicates, the first diagnosis method uses cosine similarity and, hence, a similar mathematical approach as the diagnosability analysis in Section 2.3.3. However, as they concern two different stages of the intermittent fault diagnosis, the approaches are independent and defined separately.

Each matrix column represents an expectation vector $\mathbf{v}_{r_i}$ which is assigned to one particular resource and contains the corresponding utilization-dependent expected failure rates for all regarded plausibility tests, as defined in Equation (2.7).
$\forall i \in \{1, \cdots, |R|\}$:

$$\mathbf{v}_{r_i} = [\lambda(\tau_1, r_i), \lambda(\tau_2, r_i), \cdots, \lambda(\tau_{|\mathcal{T}|}, r_i)]^{\mathrm{T}} \tag{2.7}$$

Furthermore, an observation vector $\mathbf{v}_{O_\mathcal{T}}$ is used, containing the number of observed failures of each plausibility test. As defined in Equation (2.8) this vector has the same dimension as the expectation vectors, namely $|\mathcal{T}|$.

$$\mathbf{v}_{O_\mathcal{T}} = [O_{\tau_1}, O_{\tau_2}, \cdots, O_{\tau_{|\mathcal{T}|}}]^{\mathrm{T}} \tag{2.8}$$

Is is important to mention that the expectation vectors $\mathbf{v}_{r_i}$ do not directly represent the actual expected numbers of failed tests E_τ, because they are not scaled with the observation time interval Δ_τ. However, this has no influence on the final diagnosis results, as the calculation of the cosine similarity is independent of the magnitude of the vectors. This becomes evident when regarding the denominator in Equation (2.9) which defines the cosine similarity between the expectation and observation vector.

$$CosSim_{r_i, O_\mathcal{T}} = \frac{\mathbf{v}_{r_i} \cdot \mathbf{v}_{O_\mathcal{T}}}{\|\mathbf{v}_{r_i}\| \cdot \|\mathbf{v}_{O_\mathcal{T}}\|} \tag{2.9}$$

The analysis of the similarity between the observation vector and each expectation vector can reveal whether the regarded resource is also the faulty one. If $CosSim$ is close to 1, the certainty is high that the (possibly increased) observed test failures stem from the expectation vector of the regarded resource. This is, because it indicates that the relative distribution of the test failures among all considered tests correlates with the expected failure rates associated with this resource. Correspondingly, two vectors with a much lower correlation would result in a cosine similarity value closer to 0 or even -1. In this context, due to the unavoidable transient faults, the correct diagnosis if a resource is affected by intermittent faults must include a threshold value. As shown in Equation (2.10), we use ϵ_{cos} which defines the maximum acceptable deviation of $CosSim$.

$\forall i \in \{1, \cdots, |R|\}$:

$$1 - CosSim_{r_i, O_{\mathcal{T}}} \begin{cases} \leq \epsilon_{cos} \Rightarrow \text{ observation } O_{\mathcal{T}} \text{ originates in } r_i \\ > \epsilon_{cos} \Rightarrow \text{ observation } O_{\mathcal{T}} \text{ originates not in } r_i \end{cases} \tag{2.10}$$

2.4.3 Method II - Singular Value Decomposition

By contrast, the second vector-based method uses the linear dependence of two vectors in a broader sense rather then solely rely on their angular similarity. For this, the observation and the expectation vectors defined in Equations (2.8) and (2.7) are joint together to build the sub-matrices $\Lambda_{r_i} = (\mathbf{v}_{O_{\mathcal{T}}}, \mathbf{v}_{r_i})$ which are then used for a Singular Value Decomposition (SVD).

For real numbers, linear algebra defines a SVD as the factorization of a matrix into one diagonal matrix Σ and two orthogonal matrices U and V^{T}. Geometrically, this can be interpreted as one scaling and two rotations of the corresponding matrix[4]. Now, the idea is to use the rank of the scaling matrix Σ to determine if the observation and expectation vectors are linearly independent. As Σ only contributes to the expansion or shrinking of the corresponding matrix without changing its orientation, it forms a diagonal matrix whose rank calculation might be more efficient and precise than the cosine similarity determination introduced before.

Applying the SVD to the sub-matrices Λ_{r_i} results in a product of the orthogonal matrix U, the diagonal matrix Σ and the transposed orthogonal matrix V^{T}, which is defined in Equation (2.11).

$\forall i \in \{1, \cdots, |R|\}$:

$$\Lambda_{r_i}{}^{|\mathcal{T}| \times 2} = U \Sigma V^{\mathrm{T}}, \text{ with } \Sigma^{|\mathcal{T}| \times 2} = \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \\ & \ddots \\ 0 & 0 \end{bmatrix} \text{ and } |\mathcal{T}| \geq 2 \tag{2.11}$$

Here, the matrices U and V, whose columns contain the eigenvectors of $\Lambda_{r_i} \Lambda_{r_i}^{\mathrm{T}}$ and $\Lambda_{r_i}^{\mathrm{T}} \Lambda_{r_i}$, respectively, are not relevant for the diagnosis and shall receive no consideration during the further detection procedure. Σ has the same dimension as Λ_{r_i} (i.e., $|\mathcal{T}| \times 2$) and consists of the singular values σ_1 and σ_2 on its diagonal and of zeros otherwise.

An observation $O_{\mathcal{T}}$ is considered being caused by the resource r_i, if the rank of the corresponding diagonal matrix Σ is less than 2, as defined in Equation (2.12).

$$rank(\Sigma) \begin{cases} < 2 \Rightarrow O_{\mathcal{T}} \text{ from } r_i, & \text{if } \exists \sigma \leq \epsilon_{svd} \\ = 2 \Rightarrow O_{\mathcal{T}} \text{ not from } r_i, & \text{if } \forall \sigma > \epsilon_{svd} \end{cases} \tag{2.12}$$

[4]Note, that T represents the transpose of a matrix and should not be confused with T or $\mathcal{T}$ which represent sets of all tasks and plausibility test tasks, respectively.

Similarly to the previous method and the diagnosability analysis, a threshold value ϵ_{svd} is used to adjust the diagnosis regarding potential test failures caused by transient faults. It holds, that to associate the observation O_τ with a resource, aside from the proper rank, the matrix Σ must also contain at least one singular value σ_i which is less than or equal to the threshold. By contrast, the case when both singular values are greater than ϵ_{svd} is trivial, as it already implies a rank of 2.

In general, both the performance and correctness of Methods I and II are depending on the proper choice of the threshold levels. On the one hand, a too large value might be not sensitive enough and could cause *false positive* results. On the other hand, a too small value might not consider the noise caused by transient faults which would lead to *false negative* results. Regarding the experimental results, numerous test runs proved $\epsilon_{cos} \sim 10^{-3}$ and $\epsilon_{svd} \sim 10^{-5}$ as good ranges for the analyzed system sizes.

2.4.4 Method III - Confidence Interval

A fundamentally different detection approach can be applied when we take into account the probabilistic nature of fault occurrences. To mirror the stochastic uncertainty of the expected transient faults occurring on a resource, we can create a confidence interval around the expected test failures E_τ inside which an observation is considered to comply with the corresponding λ-values in the expectation matrix. Assuming, that the fault events are following a Poisson distribution and that during normal operation all observed test failures O_τ lie within three standard deviations from the expected value E_τ, we can define the interval according to the 3σ-rule. This is defined in Equation (2.13), where σ represents the standard deviation and μ the mean of the Poisson-distributed random variable X_τ.

$$P_\tau(\mu - 3\sigma \leq X_\tau \leq \mu + 3\sigma) \approx 0.9973 \tag{2.13}$$

Given the Cumulative Distribution Function (CDF) of X_τ, we are interested in the the lowest (λ_{lo,O_τ}) and highest (λ_{hi,O_τ}) mean test failure rate which can still be regarded as compliant with the 3σ-rule. These rates can be determined by gradually approximating the corresponding CDFs $F_\lambda(X_\tau \leq O_\tau)$ towards the boundaries of the interval defined in Equation (2.13). Formally, this process can be described by an optimization problem searching for minimal observed mean test failure rates (compare Equations (2.14a) and (2.15a)) complying with the constrains given by Equations (2.14b) and (2.15b), respectively.

$\forall \tau \in \mathcal{T}$:

$$\underset{\lambda_{lo,O_\tau} \in \mathbb{R}_0^+}{minimize} \ \lambda_{lo,O_\tau} \tag{2.14a}$$

subject to:

$$F_{\lambda_{lo,O_\tau}}(X_\tau \leq O_\tau) \leq \frac{P_\tau + 1}{2} \tag{2.14b}$$

$\forall \tau \in \mathcal{T}:$

$$\underset{\lambda_{hi,O_\tau} \in \mathbb{R}_0^+}{\mathit{maximize}} \ \lambda_{hi,O_\tau} \tag{2.15a}$$

subject to:

$$F_{\lambda_{hi,O_\tau}}(X_\tau \leq O_\tau) \geq \frac{1 - P_\tau}{2} \tag{2.15b}$$

The result is a confidence interval $[\lambda_{lo,O_\tau}, \lambda_{hi,O_\tau}]$ which determines the lower and upper limits for the expected test failures E_τ during an observation time Δ_τ in a faultless system. Now, in order to identify a potentially faulty resource which would cause a shift of the corresponding confidence interval, we can formulate an ILP-based optimization problem as shown in Equations (2.16a) through (2.16e)[5].

$$\underset{\mathbf{y}_r \in \{0,1\}}{\mathit{minimize}} \quad \sum_{r \in R} \mathbf{y}_r \tag{2.16a}$$

subject to:

$$\forall \tau \in \mathcal{T}: \quad \lambda_{lo,O_\tau} \leq \frac{\boldsymbol{E_\tau}}{\Delta_\tau} \leq \lambda_{hi,O_\tau} \tag{2.16b}$$

$$\forall \tau \in \mathcal{T}: \quad \boldsymbol{E_\tau} = \sum_{r \in R} \mathbf{x}_r \cdot \lambda(\tau, r) \cdot \Delta_\tau \tag{2.16c}$$

$$\forall r \in R: \quad \mathbf{x}_r \geq \delta_r \cdot \mathbf{y}_r \tag{2.16d}$$

$$\forall r \in R: \quad \mathbf{x}_r \leq \delta_r + 10^{10} \cdot \mathbf{y}_r \tag{2.16e}$$

The aforementioned relation between E_τ and the confidence interval is defined as Constraint (2.16b). The expected test failures E_τ are calculated as sum of weighted λ-values over all resources, where the weight is defined by the resource-dependent stress variable $\mathbf{x}_r$ (Constraint (2.16c)). In this context, we use $\mathbf{y}_r$ as a switch variable to decide whether $\mathbf{x}_r$ is significantly exceeding ($\mathbf{y}_r = 1$) or still within an acceptable range ($\mathbf{y}_r = 0$) in order to keep E_τ inside the confidence interval (Constraints (2.16d) and (2.16e)). The threshold for this decision is set by the parameter δ_r. Applying these constraints, the objective function (2.16a) searches for cases where a minimal number of stressed resources (ideally a single one) is responsible for the observed test failures. To rule out additional resources being responsible for the failed

[5]Note that in this thesis, bold characters in an ILP denote variables alterable by the solver while normal characters represent constant and precalculated values, respectively.

tests, the optimization algorithm must be started a second time with the first solution excluded from the test. The correctness of the first run can only be confirmed if the latter test run finished without a solution.

Although E_τ and O_τ intuitively depict countable integers, the ILP-based optimization algorithm considers both as values from the set of positive rational numbers $\mathbb{R}_0^+$. Similarly, the variables $\lambda(\tau, r)$ and $\mathbf{x}_r$ are from $\mathbb{R}_0^+$, whereas the switch parameter $\mathbf{y}_r$ is defined as a binary variable with the two values 0 and 1. Finally, the threshold δ_r is crucial for the goodness of the fault diagnosis, such that several different values have been applied during the experimental results in Section 2.5.

2.4.5 Method IV - Pearson's χ^2-Test

As a fourth diagnosis method, an approach based on a statistical hypothesis test shall be investigated. To determine whether the observed test failures correspond with the expected failure rate, the *Pearson's χ^2-Test* shall be used as a statistical model to describe the goodness of fit. Within this test, χ^2 describes the value of the test statistic representing the normalized sum of squared differences between observed and expected test failures, as shown in Equation (2.17).

$$\chi^2 = \sum_{\tau \in \mathcal{T}} \frac{(O_\tau - E_\tau)^2}{E_\tau} \tag{2.17}$$

At the same time, Equation (2.18) defines a null hypothesis H_0 for the χ^2-Test which indicates that all observed test failures O_τ result from the expected test failures E_τ.

$$H_0 : E_\tau \rightarrow O_\tau \tag{2.18}$$

This hypothesis shall be accepted if the p-value of the χ^2-Test is higher than a predefined significance level and rejected otherwise. The p-value describes that the probability of a corresponding test statistic is equal as or higher than the actual observation, given that the null hypothesis is true. We chose the significance level of 0.05, as it is commonly used in science [126]. Consequently, the p-value can be described as in Equation (2.19), where $F(\chi^2, |\mathcal{T}| - 1)$ represents the CDF of the χ^2 distribution and $|\mathcal{T}| - 1$ defines the number of degrees of freedom.

$$1 - F(\chi^2, |\mathcal{T}| - 1) \geq 0.05 \tag{2.19}$$

Based on the definitions above, we can formulate an ILP-based optimization problem to find solutions for the χ^2-Test. Similar to the approach in Method III, we are trying to minimize the number of cases, where the expected test failure rates associated with potentially faulty resources need to be weighted with a stress variable $\mathbf{x}_r$ exceeding a predefined limit δ_r. The optimization problem is defined in Equations (2.20a) through (2.20g).

$$\underset{\mathbf{y}_r \in \{0,1\}}{\mathit{minimize}} \quad \sum_{r \in R} \mathbf{y}_r \tag{2.20a}$$

subject to:

$$\chi^2 = \sum_{\tau \in \mathcal{T}} O_\tau{}^2 \cdot \boldsymbol{R_\tau} - 2 \cdot O_\tau + \boldsymbol{E_\tau} \tag{2.20b}$$

$$\chi^2 \leq F^{-1}(1 - 0.05) \tag{2.20c}$$

$$\forall \tau \in \mathcal{T}: \quad \boldsymbol{E_\tau} \cdot \boldsymbol{R_\tau} = 1 \tag{2.20d}$$

$$\forall \tau \in \mathcal{T}: \quad \boldsymbol{E_\tau} = \sum_{r \in R} \mathbf{x}_r \cdot \lambda(\tau, r) \cdot \Delta_\tau \tag{2.20e}$$

$$\forall r \in R: \quad \mathbf{x}_r \geq \delta_r \cdot \mathbf{y}_r \tag{2.20f}$$

$$\forall r \in R: \quad \mathbf{x}_r \leq \delta_r + 10^{10} \cdot \mathbf{y}_r \tag{2.20g}$$

In contrast to the confidence interval (see Constraint (2.16b)), the correlation between the observed and the expected test failures are defined by the χ^2-Test. The Constraint (2.20b) refers to the test statistic from Equation (2.17) where the squared difference is resolved and the quotient $\frac{1}{E_\tau}$ is substituted with R_τ. The comparison of the test statistic with the χ^2 distribution is represented in Constraint (2.20c) where the inequality in Equation (2.19) is solved for χ^2 with F^{-1} describing the corresponding inverse cumulative probability. To define the reciprocal of E_τ, Constraint (2.20d) consists of the multiplication of two variables which leads to a non-linear problem in the ILP. In order to still be able to apply a linear solver, the variable R_τ must be fragmented into a sum of bit-variables representing the coefficients of the corresponding powers of two. This enables an addition-based calculation of $\frac{1}{E_\tau}$. The last three Constraints (2.20e) to (2.20g) are identical to the Constraints (2.16c) to (2.16e) in Method III and are serving the same purpose. Finally, similar to the confidence interval approach, the χ^2-Test has to be executed twice to rule out alternative solutions.

2.5 Diagnosis Evaluation

In this section, the four intermittent fault diagnosis methods defined in Section 2.4 shall be analyzed and evaluated with the aim to verify their general feasibility as well as compare their efficiency and performance.

2.5.1 Test Case Generation

The test cases used for the following experimental results are inspired by current automotive E/E architectures. As already introduced in Figure 2.4, they use graph-based representations of the system specifications (i.e., applications, consisting of data-dependent tasks and architectures consisting of interconnected resources) and are implemented based on a previously presented model [86]. In the following, the generation of test cases for the experiments shall be briefly explained.

2.5.1.1 Generating Expected Faults

Expected test failure rates for all resources are represented as λ-values of a Poisson distribution, as defined in Section 2.3.1. Each resource is initially given a randomly calculated λ-value which represents its utilization-independent susceptibility for transient faults. The mapping of an application to an architecture also assigns the test tasks τ to particular resources. For the experiments, test tasks are always depicted by the final element of a task chain. Based on this, the Λ-matrix for a specific test case can be constructed.

2.5.1.2 Generating Observed Faults

The fault observations are identified by failures of plausibility tests within a predefined period of time. As the fault occurrence on a resource is assumed to follow the Poisson-distribution, the inter-arrival times are exponentially distributed with a mean represented by a (possibly stressed) λ-value. To simulate a resource with an increased intermittent fault rate, the corresponding λ-values are multiplied with a stress factor in the range of 10^3. In each test case one such resource is randomly selected. Here, it must be ensured that all tasks which are a predecessor of the plausibility tests are affected by the faults with a frequency related to their utilization on the corresponding resources. Based on that, the numbers of observed test failures O_τ are simulated for each test case.

Altogether, 240 specifications with realistic sizes and topologies have been generated. These specifications comprise various system definitions with 10-100 resources and 3, 5 and 10 tasks executed on each resource, respectively. The ratio of plausibility tests to resources varies between $\frac{1}{4}$ and 1. The four fault diagnosis methods presented in Section 2.4 are applied to each of the 240 specifications in different variants described by three sets of the parameters resulting in a total of 2880 distinct test runs. The parameters are ϵ_{cos} and ϵ_{svd} in case of the vector-based approaches (Methods I and II) as well as δ_r in case of the ILP-based approaches (Methods III and IV). The parameter values are listed in Table 2.2. As already mentioned, the choice of these values can have a significant impact on the diagnosis results. Their selection used for the following evaluation is based on several preliminary test-runs in order to rule out values which would provide little or no insight into the intermittent fault diagnosis. In other words, the parameters shall lead to a better assessment of the presented methods rather than provide optimal thresholds to be applied on real architectures.

Table 2.2: *List of three different sets of parameter values for the proposed diagnosis methods.*

Method	Parameter	Set 1	Set 2	Set 3
I	ϵ_{cos}	$3.0 \cdot 10^{-3}$	$6.0 \cdot 10^{-3}$	$1.2 \cdot 10^{-2}$
II	ϵ_{svd}	$4.3 \cdot 10^{-5}$	$8.50 \cdot 10^{-5}$	$1.70 \cdot 10^{-4}$
III & IV	δ_r	1.1	1.5	1000

For each test case, the observation time varies, ranging from 10 to 1000 times of an average task period, as shown in Equation (2.21) (see also discussion in Section 2.7).

$$\Delta_\tau = 10^n \cdot h_t, \text{ with } n \in \{1,2,3\} \tag{2.21}$$

Finally, in order to additionally indicate false positive results, we assume that only 50 % of all test cases (i.e., 120) actually contain a faulty resource.

2.5.2 Experimental Results

All experiments were carried out on an Intel Core i5 with 2.6 GHz and 8GB RAM. For the methods that required an ILP solver, GUROBI version 5.0 [39] was used.

2.5.2.1 Correctness

First, the correctness of the introduced intermittent fault diagnosis shall be analyzed, which gives evidence of the overall feasibility of our approach. For this, the chart in Figure 2.5 shows for which percentage of the test cases the system has been correctly diagnosed as faulty and healthy, respectively. Here, as denoted by the labels on the y-axis, results from each of the four methods are further divided into the parameter sets, such that each bar represents a particular diagnosis method with one specific parameter value. Furthermore, as each outcome can be assigned to one of four different results, the bars are subdivided into four categories, namely, *correct* (filled), *false positive* (north-west lines), *false negative* (north-east lines) and *timeout* (empty), which correspond to a number of test cases between 0 and 240 for each single bar (illustrated in percent on the x-axis). *Correct* results include both the exact detection of a faulty resource and the proper conclusion that a system is not affected by intermittent faults. In contrast to this, a *false positive* result indicates that an actually healthy resource has been diagnosed as faulty. Moreover, this category also contains results where several resources are considered faulty, which can also include the one initially stressed. Finally, *false negative* results are cases where the stressed resource is not detected and the system is considered fault-free. The timeout for each test case was set to 60 seconds, which means that a test run was aborted if no solution had been found within this period. Inside the chart, these cases are illustrated by an empty bar.

To begin with, it can be stated that overall the results in Figure 2.5 give evidence of the feasibility of the presented intermittent fault diagnosis approach. Comparing the different methods

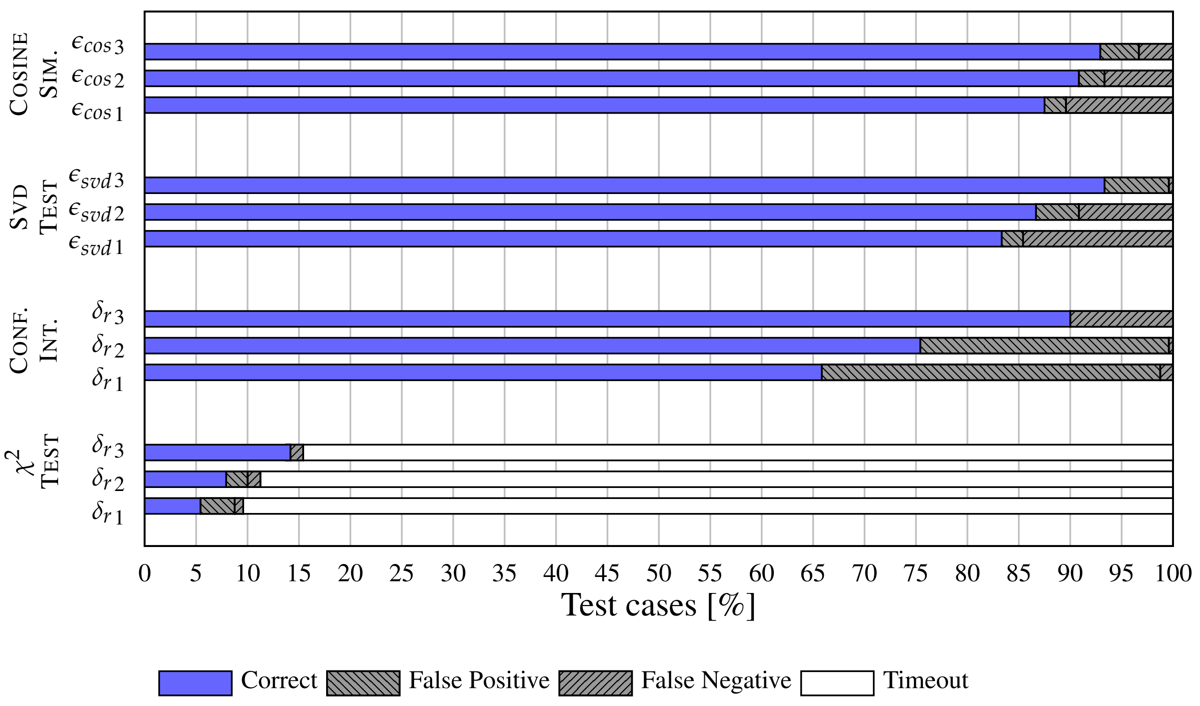


Figure 2.5: *Bar chart depicting the amount of correctly diagnosed test cases. Each bar represents the result for one fault diagnosis method with one specific parameter value. False positive and false negative results refer to test cases which could not be solved correctly. The timeout for finding a solution for one single test case was set to* 60 *seconds.*

among each other, shows the best results for the *Cosine Similarity* with about 90 % correctly diagnosed cases for all three parameter configurations, however, with a slightly increasing number of false negatives when the threshold ϵ_{cos} becomes smaller. With the overall best result in its upper bar, the *SVD* method shows a similar outcome, although it contains some more false negative test case results for the larger values of ϵ_{svd}. The decrease of false positive detections with lower threshold values can be explained by the narrowed tolerance scope for the comparison between observed and expected vectors. Since at the same time the false negative detections are increasing, which for a safety-critical system might be regarded more severe, the parameter values must be chosen with care. In contrast to this, the *Confidence Interval* method shows a larger variation between the parameters but can achieve similar diagnosis results with the right choice of the decision threshold. Here, however, the correlation is reversed, as a higher δ_r leads to significantly more false negative detections with overall more correct results. This behavior can be explained by the fact that in the ILP-based methods the decision threshold in a certain way reflects the level of the assumed system noise (i.e., the amount of transient faults). For this reason, a similar trend can be seen in the χ^2-*Test* results, where false positive outcomes appear only at low δ_r values.

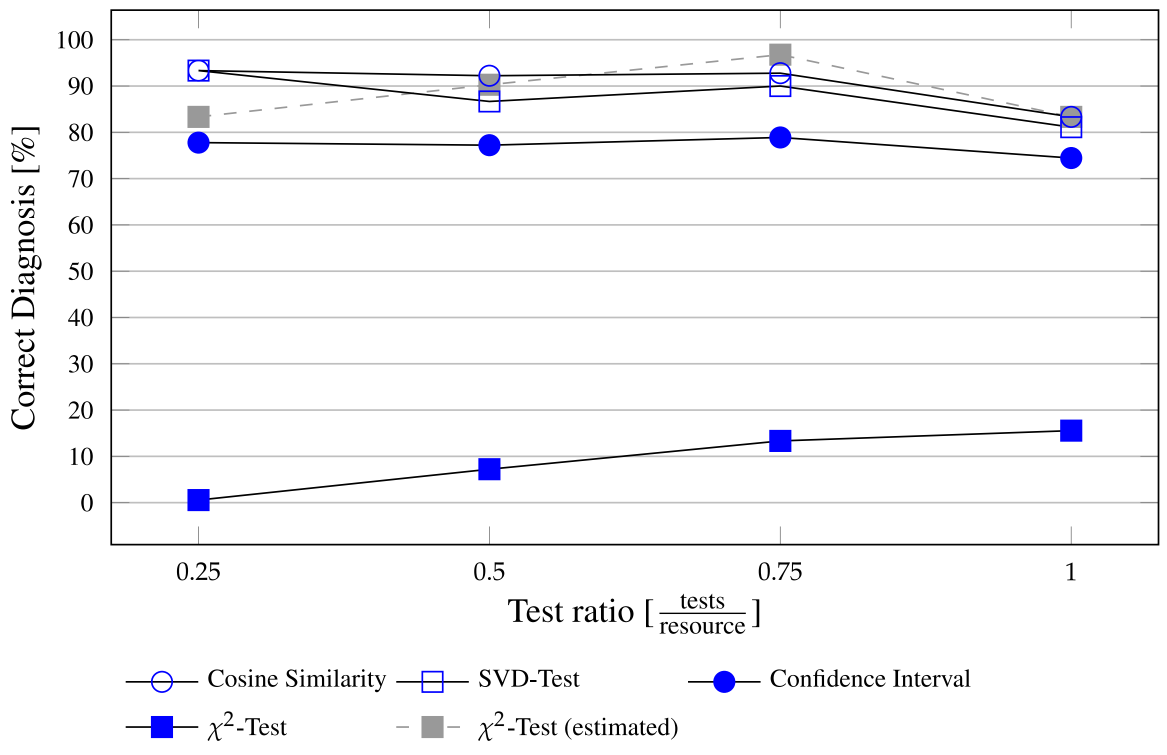


Figure 2.6: *Graph illustrating the relative amount of correctly diagnosed test cases over the corresponding test-to-resource ratios. The graphs combine the results of all regarded diagnosis methods and parameters. The gray plot represents an estimation of the χ^2-Test results without timeouts.*

The evident limitation of the fourth method is the large number of test cases resulting in a time out error, which most probably are caused by the linearization of the quotient term in the ILP, discussed in Section 2.4.5. Nevertheless, aside from the timeouts, the results for the χ^2-*Test* look promising and it is expectable that this method would deliver detection rates comparable to the *Confidence Interval* results or even better, when using an alternative implementation, such as a non-linear optimization.

2.5.2.2 Plausibility Test Distribution

The intermittent fault diagnosis uses a distributed approach where the outcomes of existing plausibility tests located on various resources are evaluated collectively. As a reduction of (possibly redundant) plausibility tests might contribute, on the one hand, to a lower utilization of the system at runtime and, on the other hand, to a faster diagnosis process, it is interesting to investigate the correlation between the average number of tests per resource and the diagnosis efficiency. In this regard, the chart in Figure 2.6 depicts the percentage number of correct test results on the y-axis and marks four points with different test-to-resource ratios on the x-axis.

Here, each diagnosis method includes the results of all three parameter sets. The ratios lie between 0.25 and 1 which means that we split the results in cases where each four resources in a system share one, two, three or four plausibility tests, respectively.

Overall, all methods except for the χ^2-*Test* show a good detectability in a high two-digit percentage range with only small deviations between the different ratios. However, besides the much lower values for the χ^2-*Test* resulting from a high number of timeouts, the progressions of the four curves can also seem unexpected due to two other aspects. Firstly, the differences between the different ratios seem rather small, as one might expect the curves increasing more clearly when higher test-to-resource ratios are used. Secondly, the three upper curves show the worst results on system with one test per resource. A possible explanation for low variation between the four ratios can be, that the here regarded system sizes with maximal 100 resources are not large enough. As will be shown in Section 2.6.3, the expected increase is clearly visible when considering systems roughly larger by one order of magnitude. The decline of the upper curves at the highest ratio, on the other hand, most probably derives from an increased number of false positive detections, when many tests are applied and could be reduced by adapting the corresponding threshold parameter values accordingly. Regarding solely the χ^2-*Test*, a higher number of tests seems to improve the detectability. These results should be interpreted with care, as that correlation may be distorted by the limited number of considered test results caused by the timeout, which is also responsible for the low detection rate. To get an idea how the outcome for the χ^2-*Test* could look like without timeout errors, the gray curve in the graph illustrates estimated results projected on the basis of the available test cases including all analyzed methods. However, it must be noted, that even though the projected curve seems in accordance with the other results, it is only a rough estimation and should not be considered for implementation or further refinement of the χ^2-*Test* method.

Summing up, for the results in Figure 2.6 in can be concluded that the choice of the right parameter values has a greater impact on the fault diagnosis than the number of plausibility tests. It also shows that for system sizes used in the automotive area, our approach works well also for lower test to resource ratios.

2.5.2.3 Runtime

As a third experimental result, the computation times of the four proposed diagnosis methods shall be analyzed. Similar to Figure 2.5, the graphs in Figure 2.7 are divided into the four diagnosis methods as well as the corresponding three parameter values. The x-axis illustrates the runtimes of the computation in seconds where each cross denotes one single test case. Here, the lowest values are in the range of microseconds and the timeout is set to 60 seconds. The numbers in brackets at the χ^2-*Test* results represent test cases which have been aborted due to a timeout error.

Overall, the graph illustrates that the results of the two vector-based diagnosis methods are in a lower millisecond range even for the slowest test cases. Also the results for the *Confidence Interval* method are still below one second and, hence, do not cause severe runtime issues. On

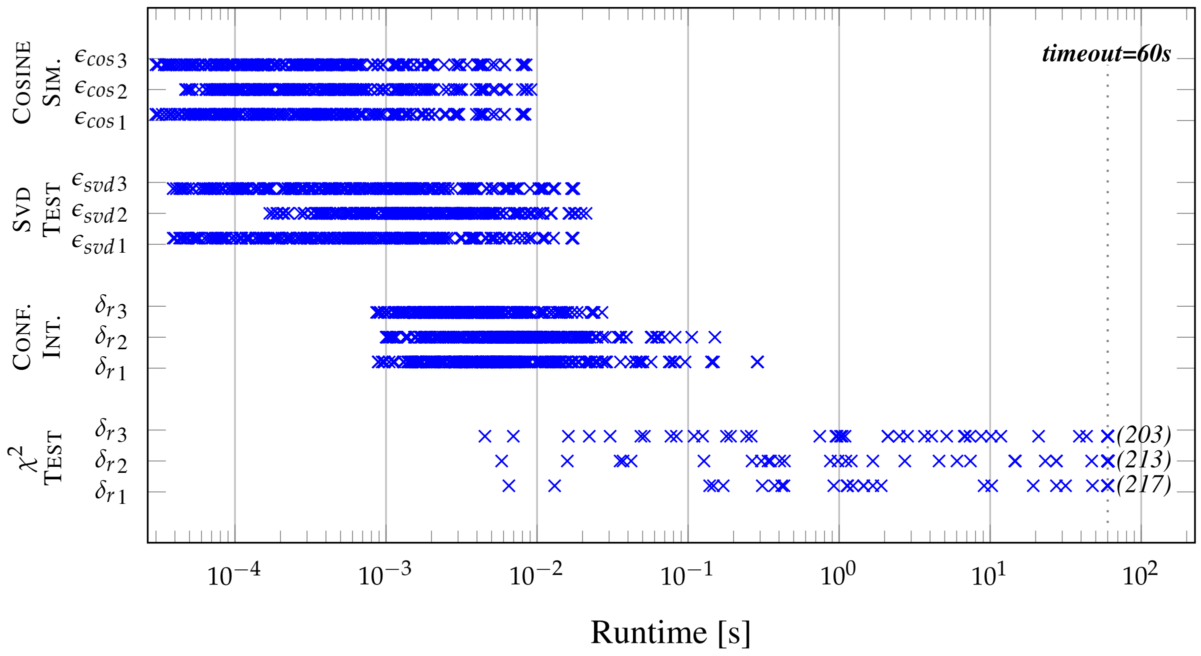


Figure 2.7: *Graph illustrating the computation times of the analyzed test cases with a timeout at 60 seconds. It is divided according to the four detection methods with three different parameters. The numbers of test runs that are aborted due to a timeout are annotated in brackets.*

the contrary, the computation times for the χ^2-*Test* are in sub-second range only for very few test cases, while most of the remaining ones deliver no proper results due to a timeout error. However, as already mentioned before, this outcome could be possibly improved by using a non-linear solver, which would eliminate the additional computationally expensive linearisation of the reciprocal of expected failures E_τ (compare Equation (2.20d)).

Considering the parameter values, it can be seen that there are only small differences in runtimes in each method. The small gaps at the beginning of ϵ_{cos2} and ϵ_{svd2}, respectively, are in a negligible time range and most probably due to small differences in the implementation. On the other hand, the slightly faster results of the *Confidence Interval* for δ_{r3} reflect the use of the computational intensive ILP solver. Obviously, the solver is able to find a solution faster when the threshold δ_r is set higher and, hence, the search space is smaller.

Finally, although there seem to be almost no differences between the runtime results for the *Cosine Similarity* and the *SVD*, it can be expected that with a higher number of resources the *SVD-Test* will experience longer computation times caused by a more intensive SVD calculation compared to the relatively simple cosine similarity. Generally, it should be considered that the runtime of the diagnosis does not have to support any real-time requirements since already the observation time of a system can be rather long and compared to that the presented methods are very fast.

2.6 Intermittent Fault Diagnosis in Many-Core Systems

The intermittent fault diagnosis has been primarily designed for automotive E/E architectures at system level in order to detect faulty ECUs. However, the approach introduced before is not necessarily restricting the considered resources to ECUs but could be also applied to distributed systems on a different scale, for instance, in many-core NoCs. As these architectures are becoming more and more relevant for the automotive domain, in the remaining part of this chapter we want to investigate the usefulness and efficiency of the intermittent fault diagnosis in many-core systems. More precisely, we apply the methodology proposed in Section 2.4 to implicitly detect an affected core by diagnosing an increased fault-rate while taking into account the distributed application tasks and their data-dependencies. The four diagnosis methods are evaluated and compared in terms of runtime and detectability, among other things, considering a significantly higher number of resources than in Section 2.5.

2.6.1 Motivation and Illustrative Example

Nowadays, the multi- and many-core paradigm is not an exclusive characteristic of servers or workstations any more. Due to higher performances and lower energy consumptions, CPUs with multiple cores find their way into embedded SoCs architectures and, thus, into domains where safety aspects are of high relevance, such as avionics and automotive. However, to fulfill the corresponding reliability requirements of safety-critical systems, efficient fault diagnosis and fault tolerance mechanisms are inevitable. Here, especially the minimized component sizes and lower supply voltages resulting from a higher number of cores on one die could lead to an increased susceptibility for faults. The latter, in turn, can be caused by environmental phenomena, like EMI or ESD, as well as marginal hardware and variations during the production process. Whatever their cause is, these faults have a negative impact on the system reliability [23]. Moreover, in many-core systems, the faults in one core can influence the behavior of the running application and, in the worst-case, lead to the failure of the entire processor. It is therefore desirable to obtain knowledge about potential core failures before they actually happen, and apply precautionary measures, which can vary from restarting the affected task on a different (redundant) core to a controlled shutdown of the entire processor. For such an early fault detection, an increased number of non-permanent faults is a suitable indicator to determine unhealthy cores.

Consequently, just as for ECUs within an automotive architecture, we are interested in an approach to implicitly detect cores in a many-core system that show an increased number of non-permanent faults. In other words, the presence of intermittent faults shall be verified. Assuming that in the course of the aging process a deficient hardware causes the presence and accumulation of intermittent faults, a potential imminent permanent fault of a specific core can be projected by analyzing the results of a set of plausibility tests running within regular tasks or as discrete applications. As mentioned earlier, one goal of the proposed diagnosis approach is to perform such a detection implicitly in order to keep the additional resource utilization low.

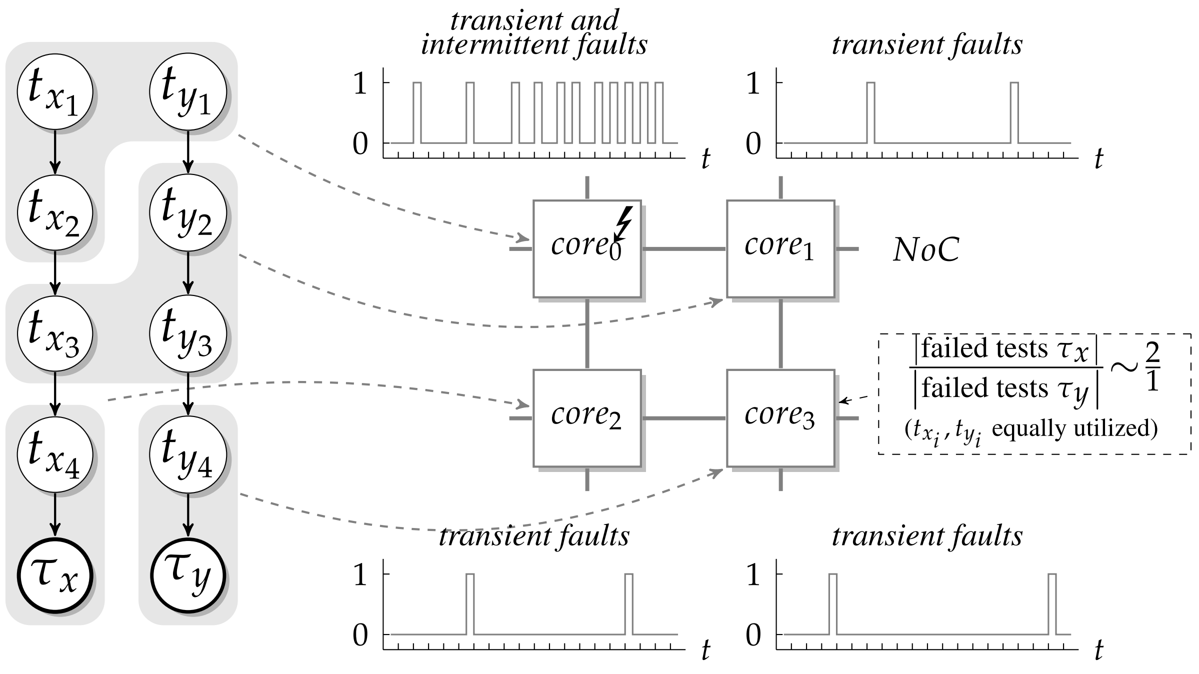
In contrast to an explicit diagnosis which would require additional tests for each component, the proposed fault detection relies on existing plausibility tests being part of the corresponding many-core applications. Moreover, heterogeneous CPU architectures might often only be able to check the consistency of test results on specific cores and components, respectively. For this purpose, the distribution of applications, the runtime of tasks, and their data-dependencies are taken into account to determine the core that shows an increased number of intermittent faults. However, the fault diagnosis is not basically limited to existing tests and could be extended to work with explicit test task when necessary, as will be briefly discussed in Section 2.7.

Based on the above considerations, we apply the intermittent fault diagnosis methods, presented in the context of faulty ECUs in automotive E/E architectures in the previous sections, to many-core systems. As the number of cores in many-core systems quickly increases and might reach ranges of 10^3 and higher within the next decade [73], the scalability of our approach is crucial. Therefore, we aim at optimizing two conflicting objectives: on the one hand, our method shall provide an optimal diagnosability of faulty cores and, on the other hand, the runtime of the diagnosis shall be minimal. In the experimental results, we show the general feasibility of the proposed approach and compare the methods in terms of runtime and correctness of the detection. For the sake of simplicity, we consider a system with at most one affected core. However, as mentioned earlier, the presented approach can be extended to a concurrent detection of multiple faulty resources.

Motivating example. Analogue to the E/E architecture scenario, Figure 2.8 illustrates the intermittent fault diagnosis principle in a many-core system. It shows four cores of a (possibly larger) NoC-based many-core system where an increased fault rate occurs on $core_0$. More precisely, the considered system consists of two functions each executing four application tasks τ_{x_i/y_i} and two test tasks $t_{x/y}$ as well as a processor architecture containing at least four cores. The dashed arrows and shaded background areas indicate which tasks are mapped to which cores. Depending on the particular assignment and utilization of the tasks, a higher rate of non-permanent faults on one core can be detected by analyzing the failure ratio of particular test tasks. In this trivial example, we assume that all cores are equally utilized by the application tasks and a specific observation time is defined. Now, given that all faults will finally lead to failures of the plausibility tests, an increased fault rate on $core_0$ will cause a failure ratio between t_x and t_y of 2:1, since it is running two tasks from the t_x-task chain and only one from the t_y-task chain. Correspondingly, a failure ratio of 1:2 indicates a faulty $core_1$, whereas the occurrence of failures of just one test t_x or t_y shows a faulty $core_2$ and $core_3$, respectively.

2.6.2 Many-Core System Model

Generally, the system model for the intermittent fault diagnosis in many-core systems is formally equal to that introduced in Section 2.3.1, including the Poisson-distributed fault model, the expectation matrix as well as a possible diagnosability analysis. As a consequence, also



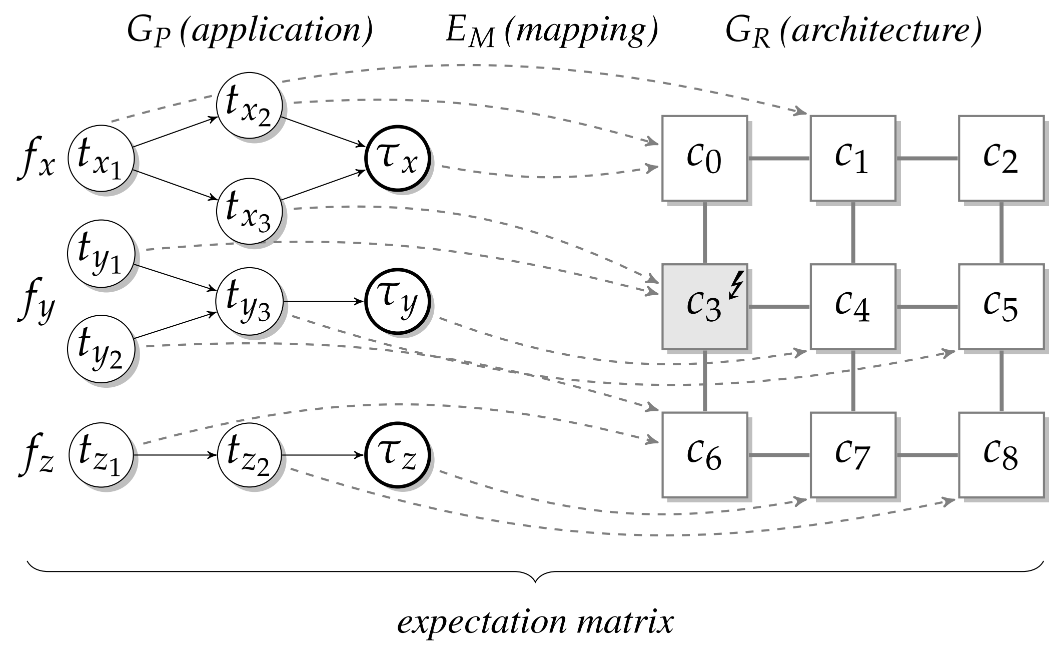
transient and intermittent faults
transient faults
transient faults
transient faults
1
0
t
core0
core1
core2
core3
NoC
|failed tests τx| / |failed tests τy| ∼ 2/1
(txi, tyi equally utilized)
tx1
tx2
tx3
tx4
τx
ty1
ty2
ty3
ty4
τy

Figure 2.8: *Example illustrating the principle of our detection approach on four cores of a NoC-based many-core system. The application tasks* t_{x_1}, t_{x_2}, t_{y_1} *are mapped to* $core_0$ *and* t_{x_3}, t_{y_2}, t_{y_3} *are mapped to* $core_1$*, whereas the remaining* t_{x_4} *and* τ_x *as well as the* t_{y_4} *and* τ_y *are assigned to* $core_2$ *and* $core_3$*, respectively. The increased fault rate on* $core_0$ *results in a test failure ratio between* τ_x *and* τ_y *of 2:1.*

the formal definitions presented in Section 2.4 can be applied for many-core systems. However, given the different dimensional and architectural constraints, there are several differences regarding the corresponding requirements, which shall be briefly discussed below.

Unlike in single-core architectures, the system performance in many-core systems arises from the number of cores rather than their complexity or clock-frequency. While beneficial with respect to the energy consumptions and computational efficiency, this also results in an absence of hardware-based fault diagnosis and fault tolerance mechanisms for each single core [131]. Consequently, to keep or increase the reliability level of single-core architectures, it becomes inevitable to implement this functionality in software for which the computational overhead should be kept as low as possible. Within the many-core system model, we consider a single core in a many-core architecture as one particular resource. However, by adapting the expected fault rates, our fault diagnosis can also be used for other system granularity levels where a resource is regarded, for example, as a whole tile comprising a computation core, cache memory and router in a NoC or, in a fine grained model, just one of these elements.

As discussed earlier, the diagnosis methods themselves do not need to comply with strict real-time requirements. However, in contrast to the intermittent fault diagnosis at ECU-level,



$$\begin{pmatrix}
\lambda(\tau_x,c_0) & \lambda(\tau_x,c_1) & 0 & \lambda(\tau_x,c_3) & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \lambda(\tau_y,c_3) & \lambda(\tau_y,c_4) & \lambda(\tau_y,c_5) & \lambda(\tau_y,c_6) & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \lambda(\tau_z,c_6) & \lambda(\tau_z,c_7) & \lambda(\tau_z,c_8)
\end{pmatrix}$$

Figure 2.9: *Figure illustrating a simplified many-core system specification and the corresponding expectation matrix. It consists of three application functions (f_x, f_y, f_z), the cores $core_0$ to $core_8$ and mappings between tasks and cores ($\dashrightarrow$). The expectation matrix derived from the specification points out the affected λ-values in case of a faulty core c_3.*

the memory necessary to store plausibility test outcomes may be a crucial issue. Hence, reducing the number of regarded tests or rather minimizing the tests-to-core ratio becomes an even more important characteristic of the diagnosis quality. Finally, the determination of the λ-values might turn out to be very challenging. On the one hand, the small sizes and similar structure of cores on a single will lead to only minimal differences between the single entires of the Λ matrix. On the other hand, effects like an uneven distribution of heat throughout the SoCs must be considered as they can influence the aging and, thus, the fault rate of particular cores.

The many-core system model used for the experimental results in Section 2.6.3, can be considered as according to the example in Figure 2.9. It shows an simplified system specification consisting of an process graph G_P representing the application with three functions f_x, f_y and f_x with two or three application tasks $t_{x/y/z}$ and one test task $\tau_{x/y/z}$ each. The architecture graph G_R represents a many-core system with nine cores $core_0$ to $core_8$ assigned to the tasks according to the mapping definition E_M. Below the system specification, the expectation matrix represents the expected failure rates of the three plausibility tests caused by each core. In this example, core c_3 is considered faulty, which would affects the observations associated with the highlighted λ-values in the third matrix column.

2.6.3 Diagnosis Evaluation

This section shall investigate the feasibility and performance of the proposed intermittent fault diagnosis in many-core systems.

2.6.3.1 Test Case Generation

For the following evaluation a similar testing environment has been used as for the previous analysis in Section 2.5. This includes the generation of expected and observed faults as well as the choice of the threshold parameters listed in Table 2.2 which might facilitate a possible comparison between the two result sections. However, the goal of the evaluation is not to merely provide the comparability between large distributed systems, such as automotive E/E architectures, and small distributed systems, such as NoCs, but also and especially to investigate the applicability of our intermittent fault diagnosis for those different scenarios as well as its scalability.

Consequently, the main modifications for the many-core scenario are in the experimental test cases. Based on the system model described above, 240 test cases have been generated with topologies inspired by NoCs. The architecture sizes are ranging from 3x3 to 48x48 cores and, thus, the corresponding system specifications comprise 9 to 2304 resources executing 3, 5 and 10 tasks on each resource on average. Due to the larger systems, the ratio of plausibility tests to cores varies between $\frac{1}{2}$ and $\frac{1}{16}$. Analogue to the previous evaluation of the intermittent fault diagnosis, the observation time varies according to Equation (2.21) and we assume that only half of the test cases have faulty core in order to detect false positive results.

2.6.3.2 Experimental Results

Correctness. First, the overall fault diagnosis results shall be investigated based on the bar chart in Figure 2.10. Like for the graph in Figure 2.5, each bar represents one detection method with one specific parameter value and each test outcome can be assigned to one of four different results, *correct*, *false positive*, *false negative* and *timeout*, with the latter one set to 60 seconds.

Inspecting the absolute numbers of correct test case results indicates a superiority of the *Cosine Similarity*, *SVD-Test* as well as the *Confidence Interval* methods ($49 - 73$ % correct detections) over the χ^2-*Test* ($11 - 16$ % correct detections). The two vector-based methods show a comparable distribution of the four regarded result categories with the *Cosine Similarity* being able to correctly detect $2 - 8$ % more test cases. The *Confidence Interval*, has, on the one hand, the overall highest number of correct detections for the parameter δ_{r3}, on the other hand,

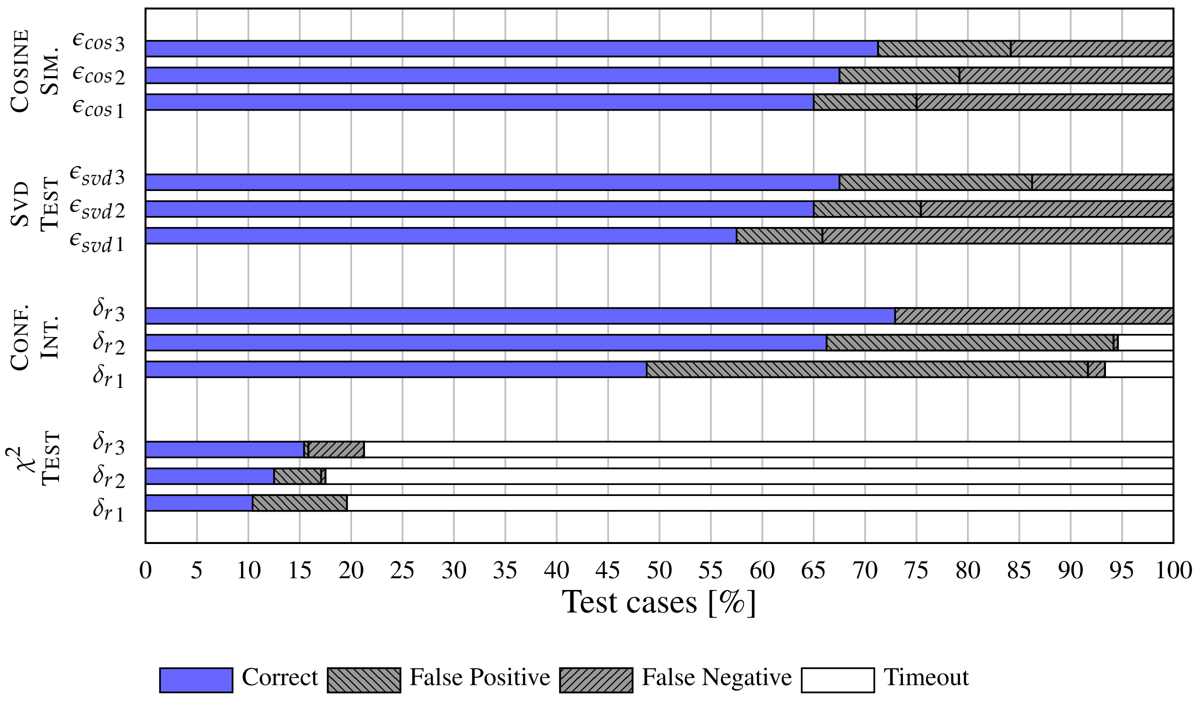


Figure 2.10: *Graph comparing the numbers of all test case results of the intermittent fault diagnosis in many-core systems. Each bar represents the outcome for a fault diagnosis method with one specific parameter value. False positive and false negative results comprise test cases which could not be solved correctly. The timeout for finding a solution for one single test case is* 60 *seconds.*

it also presents the highest variance between the different parameters. Here, the experiments using the parameters δ_{r2} and δ_{r3} ran into a timeout for about 5 % of the test cases. In contrast to this, for the χ^2-*Test* the calculations have been aborted in almost 80 % of the cases but otherwise show a comparable distribution of the results as the *Confidence Interval*. Again, using a different solver for the corresponding ILP might reduce the number of timeouts making the χ^2-*Test* still an interesting method for the intermittent fault diagnosis. Due to equal test parameter values and diagnosis methods, the distribution of correct, false positive and false negative test results for all test configurations accords with the discussion in Section 2.5.2. Nevertheless, although the presented outcome confirms the general feasibility of our approach for many-core systems and shows a qualitative similarity to the results for smaller architectures, such as the in-vehicle networks, the quantitative detection rate appears to be worse. As a consequence, the diagnosis behavior in relation to the system size, namely the number of cores, shall be investigated in the following.

Figure 2.11 illustrates the detectability as a function of the number of cores for all four methods with the results from all three parameter values combined. The y-axis represents the

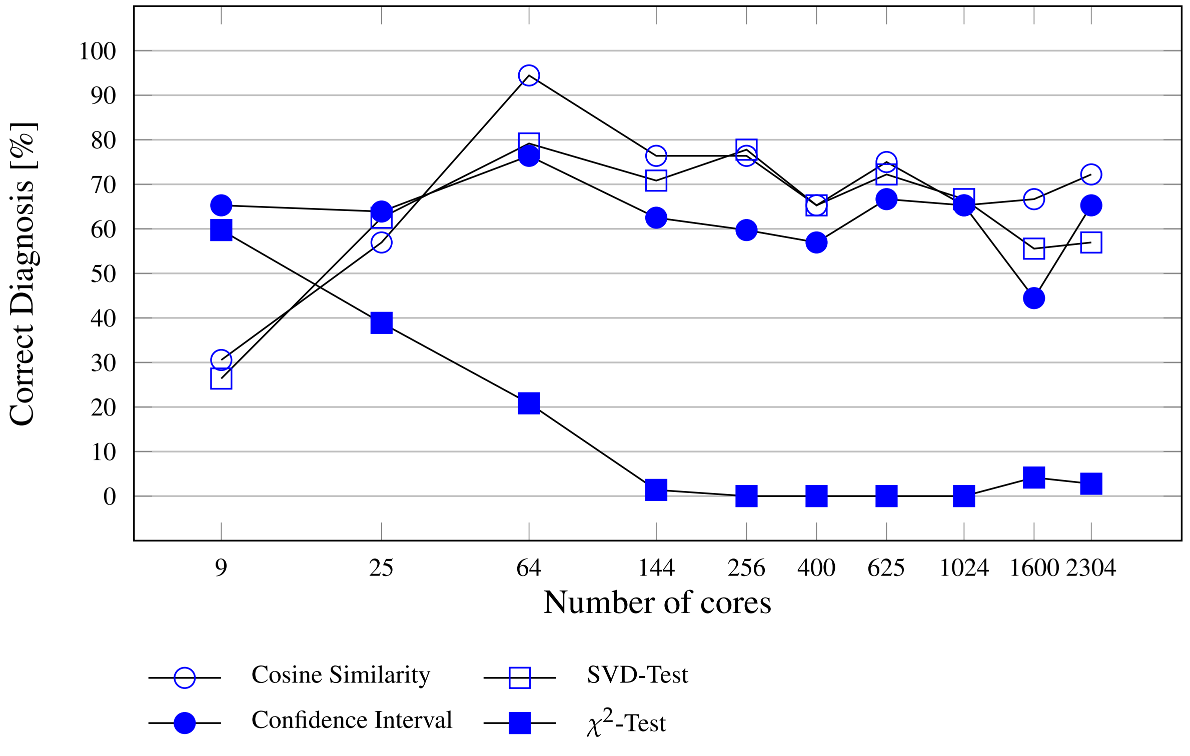


Figure 2.11: *Diagnosis rate as a function of the system size. The curves represent only correct fault diagnosis outcomes without considering false positives results and the outcome of each method combines the results of all three parameter values. The number of cores on the x-axis is depicted in logarithmic scale.*

rate of correct test outcomes, however not including false positive results and the x-axis depicts the number of cores on a logarithmic scale. Interestingly, the *Cosine Similarity* and the *SVD* are both outperformed by the ILP-based methods when less than 25 cores are being analyzed but rise quickly afterwards when the systems sizes increase, making the latter two methods a good choice only for smaller multi-core architectures. The unsteady and irregular behavior of the three upper curves towards higher core numbers is mainly caused by the nonuniform number of test cases for the different system sizes. Additionally, this might also be intensified by the random distribution of task chains during the test case generation. In general, all implementations except for the χ^2-*Test* provide a detectability which lies clearly above 50 % in almost all cases for all system sizes. Hence, the above-mentioned inferior results for many-core systems do not merely come from the larger system sizes but rather from a lower total number of plausibility tests, as will be discussed later. Regarding the χ^2-*Test*, the rapid fall and lack of correct detections for more than 144 cores can be explained by the unfinished experiments which ran into a timeout.

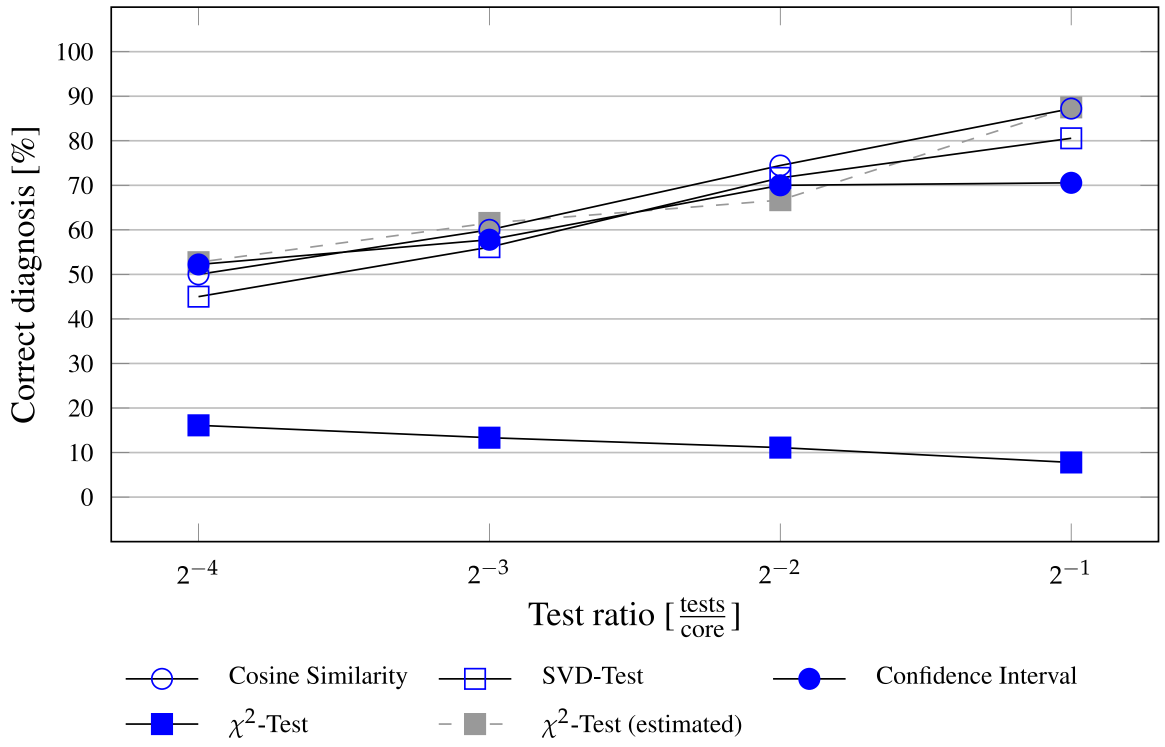


Figure 2.12: *Diagnosis rate as a function of the tests-to-core ratio. A higher ratio on the x-axis indicates that less cores share one plausibility test. The curves represent only correct fault diagnosis outcomes without considering false positives results and the outcome of each method combines the results of all three parameter values. The test ratio on the x-axis is depicted in logarithmic scale.*

Test distribution. As the proposed fault diagnosis follows an implicit paradigm, and by this, utilizes already existing plausibility tests, we want to investigate the influence of the number of tests per core on the diagnosis quality.

In Figure 2.12 the y-axis depict the percentage number of correct test results and the x-axis represents the tests-to-core ratios with the four values between 2^{-4} and 2^{-1}. This indicates that a single plausibility test per 16, 8, 4, and 2 cores, respectively, is considered for the fault diagnosis. Like for the previous plots, in this graph the results for all parameters are combined. Overall, the curves demonstrate that the diagnosis rate is increasing with a higher tests-to-core ratio for all methods except for the χ^2-*Test* for which this behavior is reversed and quantitatively less distinct. The most probable reason for this is that for a higher number of tests the detection problem becomes more complex and, thus, more χ^2-*Test* test cases result in a timeout error. This fact is also responsible for the overall low detectability of this method, as already discussed. By contrast, for the remaining three approaches it can be seen that even a relatively low rate of one plausibility test per 16 cores still correctly detects every second test case while applying one

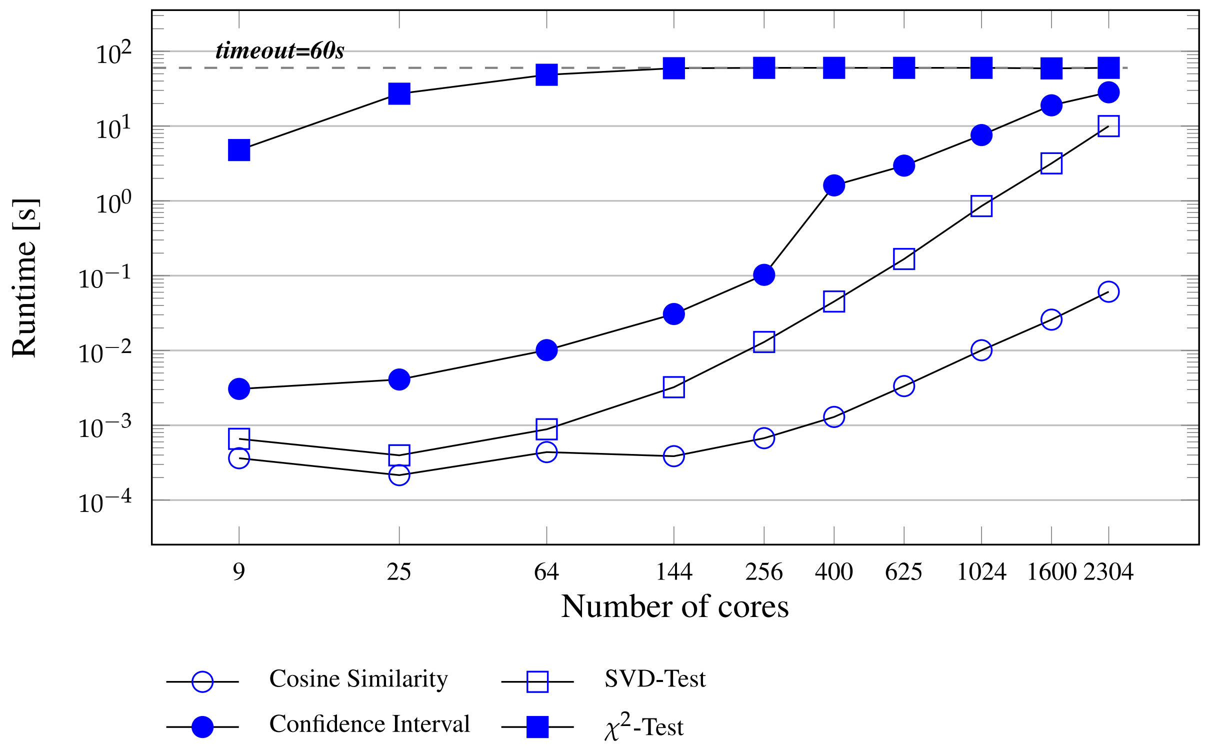


Figure 2.13: *Average test case runtimes as a function of the system size. The outcome of each method combines the results of all three parameter values. The timeout for the test case calculation was set to* 60 *seconds. Both axes are in logarithmic scale.*

test per two cores on average leads to a detectability of up to 90 %. For illustration purpose, the gray curve represents estimated results for the χ^2-*Test* without timeouts, but should not be considered for implementation or further refinement due to its possible imprecision.

Runtime. In the next step, the diagnosis methods shall be evaluated regarding their computational efficiency. For this, the graph in Figure 2.13 depicts the runtime in seconds on the y-axis and the number of cores on the x-axis. Both axes are displayed in a logarithmic scale. The result curves indicate very short computational times for the *Cosine Similarity* lying below 100 ms for all system sizes. The runtime of the *SVD-Test* method is comparable up to 64 cores but becomes slower afterwards with two orders of magnitude above the *Cosine Similarity* for the largest systems. Compared to the vector-based methods, the runtime of the *Confidence Interval* is lower but still in a reasonable range. In contrast to this, the χ^2-*Test* shows a low efficiency already for the smallest architectures and quickly becomes infeasible for test cases with more than 64 cores.

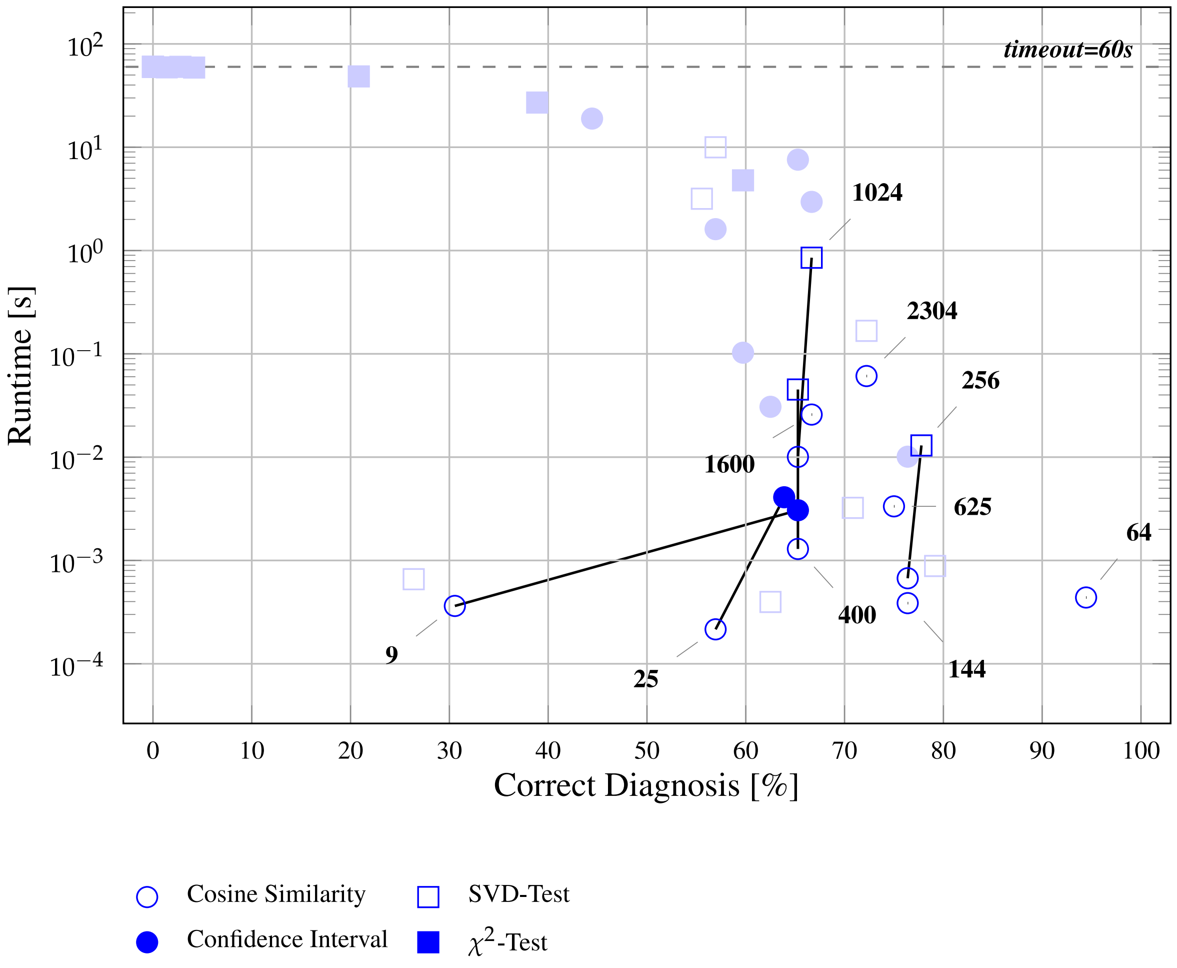


Figure 2.14: *Illustration of Pareto efficient points with respect to a high detection rate at low test case runtimes. The pinned labels show the core numbers associated with the particular results and two connected points represent a Pareto-front. The outcome of each method combines the results of all three parameter values. The runtime on the y-axis is in logarithmic scale.*

Trade-off analysis. As mentioned previously, the design of distributed systems with a focus on efficient fault diagnosis requires both a high fault detection rate without additional software or hardware extensions as well as a fast corresponding computation time. Hence, a system designer might be especially interested in the results from Figure 2.11 and 2.13 in order to select and implement the most suitable fault diagnosis method. For this, the graph in Figure 2.14 combines these two results and, thus, helps to analyze the Pareto efficiency with respect to the diagnosis quality and runtime. Here, the deep blue marks represent optimal results for a particular system size, with the number of cores pinned to the corresponding marker. In the case, where two detection methods are optimal for both objectives (i.e., runtime and correct diagnosis), the corresponding marks are connected with the resulting line forming the Pareto-front for the corresponding system size. All grayed out points are dominated by the full blue ones and, hence, are regarded as Pareto inefficient.

The graph shows, that for the most system sizes the *Cosine Similarity* seems to be the best diagnosis method. Although the *SVD* shows slightly better detection rates for 256, 400 and 1024 cores, its runtime is one or two orders of magnitude slower (however, still in a millisecond or low second range). When considering small systems with 9 and 25 cores, then, clearly, the *Confidence Interval* is the best choice, as in this time range the difference to the vector-based method is negligible. Due to its long computation time, using the χ^2-*Test* seems not beneficial for the regarded test cases.

2.7 Conclusion

In this chapter, an approach was presented that enables an implicit detection of resources in a distributed system affected by intermittent faults. Multiple methods, which analyze existing plausibility test outcomes collected at runtime, are applied off-line and during low system utilization, respectively. Overall, the diagnosis is aiming at an early detection of increased fault rates indicating a potential imminent permanent fault. Although in the first instance intended for larger automotive E/E architectures, it has been shown that the presented approach can be used for distributed systems on IC level, such as modern many-core NoC-based architectures, as well. The latter application is relevant for the automotive industry, since the use of multi- and many-core systems in safety-critical domains is constantly growing.

Four different fault diagnosis methods are proposed. Two of them use a vector-based approach (Cosine Similarity and SVD) and two use a probabilistic approaches (Confidence Interval and χ^2-Test). All methods are evaluated based on a number of synthetic test cases covering both the automotive in-vehicle networks and many-core systems. The experimental results give evidence of the effectiveness and efficiency of the proposed approach. For instance, in can be shown that, depending on the analyzed method, a correct fault diagnosis result can be obtained in up to 95 % for the automotive test cases and up to 75 % for the many-core systems. Especially regarding the wide range of the system sizes, the approach shows a good scalability and satisfactory runtimes. The main limitation is resulting from an insufficient scalability of the computational intensive χ^2-Test method.

To the best of our knowledge, this is one of the first approaches analyzing and comparing distributed plausibility test failures to implicitly detect an increased occurrence of non-permanent faults on faulty resources. Therefore, the main goal of the intermittent fault diagnosis is the demonstration of its general feasibility and its possible use in the area of safety-critical distributed systems. Finally, the presented approach has the potential for future extensions, as discussed below.

Future work. One of the aspects which shall be investigated in more detail in the future, is the influence of the observation time on the diagnosis quality. Although intermittent faults may be not as safety-critical as permanent faults, it is still very important to guarantee a fast diagnosis which, for the presented approach, depends on the choice of the observation time. As already mentioned, in this stage of the work, we only consider a single faulty resource (i.e., one ECU or one core) at a time. However, ILP approaches innately support the detection of multiple stressed resources and the methods based on the analysis of linear dependencies can be adapted appropriately. Hence, an extension of the approach towards a concurrent detection of multiple faulty resources is basically possible and shall be included in the future work. For the evaluation part of our work, the generation of synthetic test cases involves a random distribution of tasks and plausibility tests to resources. This is mainly to reflect a given and unmodified system supporting the implicit character of the diagnosis. Here, it would be interesting to investigate the influence of a deliberate and optimized placement of tasks and plausibility tests, which could improve the detection rate, however, for the price of an increased effort during design time. In this context, an option could also be to add explicit plausibility tests in order to enable or enhance the diagnosis for particular system specifications. Finally, we want to investigate the behavior of the fault diagnosis approach for different fault rates and also analyze various fault propagation models, possibly with the help of an Instruction Set Simulator (ISS).

CHAPTER 3

Diagnosis of Permanent Faults

The previous chapter presented an approach for the diagnosis of intermittent faults in distributed systems. It has been shown how the analysis of existing monitoring and testing features integrated in particular distributed resources can be used to detect and locate an increased occurrence of non-permanent faults on system-level. Furthermore, by using task dependency knowledge, resources without self-testing functionality are diagnosable as well.

This chapter presents a novel decentralized approach for the diagnosis of permanent faults in automotive E/E architectures. As already discussed in Chapter 2, both the safety-critical real-time requirements and the distributed nature of these architectures make novel fault tolerance approaches on system-level a crucial and challenging issue. At the same time, high unit numbers in manufacturing add cost efficiency as an important criterion during system design, which is conflicting with the use of often expensive explicit fault diagnosis hardware or software.

To address these challenges, we propose a diagnosis framework for permanent faults that mainly consists of two stages. In the first stage, called *diagnosis determination*, a given architecture is analyzed and all potential fault scenarios, such as a faulty ECU, are investigated to obtain a set of special diagnosis functions. These diagnosis functions are used at runtime on each suitable resource in the in-vehicle network to determine whether a certain fault scenario occurred or not, and thus, to detect a permanent fault. In the second stage, called *diagnosis optimization*, an optimization approach is proposed to determine trade-offs between diagnosis times and the number of monitored message streams. The feasibility and efficiency of the presented framework is evaluated based on a number of synthetic test cases as well as an automotive case study. Furthermore, the results of an implementation on a research platform for distributed automotive systems shall investigate its real hardware utilization and demonstrate its practical usefulness.

In addition to the general approach, we also propose a schedule synthesis method specially designed for the presented permanent fault diagnosis. It is capable of reducing the necessary diagnosis time compared to a given unchanged schedule and also, under particular circumstances, to improve the overall diagnosability by increasing the number of diagnosable resources.

Chapter outline. Chapter 3 contains seven sections which can be divided into two parts. In the first part, Section 3.1 introduces the topic, gives an illustrative example and presents the main contribution. Section 3.2 discusses the related work. The main principle of the diagnosis and the corresponding framework are presented in Section 3.3. Mathematical formulations of both the diagnosis determination and optimization are given in Section 3.4 and the corresponding experimental evaluation is discussed in Section 3.5. In the second part, Section 3.6 proposes a diagnosis-aware schedule synthesis. Being basically self-contained, it is further divided in the four subsections: Motivation (3.6.1), Communication Model (3.6.3), Methodology (3.6.4) and Evaluation (3.6.5). Finally, the concluding remarks for this whole chapter are given in Section 3.7.

3.1 Introduction

This section introduces the proposed approach for the distributed diagnosis of permanent faults. It first motivates the topic and gives an illustrative example before discussing the specific contributions.

3.1.1 Motivation

In the previous chapters, it has been highlighted how the increasing complexity of automotive E/E architectures together with the simultaneously growing use of software and hardware for safety-critical applications, such as ADAS, make strict guarantees on reliability, fault tolerance and diagnosis times inevitable. In case of a component failure, the defective resource, such as an ECU, a gateway or a bus, must be identified as quickly as possible in order to apply appropriate countermeasures which can range from restarting the affected tasks on other resources to a safe shutdown of the entire system. That issue becomes even more relevant in the view of shrinking component dimensions and decreasing supply voltages in modern integrated circuits which, in general, lead to an increased susceptibility to internal and external faults [13]. At the same time, it still holds that given the high number of units in the automotive industry, costs for electronics are a crucial optimization criterion restricting the use of explicit fault tolerance strategies.

In Chapter 2, a fault diagnosis approach was introduced which basically evaluated the outcome of distributed plausibility tests in order to detect resources with an increased fault occurrence indicating intermittent faults. The main goal there, was to diagnose the system early in advance, especially, in order to prevent a long-lasting damage or a complete failure of a resource. However, due to the characteristics of intermittent faults, their corresponding detection requires a sufficiently long observation time. Moreover, that detection is ineffective for sudden component failures which are not preceded by an increased intermittent fault rate but happen instantly. As a consequence, although the framework presented in Chapter 2 offers a relevant and valid diagnosis tool for intermittent faults, it is inevitable to complement it with an appropriate method for diagnosing *permanent faults*. For this, a novel approach is proposed and described in this chapter.

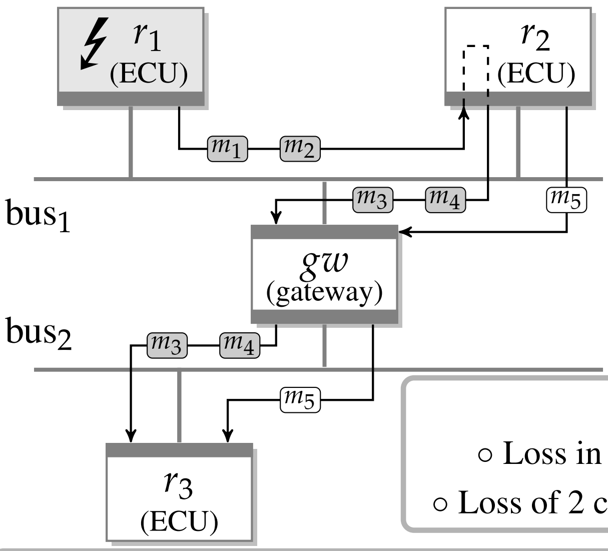
As a matter of principle, the introduced permanent fault diagnosis monitors and analyzes existing message streams in a distributed architecture in order to identify defective resources. Within its communication and fault model, it uses specific patterns of received and omitted messages, so-called *diagnosis functions*, generated by the presented framework. The approach does not rely on a single observing entity, but rather each message-processing resource in the architecture, like an ECU or a gateway, may be capable of diagnosing the entire system or parts of it. Consequently, this decentralized approach removes the single point-of-failure and, thus, improves the reliability of the fault diagnosis itself. By using only existing system messages, the induced diagnosis overhead for each observing resource is minimal.

3.1.2 Illustrative Example

The general concept of the proposed permanent fault diagnosis is illustrated in Figure 3.1 which contains a simple automotive architecture with three ECUs, two buses and one gateway connecting these buses. In this example, ECU r_3 can diagnose a permanent fault on ECU r_1 by detecting that single messages in the streams m_3 or m_4 have been omitted for two consecutive periods. It is assumed that the transmission of the message streams m_3 and m_4 strictly depends on a previous correct reception of messages in the stream m_1 and m_2, respectively, by the resource r_2. Consequently, by using explicit diagnosis functions which consider both omitted ($\neg\mathbf{m_i}$) and received ($\mathbf{m_i}$) messages[1], we are able to pinpoint the source of fault (in this case by excluding r_2) and reduce the diagnosis time.

More precisely, ECU r_1 is transmitting two periodic message streams m_1 and m_2 to ECU r_2 where they are processed and forwarded to ECU r_3 as message streams m_3 and m_4, respectively. In this way, there is a dependency between m_1 and m_3 as well as m_2 and m_4, such that when the transmission of messages from the primary stream fails, then the depending stream will be interrupted as well. Furthermore, r_2 sends another periodic message stream to r_3, namely, m_5. As the ECUs r_1 and r_2 use a different bus than r_3, the communication between them is routed via an automotive gateway. Based on the periods and jitters of the messages listed as *communication parameters* in a table in Figure 3.1, we are able to diagnose faulty ECUs and

[1]Note that while m_i normally denotes a *message stream* (i.e., the periodic occurrence of the message), in some cases it might also refer to single *message instances*. The use of the terms should be unambiguous due to the context. By contrast, the binary variable $\mathbf{m_i}$ describes the considered state of detection of a message stream.



COMMUNICATION PARAMETERS:

message stream $[m_i]$	*period* $[p_i]$	*jitter* $[j_i]$
m_1 and m_3	20ms	5ms
m_2 and m_4	15ms	5ms
m_5	10ms	5ms

ASSUMPTIONS:

∘ Loss in stream $m_1/m_2 \rightarrow$ interruption of m_3/m_4

∘ Loss of 2 consecutive messages $\rightarrow$ permanent fault

PERMANENT FAULT DIAGNOSIS OF ECU r_1 ON ECU r_3:

1. Observing single message losses	2. Using proposed diagnosis functions
$\neg\mathbf{m_3}$: $2 \cdot p_3 + j_3 = 45\text{ms}$	$\varphi = (\neg\mathbf{m_3} \vee \neg\mathbf{m_4}) \wedge \mathbf{m_5}$:
$\neg\mathbf{m_4}$: $2 \cdot p_4 + j_4 = 35\text{ms}$	$p_3 + j_3 = 25\text{ms} \;\; (> p_4 + j_4)$

Figure 3.1: *Example for an automotive E/E architecture consisting of three ECUs, r_1 to r_3, two buses and one gateway. ECU r_3 is an observing resource capable of diagnosing permanent faults (↯) by monitoring patterns of omitted ($\neg\mathbf{m_i}$) and received ($\mathbf{m_i}$) messages. Often, these* diagnosis functions *allow a reduction of the diagnosis time.*

minimize the diagnosis times not only on the common bus but also across the gateway borders, and hence, on network-level.

For instance, a permanent fault on r_1 will cause an interruption (grayed out message blocks) of the message streams m_1 and m_2 and, due to the aforementioned dependency, also m_3 and m_4. In order to distinguish a permanent fault from a transient fault, the number of omitted message instances is monitored by taking into account both the measured and the expected amount of time in which the corresponding message streams are interrupted. Equivalent to the message transmission model *periodic-with-jitter*, this time is represented as an integer multiple of the number of periods and must also include a possible worst case jitter. As will be explained in detail in Section 3.3, two omitted messages shall indicate a permanent fault. As a consequence, when r_3 monitors the streams m_3 and m_4 separately, it needs at least 45 ms ($2 \cdot p_3 + j_3$) or 35 ms ($2 \cdot p_4 + j_4$), respectively, in order to decide if a permanent fault occurred. Besides, it still cannot be decided if the fault stems from r_1 or r_2. However, a concurrent analysis of all message streams on bus$_2$ can reduce the diagnosis time and correctly pinpoint the defective ECU. With the help of a special diagnosis function, in this example $(\neg\mathbf{m_3} \vee \neg\mathbf{m_4}) \wedge \mathbf{m_5}$, one only needs to wait until the first loss of a message instance from the stream m_3 as it has the longer period, during which a message from m_4 must have been inevitably omitted or received as well. Hence, the corresponding diagnosis time decreases to 25 ms, because $(p_3+j_3) > (p_4+j_4)$. At the same time, by detecting a properly received m_5, one can exclude r_2 as potentially faulty ECU.

For a better comprehension of the diagnosis methodology, the architecture and communication parameters described above will serve as *running example* throughout Section 3.4.

3.1.3 Contributions

This chapter proposes a novel approach towards the diagnosis of permanent faults in automotive E/E architectures, which is based on the analysis of the monitored network traffic. Modern in-vehicle networks use different buses, such as CAN, LIN or FlexRay, operating in different domains and connected via automotive gateways [121]. This can impede a message-based fault diagnosis especially when the monitoring resource and the defective resource are attached to different buses, as in the example in Figure 3.1. By regarding the entire distributed architecture at *network-level*, including all buses and gateways (i.e., considering all system messages and their dependencies) the introduced fault diagnosis extends the number of detectable resources and enables a low diagnosis time. Moreover, the abandonment of a single diagnosis entity increases the overall system reliability and, hence, improves the safety-critical aspects of automotive E/E architectures. At the same time, by using only existing network communication, there is no need for special diagnostic messages which would involve additional computation overhead and possibly extend the diagnosis time. Such a decentralized and implicit fault diagnosis is especially (but not solely) beneficial in the cost-driven automotive industry where an early diagnosis of defective hardware is crucial but each dedicated monitoring and testing device entails increased hardware costs due to a high number of units. However, it cannot be ruled out that an analyzed architecture contains some resources which are not transmitting any suitable messages for the diagnosis process. For this purpose, Section 3.6 proposes a method for a diagnosis-aware system design which optimizes existing message streams for a lower diagnosis time and allows the insertion of efficient diagnostic messages.

In particular, we introduce a fault diagnosis approach which determines the effect of potential permanent faults on the existing message-based communication. For a given system architecture, we generate unique diagnosis functions for each possible fault scenario which will be implemented on suitable resources (see Section 3.5.3). In order to find trade-offs between diagnosis times (i.e., the observation times necessary to detect particular fault scenarios) and the number of monitored message streams, we propose an optimization method for the diagnosis functions.

In summary, the contribution of this chapter is twofold:

1. Determination of light-weight diagnosis functions for an implicit and decentralized real-time fault diagnosis in distributed systems.

2. Optimization of diagnosis functions to determine trade-offs between diagnosis times and the number of monitored message streams.

We address fault diagnosis as a self-contained topic and the system behavior subsequent to it is not in the focus of this work. However, our approach is not limited to a specific fault recovery method and, hence, applicable to any fault tolerance strategy in a distributed system which considers a fail-silent behavior of its resources. To the best of our knowledge, this work represents the first approach towards a fully implicit and decentralized permanent fault diagnosis for automotive E/E architectures, including a trade-off analysis of the real-time and performance aspects.

3.2 Related Work

In safety-critical automotive systems it is important to detect and diagnose faults early in order to provide appropriate countermeasures. This is even more crucial for permanent faults which usually involve an immediate loss of a particular functionality. For this reason, there exists a variety of work in the area of reliability-aware system design. The approaches comprise, on the one hand, component-oriented strategies which try to enhance reliability by the proper selection of system components, taking into account constraints such as the chip area, energy consumption and performance [142, 36]. On the other hand, there are task-oriented methods where a better reliability is achieved through specific process allocation, additional tasks or schedule optimization [42, 152]. All these works cover differently elaborated fault diagnosis strategies and have been discussed in more detail in Section 2.2.

Furthermore, [47] goes beyond the assumptions of an ideal diagnosis and investigates the use of imperfect software fault detectors. The paper presents a design optimization which considers both detectable and undetectable faults, assuming that, although imperfect detectors might not prevent all silent data corruptions, they may provide more capacity for fault tolerance methods due to a lower resource utilization. Enhanced reliability techniques for system-level synthesis have been proposed in [24]. Here, multiple fault management requirements for different target platforms can be specified at design time together with the specific available reliability-oriented techniques. A subsequent design space exploration identifies the optimal solutions for a hardened system regarding performance and reliability requirements. An extension of this work has been published in [10] especially considering mixed-criticality fault management and an optimal placement of the voting/checking algorithm. Regarding the actual fault detection methods, the three works above consider well-known approaches like task duplication with concurrent error detection units or particular checking and voting mechanisms.

By contrast, our work presents a diagnosis of permanent faults which analyzes the message-based communication in order to identify defective resources. In this context, a detection algorithm using corrupted messages to discriminate functioning from defective resources is introduced in [129, 128]. It is designed as an add-on protocol for a time-triggered communication platform and to some extent resembles our approach as it also uses omitted messages to detect faults. However, while their detection methods are applied at bus-level, our approach considers the entire system at network-level, which means that it includes all buses, gateways and other dedicated links. This increases the diagnosis capability of our methods but makes them also more challenging by requiring, inter alia, special diagnosis functions. Furthermore, the detection time in [129, 128] is bound by the sending slot cycles of the TDMA-architecture, while our approach supports event-triggered communication as well.

The idea of monitoring conjoint event arrivals has been used in [103] in the context of mixed-criticality systems. In this work the activation of a group of low priority tasks is modeled through the sums of their workload arrival functions to bound their influence on high priority tasks. While the concept of combined workload arrival functions is similar to our approach of summing up message arrival curves, it is used for integrating applications aiming at increasing the system utilization. On the contrary, in our work the arrival curves are used to determine the minimal diagnosis time for the detection of a potential permanent fault.

A distributed fault diagnosis for automotive architectures is presented in [89]. Here, single nodes send the outcome of a local fault diagnosis to neighboring nodes in order to achieve a global fault diagnosis. Although the paper highlights the high communication costs and delays of a centralized diagnosis, the approach is still relying on diagnosis status broadcasts and diagnosis times are not explicitly discussed. Our method uses solely existing messages and is based on a decentralized principle to reduce system overhead as well as remove a possible single point-of-failure. It also puts an emphasis on a minimal diagnosis time.

In [144] the authors present an on-line/off-line diagnosis framework motivated by the automotive domain. The approach uses propositional logic to describe the relationship between specific system properties and potential faults. However, in contrast to our work which provides a concrete massage-based detection method, the presented framework is a purely formal specification depending on user-defined tests and monitors to produce a valid diagnosis outcome.

Fault diagnosis has been also addressed in the area of Discrete Event Systems (DESs), for instance in [33, 91, 137]. Such diagnosis methods often apply a model-oriented approach using logical models, such as Petri nets or timed automata. [33] proposes an on-line diagnosis to detect and localize multiple event and state faults. For this a special diagnosis tree representation for a Petri net (Coverability tree) is used to identify faulty transitions as well as to isolate faulty places. Similarly, Petri nets are used in [91] to detect permanent faults based on the observation of a subset of fault transitions, where the latter ones are classified according to three different diagnosis states (no fault, uncertain, sure fault). In [137], first a temporal, functional and fault-based analysis of a system with a timed automaton model is performed and a special automaton called diagnoser is built. Then, the model-based diagnosis uses characteristic times during normal behavior to detect system failures. Although we use a graph-based system description to determine diagnosis functions, at runtime the fault diagnosis does not depend on a fault-model but rather on precalculated diagnosis functions and a real-time monitoring of the network traffic to keep the computational and memory overhead low. As has been pointed out in Section 1.3.1, model-oriented diagnosis methods are not suitable and can be hardly applied in our work.

Finally, the diagnosis-aware system design approach presented in Section 3.6 uses a time predictable scheduling paradigm and, therefore, is inspired by the work in [119, 118]. These papers discuss integration strategies for independently defined subsystem schedules into a global schedule, which are applied for modular architectures based on a time-triggered and data-centric system model. As the design strategy proposed in this thesis targets time-triggered systems as well, to some extent the corresponding message stream adaptation has to meet similar constraints during the synthesis process as the schedule integration approach in [118].

3.3 Diagnosis Description

This section describes the distributed system model especially emphasizing the considered representation of permanent faults. Furthermore, the corresponding diagnosis framework is presented.

3.3.1 Fault Model

As illustrated in Figure 3.1, we generally regard automotive E/E architectures as distributed systems where ECUs are linked to different gateway-connected buses and use message-based communication. Within the considered system model, each message stream m_i is defined by its nominal period p_i and a jitter j_i which depicts the variation of the measured period from the nominal period caused by task delays. For the proposed fault diagnosis strategy, these message streams are monitored and analyzed in order to identify and properly interpret particular traffic patterns. In this context, among other things, it is an important task to distinguish potential permanent faults from the less critical transient faults.

In electronic components faults originate from physical phenomena and usually manifest themselves as bit flips. In Section 2.3.1, it has been explained that the cause of a fault can be both external, such as a temporary change in the environmental conditions, and internal, such as an unstable or marginal hardware. The first case can be often linked to transient faults while the second case normally results in intermittent and permanent faults. In the approach presented in this chapter, intermittent faults, which might cause the interruption of several consecutive messages, are considered severe and are practically not distinguishable from permanent faults.

Fault assumptions. Generally, we assume a fail-silent behavior where a fault will result in an error leading to a transmission failure of messages. For instance, this could be the effect of a total breakdown of a resource. However, it is also possible that a fault will not automatically lead to a message loss but, for example, will entail a message corruption or other erroneous behavior. In order to guarantee a complete and correct fault diagnosis, especially for the latter cases it is expected that an internal fault detection mechanism prevents a resource from sending faulty messages. Although, this could require an extension of the current functionality of a monitoring resource, the gain would be an improved and simplified fault diagnosis at system-level. Moreover, most ECUs usually posses dedicated monitoring and diagnostic components for self-diagnosis, as has been discussed in Section 1.2.1. In this way, our approach only needs to consider existing communication without the necessity for special diagnostic messages. And even if a system already uses diagnostic messages it can further benefit from the proposed method by a reduction of the diagnosis time.

Basically, we discriminate between a healthy (functioning) resource, which only exhibits transient faults, and an unhealthy (defective) resource, which is affected by permanent faults. The distinction is made on the basis of the number of consecutive message losses in a message stream. For this, we state that the loss of a single message indicates any fault (also transient), while several consecutively omitted messages in one message stream indicate a permanent fault. However, depending on its specific manifestation within the system, a transient fault might also cause more than one message loss such that each additional omitted message increases the certainty that a permanent fault has been diagnosed. As a consequence, in a trivial case when regarding merely one single message stream, the minimum diagnosis time for a permanent fault can be calculated as $N \cdot p_i + j_i$, where N specifies the number of consecutively omitted messages in this message stream (and accordingly the number of omitted periods) and $N \geq 2$. This means, that for $N = 2$ one needs to detect an interruption in the message stream m_i for at least two periods p_i (plus the worst case jitter j_i) in order to conclude that a permanent fault occurred at the corresponding resource.

3.3.2 Diagnosis Framework

Figure 3.2 illustrates the proposed diagnosis framework which can be divided into two parts: the fault diagnosis determination (Stage I) and the fault diagnosis optimization (Stage II). A general outline of the framework is given below and the relevant formulations will be discussed in detail in Section 3.4.

3.3.2.1 Stage I: Fault Diagnosis Determination.

Diagnosis functions are a fundamental element of the proposed permanent fault diagnosis and thus, play an essential role within our diagnosis framework. They are represented by Boolean functions of message observations that are derived from a given system specification and can be generated at design time. The system specification is a graph-based representation of a distributed architecture, where applications, consisting of tasks and messages, are mapped onto architecture resources, such as ECUs and buses. With the help of a graph-oriented search algorithm, the system is inspected for all potential sources of failure, which are then defined as *fault scenarios*. The latter are represented as sets which can consist of a single resource or a group of resources with a common potential source of fault, such as a shared power supply[2]. For each fault scenario a unique pattern of received and omitted messages per *observing resource* is

SYSTEM ARCHITECTURE
(DETERMINING COMMUNICATION PARAMETERS)

Fault diagnosis determination $\downarrow$ Section 3.4.1

DIAGNOSIS FUNCTIONS
(DEFINING BOOLEAN FUNCTIONS OF MESSAGE STREAM OBSERVATIONS PATTERNS)

Fault diagnosis optimization $\downarrow$ Section 3.4.2

DIAGNOSIS TRADE-OFF
(DETERMINING DIAGNOSIS TIMES VS. MONITORED MESSAGE STREAMS)

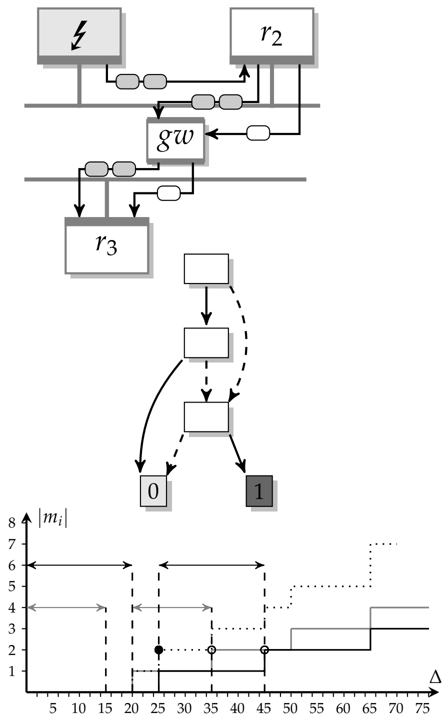


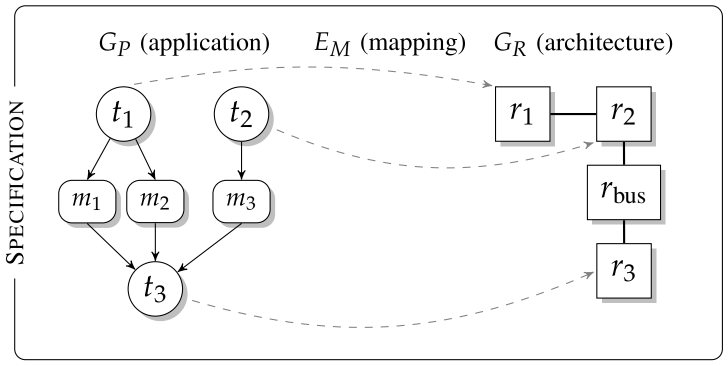
Figure 3.2: *Illustration of the proposed permanent fault diagnosis framework including a determination and optimization stage. At design time, a given system specification is analyzed in order to generate special diagnosis functions that are applied at runtime. In the second stage, these functions are used to optimize the diagnosis times and the number of monitored message streams.*

determined that allows to indicate the occurrence of a permanent fault. It is assumed, that each bus-attached resource can act as an observing resource.

In order to efficiently integrate our fault diagnosis in a runtime system, the observation patterns are initially encoded as Binary Decision Diagrams (BDDs). With the help of a *minimum cut* algorithm potential suboptimal paths are excluded from the BDDs and the final *diagnosis functions* $\varphi_{r_o,s}(M_{r_o,s})$ are obtained. The observing resources r_o use the diagnosis functions to identify a defective system component by indicating the occurrence of a particular fault scenario s. This is done by evaluating the state of messages from a set of monitored streams $M_{r_o,s}$ and calculating the outcome of $\varphi_{r_o,s}(M_{r_o,s})$, as defined in Equation (3.1).

The corresponding methods and formulations for Stage I are described in Section 3.4.1.

[2]Although in principle the diagnosis methods could handle multiple faults (e.g., by generating specific diagnosis functions), in the scope of this work we apply the single-fault assumption which is common in automotive [77] and has been also discussed in Chapter 2.



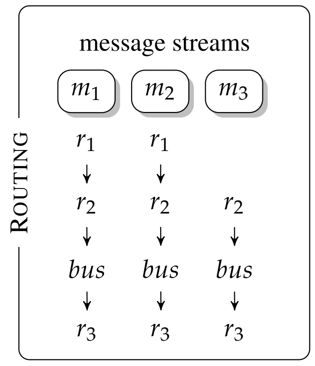


Figure 3.3: *Example of a system specification and message routing: an application consisting of three tasks t_1, t_2 and t_3 is mapped onto the resources r_1, r_2 and r_3. The routing defines the message communication where m_1, m_2 are sent over an exclusive connection between r_1, r_2 as well as over the bus resource and m_3 is using only the bus.*

$$\forall r_o \in R, s \in S :$$

$$\varphi_{r_o,s}(M_{r_o,s}) = \begin{cases} 1, & \text{fault scenario } s \text{ is diagnosed by observing resource } r_o \\ 0, & \text{fault scenario } s \text{ is } \textit{not} \text{ diagnosed by observing resource } r_o \end{cases} \tag{3.1}$$

Within the presented decentralized approach, the diagnosis functions for one fault scenario s can vary for different observing resources r_o.

3.3.2.2 Stage II: Fault Diagnosis Optimization.

In the second stage, our framework uses the diagnosis functions to investigate trade-offs between the fault diagnosis times and the number of message streams that have to be monitored. To allow a general description of message streams and facilitate the addition of message times, the corresponding periods and jitters are modeled by means of event arrival curves. At the same time, the Boolean representation of $\varphi_{r_o,s}(M_{r_o,s})$ enables to determine message streams in $M_{r_o,s}$ which can be excluded from a diagnosis function without downgrading its diagnosis capability. However, while this exclusion is done without affecting the diagnosis itself, it may increase the corresponding diagnosis time. This stage applies ILP with the objective to minimize the observation time for each diagnosis function. The optimization problem is repeated multiple times where the cardinality of the set $M_{r_o,s}$ is successively reduced.

The corresponding methods and formulations for Stage II are described in Section 3.4.2.

3.4 Methodology

This section provides the formal descriptions and detailed explanations about the determination of diagnosis functions as well as their subsequent optimization. The permanent fault diagnosis framework uses a graph-based representation for applications and architectures where nodes depict processes (e.g., tasks or messages) and resources (e.g., ECUs or buses) while edges constitute dependencies and links, respectively. Figure 3.3 illustrates this and shows that applications are mapped onto architectures and from this mapping an appropriate routing for the messages can be obtained. A more detailed explanation of a system specification is given for the diagnosis-aware system design, in Section 3.6.3. In the following, the focus lies on the communication aspects, namely, the corresponding message streams, their routings and timings. For a better understanding of the presented methods, the architecture in Figure 3.1 will be used as running example.

3.4.1 Fault Diagnosis Determination: Formulation

The definitions in this section are mainly based on the sets and functions listed below.

$r \in R$	system resource from the set of all resources R to be considered for the permanent fault diagnosis
$r_o \in R$	observing resource from the set of all resources R to be considered for monitoring all observable message streams
$s \in S$	fault scenario from the set of all scenarios S, each representing a subset of (potentially faulty) resources (i.e., $s \subseteq R$)
$m \in M$	message stream from the set of all message streams M, where $M_{r_o} \subseteq M$ contains all messages observable by r_o and $M_{r_o,s} \subseteq M$ contains messages observable by r_o for a particular fault scenario s
$c : R \times M_{r_o} \rightarrow 2^R$	*cause function* determining all resources which can cause the loss of a message in m observable by r_o
$o : R \times S \times M_{r_o} \rightarrow \{0,1\}$	*observation function* determining if a message in m observa ble by r_o will be omitted (0) or detected (1) when a fault scenario s occurs
$b : R \times S \rightarrow 2^{M_{r_o}}$	*BDD function* constructing the Binary Decision Diagram $b(r_o, s)$ for an observing resource r_o and a fault scenario s
$\varphi_{r_o,s} : 2^{M_{r_o,s}} \rightarrow \{0,1\}$	*diagnosis function* indicating if a fault scenario s has been diagnosed (1) or not (0)

3.4.1.1 System Analysis

At first, we need to define all message streams which can be detected by each specific observing resource. If r_o is attached to a bus then, besides the own dedicated messages streams, M_{r_o} usually contains all other streams on this bus as well. For example, in Fig. 3.1, the message set for r_3 is defined as $M_{r_3} = \{m_3, m_4, m_5\}$.

Now, for each message stream $m \in M_{r_o}$, all resources $r \in R$ shall be identified which could lead to the loss of m when affected by a permanent fault. Within our framework, *cause functions* $c(r_o, m)$ are used to assign these resources. In particular, a cause function performs a Depth-First Search (DFS) on the graph G_R representing the system architecture, starting at the observing resource r_o. As all dependencies in the application graph G_P are taken into account, also remote resources which are not directly attached to a respective bus can be identified as causes. In our example architecture the cause function $c(r_3, m_3)$ will result in the set $\{r_1, bus_1, r_2, gw, bus_2\}$.

Using the cause functions, it shall be determined which message streams can be affected by a particular fault scenario $s \in S$, e.g., a faulty resource, and which not. For this, we use *observation functions* $o(r_o, s, m)$, formally defined in Equation (3.2) and explained below.

$\forall r_o \in R, s \in S, m \in M_{r_o}$:

$$o(r_o, s, m) = \begin{cases} 1, \text{ if } c(r_o, m) \cap s = \emptyset \\ 0, \text{ otherwise} \end{cases} \tag{3.2}$$

An observation function indicates if any of the resources which can cause the interruption of a particular message stream m are also contained within a fault scenario s. It delivers 1 if a fault scenario s and a cause function $c(r_o, m)$ have no common intersection, else it returns 0. In other words, $o(r_o, s, m) = 1$ indicates that a message stream m will not be affected by a permanent fault in s (observation of m is *true*) while $o(r_o, s, m) = 0$ means that a fault in s will cause an interruption of m (observation of m is *false*). For example, in the specification in Figure 3.1, an observation function for the fault scenario $s_1 = \{r_1\}$ and the observing resource $r_o = r_3$ will result in $o(r_3, s_1, m_3) = 0$, $o(r_3, s_1, m_4) = 0$ and $o(r_3, s_1, m_5) = 1$, as only m_3 and m_4 are affected by a defective ECU r_1.

Although each observation function unambiguously defines received and omitted messages for a particular fault scenario and observing resource, it comprises all detectable messages from M_{r_o}. However, our goal is to find representations of diagnosis functions $\varphi_{r_o,s}$ which not only provide low diagnosis times but also a low computational effort which, in turn, can be achieved when fewer message streams have to be monitored. In this regard, we use BDDs to efficiently encode the observations determined by $o(r_o, s, m)$. As described in [16], a BDD represents Boolean functions in the form of propositional directed acyclic graphs and enables an easy and efficient manipulation and analysis of the underlying data structure. Each node in a BDD represents a binary variable and the terminal nodes indicate the function result, i.e., 0 or 1. An example for a BDD is shown in Figure 3.4, which will be discussed later. To obtain one of

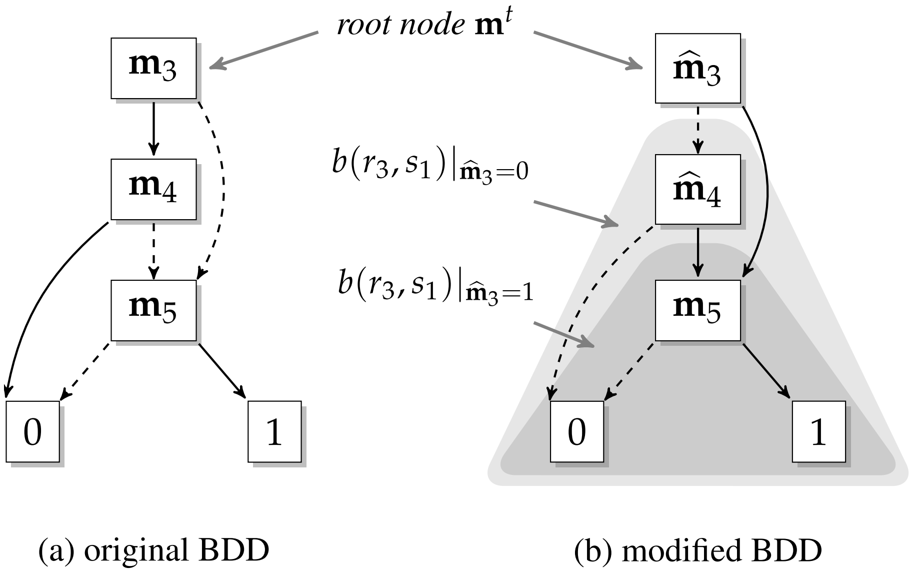


Figure 3.4: *Illustration of the original (a) and the modified (b) BDD $b(r_3, s_1)$ representing the diagnosis function for a faulty r_1 and an observing r_3 in the specification in Figure 3.1. A solid line denotes a received message, a dashed line denotes an omitted message. Note, that the edges for auxiliary variables $\widehat{\mathbf{m}}$ are inverted.*

potentially multiple variable assignments for a positive result of the Boolean function, one has to follow a path from the root node to the 1-terminal and set each variable according to the outgoing edge (usually, 1 for a solid line and 0 for a dashed line). Equation (3.3) shows the formal definition for the construction of BDDs for all observing resources r_o with the corresponding observed fault scenarios s.

$\forall r_o \in R, s \in S:$

$$b(r_o, s) = \bigwedge_{\tilde{s} \in S, s \neq \tilde{s}} \Big(\big(\bigvee_{\substack{m \in M_{r_o} \\ o(r_o,s,m)=0 \\ o(r_o,\tilde{s},m)=1}} \neg \mathbf{m} \big) \vee \big(\bigvee_{\substack{m \in M_{r_o} \\ o(r_o,s,m)=1 \\ o(r_o,\tilde{s},m)=0}} \mathbf{m} \big) \Big) \tag{3.3}$$

Basically, within the permanent fault diagnosis, a BDD encodes observation differences between the analyzed scenario s and all other fault scenarios $\tilde{s}$ for a particular observing resource r_o. In this context, messages from a message stream m are modeled one-to-one by the binary variables $\mathbf{m}$. More precisely, $\mathbf{m} = 1$ indicates a *positive observation* and $\mathbf{m} = 0$ indicates a *negative observation* representing a received and an omitted message, respectively.

The implementation (and possibly a more legible description) of the BDD function is given in Algorithm 1, where the variables $\mathbf{b}_{tar}$ and $\mathbf{b}_{aux}$ are used for the target and auxiliary BDDs, respectively, and the values for the observation of a message stream m are represented by the

Algorithm 1 Construction of the BDD for one fault scenarios s

```
Require: $S$, $M_{r_o}$, $o(r_o, \tilde{s}, m)$, $o(r_o, s, m)$, $r_o \in R$, $s \in S$
Return: $\mathbf{b}_{tar}$                         ▷ Boolean function representing the target BDD
 1: function $b(r_o, s)$
 2:     $\mathbf{b}_{tar} = 1$                            ▷ initialization of a new target BDD
 3:     for all $\tilde{s} \in S$ and $\tilde{s} \neq s$ do
 4:         $\mathbf{b}_{aux} = 0$                        ▷ initialization of a new auxiliary BDD
 5:         for all $m \in M_{r_o}$ do
 6:             if $o(r_o, s, m) \neq o(r_o, \tilde{s}, m)$ then
 7:                 if $o(r_o, s, m) == 0$ then
 8:                     $\mathbf{b}_{aux} := \mathbf{b}_{aux} \vee \neg L(m)$ ▷ disjunction of a negated message li-
 9:                                                 teral for a negative observation
10:                 else
11:                     $\mathbf{b}_{aux} := \mathbf{b}_{aux} \vee L(m)$  ▷ disjunction of a non-negated message
12:                                                  literal for a positive observation
13:                 end if
14:             end if
15:         end for
16:         $\mathbf{b}_{tar} := \mathbf{b}_{tar} \wedge \mathbf{b}_{aux}$      ▷ conjunction of target BDD with auxiliary BDD
17:     end for
18: end function
```

binary literals $L(m)$. Following the definition in Equation (3.3) as well as Algorithm 1, the function $b(r_3, s_1)$ for our example specification will result in the BDD illustrated in Figure 3.4a.

3.4.1.2 Diagnosis Function Generation

The immediate way to retrieve a Boolean function from a BDD is to disjunctively combine all possible paths towards the 1-terminal. However, as this procedure may result in non-minimal solutions, for the eventual generation of diagnosis functions we make use of a *minimum cut* algorithm which excludes redundant paths within the BDD. The recursive computation of the algorithm is defined in Equation (3.4) and illustrated in Algorithm 2.

$$\begin{aligned} &\text{MinCut}\left(b(r_o, s)\right) = \\ &\qquad \{C \cup \{\mathbf{m}^t\} \mid C \in \text{MinCut}\left(b(r_o, s)|_{\mathbf{m}^t=0}\right) \setminus \text{Cut}\left(b(r_o, s)\,|_{\mathbf{m}^t=1}\right)\} \\ &\qquad \cup\ \text{MinCut}\left(b(r_o, s)|_{\mathbf{m}^t=1}\right) \end{aligned} \tag{3.4}$$

Here, the variable $\mathbf{m}^t$ denotes the root node of $b(r_o, s)$, and its two sub-BDDs on the 1-path and 0-path are referred to as $b(r_o, s)|_{\mathbf{m}^t=1}$ and $b(r_o, s)|_{\mathbf{m}^t=0}$, respectively (compare the shaded areas in Figure 3.4b). Broadly speaking, redundant paths in the BDD are removed by applying the relative complement of $\text{Cut}\left(b(r_o, s)|_{\mathbf{m}^t=1}\right)$ in $\text{MinCut}\left(b(r_o, s)|_{\mathbf{m}^t=0}\right)$. The operation eliminates the nodes which reside on paths towards the 0-terminal in $b(r_o, s)|_{\mathbf{m}^t=1}$. Additional explanations as well as the proof of correctness of the algorithm can be found in [122].

An essential prerequisite for the minimal cut algorithm is the *monotonicity* of the analyzed Boolean function, which, simply said, restricts the function to using non-negated variables only. We can achieve monotonicity by substituting negative observations with positive auxiliary variables $\widehat{\mathbf{m}}$ and switch them back after the calculation is finished. The auxiliary variables invert the edges of their nodes, as can be seen in the modified BDD in Figure 3.4b.

In principle, the minimal cut algorithm provides sets of Boolean variables representing *minterms* (i.e., conjunctions of binary variables) in a Disjunctive Normal Form (DNF) function. However, as our goal is to analyze the *sums* of message arrival curves, it is more beneficial to have diagnosis functions in a Conjunctive Normal Form (CNF). An uncomplicated circumvention of this problem can be achieved by applying *De Morgan's laws*, which describe the transformation between disjunctive and conjunctive expressions via negations of terms. More precisely, we modify the algorithm to follow paths to the 0-terminal instead of the 1-terminal and interpret the resulting minterms as *maxterms* (i.e., disjunctions of binary variables).

The entire process is described in detail in Algorithm 2. There, the function *addMinCuts* recursively cuts subtrees from the BDD in order to find minimal sets of nodes leading to 0. For this, four cases are distinguished which correspond to different states of the 0-edge and 1-edge subtrees (lines 7, 15, 21 and 25). Finally, the algorithm provides a family of sets $\mathcal{B}_{tar}$, each of which contains the observation variables of one particular maxterm in a CNF function. The latter one represents the desired diagnosis function $\varphi_{r_o,s}$ and its corresponding formal description is shown in Equation (3.5).
$\forall r_o \in R, s \in S$:

$$\varphi_{r_o,s}(M_{r_o,s}) = \bigwedge_i \Phi_i(M_i) \quad , \quad i \in \mathbb{N} \tag{3.5a}$$

where
$$\Phi_i(M_i) = (\bigvee_{m \in M_i^-} \neg\mathbf{m}) \vee (\bigvee_{m \in M_i^+} \mathbf{m}) \tag{3.5b}$$

and
$$M_i = M_i^- \cup M_i^+ \quad , \quad \bigcup_i M_i = M_{r_o,s} \tag{3.5c}$$

Here, $\Phi_i(M_i)$ defines the i-th *maxterm* in $\varphi_{r_o,s}$ that consists of positive and negative message observations (Equation (3.5b)). The sets M_i^- and M_i^+ contain the omitted and received message streams, respectively, for the i-th maxterm (Equation (3.5c)). Relating to the example specification in Figure 3.1, the diagnosis function for the observing resource r_3 and a faulty r_1 is defined as $\varphi_{r_3,s_1} = (\neg\mathbf{m}_3 \vee \neg\mathbf{m}_4) \wedge \mathbf{m}_5$.

Algorithm 2 Recursive function for an efficient determination of maxterm sets from a BDD

```
Require: $\mathbf{b}_{tar}$
Return: $\mathcal{B}_{tar}$   ▷ Set of sets representing maxterms in final diagnosis function
1: function $addMinCuts(\mathbf{b}_{tar})$
2:     $\mathcal{B}_{tar} = \emptyset, \mathcal{B}_{lo} = \emptyset, \mathcal{B}_{hi} = \emptyset$   ▷ initialize sets of sets
3:     $\mathbf{m}^t = source(\mathbf{b}_{tar})$   ▷ get source node of BDD
4:     $\mathbf{b}_{lo} = low(\mathbf{b}_{tar})$   ▷ get subtree following the 0-edge of $\mathbf{m}^t$
5:     $\mathbf{b}_{hi} = high(\mathbf{b}_{tar})$   ▷ get subtree following the 1-edge of $\mathbf{m}^t$
6:     switch state$(\mathbf{b}_{lo}, \mathbf{b}_{hi})$   ▷ different cases for different subtree states
7:         case $\{\mathbf{b}_{lo} \neq 0 \wedge \mathbf{b}_{hi} \neq 1\}$
8:             $\mathcal{B}_{lo} = addMinCuts(\mathbf{b}_{lo})$   ▷ add mincuts of 0-edge subtree to $\mathcal{B}_{lo}$
9:             $removeCuts(\mathcal{B}_{lo}, \mathbf{b}_{hi})$   ▷ remove cuts from $\mathcal{B}_{lo}$ leading to 1 in $\mathbf{b}_{hi}$
10:            for all $\mathrm{B} \in \mathcal{B}_{lo}$ do
11:                $\mathrm{B} = \mathrm{B} \cup \{\mathbf{m}^t\}$   ▷ add root node to each maxterm B in $\mathcal{B}_{lo}$
12:            end for
13:            $\mathcal{B}_{hi} = addMinCuts(\mathbf{b}_{hi})$   ▷ add mincuts of 1-edge subtree to $\mathcal{B}_{hi}$
14:        end case
15:        case $\{\mathbf{b}_{lo} \neq 0 \wedge \mathbf{b}_{hi} = 1\}$
16:            $\mathcal{B}_{lo} = addMinCuts(\mathbf{b}_{lo})$   ▷ add mincuts of 0-edge subtree to $\mathcal{B}_{lo}$
17:            for all $\mathrm{B} \in \mathcal{B}_{lo}$ do
18:                $\mathrm{B} = \mathrm{B} \cup \{\mathbf{m}^t\}$   ▷ add root node to each maxterm in $\mathcal{B}_{lo}$
19:            end for
20:        end case
21:        case $\{\mathbf{b}_{lo} = 0 \wedge \mathbf{b}_{hi} \neq 1\}$
22:            $\mathcal{B}_{hi} = addMinCuts(\mathbf{b}_{hi})$   ▷ add mincuts of 1-edge subtree to $\mathcal{B}_{hi}$
23:            $\mathcal{B}_{hi} = \mathcal{B}_{hi} \cup \{\{\mathbf{m}^t\}\}$   ▷ add root node as maxterm in $\mathcal{B}_{lo}$
24:        end case
25:        case $\{\mathbf{b}_{lo} = 0 \wedge \mathbf{b}_{hi} = 1\}$
26:            $\mathcal{B}_{tar} = \mathcal{B}_{tar} \cup \{\{\mathbf{m}^t\}\}$   ▷ add root node to $\mathcal{B}_{tar}$
27:        end case
28:    end switch
29:    $\mathcal{B}_{tar} = \mathcal{B}_{lo} \cup \mathcal{B}_{hi}$   ▷ unify the calculated sets and return the result
30:    return $\mathcal{B}_{tar}$
31: end function
```

3.4.2 Fault Diagnosis Optimization: Formulation

The proposed diagnosis functions are the foundation for an efficient decentralized diagnosis of permanent faults. In a runtime system (e.g., the automotive E/E architecture), each observing resource will be provided with a set of diagnosis functions for different fault scenarios. By monitoring the bus with respect to the message arrival times and updating the message observations within the diagnosis functions accordingly, an observing resources can reliably detect faulty resources. However, as the number of monitored message streams affects the computational performance of the system, we propose an optimization-based determination of minimal diagnosis times for different lengths of the corresponding diagnosis functions, i.e., numbers of monitored message streams. Here, the correctness and accuracy of the diagnosis shall not be degraded by excluding particular message streams from being monitored.

As discussed in Section 3.3, diagnosis times are defined by both the periods and the worst-case jitters of the corresponding message streams in $M_{r_o,s}$. Their determination and optimization is based on the sets and variables listed below.

$\mathbf{t}_{\varphi_{r_o,s}} \in \mathbb{R}$	variable defining the diagnosis time of a diagnosis function $\varphi_{r_o,s}(M_{r_o,s})$
$\Phi \in \varphi_{r_o,s}$	one disjunction (maxterm) of a set of binary variables $\mathbf{m}$ within a diagnosis function $\varphi_{r_o,s}(M_{r_o,s})$
$N \in \mathbb{N}$	number of consecutively omitted messages in a message stream (equivalent to the number of omitted periods)
$R_F \subseteq R$	set of all faulty resources that is a subset of all resources R
M^-, M^+	sets of message streams in $M_{r_o,s}$ with omitted (M^-) and received (M^+) messages
α^l_m	lower arrival curve for a message stream $m \in M_{r_o,s}$
$\mathbf{s}^k_m$	binary variable describing the k^{th} step of a corresponding lower arrival curve

3.4.2.1 Diagnosis Time Determination

To determine the arrival times of messages, we use so-called arrival curves that are based on an event model introduced in [19]. As a matter of principle, this model represents the earliest and latest possible arrival times of an event as an upper (α^u) and a lower (α^l) arrival curve, respectively. For example, Figure 3.5 illustrates the arrivals of the message streams m_3 and m_4 from Figure 3.1 where, for reasons of clarity, only lower arrival curves α^l_m are depicted. Although in the scope of this work we regard the communication model *periodic-with-jitter*, the use arrival curves allows us to apply more complicated traffic patterns, for instance those

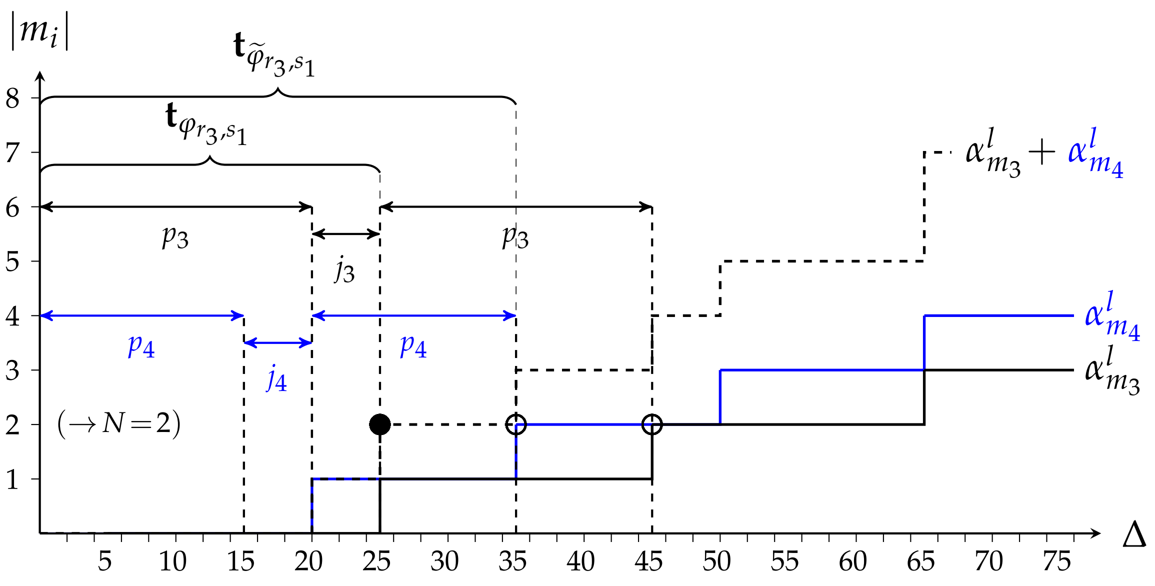


Figure 3.5: *Lower arrival curves (α^l_m) for the message streams m_3 and m_4 from Figure 3.1. They define the minimum number of messages (y-axis) observed during an arbitrary time interval Δ (x-axis). Diagnosis times for the single message observations (rings) as well as the reduced diagnosis time (dot) defined by the sum of the curves (dotted line) for $N = 2$, are highlighted.*

containing bursts or even non-periodic processes. A more detailed explanation of arrival curves is presented in the context of a message-based security diagnosis in Chapter 4.

Considering, that for each message stream $m \in M_{r_o,s}$ the arrival curves are known through the corresponding periods and jitters, then the set M^+ contains all messages whose arrival times are above their lower arrival curve α^l_m and M^- holds all messages with arrival times below α^l_m. Here, intersections are excluded for reasons of consistency of the diagnosis, which is defined in Equation (3.6).

$$M^- \cap M^+ = \emptyset \tag{3.6}$$

Now, the general diagnosis condition can be formulated in Equation (3.7).

$$\forall r_o \in R, \breve{m} \in M^-, \forall \widehat{m} \in M^+ :$$

$$\Big(c(r_o, \breve{m}) \cap R_F \neq \emptyset\Big) \wedge \Big(c(r_o, \widehat{m}) \cap R_F = \emptyset\Big) \tag{3.7}$$

The condition implies that each faulty resource must result in omitted messages while, at the same time, there can be no message loss from a functioning resource. Based on this general premise, we can formulate an ILP which determines the minimum diagnosis time of a diagnosis function. The objective of the optimization problem is to minimize the variable $\mathbf{t}_{\varphi_{r_o,s}}$

representing the diagnosis time of $\varphi_{r_o,s}(M_{r_o,s})$. The Objective (3.8a) and the corresponding Constraints (3.8b) through (3.8g) are defined below.

$\forall m \in M_{r_o,s},\ \mathbf{s}_m^k \in \{0,1\},\ \mathbf{t}_{\varphi_{r_o,s}} \in \mathbb{R}:$

$$\textit{minimize} \quad \mathbf{t}_{\varphi_{r_o,s}} \tag{3.8a}$$

subject to:

$$\forall \Phi \in \varphi_{r_o,s} : \sum_{m \in M^-} \alpha_m^l \geq N \tag{3.8b}$$

$$\forall \Phi \in \varphi_{r_o,s} : \sum_{m \in M^+} \alpha_m^l \geq 1 \tag{3.8c}$$

$$\alpha_m^l = \sum_{k=1}^{N} \mathbf{s}_m^k \tag{3.8d}$$

$$\mathbf{s}_m^k \geq \mathbf{s}_m^{k+1} \tag{3.8e}$$

$$\mathbf{t}_{\varphi_{r_o,s}} \geq (p_m + j_m) \cdot \mathbf{s}_m^1 + \sum_{k=2}^{N} p_m \cdot \mathbf{s}_m^k \tag{3.8f}$$

$$\mathbf{t}_{\varphi_{r_o,s}} < (p_m + j_m) + p_m \cdot \mathbf{s}_m^1 + \sum_{k=2}^{N} p_m \cdot \mathbf{s}_m^k \tag{3.8g}$$

The Constraints (3.8b) and (3.8c) state that the message arrivals defined by sums of lower arrival curves α_m^l must add up to at least N for the omitted messages ($m \in M^-$) and at least 1 for received messages ($m \in M^+$) in order to identify the corresponding observations as negative and positive, respectively. For this, each maxterm Φ in a diagnosis function is regarded separately. As can be seen in the following Constraints (3.8d) through (3.8g), a lower arrival curve itself is defined by means of a monotonically increasing step function with time $\mathbf{t}$ representing the independent variable of the step function. More precisely, Constraint (3.8d), allocates the single steps described by binary variables $\mathbf{s}_m^k$ as a sum to an arrival curve. It delivers 1 for the latest arrival time of the k^{th} message instance from stream m and returns 0 otherwise. Constraint (3.8e) is necessary to guarantee that all previous steps are set before a new step can be added to the arrival curve. Finally, the last two Constraints (3.8f) and (3.8g) define the boundaries for the time variable $\mathbf{t}_{\varphi_{r_o,s}}$ which are depending on the predefined messages periods p_m and worst case jitters j_m. Here, the lower and upper limits are regarded for each single step of the arrival curve.

The consideration of multiple message streams in the diagnosis function $\varphi_{r_o,s}$ when monitoring omitted messages is an essential aspect in our approach as it allows to reduce the diagnosis time for particular functions. The principle is illustrated in Figure 3.5, where the sum of the arrival curves $\alpha^l_{m_1}$ and $\alpha^l_{m_3}$ results in a minimal diagnosis time $\mathbf{t}_{\varphi_{r_3,s_1}} = 25$ ms for $N = 2$.

The ILP (3.8) is formulated for a *periodic with jitter* representation of the message stream, which is common for automotive real-time communication. Nevertheless, the proposed diagnosis approach can be applied to other arrival curve descriptions as well in which case the ILP can be easily adapted.

3.4.2.2 Trade-Off Analysis

Based on the determination of minimal diagnosis times, we want to evaluate trade-offs between the diagnosis time and the number of monitored message streams. For an efficient approach, low values are preferable for both parameters in order to guarantee a fast diagnosis of a fault scenario and a low computational overhead. However, it must be considered that a reduced message set may (depending on the size of its reduction) result in an increase of the diagnosis time, defined by Equation (3.9).

$$\mathbf{t}_{\widetilde{\varphi}_{r_o,s}(\widetilde{M}_{r_o,s})} \geq \mathbf{t}_{\varphi_{r_o,s}(M_{r_o,s})}\ , \quad \text{with} \quad \widetilde{M}_{r_o,s} \subseteq M_{r_o,s} \tag{3.9}$$

The trade-off analysis is performed by successively reducing the cardinality of the message set $M_{r_o,s}$ and repeating the diagnosis time optimization for each reduced set $\widetilde{M}_{r_o,s}$. For this, the ILP is extended with an additional binary variable $\mathbf{c}_m$ which determines whether a message m shall be considered for the optimization (1) or not (0). This is illustrated in the optimization problem below, where the objective function is defined in Equation (3.10a), and two additional Constraints (3.10b) and (3.10c) are used for the message stream reduction. Note that all constraints defined for the ILP (3.8) also hold in this ILP (3.10).

$\forall k \in \{|M_{r_o,s}|, \ldots, 1\}$:
$\forall \widetilde{M}_{r_o,s} \subseteq M_{r_o,s},\ |\widetilde{M}_{r_o,s}| = k,\ \mathbf{c}_m, \mathbf{s}_m \in \{0,1\},\ \mathbf{t}_{\widetilde{\varphi}_{r_o,s}} \in \mathbb{R}$:

$$\textit{minimize} \quad \mathbf{t}_{\widetilde{\varphi}_{r_o,s}(\widetilde{M}_{r_o,s})} \tag{3.10a}$$

subject to:

$$\mathbf{c}_m - \mathbf{s}^1_m \geq 0 \tag{3.10b}$$

$$\sum_{m \in M_{r_o,s}} \mathbf{c}_m = |\widetilde{M}_{r_o,s}| \tag{3.10c}$$

This stage of our framework results in diagnosis functions with a reduced number of monitored message streams but still capable of correctly diagnosing the corresponding fault scenario.

Regarding the example in Figure 3.1, the optimization removes m_3 and, hence, results in a more compact diagnosis function $\widetilde{\varphi}_{r_3,s_1} = \neg \mathbf{m}_4 \wedge \mathbf{m}_5$. However, the diagnosis time $\mathbf{t}_{\widetilde{\varphi}_{r_3,s_1}} = 35$ ms for $N=2$ increases by 10 ms with respect to the original function (compare also Figure 3.5).

3.5 Diagnosis Evaluation

To provide a significant evaluation of the proposed diagnosis approach, a selection of 100 synthetics test cases and a case study, both based on automotive E/E architectures, are used. The test cases comprise systems with more than 20 ECUs, several gateway-connected buses and functions with different task and message stream numbers, as indicated in Table 3.1. Overall, up to 24 fault scenarios per test case have been determined and each of them has been analyzed for all possible observing resources as well as for 1 to 5 omitted message periods. For the presented experimental results, each fault scenario consists of one resource, such that $s_i = \{r_i\}$. All experiments including the system analysis, diagnosis function generation and optimization were carried out on an Intel Core i5 with 2.6 GHz and 8 GB RAM. For the ILP-based optimization, GUROBI version 5.6 was used as solver [39].

3.5.1 Synthetic Test Cases

3.5.1.1 Diagnosis Times

To demonstrate the general feasibility of the permanent fault diagnosis framework, Figure 3.6 illustrates diagnosis times as function of the number of monitored message streams. In the graph, each vector represents the duration-complexity relation for the diagnosis of a permanent fault. It starts with a minimal number of monitoring message streams and a maximal diagnosis time and points towards a minimal observation time with a maximum number of streams. As defined in Sec. 3.4.2, the diagnosis time reduction represents the difference in observation time between the diagnosis functions $\widetilde{\varphi}_{r_o,s}$ with reduced numbers of message streams and the original function $\varphi_{r_o,s}$. The experiments have been performed with all generated diagnosis functions for each analyzed test case, however, results where a reduction of the diagnosis time was not possible or less than 50 ms have been omitted to maintain legibility. The resulting graph indicates that diagnosis functions can be optimized either towards a minimal diagnosis time or towards

Table 3.1: *Essential test case parameters for the permanent fault diagnosis and their minimum and maximum values.*

	ECUs	*Buses*	*Tasks*	*Functions*	*Message Streams*	*Fault Scenarios*	*Omitted Messages*
MIN	2	1	10	2	7	2	1
MAX	23	4	132	25	78	24	5

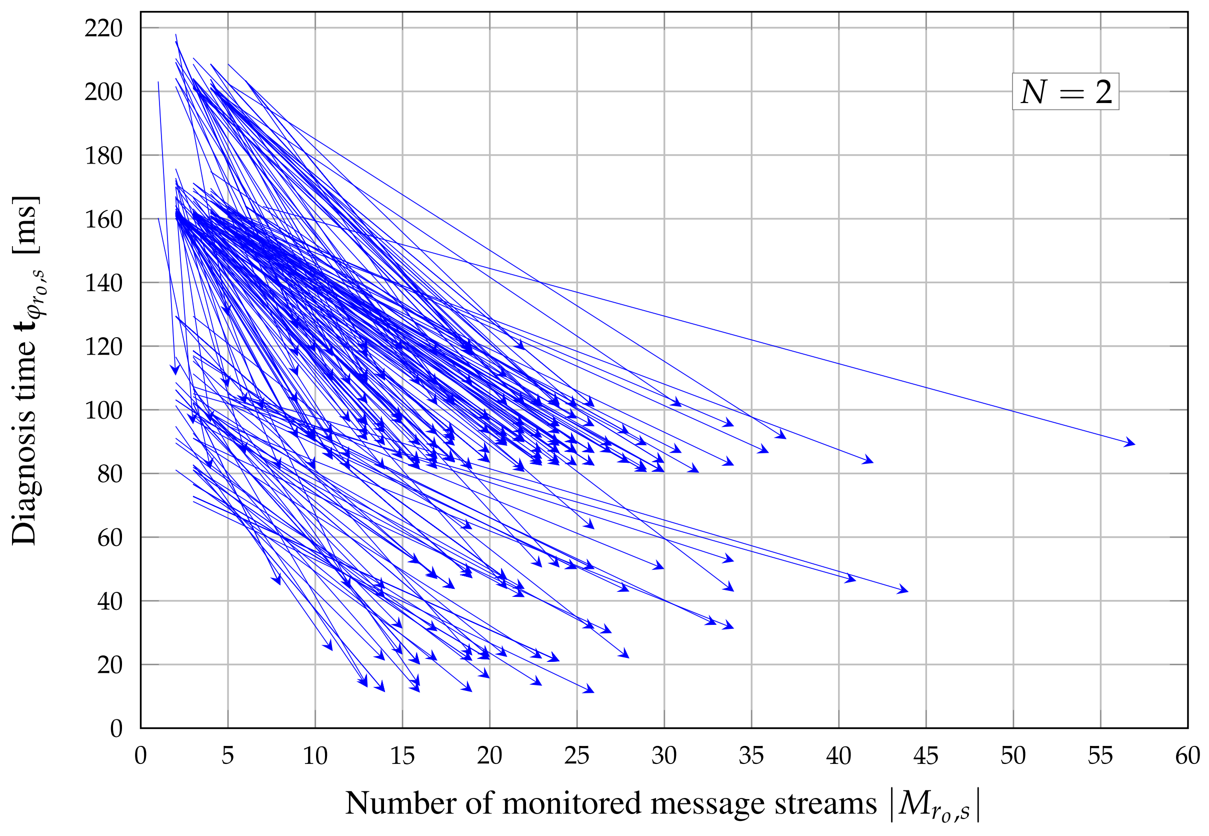


Figure 3.6: *Relation between the diagnosis time and the number of monitored message streams for the permanent faults. Each vector represents the decrease from a maximum to a minimum diagnosis time indicating the corresponding message numbers. The graph comprises results for* $N = 2$ *with a time reduction of more than* 50 *ms.*

a lower computational performance by monitoring fewer message streams. As a consequence, during system design, an emphasis can be put on hard real-time requirements or energy/cost efficiency, depending on the current preferences.

Generally, the amount of diagnosis functions allowing a diagnosis time reduction depends on the system specification of a particular test case. Here, the experiments show that the optimization step can have a large impact by reducing the observation times in some cases by more than 50 % or, on the other hand, removing more than 90 % of the monitored message streams. Furthermore, results with an irreducible diagnosis time provide important information about which messages can be removed from the diagnosis function without affecting the fault diagnosis itself. Such results would be depicted as horizontal vectors in the graph and have been left out for the sake of clarity, as mentioned above.

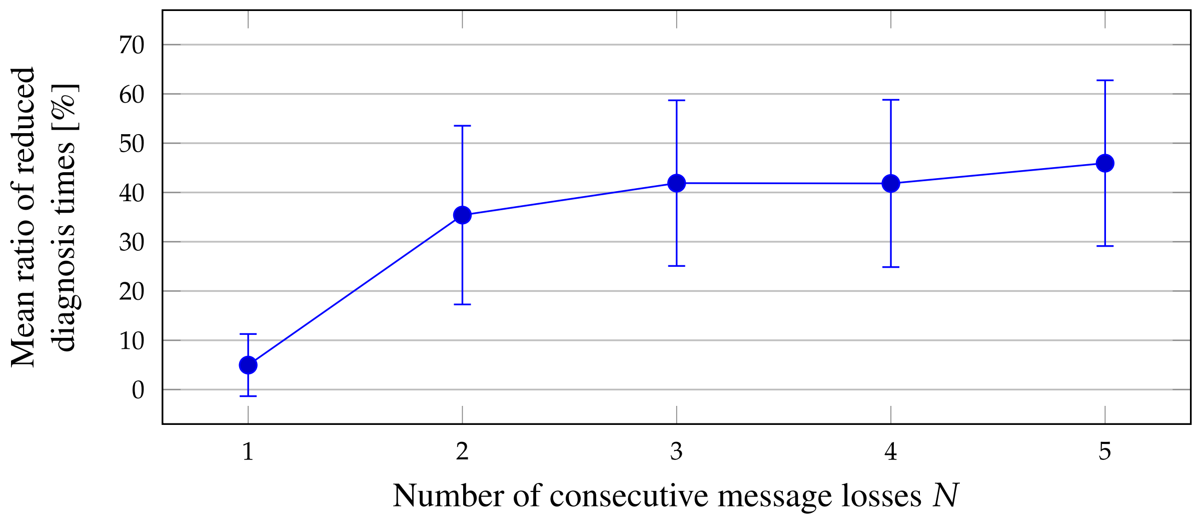


Figure 3.7: *Amount of diagnosis functions with reduced diagnosis times for different numbers of consecutive message losses. The y-axis represents the mean ratio of reduced diagnosis functions to all determined functions and the x-axis indicates numbres of omitted periods. The error bars illustrate the corresponding standard deviations.*

3.5.1.2 Omitted Message Streams

Figure 3.6 depicts the outcome where two message losses imply a permanent fault, i.e., $N = 2$. To investigate the influence of different numbers of omitted periods, Figure 3.7 illustrates the mean ratios of reducible to irreducible diagnosis times for up to 5 omitted periods (i.e. consecutive message losses). It can be seen, that while a diagnosis time improvement is very improbable when only one omitted period is regarded, the amount of reducible diagnosis functions strongly increases for two and more omitted periods. This is due to the summation of message arrival curves where for a higher period number more messages can contribute to a potential diagnosis time reduction. For the same reason, an increased number of omitted periods does not necessarily proportionally increase the diagnosis time. Overall, the outcome indicates that considering a higher number of omitted periods might be an important design criterion, especially, as the certainty for detecting a permanent fault increases with the number of consecutively observed message losses, as has been discussed in Section 3.3.1.

The results in Figure 3.6 and 3.7 are based on approximately 8000 distinct diagnosis functions for each N. Altogether, based on 100 test cases, a total of 40380 distinct fault observations have been analysed from which 12551 (31 %) could be optimized with respect to diagnosis time and number of monitored message streams, respectively.

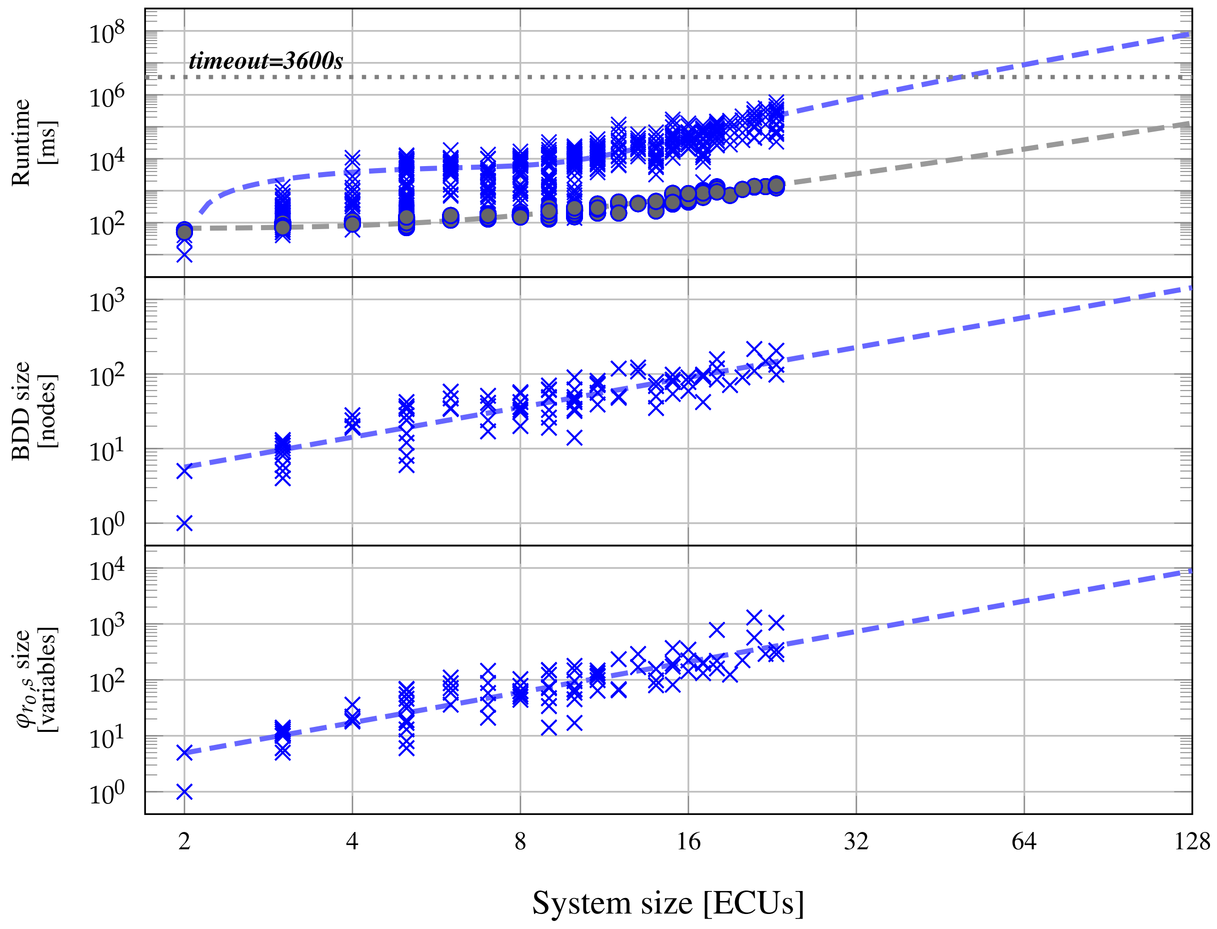


Figure 3.8: *Performance analysis of the presented permanent fault diagnosis approach. The three graphs illustrate how the size of an architecture (i.e., number of ECUs) influences the runtimes of the system analysis (top, o-marks) and optimization (top, x-marks), the BDD size (middle), and the size of the diagnosis functions (bottom). Additionally, dashed lines represent trend curves indicating the estimated behavior for larger systems.*

3.5.1.3 Framework Performance

As the fault diagnosis determination and optimization are performed off-line and the target system only stores the diagnosis functions $\varphi_{r_o,s}(M_{r_o,s})$, a fast and memory-efficient computation for our framework is not directly required to guarantee a fast diagnosis at runtime. However, both the BDD-based diagnosis function determination and their ILP-based optimization might exhibit relatively large time complexities in the number of regarded message streams. For this reason, it is worth to analyze the corresponding performance behavior of the diagnosis framework.

In this context, three graphs in Figure 3.8 show the computation times (top), the BDDs sizes (middle), and the diagnosis function sizes in terms of variable numbers (bottom). All axes are in logarithmic scale and the system sizes are depicted for up to 128 ECUs to indicate the estimated scalability.

The runtime results in the upper graph are divided into the system analysis (i.e., the generation of diagnosis functions) (o-marks) and the diagnosis function optimization (x-marks). Here, the timeout for one test case after which the computation is canceled is set to one hour. It can be seen, that even for systems beyond 20 ECUs both runtime values do not exceed 100 s and 600 s, respectively. The BDD sizes in the middle graph are measured in number of nodes per tree and their extent reaches slightly above 200 at the most. The lower graph shows the maximum sizes of non-optimized diagnosis functions. Although for some large systems the numbers can exceed 1000 variables, in many cases our optimization can reduce this values by a factor of 10 or more without affecting the diagnosis time. The functions can be minimized even further when taking an increase of the diagnosis time into account.

Furthermore, each of the three graphs contains dashed curves indicating an estimated behavior for system sizes up to 128 ECUs and, hence, several hundred message streams. Each curve represents the linear regression for the appropriate measuring points and is determined by means of the method of least squares. Evaluating the corresponding coefficients of determination suggests, that a good estimation for both the system analysis approach and the diagnosis optimization is a polynomial-time complexity in the number of message streams. Regarding the necessary memory capacity, which increases with the number of BDD nodes or diagnosis function variables, our method seems to scale linearly. All four curves indicate a manageable scalability with a worst case optimization time of less than three hours for a system with 128 ECUs. Also the estimated sizes of BDDs and diagnosis functions should not have a too large impact on the performance of the diagnosis framework. However, the presented trend curves are only a rough estimation and cannot replace a proper extension of the test cases for future analyses.

Summing up, the performance results show that a deployment of the diagnosis functions to today's automotive hardware should be feasible in principle and can be further considered for a prototypical implementation as discussed in Section 3.5.3.

3.5.2 Case Study

Finally, a case study shall demonstrate the usability of our method for an in-depth reliability analysis. For this, an assignment of observing resources r_o to fault scenarios s is depicted by a matrix of dots and circles in Figure 3.9.

The case study is based on a realistic automotive E/E architecture comprising 106 tasks and 61 messages and a distributed system with 22 ECUs. The architecture uses 4 buses connected to a gateway (gw) where resources r_1 to r_4, r_5 to r_{12}, r_{13} to r_{15} and r_{16} to r_{22} are sharing a bus, respectively. Message periods can range from 5 ms to 80 ms and their worst case jitters are determined according to the priority and delay calculation for event-triggered systems proposed in [88]. Based on the system specification, our framework generated 268 explicit diagnosis

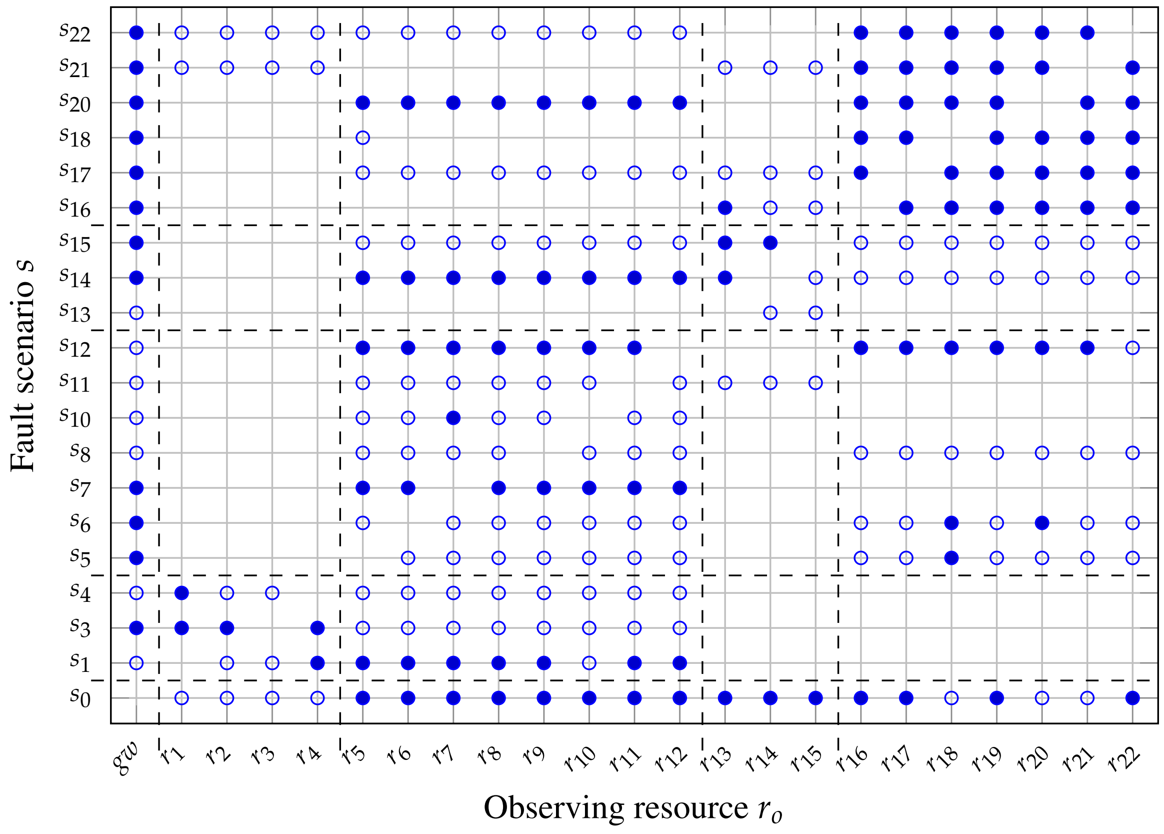


Figure 3.9: *Assignment of the determined diagnosis functions to different fault scenarios (y-axis) and the corresponding observing resources (x-axis). It holds that,* $s_i = \{r_i\}$ *and* $s_0 = \{gw\}$*. The depicted results cover the case study with* $N = 3$*. Diagnosis times can be reduced for* 119 *diagnosis functions (dots) while the remaining* 149 *cases (rings) are irreducible.*

functions, from which 119 allow a diagnosis time reduction (dots) and the remaining ones cannot be minimized (rings). Here, each scenario consists of one potentially faulty ECU and the dashed lines delimit the ECUs which are attached to different buses. For instance, the four groups lying on the diagonal represent cases where the observing resource and the fault scenario are attached to the same bus.

The matrix illustrates, that each fault scenario can be diagnosed by at least two observing resources as well as the gateway (see s_{13}). In many cases, however, the monitoring possibilities are much larger and besides the plain diagnosis, on many resources also optimized functions can be applied. Moreover, we can clearly recognize the ability of our method to diagnose faulty resources (as well as optimize the diagnosis) beyond the limits of the own bus, namely, at network-level. The high density of reducible diagnosis functions in the upper right corner mainly results from an overall larger amount of data exchange between the corresponding resources and, consequently, more freedom for the optimization approach.

On the other hand, the empty areas in the matrix indicate that there are several cases where the permanent fault diagnosis is not applicable. For three resources the suitable fault scenarios (s_2, s_9, s_{19}) are missing completely, as they are not transmitting any diagnosable messages. As a remedy, the proposed fault diagnosis approach could be considered already during system design in order to provide improved diagnosis times and increase the number of diagnosable resources. A corresponding approach is proposed and described in Section 3.6.

In summary, an off-line diagnosis analysis as demonstrated by this case study gives the system designer the possibility to implement the permanent fault diagnosis in a most suitable and efficient way.

3.5.3 Prototypical Implementation

To complement the evaluation of the proposed diagnosis approach, this section presents the design and outcome of a prototypical implementation on a research platform for automotive E/E architectures. The results demonstrate the practical feasibility of the permanent fault diagnosis and provide first measurements of the hardware utilization. For this reason, even though the implementation itself is only a minor part of the research work presented in this thesis, its main results shall not be withheld at this point [3].

3.5.3.1 Research Platform and Conceptual Design

The permanent fault diagnosis is implemented on a research platform for distributed predictable automotive E/E architectures consisting of four ECUs and using Ethernet-based communication for data exchange (see Figure 3.10). Each of the ECUs is represented by a development board with an ARM-based microcontroller [108] (Cortex-M4 with 168 MHz, 1 MB Flash and 196 KB RAM) and using the *Micrium µc/OS-III* as RTOS [74]. Basically, the platform architecture corresponds to that in Figure 3.1 where three processing ECUs are attached to two separate buses connected to an automotive gateway. To model a suitable bus behavior, the switched Ethernet connection uses a multicast function to transmit messages appropriately. The test system runs a distributed steer-by-wire application where sensor data from a steering wheel connected to r_1 and r_2 is first processed on r_2 and then, together with other messages, transmitted via the gateway gw to r_3. The latter ECU is logically connected to a different bus and controls the simulated vehicle wheels (compare right screen in Figure 3.10). One investigated fault scenario includes a permanent fault on r_1. After it is diagnosed, all relevant tasks are automatically switched to r_2 which now takes control over the steering wheel input. Given the minimal diagnosis times of our proposed approach, the fault-tolerant application is able to handle this scenario correctly and with a imperceptible delay for the driver.

One of the main challenges for the implementation of the permanent fault diagnosis is to keep the additional monitoring overhead (i.e., computation time and memory) as low as possible

[3]The implementation of the proposed permanent fault diagnosis and the corresponding performance measurements have been conducted for a Master's Thesis [138] supervised in the context of this thesis.

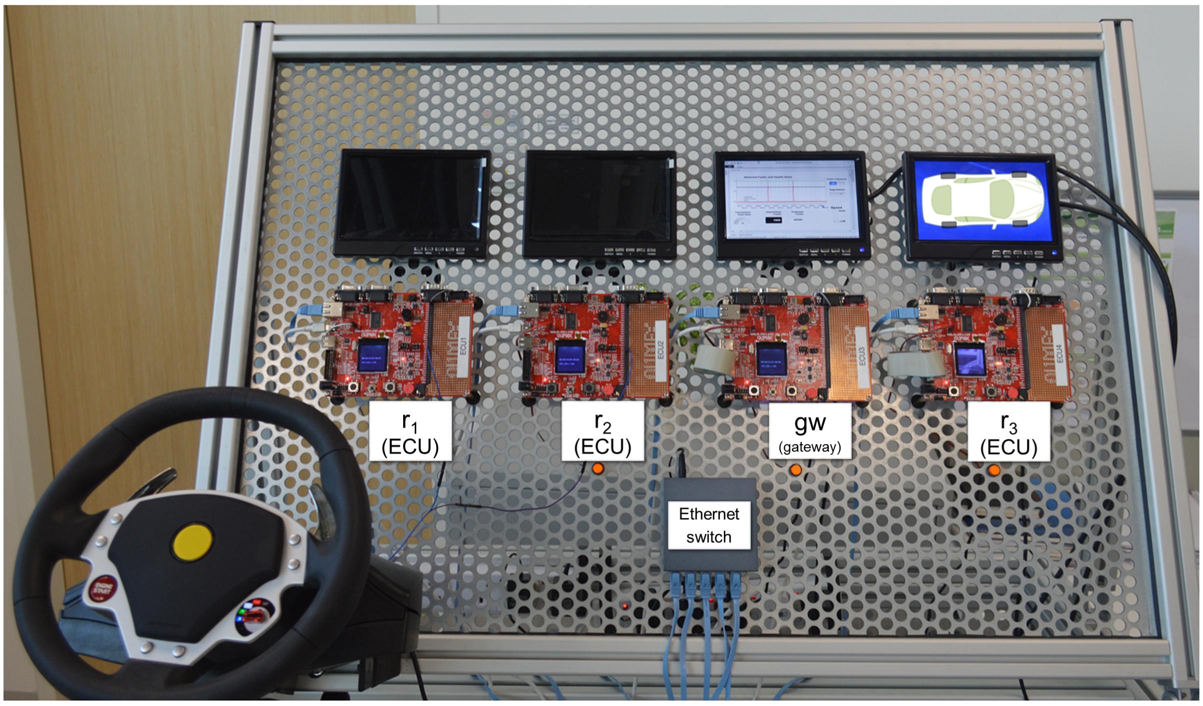


Figure 3.10: *Picture of a research platform for a distributed steer-by-wire application. This setup is used to test an implementation of the proposed permanent fault diagnosis.*

such that it does not restrict or impede the execution of other system and application tasks on the ECUs. For this, [138] investigates possibilities of monitoring message stream receptions on observing resources in order to correctly and efficiently detect message omissions. Two methods are considered in particular, a baseline approach and an extended approach. Broadly speaking, the *baseline approach* applies a simple time-sliced concept where the diagnosis tasks are executed periodically, for instance, one every millisecond. In this context, a diagnosis task mainly analyses the outcome of a diagnosis functions $\varphi_{r_o,s}$ by setting its variables $\mathbf{m}$ according to the current message reception status and calculating the Boolean function. By contrast, for the *extended approach* the diagnosis tasks are executed aperiodically, only at the specified deadline times of the appropriate message streams. This is achieved by using a binary tree (min-heap) which allows to efficiently sort all message data structures according to the earliest absolute deadline. Here, the trade-off to be investigated lies between low memory costs of the basic approach and the low computation time costs of the extended approach.

3.5.3.2 Experimental Results

As our main interest lies in the computational overhead of the permanent fault diagnosis, those results from [138] which prove the feasibility and correctness of the implementation are omitted here. For the overhead evaluation the baseline and the extended monitoring approach are compared. Additionally, to get an indication for the scalability of the implementation, two bus

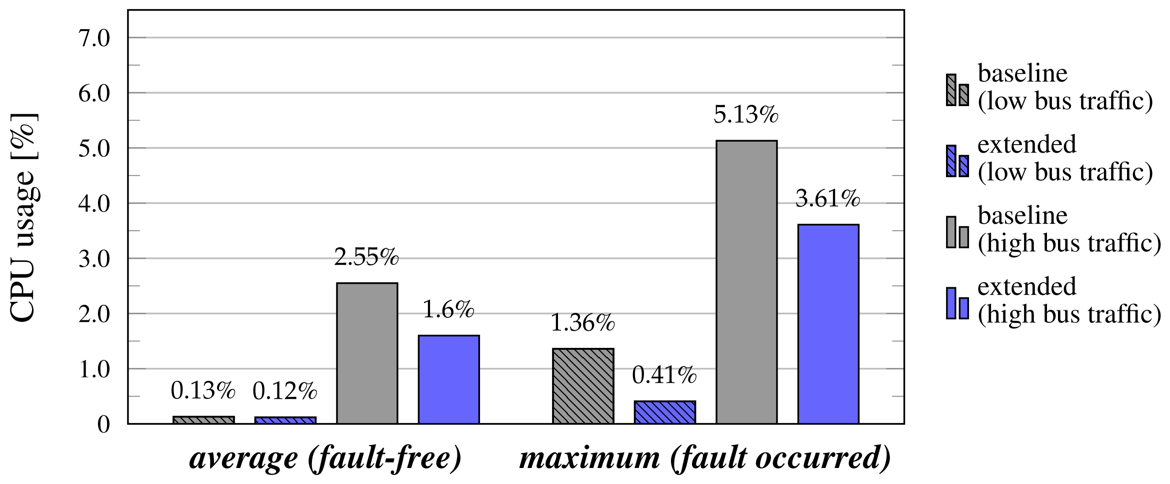


Figure 3.11: *Comparison of the computational overhead of the fault diagnosis implementation. The graph illustrates the average and the maximum CPU utilization for both the baseline approach and the extended approach, respectively. Furthermore, a low and a high network traffic help to investigate the scalability.*

utilization scenarios are considered, a *low traffc* with 5 message streams and a *high traffic* with 103 message streams.

Computational overhead. Figure 3.11 depicts the additional CPU usage on an ECU while performing the permanent fault diagnosis. It illustrates the average overhead during a fault-free operation and and the maximum overhead when a fault occurs. Clearly, in all configurations the baseline approach utilizes more computational resources then the extended approach. This reflects the inefficiency of the periodic inspection of the current message states in contrast to the aperiodic monitoring based on the expected message deadlines.

Generally, in this graph the average utilization results are more relevant, as faults occur rather seldom and not regularly. Here, regarding the scalability (i.e., the number of message streams increase by a factor of 20), the extended approach shows a better behavior causing a computation increase by a factor of about 13, compared to a factor of 19 for the baseline approach.

Memory overhead. Aside from the computation, also a low memory usage is important when dealing with the limited resources in automotive E/E architectures. Figure 3.12 illustrates the memory allocation on an ECU for the relevant diagnosis data (e.g., structures for message streams, maxterms and diagnosis functions). In both traffic scenarios the extended approach shows an increase in memory usage by $40 - 50$ % compared to the basic method. This is mainly caused by the additional data structures and the corresponding memory allocation necessary for implementing binary trees. Nevertheless, even for the high traffic the extended approach utilizes

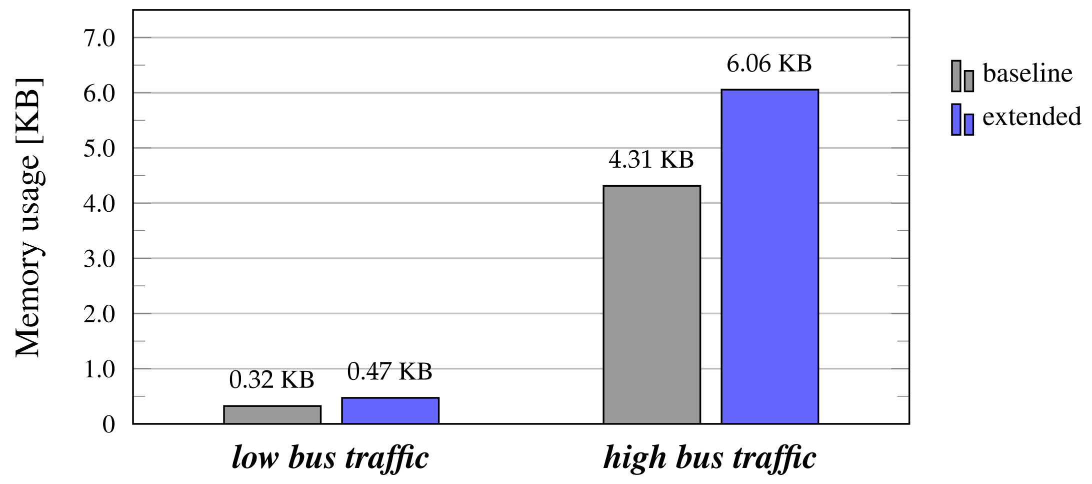


Figure 3.12: *Comparison of the memory allocation of the fault diagnosis implementation. The graph depicts the low and high traffic results for both the baseline and extended monitoring approach.*

only about 6 KB of memory. Regarding scalability, both approaches indicate an increase by a factor of 13 when switching from 5 message streams to 103.

Summing up, for both methods the average computational overheads of 1.6 % and 2.55 %, respectively, seem to be small enough to be applied in automotive ECUs without a necessary increase of the hardware performance. Similarly, the memory costs are in a reasonable range even for systems with low memory resources, for instance, only a few tens of kilobytes.

3.6 Diagnosis-Aware System Design

This section introduces a schedule synthesis approach which specifically considers the permanent fault diagnosis strategy described in this chapter. The approach is based on a time-triggered architecture and aims at a decentralized and non-intrusive communication-based solution. In this context, we propose a system design methodology which optimizes a subsequent permanent fault diagnosis method in terms of the necessary diagnosis time as well as the overall diagnosability. Broadly speaking, in order to increase the system reliability, during schedule synthesis a modified and adapted message distribution is taken into account which additionally considers previously not diagnosable resources. While this method might lead to a slightly increased bandwidth utilization, it clearly improves the overall diagnosis of faulty resources by offering a faster diagnosis covering previously undetectable resources.

Thematically, the diagnosis-aware system design is part of the permanent fault diagnosis. However, due to the specific time-triggered paradigm with its own mathematical formulations it is described as a self-contained topic. Consequently, this section consists of a brief introductory part, a description of the used communication model, the actual scheduling methodology and a case study evaluation.

3.6.1 Motivation

In Chapter 1 it has been discussed how an increasing number of vehicle functions, such as X-by-wire or ADAS, together with their safety-critical and hard real-time requirements call for new safety standards. At the same time, the continuous advancements in the underlying hardware architecture, e.g., shrinking geometries and reduced supply voltages, result in an increased occurrence of faults. The consequence can be clearly seen within the V-model illustrated in Section 1.2.3, where a large part of the development process is dealing with testing of single components and the entire system. The corresponding system integration effort can become very complex and time consuming, especially when addressing event-triggered systems where the communication is not predictable and many different execution scenarios have to be tested in order to guarantee a safe system behavior [68]. By contrast, as time-triggered systems use predefined schedules, the task executions and timings can be easily determined and the system behavior is more deterministic. This allows to use a more systematic testing approach and greatly simplify the verification and validation process, for instance, when different variants of one vehicle are offered [120]. Hence, this deterministic paradigm and its advantages for system safety is one of the main reasons why the automotive industry is slowly shifting from event-triggered towards time-triggered architectures.

During the diagnosis-aware system design, both a time-triggered scheduling and a reliable fault diagnosis method are regarded as part of a common system design approach where the synthesis of a system schedule is explicitly considering a message-based fault diagnosis at runtime. As a matter of principle, besides decreasing the fault diagnosis time and increasing the component coverage, the presented approach can be also beneficial during the design and maintenance phase. There, by providing strictly defined and consistent diagnosis times it can improve and reduce the testing and verification efforts.

3.6.2 Contribution

We propose a novel approach towards an improved permanent fault diagnosis in a distributed network-based system, such as an automotive E/E architecture. It is mainly designed for time-triggered architectures and uses a framework which was previously introduced in the context of schedule integration [118]. Within the proposed diagnosis-aware system design, that framework is extended and adapted for a message-based fault diagnosis.

The main contribution of this section is the modification of a time-triggered system schedule in order to improve a diagnosis of permanent faults at runtime. This covers, first, a determined and reduced diagnosis time and, second, the inclusion of resources which could not be diagnosed earlier. The first goal is achieved by adapting the message transmission times of existing message streams in the network in order to obtain a better distribution for their detection. The second goal uses the available bandwidth of the communication channel to insert auxiliary light-weight diagnostic messages for naturally undetectable resources, for example, those executing reception tasks only. Furthermore, the insertion of additional message streams can be used to resolve infeasible constraints in terms of a too short observation time[4]. Finally, the approach does not compromise the predefined system specifications, such as end-to-end delays of applications. The modification of system schedules in order to enhance a message-based fault diagnosis can be also seen as a first step towards Design for Testability (DFT) which is a relevant topic for the automotive industry.

3.6.3 System Model

Similar to the diagnosis method discussed in Section 3.4, the proposed system design approach uses a graph-based description where software functionality and hardware components are modeled by process graphs and architecture graphs, respectively. The upper part of Figure 3.13 shows a simple example for the specification of a distributed system with three applications a_i and an architecture consisting of three bus-attached resources r_i where the bus is considered an own resource. Here, a process graph $G_P = (P, E_P)$ consists of the vertices $P = T \cup M$, where a process $p \in P$ can define both a tasks $t \in T$ and a message $m \in M$, respectively. The edges E_P represent the corresponding data-dependencies. Accordingly, an architecture graph $G_R = (R, E_R)$ connects resources $r \in R$ (e.g., ECUs, buses and gateways) through architectural links $l \in E_R$. In order to assign processes to the appropriate resources, both graphs share mappings defined by the edges $E_M = (P, R)$.

3.6.3.1 Time-Triggered Scheduling

The system model comprises a fully synchronous *time-triggered* schedule where at design-time each process, i.e., task or message, is assigned a start time which defines the instant of its execution within a periodic cycle. An application, on the other hand, is defined by its end-to-end delay which is described by the interval between the beginning of its first task (e.g., a sensor process) and the end of the last task (e.g, an actuator process). Generally, automotive applications define distributed control functions that rely on maximal end-to-end delays rather than on single deadlines for each task. As a consequence, given a sufficient bandwidth on the communication channel, the start-times of single messages within the end-to-end boundaries might be shifted without affecting the control performance. This aspect is used for the approach proposed here, as our main goal is to modify system schedules in order to improve the diagnosis times and increase the diagnosability without compromising the original constraints specified during system design. In this context, we assume the required maximum diagnosis time to be

[4]Although, *observation time* and *diagnosis time* are usually used interchangeably, in some cases the former term refers to a pure detection of message instances while the latter one indicates the duration of diagnosing a fault scenario.

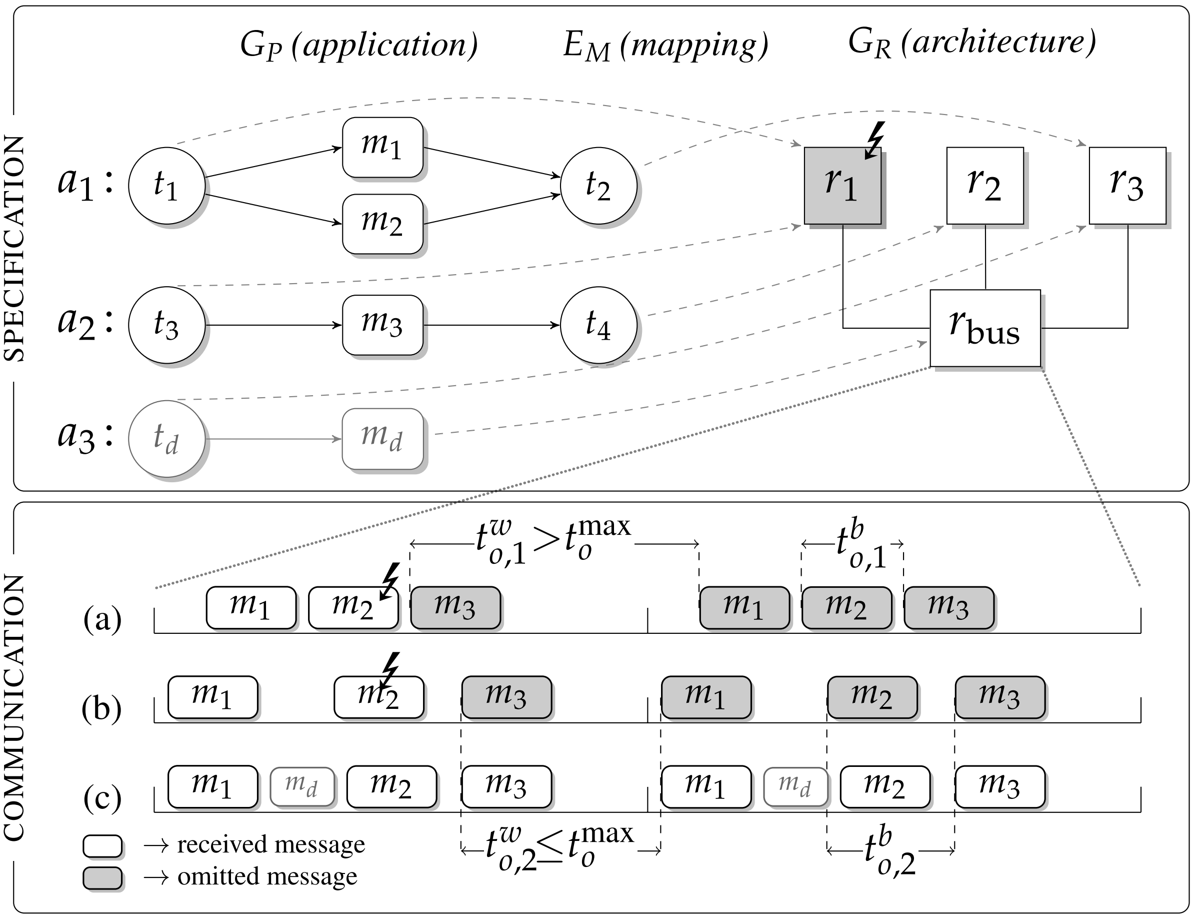


Figure 3.13: *Specification for a distributed system consisting of several applications mapped to an architecture. The bottom part shows the communication on the bus resource for three different schedules, where t^w_o and t^b_o indicate the worst and best observation time, respectively, whereas $t^{\max}_o$ represents its upper bound. A permanent fault occurs on r_1 and prevents a further message transmission.*

specified by the system designer depending on specific application requirements as well as the subsequent fault tolerance strategy.

3.6.3.2 Illustrative Example

The lower part of Figure 3.13 depicts three different allocations of messages on a bus resource for a simple system specification. Each communication pattern is illustrated for two periodic cycles and will be referred to as schedule (a), (b) and (c), respectively. Schedule (a) represents the unaltered message allocation resulting from the schedule synthesis of an original user-defined system specification. The messages m_1 and m_2 are carrying data from task t_1 to task t_2 and, hence, will be transmitted from resource r_1 to r_3, whereas m_3 is sent from r_1 to r_2.

Now, let us assume that during the transmission of m_2, a permanent fault occurs on resource r_1 (symbolized by a *lightning* in Figure 3.13) and resulting in the transmission failure of all following messages. At this point, it shall be also presumed, that the system designer defined a maximum diagnosis time $t_o^{\max}$ which cannot be exceeded. As discussed before, in order to rule out that the resource was only affected by a transient fault, our diagnosis method needs to observe at least two consecutively omitted messages from a message stream whose diagnosis function is targeting the potentially faulty resource. Thus, for schedule (a) the diagnosis of a faulty r_1 would require a worst case observation time of $t_{o,1}^{w}$, which is supposed to be higher than the predefined maximum observation time $t_o^{\max}$. By contrast, the proposed approach uses the idle times on the bus to distribute messages from each resource in such a way that $t_o^{\max}$ between two consecutive messages in a stream is not exceeded. This is demonstrated in schedule (b) where the single messages are distributed more evenly, preventing larger gaps between them. In many cases the proposed method can not only reduce the observation time itself, but it also enables a better overall system predictability by defining un upper bound for the observation time, as in this example where $t_{o,2}^{w} \leq t_o^{\max}$.

So far, only message streams from the original system specification have been considered. This is in accordance with the overall methodology introduced in Section 1.3, that requires a diagnosis method to be as less intrusive as possible. However, a complex distributed system, such as an E/E architecture, can contain a number of resources whose faults might not be detectable with the help of a fully implicit and non-intrusive fault diagnosis approach. For instance, in the specification in Figure 3.13 the resources r_2 and r_3 are both endpoints in the corresponding task communication and, thus, not transmitting messages on their own. For this purpose, in order to improve the diagnosability, the diagnosis-aware system design is able to identify such resources and integrate auxiliary tasks which can broadcast short diagnostic messages, such as the application a_3. These auxiliary tasks are supposed to be lightweight and transmit the diagnostic message in time intervals that do not influence the execution of regular system applications (e.g., once a hyper period). Schedule (c) illustrates this scenario, where the task t_d is mapped to resource r_3 and transmits a diagnostic message m_d. Assuming that, due to a limited bandwidth or other timing constraints resource r_2 remains undiagnosable, then the overall diagnosability of the system in Figure 3.13 has been increased from one resource to two resources.

3.6.4 Methodology

This section formally describes the proposed diagnosis-aware system design, divided into the general requirements, message stream adaptations and message insertion. Besides the process and architecture graphs introduced in Section 3.6.3, it mainly uses parameters and variables describing time-triggered schedules, as listed below.

$p \in P$	system process from the set $P = T \cup M$ that is either a task $t \in T$ or a message $m \in M$

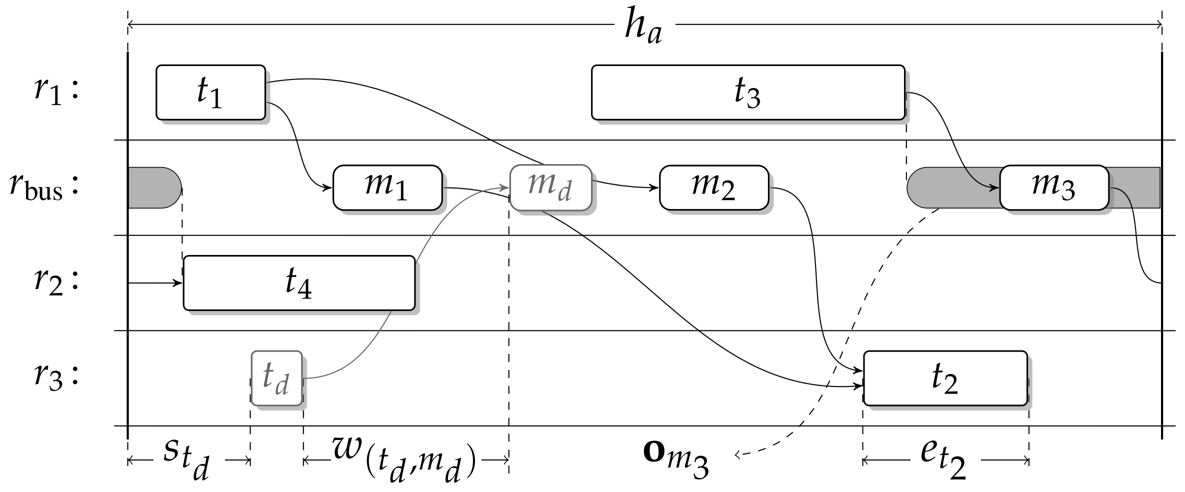


Figure 3.14: *Example for a time-triggered schedule of a system based on the specification in Figure 3.13. The schedule excerpt depicts one application period h_a showing the appropriate positions and data dependencies of all processes. Moreover, it illustratively describes the role of the main variables and parameters used for the diagnosis-aware system design.*

$w_{(\tilde{p},p)} \in \mathbb{R}$	waiting time between the end of a process $\tilde{p}$ and the start of a data-dependent process p
$\mathbf{o}_p \in \mathbb{R}$	variable describing the offset interval of a single process p
$\mathbf{o}_a \in \mathbb{R}$	variable describing the offset interval of an entire application a
$h_a \in \mathbb{R}$	period defining the recurring execution of an application a
$s_p, e_p, h_p \in \mathbb{R}$	start time, execution time, and period defining a single process p
$H_r \in \mathbb{R}$	hyper period (i.e., least common multiple) of all processes executed on a resource r
$\alpha(p) : P \to A$	function returning the respective application a a process p is part of
$\rho(p) : P \to R$	function returning the respective resource r a process p is executed on
$t_{o,r}^{max}$	maximum observation time to detect message omissions on a potentially faulty resource r

In Figure 3.14 the parameters listed above are shown within a hypothetical time-triggered schedule based on the specification in Figure 3.13. Each row illustrates the processes on one particular system resource (including the bus resource in the second row from the top) and the solid arrows represent their data dependecies corresponding to the process graph. In this example, the schedule excerpt comprises one period h_a which is the same for all three example applications a_1, a_2 and a_3. Here, optimization variables, such as the message offset $\mathbf{o}_{m_3}$, are depicted in bold characters to distinguish them from previously specified parameters, such as the task execution time e_{t_2}.

3.6.4.1 Requirements

Broadly speaking, in order to adapt a schedule for better diagnosis performance, both single messages and entire applications are shifted in terms of their start times, ensuring that the defined maximal end-to-end delay is met. As our approach uses previously specified system schedules, it basically only requires modifications on the communication resources, leaving the originally defined time constraints for task processes unchanged. However, to gain an additional degree of freedom for the following message stream adaptation, the applications are allowed to be shifted as a whole without changing their end-to-end delay. Hence, to begin with, we define the boundaries of the individual message offsets $\mathbf{o}_m$ as well as the application offsets $\mathbf{o}_a$.

$\forall a \in A, \forall m \in M, (\tilde{p}, m) \in E_P, (m, p') \in E_P$:

$$0 \leq \mathbf{o}_a < h_a \tag{3.11a}$$

$$-w_{(\tilde{t},m)} \leq \mathbf{o}_m \leq w_{(m,t')} \tag{3.11b}$$

According to Equation (3.11a), the offset of an application can amount to the length of its period, where the latter one itself is excluded from the interval. Because shifting an application is equivalent to setting an identical offset for all its corresponding processes the relevant end-to-end delays are not affected. In contrast to this, in Equation (3.11b) the offset of one single message is bounded by the waiting times to its predecessor task $\tilde{t}$ and its successor task t'. For instance, in the schedule in Figure 3.14 the message m_3 (whose offset interval $\mathbf{o}_{m_3}$ is indicated by the subjacent gray area) can be maximally shifted by $-w_{(t_3,m_3)}$ towards t_3 and by $w_{(m_3,t_4)}$ towards t_4.

Without loss of generality, in the following we consider a non-preemptive single-threaded execution model where a resource can execute concurrently at most one process. For this reason, it must be guaranteed that both tasks and messages are not scheduled on a shared resource at the same time. The constraints assuring non-interference between the corresponding messages and tasks are defined for the Equation (3.12) and (3.13), respectively, which, in turn, determine the message offsets $\mathbf{o}_m$ and the application offsets $\mathbf{o}_a$. In other words, in both cases the two exclusively disjunctive ($\oplus$) inequalities are preventing two processes from overlapping. In order to ensure that a process is compared with every other process executed on the shared resource, it is necessary to iterate over the process periods up to a duration of three and two hyper periods, respectively, as defined by the counting variables i and j.

$\forall m, \widetilde{m} \in M, m \neq \widetilde{m}, a = \alpha(m), \tilde{a} = \alpha(\widetilde{m}), \rho(m) = \rho(\widetilde{m}),$

$i = \{0, \cdots, \frac{3H_{\rho(m)}}{h_m} - 1\}, j = \{0, \cdots, \frac{3H_{\rho(\widetilde{m})}}{h_{\widetilde{m}}} - 1\}$:

$$\begin{aligned} &\mathbf{o}_a + \mathbf{o}_m + i \cdot h_m + s_m + e_m \leq \mathbf{o}_{\tilde{a}} + \mathbf{o}_{\widetilde{m}} + j \cdot h_{\widetilde{m}} + s_{\widetilde{m}} \\ \oplus\ &\mathbf{o}_{\tilde{a}} + \mathbf{o}_{\widetilde{m}} + j \cdot h_{\widetilde{m}} + s_{\widetilde{m}} + e_{\widetilde{m}} \leq \mathbf{o}_a + \mathbf{o}_m + i \cdot h_m + s_m \end{aligned} \tag{3.12}$$

$\forall t, \tilde{t} \in T, t \neq \tilde{t}, a = \alpha(t), \tilde{a} = \alpha(\tilde{t}), a \neq \tilde{a}, \rho(t) = \rho(\tilde{t}),$
$i = \{0, \cdots, \frac{2H_{\rho(t)}}{h_t} - 1\}, j = \{0, \cdots, \frac{2H_{\rho(\tilde{t})}}{h_{\tilde{t}}} - 1\}$:

$$\begin{aligned} &\mathbf{o}_a + i \cdot h_t + s_t + e_t \leq \mathbf{o}_{\tilde{a}} + j \cdot h_{\tilde{t}} + s_{\tilde{t}} \\ \oplus\ &\mathbf{o}_{\tilde{a}} + j \cdot h_{\tilde{t}} + s_{\tilde{t}} + e_{\tilde{t}} \leq \mathbf{o}_a + i \cdot h_t + s_t \end{aligned} \tag{3.13}$$

3.6.4.2 Message Stream Adaptation

Having assured that no two processes can utilize a resource at the same time, we can now formulate the diagnosis-aware message stream adaptation for an improved fault diagnosis. Its main constraint is shown in Equation (3.14).

$\forall r \in R, \forall m \in M_r, a = \alpha(m), i = \{0, \cdots, \frac{H_{\rho(m)}}{h_m}\}$:

$$\bigvee_{\widetilde{m} \in M_r} e_m \leq \Big(\mathbf{x}_{m,i} \cdot h_{\widetilde{m}} + s_{\widetilde{m}} + \mathbf{o}_{\widetilde{m}} + \mathbf{o}_{\tilde{a}=\alpha(\widetilde{m})}\Big) - \Big(i \cdot h_m + s_m + \mathbf{o}_m + \mathbf{o}_a\Big) \leq t_{o,r}^{max} \tag{3.14}$$

As mentioned before, one of our goals is to guarantee that a fault diagnosis does not exceed predefined diagnosis times which is equivalent to setting specific upper bounds for the observation times of the message streams on the appropriate resources. More precisely, the system designer selects a maximum observation time $t_{o,r}^{max}$ for each resource r during which a potential permanent fault shall be diagnosed with the help of the corresponding diagnosis function.

In this context, the observation time is defined as the interval between the expected arrival times of two consecutive messages from the same resource. However, as the individual process periods may differ between applications, it is not sufficient to compare adjacent messages only. As can be seen in Equation (3.14), an auxiliary integer variable $\mathbf{x}_{m,i}$ is used in such way, that for each iteration i of a message occurrence m, the defined equation finds at least one closest neighbor $\widetilde{m}$ within the regarded hyper period. In contrast to Equation (3.12) and (3.13), where all message streams are analyzed concurrently, here each resource is analyzed separately, which means that only message streams from the appropriate set M_r are considered. Consequently, given a sufficiently large set M_r, applying the constraint in Equation (3.14) will ideally result in a schedule which complies with the maximum observation time as well as exhibits an even distribution of the message start times for streams from the same resource.

3.6.4.3 Message Insertion

Our second goal is to extend the system schedule by inserting diagnostic messages and, hence, include previously undiagnosable resources $r \in R_d$ into the fault diagnosis process. The retrieval of these resources is performed with the help of a DFS algorithm on the architecture graph G_R that basically looks for resources with an insufficient amount of outgoing message streams. Having determined the set R_d, we can extend the system specification with particular diagnostic tasks t_d that periodically broadcast short diagnostic messages m_d. As defined in Equation 3.15, both the task period h_{t_d} and the message period h_{m_d} are bound by the maximum observation time to be consistent with the previous constraints.

$\forall r \in R_d, t_d \in T_{d,r}, m_d \in M_{d,r}$:

$$h_{t_d} = h_{m_d} \leq t_{o,r}^{max} \tag{3.15}$$

The new diagnostic applications a_d, consisting of one task t_d and one message m_d each, are treated as normal system applications such that they are added to the extended message and task sets, as defined in Equation (3.16a) and (3.16b). In this manner, it is guaranteed that the inclusion of the new processes into the existing schedule is performed in compliance with all previous constraints and the initial values for the start times, s_{t_d} and s_{m_d}, are adjusted based on the corresponding application and message offsets $\mathbf{o}_{a_d}$ and $\mathbf{o}_{m_d}$, respectively.

$$T' = T \cup T_d \tag{3.16a}$$

$$M' = M \cup M_d \tag{3.16b}$$

Although not discussed in detail, the insertion procedure described above makes use of a special conflict refinement, presented in [118]. In cases, where the adaptation of the message distribution is not able to comply with the specified constraints (e.g., there are not enough messages to guarantee a defined maximal observation time), auxiliary diagnostic messages can be inserted to decrease the observation time. This approach is used for the case study in Section 3.6.5. It is important to keep in mind that, depending on the used hardware, the proposed message insertion might require adaptations on the respective resources, e.g. in terms of their bus interfaces, to allow a transmission of messages in the first place.

3.6.5 Case Study

The presented case study is an automotive application where multiple tasks and messages are mapped to an architecture with four bus-attached ECUs. It use the research platform for automotive E/E architectures introduced in Section 3.5.3 executing the corresponding steer-by-wire application. Here, the maximal tolerable observation times for the ECUs are required to stay below 2.7 ms, 5.0 ms, 4.0 ms and 3.0 ms, respectively. All other relevant case study parameters

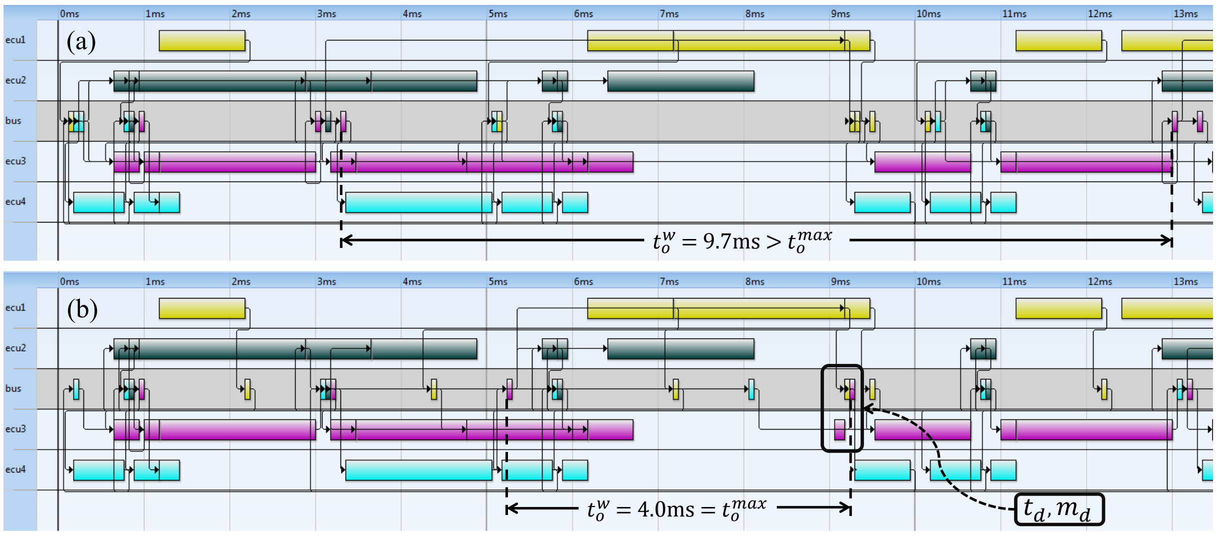


Figure 3.15: *Screenshot of a schedule visualization tool, showing the resulting schedules for the presented case study. Messages in schedule (a) follow solely the constraints of the initial specification and seem more clustered, whereas our adapted schedule (b) distributes them according to a defined maximum observation time* t_o^{max}.

Table 3.2: *Case study parameters and maximal tolerable observation times* t_o^{max}.

ECUs	*Buses*	*Functions*	*Tasks*	*Messages*	t_o^{max} *for ECU 1/2/3/4*
4	1	5	25	15	2.7/5.0/4.0/3.0 ms

are listed in Table 3.2. The calculations were carried out on an Intel Core i5 with 2.6 GHz and 8 GB RAM. For the methods that required solving a decision problem, *Microsoft Z3* version 4.3.0 as Satisfiability Modulo Theories (SMT) solver has been used [26].

Figure 3.15 depicts the case study outcome for both the unaltered system schedule (a) and the adapted schedule (b), which results after applying the diagnosis-aware system design. The bus resource in the middle of each schedule contains the entire message communication and the task schedules for the ECUs 1 and 2 as well as 3 and 4 are arranged above and below, respectively. For clarity reasons, from the entire hyper period of 20 ms only 13 ms are depicted and all processes (i.e., tasks and messages) which belong to one particular ECU are shown as bars in the same color. The worst case observation time t_o^w for ECU 3 is highlighted in both schedules and the processes t_d and m_d, inserted by our algorithm, are framed.

Generally, comparing the bus resources only, it can be clearly seen that the proposed method distributes messages more evenly, which is a first indicator that the observation times might have been decreased. The relative positions of single task processes to each other are unaltered showing that the end-to-end delays of the individual applications have not been affected. As an example, it is pointed out that the initial worst case observation time of 9.7 ms for two message

Table 3.3: *Initial and improved fault diagnosis observation times.*

observation time t_o^w	*ECU 1*	*ECU 2*	*ECU 3*	*ECU 4*
initial	5.0 ms	5.0 ms	9.7 ms	4.5 ms
improved	**2.7 ms**	**5.0 ms**	**4.0 ms**	**3.0 ms**

omissions from ECU 3 has been reduced to the required maximum observation time of 4.0 ms. This would decrease the corresponding diagnosis time of a permanent fault on this resource by more than 50 %. At the same time, the observation times for the remaining three ECUs could also be decreased or remained unchanged in order to comply with the required maximum times. The remaining results are shown in Table 3.3 which lists both the initial and the improved observation times.

The computation time of our algorithm did not exceed 1.3 seconds for the presented small case study. As our method introduces relatively few additional scheduling constraints, a good scalability for larger systems can be expected. Nevertheless, more experimental results with larger test cases need to be performed in the future.

3.7 Conclusion

This chapter presented a novel approach for the diagnosis of permanent faults in automotive E/E architectures. A faulty resource can be detected instantly on the basis of special diagnosis functions, which are distributed among the other system resources to prevent a single point-of-failure. The diagnosis method is based on the monitoring and analysis of existing message streams and, thus, is non-intrusive with respect to the network communication. Only in special cases, for instance, when particular resources are undetectable, additional diagnostic messages can be included to improve the general diagnosability.

The proposed diagnosis framework covers two stages. First, during a diangosis determination stage, a given system architecture is analyzed in order to identify potential fault scenarios. Diagnosis functions are then generated based on BDD-encoded observations of traffic patterns to detect these scenarios. Second, during an optimization stage, the diagnosis functions are used to determine trade-offs between the diagnosis times and the numbers of monitored message streams.

The general feasibility of our approach is demonstrated based on the evaluation of 100 synthetic test cases covering automotive architectures of up to 23 ECUs and more than 40000 distinct fault observations. The experimental results show, that an optimization of the generated diagnosis functions can have a large impact on the efficiency by reducing the observation times by up to 50 % or removing up to 90 % of the monitored message streams. The practical use is illustrated on a case study highlighting the ability of diagnosing permanent faults beyond the limits of the bus the observing resource is connected to (i.e., on network-level). Moreover, the implementation of the diagnosis method on a research platform for automotive E/E architectures demonstrates low CPU and memory overheads of maximum 5.13 % and 6.06 KB, respectively.

Finally, this chapter also presents an approach to extend and improve the permanent fault diagnosis by applying a diagnosis-aware system design. Here, the communication of a given system schedule is modified by adapting start times of periodic messages and including light-weight diagnostic applications for undiagnosable resources. As a result, the evenly distributed messages offer a more consistent and deterministic fault diagnosis which is able to further reduce the initial observation times.

Future work. Similar to the intermittent fault diagnosis in Chapter 2, the approach presented here basically uses the single-fault assumption. Even though a fault scenario can contain multiple resources, it is presumed that a single permanent fault causes their failure and therefore they are all covered by one specific diagnosis function per observing resource. In future, an extended set of diagnosis functions together with a possible consideration of delays between two successive faults could facilitate the diagnosis of multiple resources affected by both dependent and independent permanent faults.

Considering the diagnosis-aware system design, the proposed method mainly targets a fault diagnosis at runtime. However, this work could be regarded as an initial step towards future DFT techniques for distributed architectures. By considering both the verification and validation during manufacturing as well as the maintenance tests after delivery, the proposed approach could be extended to support diagnoses and tests during the whole product life time.

CHAPTER 4

Diagnosis of Security Attacks

In the previous chapter a strategy for a permanent faults diagnosis in distributed systems has been introduced. It has been shown how, based on the observation of message stream patterns, special diagnosis functions can be used and further optimized for a decentralized detection of faulty resources. Furthermore, a diagnosis-aware system design approach was presented which allows both a reduction of the necessary diagnosis times and an inclusion of previously undetectable resources into the fault diagnosis. Now, still considering the general idea of a decentralized monitoring of message streams, we want to shift from a pure fault diagnosis towards the diagnosis of deliberate system infringements. For this, the present chapter introduces a novel approach towards the distributed diagnosis of security attacks in automotive E/E architectures.

Due to the growing interconnectedness and complexity of in-vehicle networks, in addition to safety, *security* is becoming a more than ever relevant topic in the automotive domain. We propose a diagnosis method for the detection of manipulated message streams in which the contents of the messages themselves are not considered problematic. The approach is based on a decentralized principle where the diagnosis tasks are distributed among suitable system resources, such as ECUs, aiming at an overall improvement of the diagnosis reliability and robustness. The proposed security framework consists of two stages. First, a given system architecture is analyzed and the necessary communication parameters and properties for the subsequent diagnosis are determined. In the second stage, this system knowledge is used for both a redundancy-aware distribution of diagnosis tasks and an appropriate parametrization of the diagnosis algorithms executed on the monitoring resources. By using a light-weight detection method and an optimization-based task distribution, our approach guarantees a full coverage and timeliness of the diagnosis while imposing a low additional overhead.

Chapter outline. Chapter 4 consists of the seven following sections. First, in Section 4.1 an introduction is given, which also contains an illustrative example and a description of the main contributions of the diagnosis. Section 4.2 discusses the related work. Section 4.3 presents the diagnosis framework and provides a description of the communication and architecture models underlying the proposed approach. The diagnosis itself is introduced in Section 4.4 which also contains an overview of possible attack scenarios. The methodology and formal definitions are presented in Section 4.5, which is further divided into two parts: real-time detection and redundancy-aware distribution. Section 4.6 contains the experimental results which evaluate our approach based on synthetic test cases and a case study. Finally, the conclusion and future work are presented in Section 4.7.

4.1 Introduction

This section provides the introduction to the diagnosis of security attacks. Following the motivation, which includes a brief literature overview, the overall idea is explained with the help of a realistic automotive attack scenario. Afterwards, the main contributions of this chapter are presented.

4.1.1 Motivation

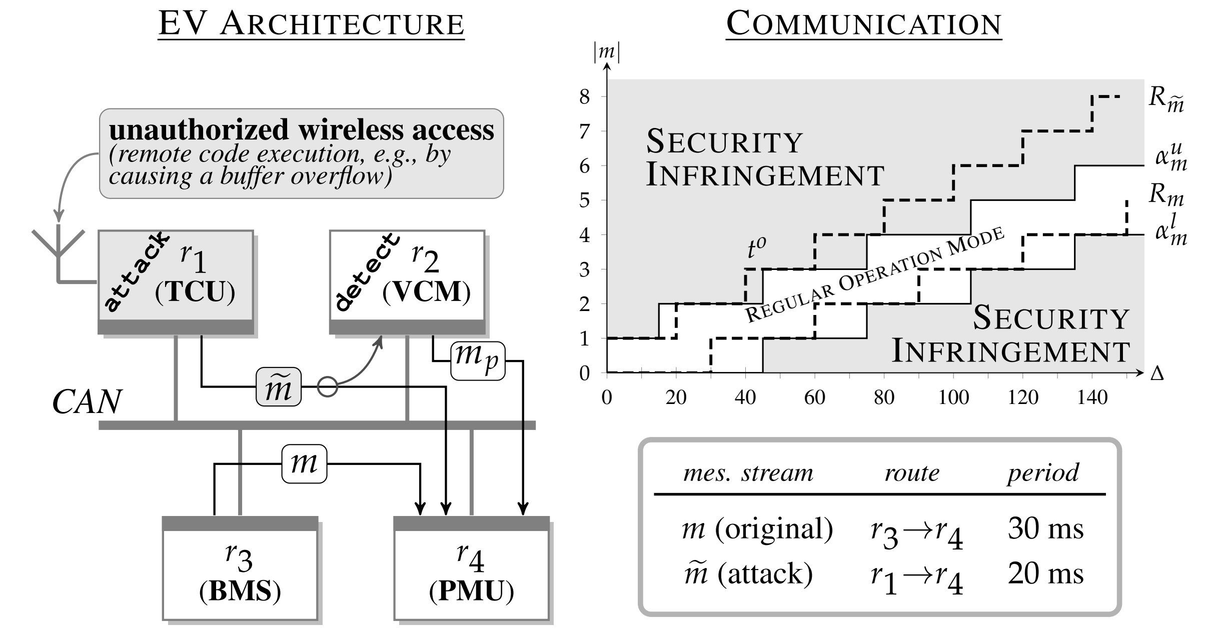
Over the past decades, automobiles have been transforming from mere functional means of conveyance into connected, electronically controlled vehicles. They are expected not only to transport passengers but also to entertain and inform in a safe and protected environment. As a consequence of the specific requirements on safety, efficiency and customers' demands, the complexity of automotive E/E architectures has been constantly growing. In-vehicle networks with more than 100 ECUs communicating over multiple automotive-specific buses and gateways are now common in the industry. At the same time, novel functionality like C2C and C2I as well as modern infotainment systems have resulted in an increasing amount of vehicle components with interfaces to the outside world, for instance, via radio-based communication links. Integrated in an automotive architecture, such devices make cars more than ever vulnerable to security infringements, as presented in [45]. Recent demonstrations of security violations in production vehicles, for instance by [69] and [21], clearly illustrate the growing importance of this topic. An equal treatment of safety and security as part of the overall reliability of an automotive E/E architecture seems crucial, as an exposed electronic component might contribute to the passengers' safety risk at the same extent as a faulty device [119]. However, in the competitive automotive domain, the related increased costs at design-time are an obstacle for the introduction of novel security features which, in turn, call for inexpensive solutions.

In recent years, the problem of automotive security has been regarded from various perspectives. First, in addition to the examples given above, the overall dangers of an insecure automotive architecture for the driver's safety are pointed out in [9] and [46], where the latter one also considers DoS-based threats. The inclusion of security measures into automotive E/E architectures at design-time requires, among other things, extended formal definitions and solution strategies for a security-aware Design Space Exploration (DSE), as shown in [135] and [82]. On the other hand, the diagnosis of security attacks at runtime uses special Intrusion Detection System (IDS) in order to detect anomalies in the system communication behavior or data content, as presented in [102] and [101]. Finally, automotive security also comprises the intra-vehicular communication which has been studied in [17] and [32] and may contribute to an early intrusion prevention for in-vehicular networks. Although monitoring of input streams has been used in the context of traffic verification and regulation in real-time systems [47], to the best of our knowledge, our approach is the first one analyzing the automotive message-based communication on system-level for security purpose. A detailed discussion of the existing work in the area of automotive security is presented in Section 4.2.

In this chapter, we present an *automotive security diagnosis* approach, that, due to its decentralized and non-intrusive principle, offers a reliable and cost-efficient solution for the emerging and fast-growing topic of vehicle security. Generally, in message-oriented networks many common forms of attacks change a message stream and manifest themselves in altered traffic patterns (e.g., DoS or message flooding). In classic computer networks which are based on a highly dynamic and unpredictable packet transmission, such as the Internet, potential traffic-manipulating attacks can only be detected in extreme cases. This is basically different for in-vehicle networks where due to highly time-critical control software, message periods and execution time bounds are given, such that delays and worst case jitters can be precisely calculated. Here, in contrast to the aforementioned generic networks, already a relatively small increase of the message rate can compromise safety-critical functions, for instance, by slackening the responsiveness in a brake-by-wire system or impede a correct power management. As a consequence, our automotive security diagnosis approach is not analyzing the message content itself, but rather the overall communication behavior in order to detect and prevent an effective attack. In addition to classic DoS, our method can also diagnose attacks where a service is not entirely rejected but, for instance, carried out incorrectly.

4.1.2 Illustrative Example

We demonstrate the relevance of the automotive security problem as well as our solution approach with the help of a realistic attack scenario. For this purpose, Figure 4.1 illustrates a part of an E/E architecture based on the in-vehicle network of an up-to-date and currently best-selling commercial Electric Vehicle (EV) [104]. It consists of four ECUs connected to a CAN bus: a telematics unit (TCU) providing a wireless connection for remote services, a vehicle control module (VCM) evaluating ECU and sensor signals, a Battery Management System (BMS) controlling the Li-Ion cells and a power management unit (PMU). In EVs, to avoid a potentially harmful battery depletion, the transmission of energy to the motor or other electrical devices is initiated by a torque request (e.g., by the PMU) to the BMS and has to be accepted or re-



mes. stream	*route*	*period*
m (original)	$r_3 \rightarrow r_4$	30 ms
$\widetilde{m}$ (attack)	$r_1 \rightarrow r_4$	20 ms

Figure 4.1: *Attack scenario illustrating our security diagnosis approach. The telematics unit of an automotive E/E architecture is attacked in order to manipulate the communication between the battery (BMS) and the power management (PMU). The attack can be detected and counter measures (e.g. high-priority warning m_p) can be applied at the vehicle control module (VCM) by testing the compliance of the manipulated stream $\widetilde{m}$ with the arrival curve of the original stream m.*

jected by a response message (compare message stream m between r_3 and r_4). Moreover, to ensure functional safety, a BMS usually contains a hard-wired shut-off switch which will cut the electrical power supply, for instance, before current or voltage levels can fall below dangerous thresholds [117]. Let us assume, that an attacker intends to bring the vehicle to an immediate halt. As recently discussed in [69], an intruder can relatively easily gain wireless access to the automotive architecture, e.g., via the telematics interface or a Bluetooth adapter for the on-board diagnostics connector (OBD-II port). Once having control over the TCU, the attacker manipulates its code to transmit a simple high frequency stream of responses accepting the torque request (compare message stream $\widetilde{m}$), such that they would in any case dominate the original, possibly rejecting, response messages on the bus. As it is receiving positive responses to its torque request at a high rate, the PMU continues to draw current from the battery beyond the designated threshold which will trigger the aforementioned shut-down switch. Disconnected from its power source, the car will abruptly slow down, lose its brake booster or power steering and finally come to a halt. This could have catastrophic consequences, for instance, during an overtaking manoeuvre, such that an instant detection of such an attack is crucial and indispensable.

Within the proposed security diagnosis approach, we use the knowledge of particular system components about essential communication parameters, such as message periods and maximum jitters. This information is specified during system design and can be stored in the form of so-called *message arrival curves* which are compact representations of the upper and lower bounds for the number of message stream events within a specific time interval. As arrival curves are easily adaptable and can be parametrized for many different message stream patterns, they help us to design efficient and distributable diagnosis tasks. Consequently, within the illustrated scenario, the VCM can very quickly detect the transgression of a predefined upper arrival curve α_m^u by monitoring and analyzing the actual message count of the manipulated stream $R_{\widetilde{m}}$ with respect to α_m^u. Such a transgression is illustrated in the arrival curve graph in the right part of Figure 4.1 at time t_o. After having detected the altered message stream and identified it as torque response (e.g., via the CAN-ID), the VCM transmits a high priority message m_p to the PMU informing it about the wrongful responses from the BMS. The subsequent countermeasures could include, among other things, a safe and controlled reduction of the torque request and at the same time warn the driver about a serious malfunction asking him or her to stop the car. By all means, it is of utmost importance that these measures are initiated as early as possible, which can be ensured with the methods presented in this chapter. Moreover, should the attacker try to disrupt the corresponding diagnosis functionality on the VCM, our decentralized redundancy-aware methodology maintains the overall diagnosis capability by using several distributed monitors for each message stream.

4.1.3 Contributions

As a matter of principle, the proposed security diagnosis approach is designed to detect communication-based attacks aiming at compromising the vehicle's E/E architecture or parts of it. For this, the network communication is monitored in order to identify deviations from the transmission specification of single message streams. In other words, it is detecting changes in the expected communication patterns. Such an approach is insofar relevant, as many common methods to harmfully infiltrate and disrupt automotive networked systems manifest themselves in altered transmission patterns. For instance, [101] explores, inter alia, two attack scenarios: an increased frequency in a regular message stream as well as message flooding using highest-priority CAN identifiers to restrict the availability of the bus. While [46, 43] describe how such attacks can cause severe dangers for road participants, e.g., by turning off headlights at night, our motivating example extends this by demonstrating that the manipulation of particular traffic patterns in an EV can easily disable its entire power supply.

In general, our security diagnosis approach follows an *implicit* paradigm and is performed in a *decentralized* manner. On the one hand, it utilizes only the existing communication without the need for additional diagnostic messages or hardware components making the proposed method non-intrusive. The used diagnosis tasks are lightweight and, hence, the induced computational and timing overhead is minimal. On the other hand, distributing the diagnosis tasks helps to remove the risk of a single point of failure and, hence, improves the overall reliability of the diagnosis itself. Additionally, different ECUs can inspect a limited number of message streams to prevent an over-utilization of particular resources. In order to guarantee a complete coverage of diagnosis, the proposed distribution method considers the entire system communication and allows to be configured in terms of redundancy and allocation of the monitored message streams.

In this work, we want to provide an efficient and reliable security diagnosis framework which can overcome the three main difficulties mentioned below. First, the formulation of a fast diagnosis algorithm which can be efficiently implemented on each considered monitoring resource, preferably close to the communication controller to minimize latencies. Here, the actual diagnosis algorithm is implemented on each monitoring ECU and is based on a method for the verification of runtime conformity of hard-real time systems to detect the violation of predefined communication parameters. For this, as already illustrated in the motivating example, message streams are represented by special arrival curves which efficiently describe the specified earliest and latest times a messages should be received. This enables a suitable detection of potential malicious attacks on the system communication. Second, the distribution of this algorithm within specific diagnosis tasks shall be carried out automatically but still enable the system designer to manually adjust necessary parameters regarding monitoring redundancy and component utilization. Third, to cover current and future distributed architectures, the overall diagnosis framework must provide a suitable scalability. In summary, we propose:

1. A real-time traffic conformity check for a decentralized and light-weight detection of message-based attacks in automotive E/E architectures.

2. An optimization-based method to efficiently distribute diagnosis tasks among the monitoring resources considering redundancy.

It shall be mentioned that the focus of this work is on the *diagnosis* process for a secure automotive E/E architecture. The system behavior subsequent to an identified attack, especially the application of specific countermeasures as discussed in the motivating example, is not part of this work. Eventually, it is important to understand, that the proposed work does not raise a claim to sufficiency in terms of a universal security strategy. While the presented approach cannot guarantee to detect all types of security infringements, the low implementation effort along with the anyway required diagnosis functionality offers an additional security measure which can be used orthogonally to other approaches.

4.2 Related Work

In this section, a comprehensive overview and discussion of the existing work in the area of automotive security is given. The main focus is on the design-time and runtime diagnosis of attacks on in-vehicle networks.

General automotive security. The automotive industry is more than ever driven by the technological progress, notably in terms of connectivity and communication. The security risks arising from this development are evident, and beyond the stage of mere thought experiments or lab setups, as pointed out in [9]. An introductory insight and discussion of automotive security is presented in [119]. On the one hand, it illustrates the threats and the challenges arising from the growing complexity of automotive electronics and its interconnections to the outside world and, on the other hand, it proposes potential concepts for future solutions comprising Ethernet-based networks and formal verification approaches.

It has been shown that the related threats can range from a relatively harmless unauthorized access to the vehicle by cracking the key-less entry system [11] to the much more severe intrusion into an in-vehicle network and infiltration of its ECUs. This, in turn, can interrupt safety-critical systems and endanger the lives of the car occupants, as shown, for instance, in [45, 69, 21]. Using the example of an Electronic Throttle Control (ETC), [46] highlights the relevance and danger arising from a DoS attack, which is also an important scenario addressed in our work. The authors demonstrate how an attacker, after having hacked the engine management ECU, can use malicious code to flood the CAN bus with a large number of spoofed messages and provoke a DoS on the ETC ECU unit, thereby potentially causing an uncontrolled acceleration. In this context, the authors also developed an adaptive strategy to communicate security related incidents to the driver [44].

Furthermore, multiple approaches examine the topic of automotive security from a larger perspective including inter-vehicular networks, Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure (V2I) communication [49, 90]. For example, the work in [17] analyses the performance of vehicular communication systems, where the car itself is regarded as a node in a network. Similarly, the authors in [32] investigate and discuss the use of the IPv6 protocol to create a secure mobile vehicular communication and demonstrate that this is possible without serious network overload. Although these intra-vehicular approaches are not covering automotive E/E architectures in particular, the presented methods and analyses can help to prevent an intrusion into in-vehicle networks in the first place.

Overall, the examples above imply that cyber-attacks on automotive systems have become a practical threat which must be considered both during the design process of E/E architectures and while the vehicle is in operation.

Securing systems at design-time. In recent years, several papers have been published addressing the automotive security at design-time. In [135], security aspects are integrated into the traditional DSE of distributed embedded systems. For this, the formal system representation is extended by a formal description of the attacker capabilities, such as reading, modifying or inserting messages, as well as security requirements, defining what kind of attacks shall be prevented, in order to obtain cost-optimized architectures and security guarantees. Similarly, in [82] the security in the DSE process is addressed but puts the focus on the mapping problem from functional models to CAN-based platforms, where the latter are additionally extended by Message Authentication Codes (MACs). Security constraints preventing direct and indirect attacks on the MAC are defined and Mixed Integer Linear Programming (MILP) formulations as well as a heuristic algorithm are used to solve the mapping problem. To improve the efficiency and define the security scenarios in a more realistic way, the authors have extended the work in [83] by redefining the MILP formulations and considering functional paths rather than signal-based constraints. The application of MACs for the Flex-Ray bus is presented in [40]. There, the corresponding security requirements are specified together with usual system parameters, such as end-to-end latencies, but a divide-and-conquer technique is used to reduce the complexity of the corresponding schedule synthesis.

In [100], the strategy for the verification of security requirements of automotive architectures is proposed. The approach uses pre-assessed security-specific parameters such as exploitability and patching rate to describe the system and determine a Markov model which is then subjected to a probabilistic model checker. Special functions describing particular demanded security aspects are fed into the model checker and allow the corresponding architecture to be analyzed in terms of confidentiality, integrity or availability.

By contrast, at design-time, our security diagnosis approach covers two tasks, first, a system analysis extracts the necessary parameters for the runtime diagnosis and second, an optimal distribution of the diagnosis tasks among the system resources is performed. As a consequence, our work follows a rather different approach from the papers above, mainly because it uses the unaltered system specification for the diagnosis process (i.e., without initial security extensions)

Securing systems at runtime. Together with its consideration during automotive system-design, security has been also addressed in the context of runtime diagnosis. A general detection scheme for attacks on in-vehicle networks is proposed in [102], where the authors define a set of eight recognition criteria for the identification of a typical behavior of automotive bus systems, such as CAN, with a minimum number of false positive detections. The goal of this work is to present a concept for an IDS for future vehicles where the associated recognition criteria, called *Anomaly Detection Sensors*, are supposed to inspect the communication in terms of formality, location, range, frequency, correlation, protocol, plausibility and consistency. In such a holistic framework, the current state of our diagnosis approach would mainly fit into a frequency detection sensor, however, it could be also extended towards a location and correlation sensor with relatively little effort in the future. Besides the general scheme, the authors of [102] have also proposed a specific anomaly detection method using an information-theoretical approach where especially their scenario of an entropy-based increased frequency detection at runtime seems to be related to our work [101]. However, the corresponding evaluation in that work merely proves the concept and does not provide the observation times needed for the diagnosis. Thus, for the time being, a direct comparison with our approach in terms of real-time efficiency is not possible.

Beyond the automotive domain, the importance of a distributed IDS, which removes the single point of failure, has been recently discussed in [143]. There, the actual differentia-

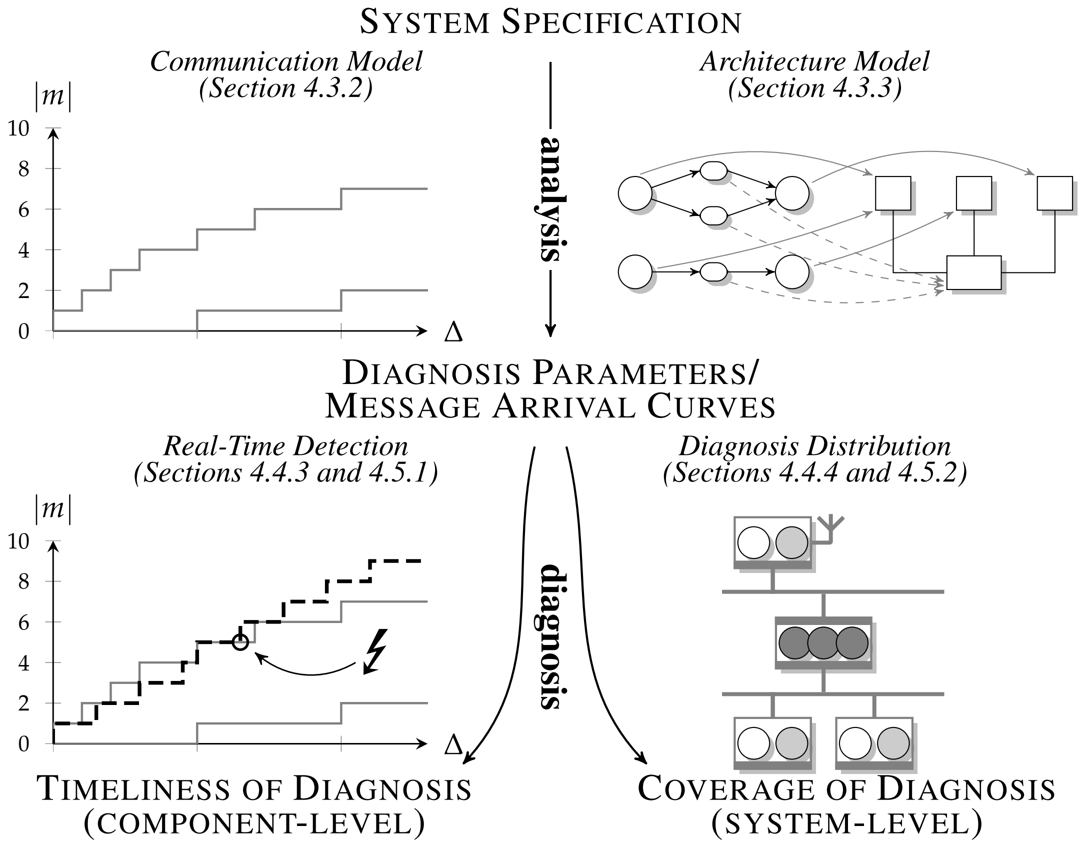

SYSTEM SPECIFICATION
Communication Model (Section 4.3.2)
Architecture Model (Section 4.3.3)
|m|
10
8
6
4
2
0
Δ
analysis
DIAGNOSIS PARAMETERS/ MESSAGE ARRIVAL CURVES
Real-Time Detection (Sections 4.4.3 and 4.5.1)
Diagnosis Distribution (Sections 4.4.4 and 4.5.2)
|m|
10
8
6
4
2
0
Δ
diagnosis
TIMELINESS OF DIAGNOSIS (COMPONENT-LEVEL)
COVERAGE OF DIAGNOSIS (SYSTEM-LEVEL)

Figure 4.2: *The proposed security diagnosis framework is built up as follows: first, the system specification is analyzed in terms of its communication and architecture in order to obtain the necessary message arrival curves as well as the diagnosis parameters, e.g. a resource-to-observation map. In a second stage, this outcome is used for both the real-time detection of compromised message streams and the distribution of the diagnosis tasks.*

tion between normal and attacked traffic is based on a feature extraction performed during a parametrization and training phase on each node using the *Naïve Bayes Classifier* and the agreement between the nodes is determined using an average consensus protocol.

In contrast to this probabilistic approach, our security diagnosis takes advantage of the fact that automotive system and communication properties are known at design-time, which facilitates a verification if the observed traffic behavior conforms to the given specification.

4.3 System Model

In this section, we first present an overview of the security diagnosis framework before introducing the corresponding communication and architecture models.

4.3.1 Framework

Figure 4.2 depicts the general structure of the proposed diagnosis framework, that can be roughly divided into two stages. The first stage (upper part of the figure) illustrates the *system analysis* while the second stage (lower part of the figure) represents the actual *system diagnosis*. More precisely, first of all, a given specification of the system is analyzed in terms of its communication parameters as well as its application and architecture structure. The goal is to obtain the necessary information for the subsequent diagnosis stage. For the latter, the resulting communication parameters, which define special message arrival curves, are used to configure the real-time diagnosis functions later implemented on the corresponding monitoring resources. Furthermore, the knowledge about the system architecture is incorporated in the distribution of the diagnosis tasks among the system resources using predefined parameters such as redundancy levels. The second stage covers two important diagnosis criteria for safety-critical systems, namely, the *timeliness* and the *coverage*, which will be explained in detail in Section 4.5.

In the following, the communication model which underlies the presented approach is introduced in Section 4.3.2 and the description of the corresponding architecture model is given in Section 4.3.3.

4.3.2 Communication Model

The presented approach for diagnosing security infringements aims at the entire in-vehicle network independent of the particular communication protocol. Nevertheless, setting the focus on event-triggered communication, such as used by the CAN bus, is reasonable as it is one of the most prevalent automotive buses. Moreover, as it is often used for safety-critical tasks, CAN-based communication brings the highest risk of being exploited for undesired purposes [66]. To model the network traffic, we adopt the concept of arrival curves presented in [19] for general event-based systems. In the context of the security diagnosis, these arrival curves are used to define the bounds for the number of observed message streams on a bus during a specified time interval.

In the following, we first generally define message streams before introducing arrival curves and discussing their determination.

4.3.2.1 Message Streams

A message stream m is generally described by its *event trace* function $\mu_m(n)$ where the n-th message arrives at time t_{n-1}, as defined in Equation (4.1). For example, Figure 4.3 shows a part of a message stream m where six message events arrive at times t_0 through t_5. Additionally, nine events of a compromised message stream $\widetilde{m}$ are depicted.

$\forall t_n \in \mathbb{R}, n \in \mathbb{N}:$

$$\mu_m(n) = t_{n-1}, \quad \text{with } n \in \{1, ..., n_{\max}\} \tag{4.1}$$

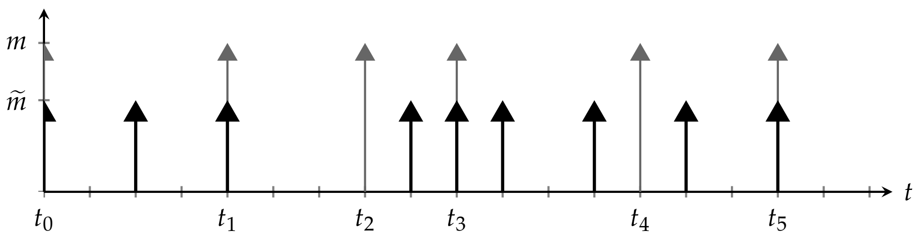


Figure 4.3: *Example for two event traces. Six event arrivals (long arrows) represent an excerpt of a regular message stream m and nine event arrivals (short arrows) stand for a possible attack on m in the form of a compromised message stream $\widetilde{m}$.*

Based on the event trace of a message stream m, the *message count* function $R_m[t_a, t_b)$ returns the actual number of events (i.e., *message instances* or *messages*) arriving at a communication resource in the time $[t_a, t_b)$, where t_a is inside and t_b outside the half-closed interval. This is formally defined in Equation (4.2a) with $R_m(t_a)$ and $R_m(t_b)$ determined by the maximum number of events of the trace function up to the time t_a and t_b, respectively (Equations (4.2b) and (4.2c)).
$\forall t_a, t_b \in \mathbb{R}^+, t_a < t_b, n \in \mathbb{N}$:

$$R_m[t_a, t_b) = R_m(t_b) - R_m(t_a) \tag{4.2a}$$

$$R_m(t_a) = \max\{n \mid \mu_m(n) < t_a\} \tag{4.2b}$$

$$R_m(t_b) = \max\{n \mid \mu_m(n) < t_b\} \tag{4.2c}$$

4.3.2.2 Arrival Curves

Obviously, for a message stream which is not purely periodic but, for example, periodic-with-jitter, the message count can differ for the same interval length $[s, t)$ but different start times s of the counting. Consequently, in order to diagnose communication inconsistencies, we are interested in determining the highest and lowest message count of m for each possible time interval. This can be facilitated by the aforementioned arrival curves. By definition, the minimum and maximum number of messages arriving within an arbitrary time interval Δ is bound by a lower arrival curve $\alpha^l_m(\Delta)$ and an upper arrival curve $\alpha^u{}_m(\Delta)$, respectively, both represented by *step functions*. Given the message count on a bus which lasts for an interval of length Δ, the lower and upper arrival curves must satisfy the two Equations (4.3a) and (4.3b), respectively, for all times t.

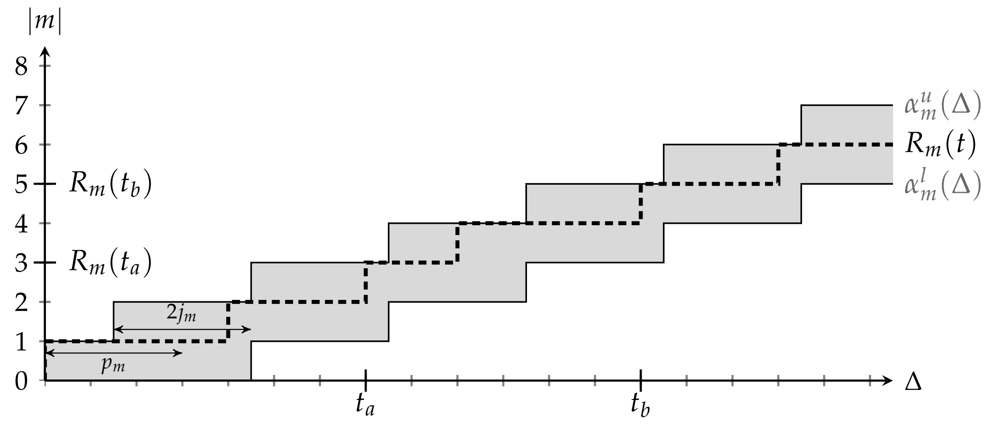


Figure 4.4: *Example of an arrival curve graph depicing the Regular Operation Mode on a communication resource. $R_m(t)$ represents the message count (not an arrival curve) for an example message stream which does not violate the bounds of a periodic-with-jitter communication defined by α_m^l and α_m^u. The adherence to Equation* (4.3) *is highlighted with the example of the two message counts $R_m(t_a)$ and $R_m(t_b)$.*

$\forall t \in \mathbb{R}^+$:

$$\alpha_m^l(\Delta) = \min_{t \geq 0}\{R_m(t+\Delta) - R_m(t)\} \tag{4.3a}$$

$$\alpha_m^u(\Delta) = \max_{t \geq 0}\{R_m(t+\Delta) - R_m(t)\} \tag{4.3b}$$

An example for a pair of arrival curves is shown in Figure 4.4. The graph depicts the number of messages in a stream m (*y-axis*) which can be observed within any arbitrary time interval Δ (*x-axis*). With the upper and lower arrival curves at its edges, the shaded area in the graph marks the region of the Regular Operation Mode which is not allowed to be left by a regular message stream, independent from the starting time of its observation. Here, $R_m(t)$ illustrates one of the infinite number of possible regular message streams. In this case it is representing the message arrivals of the stream m in Figure 4.3, where the observation begins at t_0 and t_a and t_b correspond to t_3 and t_5, respectively.

It is important to understand, that $R_m(t)$ does not represent an arrival curve for an arbitrary time interval Δ but rather the message count of a particular message stream for a specific observation time. Hence, strictly speaking, $R_m(t)$ does not refer to the x-axis of time intervals Δ but one reflecting the absolute runtime t. In the graph, a separate axis for the latter one is omitted for the purpose of clarity.

4.3.2.3 Arrival Curves Determination

Although some bus protocols, such as CAN and partly also FlexRay, follow an event-triggered communication paradigm, most of their messages are transmitted in a cyclic manner which is a common characteristic for the automotive communication with a large number of control functions in a sensor/actuator environment. As a consequence, for many applications, message streams can be modeled as periodic-with-jitter. In an event-triggered architecture, the jitter in a periodic message stream is an indicator for the variation between the inter-completion or response times of its corresponding successive release tasks. It can be caused by task preemption, changes in execution times or other delays [8]. Moreover, particular bus protocols can encounter jitters caused by the variations of the frame length due to bit-stuffing [106]. Consequently, this means that the arrival curves α^l_m and α^u_m are shifted by the maximum possible jitter j_m of a message stream m relative to a nominal period p_m, as defined by Equations (4.4a) and (4.4b).
$\forall p_m, j_m \in \mathbb{R}^+, \Delta > 0:$

$$\alpha^l_m(\Delta) = \left\lfloor \frac{\Delta - j_m}{p_m} \right\rfloor \tag{4.4a}$$

$$\alpha^u_m(\Delta) = \left\lceil \frac{\Delta + j_m}{p_m} \right\rceil \tag{4.4b}$$

In other words, the earliest moment to detect a message on the bus is at the instant of starting the observation and the latest is immediately before $p_m + j_m$ has elapsed. Hence, the definition for an upper and lower arrival curve for a periodic event-triggered system can be given by the period with jitter of the corresponding message stream. A detailed derivation of these equations is presented in [139].

Since during system design the arrival curves are initially calculated based on the task release times of the transmitting resources, the effects of existing bus traffic must be considered to assure the correct diagnosis timing. More precisely, it is necessary to add delays to the nominal intervals of the arrival curves with respect to a possible current bus occupation and higher priority messages. For instance, in [66] this has been done for the CAN bus. In case of a non-periodic communication, the curves can be determined from measurement or simulation, where a window of length Δ is slid over a traced message stream, recording the minimum and maximum message numbers. Repeating this for different window lengths allows to construct the arrival curves. Strategies for generating arrival curves have been presented, for instance, in [71]. However, it must be considered that the measurement and simulation approaches might not be sufficient as they still can miss the worst-case arrival times of messages.

In the context of automotive architectures, which are particularly regarded in this thesis, a system analysis can provide the upper and lower arrival curves for each message stream. This is possible since the communication of in-vehicle networks is generally known at design-time and

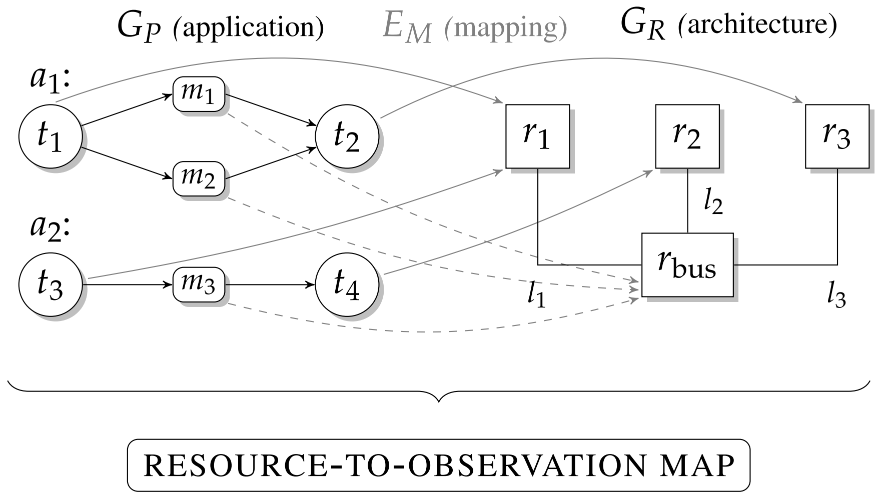


Figure 4.5: *Example for a graph-based system specification used for the security diagnosis framework. A process graph G_P consisting of two applications a_1 and a_2 is mapped onto a system architecture. The latter is defined by the resource graph G_R. The specificatios are used to define so called resource-to-observation maps.*

specified for the worst-case arrival times. Without loss of generality, in this work, we regard periodic streams with jitter.

4.3.3 Architecture Model

The presented diagnosis framework employs a graph-based system specification where software functionality and hardware are modeled by process and architecture graphs, respectively. The example in Figure 4.5 illustrates this with two applications, a_1 and a_2, being implemented to a small system architecture. The descriptions of the process graph G_P, architecture graph G_R and mapping E_M are basically the same as for the system model presented in Chapter 3, so that Section 3.6.3 can be referred to for more details.

We have already discussed that our goal is to diagnose a system in an implicit manner, which means that no additional diagnostic messages or hardware is used. Furthermore, we regard the system specification, like the one described above, as given. However, at this level of abstraction, the diagnosis functions themselves that are implemented on multiple resources can be regarded as tasks even though without any data-dependencies (see Section 4.4.4). In order to properly distribute the diagnosis tasks with respect to both the redundancy requirements and a complete coverage of the potentially attacked message streams, a preliminary system analysis is performed. The purpose is to define all observations describing the message streams and their parameters which can be monitored by a particular resource, not necessarily the transmitting one. These observations are then stored in a resource-to-observation map which basically associates sets of message stream data with the appropriate monitoring resources. Later, this map is used to retrieve the necessary configuration parameters for the distribution of diagnosis tasks, which is discussed in detail in Section 4.5.

4.4 Diagnosis Description

In the area of hard real-time systems, an important precondition is the ability to react to an asynchronous event within a predefined time interval, i.e., without violating the deadline [2]. In order to detect a security infringement, such as an unusual or unwanted communication behavior, within a guaranteed or shortest possible diagnosis time, each message stream in the system should be continuously monitored. Based on the observed occurrence of message stream events resulting, for instance, in a violation of the upper arrival curve, particular security attack scenarios can be diagnosed. In the following, we will briefly discuss some scenarios which are based on three general security principles.

4.4.1 Attack Scenarios

As already mentioned in Section 4.1, there are some fundamental differences between a classic computer network and an automotive in-vehicle network, for instance, regarding the safety-criticality or the predictability in the system communication. Nevertheless, as shown in [100], the three general security principles, namely, confidentiality, integrity and availability, can be applied in the automotive domain as well. In the context of this work, infringements of these principles can be broadly interpreted as follows:

1. *Confidentiality Breach* leads to the read access of messages by an unauthorized entity.
2. *Integrity Breach* leads to an unauthorized creation or manipulation of messages.
3. *Availability Breach* leads to an interruption of one particular or all messages streams.

An attack scenario violating confidentiality might be a hacked resource giving read access to the message content but producing no additional traffic. The diagnosis of such an infringement could benefit from possible (but not definite) increase of the computational effort during the unauthorized read access, causing additional delays in the corresponding message stream. As a result, the stream would violate the regular operation mode by falling below the lower arrival curve and could be detected accordingly.

Regarding integrity, a possible attack scenario is a hacked resource which counterfeits another resource by injecting a compromised message stream into the network (e.g., with the message ID belonging to a stream from the counterfeited resource). Assuming that the compromised transmission is not precisely overwriting the original messages, this attack would result in an increased message frequency and could be detected with the help of the method proposed here.

Finally, a scenario violating the principle of availability could be an attack, where one or more hacked resources start to transmit a large number of (possibly void) messages in order to mitigate the system's availability or to entirely disable it. Similar to the second scenario, a DoS attack will result in an increased transmission rate of the particular messages and the transgression of the upper arrival curve.

In contrast to the latter two scenarios, the first one, which deals with the non-compliance of the lower arrival curve, requires a different detection algorithm which is not discussed in the scope of the security attack diagnosis. However, the violation of lower arrival curves has been used in the context of the permanent fault diagnosis in Chapter 3 and the approaches presented there could be considered for an inclusion of the confidentiality attack detection in our security diagnosis in the future.

In the following we will consider especially the third scenario, namely availability attacks, to define the algorithm for the detection of compromised message streams.

4.4.2 Limitations

As has been briefly mentioned in Section 4.1, our attack detection does not claim sufficiency or represent a universal security strategy for automotive E/E architectures. There exist attacks, that cannot be detected with the proposed method. For example, attacks targeting confidentiality, such as message sniffing, and specific integrity breaches, such as single message injection, might not alter the traffic in a way that is detectable with the proposed algorithm. Consequently, many attacks will require own special security solutions which are not covered in the scope of this work.

The presented approach targets a class of attacks, which result in an abnormally altered frequency of transmitted messages, where the attacker has no information about the impacted message arrival curves. At the same time, due to its specific attack detection principle, this framework can improve automotive safety and reliability. For instance, it can be combined with the message-based fault diagnosis into a unified safety and security framework [149]. Nevertheless, because of the realistic threats arising from a malicious manipulation of traffic patterns (e.g., illustrated in Section 4.1 or in [101]) as well as its non-intrusive and lightweight detection principle the proposed approach is legitimate as an additional security solution orthogonal to other methods.

4.4.3 Real-Time Detection

In the framework overview in Figure 4.2, it was shown that the component-level part of the security diagnosis deals with the actual detection of a compromised message stream. As mentioned above, the detection method is explained by means of a common security infringement, namely the manipulation of the communication behavior. Figure 4.6 illustrates this scenario, with the shaded area marking the corresponding compromised region and $R_{\widetilde{m}}(t)$ representing

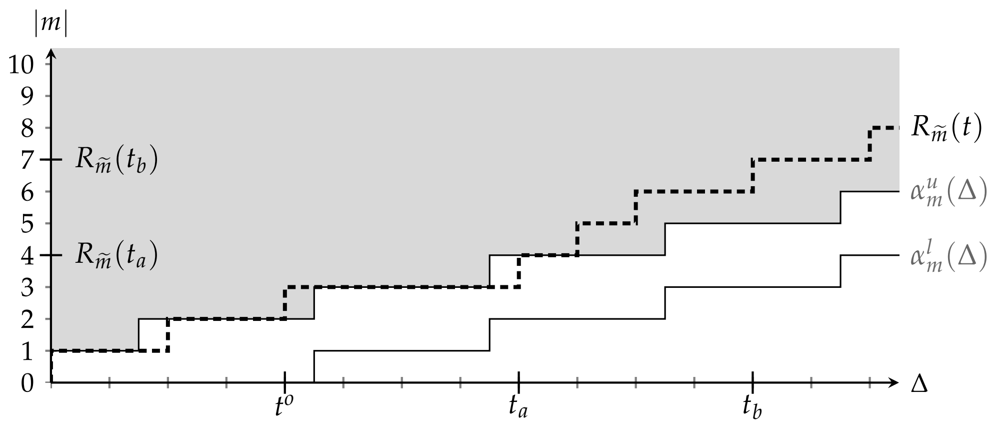


Figure 4.6: *Example for an arrival curve graph depicting an attack on a single message stream by manipulating its transmission period. The corrupted message stream violates the upper arrival curve for the first time at t^o.*

the message count from one possible stream violating the predefined upper bound. This stream equates to the event trace function in Figure 4.3 depicted by the short arrows.

Based on the arrival curve definition in Equation (4.3), we can now formulate a condition for the transgression of the upper bound, caused by an increased occurrence of message events. The attack requirement is defined in Equation (4.5).

$\exists t \in \mathbb{R}^+ :$

$$\alpha^u_m(\Delta) < \max_{t \geq 0}\{R_m(t+\Delta) - R_m(t)\} \tag{4.5}$$

Here, the inequality itself is a necessary condition to detect a corrupted message stream. However, it might be not sufficient, as there could exist observation intervals for which a corrupted stream provides a $R_m(t)$ which is on or below the upper curve $\alpha^u_m(\Delta)$. Such a situation is shown in the example in Figure 4.6 at time t_a. Hence, it is necessary to monitor inter-arrival times for more than merely two consecutive messages in order to guarantee a proper detection of the earliest upper bound violation. It shall be emphasized once more that $R_m(t)$ depicts a message stream in a normal runtime domain rather than an arrival curve in the interval domain and, hence, allows it to present decreasing periods between two events with increasing x-values. The procedure of diagnosing arrival curve violations is explained in detail in Section 4.5.1.

4.4.4 Redundancy-Aware Distribution

The system-level part of the diagnosis framework covers the distribution of the diagnosis tasks to the system resources. Due to the decentralized nature of our approach, different observing resources may diagnose different security attacks and also vary in the corresponding diagnosis time. This means, that it might not be necessary to consider all observable and potentially manipulated message streams on each monitoring resource but only a subset of them, which

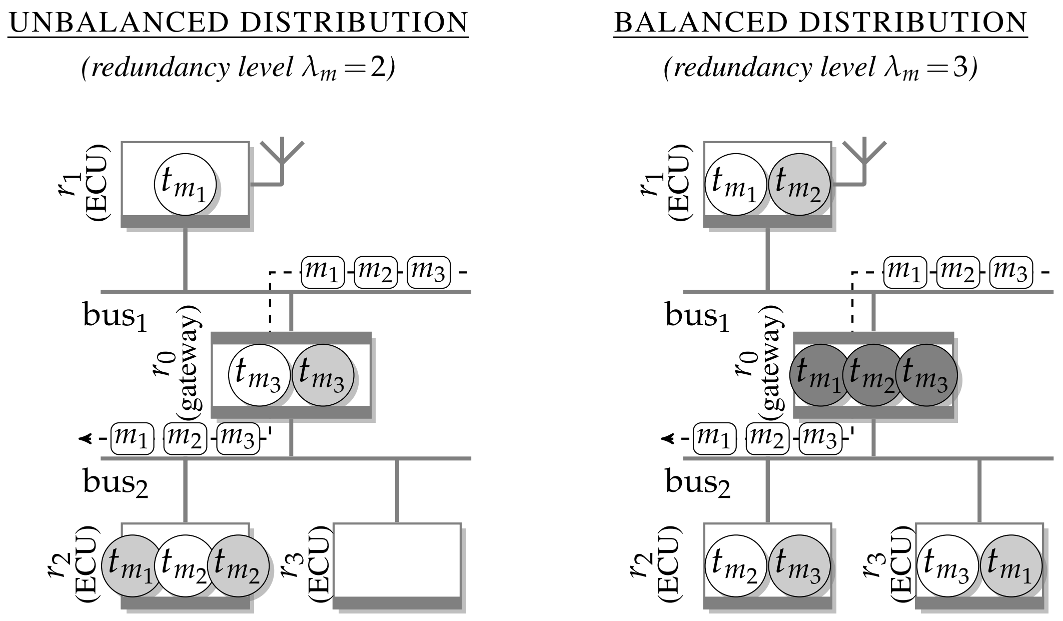


Figure 4.7: *Example for different distributions of diagnosis tasks. The left architecture uses two tasks per message stream ($\lambda_m = 2$) which are unevenly distributed. In the right architecture the tasks are placed evenly with a higher redundancy level ($\lambda_m = 3$).*

would clearly reduce the implementation and computation effort for the security diagnosis. An optimization-based diagnosis distribution at design-time, as proposed in this chapter, can help the designer to chose suitable resources and configurations for particular message stream observations. Additionally, besides the omission of a single point of failure and the reduction of the computation effort, a specific distribution of the monitors can be used in order to introduce redundancy into our diagnosis approach and, hence, improve the reliability.

As a manual distribution of the diagnosis tasks might quickly become very complex for large systems, we propose an automatic ILP-based solution. For this, the system designer can configure two parameters: a redundancy level λ_m for all or only particular message streams as well as tolerance limits θ_r for the monitoring resources. While the former increases the overall diagnosis reliability, the latter is used to balance the diagnosis task distribution. The principle is illustrated in Figure 4.7 using a simple architecture with three ECUs, two buses and a gateway, where the message streams m_1, m_2 and m_3 utilizing bus$_1$ and bus$_2$ are being monitored by the corresponding tasks. For instance, the task t_{m_1} can detect a manipulated m_1, t_{m_2} a manipulated m_2 and so on. The left side of the figure shows an unbalanced diagnosis scenario with a redundancy level of 2, which means that the tasks are not evenly distributed and there exist two diagnosis tasks per message stream. By contrast, the balanced scenario on the right uses three diagnosis tasks per message stream ($\lambda_m = 3$) which are evenly distributed among all monitoring resources. The distribution process and the corresponding ILP are formally described in Section 4.5.2.

4.5 Methodology

We regard two important criteria for the quality of a security diagnosis method, namely, the *timeliness* and the *coverage*. Timeliness guarantees a minimal diagnosis time which is essential for a safety-critical real-time system, such as an automotive E/E architecture. Coverage, on the other hand, ensures that all diagnosable attacks can be correctly and reliably detected. This section provides the formal descriptions for the real-time diagnosis and the diagnosis distribution which both cover these two criteria, respectively.

In the following, $\widetilde{m}$ refers to a manipulated version of the message stream m and is accordingly described by the period $p_{\widetilde{m}}$ and a worst-case jitter $j_{\widetilde{m}}$.

4.5.1 Real-Time Detection: Formulation

Within an in-vehicle network, the main purpose of manipulating traffic is to make one or several system resources unavailable or disrupt the corresponding services by decreasing the periods of one or more message streams beyond the allowed limits. As a timely detection of an attack is crucial in a safety-critical system, an obvious approach is, to test for arrival curve violations of specific message streams at the arrival instants of the corresponding messages. For this, at the monitoring resource, an event trance function $\mu_m(n)$ provides timestamps t_{n-1} for message instances of the stream m (see Equation (4.1)). However, as mentioned in Section 4.4, for a correct diagnosis it might not be sufficient to simply evaluate the inequality in Equation 4.5 after each received message. For example, in Figure 4.6, such a procedure would often lead to a negative test result, because in the most cases the intervals between two messages comply with the upper arrival curve interval for the first observation. In contrast to this, by checking after two consecutive message arrivals, the test would be positive, as depicted in the motivating example at time t^o. A sufficient observation time is especially crucial when considering a bursty behavior of the message stream with the upper arrival curve steeply ascending in the beginning before swinging into a periodic run. At the same time, checking for an attack only after a particular number of detected messages can also be an unfavorable strategy, as a message count can indeed fall below α^u before exceeding it again, as demonstrated in Figure 4.6 at time t_a. Clearly, it seems to be important to know the history of a monitored message stream, i.e., the arrival times of the previous messages.

4.5.1.1 Detection Algorithm

Considering the issues discussed above, in order to detect the violation of an upper arrival curve as defined by Equation 4.5, we adopt the *leaky bucket* approach used to validate the runtime conformity of event arrivals in the context of network calculus [78]. The method has been selected as it is lightweight and can be efficiently implemented, which makes it especially suitable for hard real-time systems, such as automotive E/E architectures. The basic idea of this

Algorithm 3 Testing for compromised message streams on a resource r

```
Require: $\bigcup_{m,s}(\delta_{m,s}, \nu_{m,s})$    ▷ tuples describing the arrival curves $\alpha_m^u$ for all
                                                 message streams $m$ monitored by resource $r$
 1: for $s \in \{1,..,n\}, m \in M_r$ do                    ▷ initialize timers and counters
 2:     $TIM_{m,s} \leftarrow \delta_{m,s}$
 3:     $CNT_{m,s} \leftarrow \nu_{m,s}$
 4: end for
 5: for $m \in M_r$ do                              ▷ analyzing monitored message streams
 6:     if $received(m) =$ TRUE then                         ▷ message from $m$ arrived
 7:         for $s \in \{1,..,n\}$ do
 8:             if $CNT_{m,s} = \nu_{m,s}$ then                    ▷ first message in $\delta_{m,s}$
 9:                 $TIM_{m,s} \leftarrow \delta_{m,s}$
10:             end if
11:             $CNT_{m,s} \leftarrow CNT_{m,s} - 1$                 ▷ reduce message counter
12:             if $(CNT_{m,s} < 0)$ then
13:                 $attacked(m)$                    ▷ report attack on message stream $m$
14:             end if
15:         end for
16:     end if
17:     for $s \in \{1,..,n\}$ do                   ▷ adapt message counters after timeout
18:         if $timeout(TIM_{m,s}) =$ TRUE then
19:             $CNT_{m,s} \leftarrow \min(CNT_{m,s} + 1, \nu_{m,s})$
20:             $TIM_{m,s} \leftarrow \delta_{m,s}$
21:         end if
22:     end for
23: end for
```

approach is to monitor the numbers of message arrivals in all particular time intervals which define an upper arrival curve.

Algorithm 3 illustrates the detection procedure for an observing resource r monitoring the message streams $m \in M_r$ with given upper arrival curves $\alpha_m^u(\Delta)$. It is mainly based on the parameters listed below.

p_m	nominal period of a message stream m
j_m	worst case jitter defining the maximum deviation from p_m
$\delta_{m,s}$	time interval between message arrivals for a particular curve step s
$\nu_{m,s}$	maximum message number in the interval $\delta_{m,s}$
$\rho_{m,s}$	parameter for the message counter reduction in the modified algorithm

The algorithm is based on the event stream regulation presented in [47] and uses the knowledge that each arrival curve can be conservatively approximated by a minimum on a set of periodic staircase functions [75]. More precisely, distinct *step widths* of an arrival curve are represented by n tuples $(\delta_{m,s}, \nu_{m,s})$ with $s \in \{1,..,n\}$ in ascending order, where $\nu_{m,s} = \alpha^u_m(\delta_{m,s}) - 1$ is the maximum allowed message number until the interval $\delta_{m,s}$ has elapsed.

Regarding the detection procedure, first, an initialization process (lines 1-4) allocates the time intervals $\delta_{m,s}$ and the corresponding message numbers $\nu_{m,s}$ to timer variables and message counters, respectively, for each regarded step s and message stream m. Then, during a continuous detection process (lines 5-23), the diagnosing resource is checking for arriving messages (line 6) and expired timers (line 18) of each m. When a message arrives, the timers of each *full* message counter variable are restarted and all $CNT_{m,s}$ are decreased by one (lines 8-11). If during this process a variable falls below zero, an arrival curve violation occurred and an attack on the message stream m is reported (lines 12-14). Finally, after a timer $TIM_{m,s}$ has expired, $CNT_{m,s}$ is incremented by one if it lies below the initial value $\nu_{m,s}$ and the timer is reset again (lines 17-22).

The evidence of the correctness of Algorithm 3 is related to the formal proofs presented in [47] and can be easily derived from them. It is therefore omitted here.

4.5.1.2 Complexity Reduction (for periodic message streams)

By iterating over both message streams and distinct curve steps, the presented algorithm is of complexity $\mathcal{O}(n^2)$ and, hence, its computation time increases quadratically in the number of message streams. However, this complexity seems manageable, as the presented method does not require tuples for each single step of an arrival curve, but only for the distinct periods describing the curve. For consecutive periodic steps, the timers would expire at the same time instant, leading to only one tuple per such periodic section.

Complementary to the general definition of the detection algorithm covering different variants of arrival curves, in the following we present a modification targeting the automotive E/E architectures in particular. The efficiency of the presented algorithm can be increased, given the prevalent periodicity of the message-based communication in automotive networks, where the boundaries of message streams can often be described by arrival curves composed of a nominal period p and a worst case jitter j. Based on the derivations for horizontally shifted staircase functions defined in [75], we can slightly modify Algorithm 3 and adapt it for a periodic-with-jitter model. As a result of the modification, it only needs one tuple as input but requires an additional parameter ρ for the message counter reduction. The three parameters are defined in Equations (4.6a)-(4.6c), where $\gcd(x_1, x_2)$ is a function returning the *greatest common divisor* of x_1 and x_2.

$$\delta_m = \gcd(p_m, p_m - j_m) \tag{4.6a}$$

$$\nu_m = 2 \cdot \rho_m - \frac{p_m - j_m}{\delta_m} \tag{4.6b}$$

$$\rho_m = \frac{p_m}{\delta_m} \tag{4.6c}$$

Now, after initializing the timer with δ_m and the counter with ν_m, the only change in the code needs to be done in line 11 where the counter is decreased by ρ_m rather than by 1 ($CNT_m \leftarrow CNT_m - \rho_m$). The modified algorithm requires only one timer variable (i.e., $s \in \{1\}$) per monitored message stream resulting in a linear complexity, which indicates a good scalability and promises an efficient implementation by reducing the memory and runtime overhead of the diagnosis.

4.5.1.3 Numerical Example

To illustrate the procedure, we apply it to the scenario in the motivating example in Figure 4.1, where a message stream m with period $p_m = 30$ ms and worst case jitter $j_m = 15$ ms is attacked and replaced by a compromised message stream $\widetilde{m}$ with $p_{\widetilde{m}} = 20$ ms and no jitter. The parameters for the diagnosis algorithm result in $\delta_m = 15$ ms, $\nu_m = 3$ and $\rho_m = 2$, meaning that the timer TIM_s is loaded with 15 and the counter CNT_m is set to 3 and reduced by 2 when a message arrives. Neglecting the actual computation times and assuming that the first message of $\widetilde{m}$ is detected instantly after the algorithm has been started, then the detection time (i.e., the time until the method attacked(m) is reached) will be exactly 40 ms. The fast execution time ensures that an attack can be diagnosed early in order to initiate any necessary counter measures.

4.5.2 Redundancy-Aware Distribution: Formulation

Our diagnosis approach does not use a central diagnostic component but rather is designed for a decentralized implementation where the introduced detection algorithm is simultaneously executed for multiple message streams on different resources. This approach guarantees, on the one hand, a continuous diagnosis functionality in case of a system fault or attack disabling one or more resources. Besides the removal of a single point of failure, the decentralized concept also facilitates the coverage of the entire architecture, as the diagnosis method can be easily deployed into remote parts of the system, such as different sub-networks. On the other hand, it allows the system designer to evenly distribute the diagnosis effort amongst the resources, which also increases the overall reliability. Moreover, in order to improve the security for safety-critical messages, a *redundancy level* is used, which defines how many diagnosis tasks are used on different resources to monitor one particular message stream.

First of all, an essential requirement is that each message-transmitting resource, such as an automotive bus, is connected to at least one computation resource (e.g., ECU, gateway), providing the ability for a security diagnosis implementation. For the following definitions, we make use of the graph-based specification of the system architecture. As described earlier, the graph $G_R = (R, E_R)$ connects resources $r \in R$ (e.g., ECU and buses) through architectural links $l \in E_R$ where the latter are defined as resource pairs (r_i, r_j). Using two sets R_{bus} and R_{ecu} containing the bus and the computational resources, respectively, as well as a binary function

$\kappa(r)$ indicating if a resource r is capable of implementing the diagnosis tasks ($\kappa(r) = 1$) or not ($\kappa(r) = 0$)[1], we can define the coverage of diagnosis at system level, as shown in Equation (4.7).

$$\forall m \in M : \quad \begin{aligned} &\exists r \in R_{ecu} : \kappa(r) \wedge (r, r_b) \in E_R \wedge (m, r_b) \in E_M \\ &\text{with } r_b \in R_{\text{bus}} \text{ and } R_{\text{ecu}} \cup R_{\text{bus}} = R \end{aligned} \tag{4.7}$$

As required, the equation demands that for each message stream m there exists at least one resource that provides diagnosis functionality and, at the same time, contains an architectural link $l \in E_R$ to the bus resource r_b serving the corresponding message m. It describes the minimal requirement for the capability of the system to detect security attacks. However, besides an efficient runtime performance and a satisfactory diagnosis time, the proposed decentralized approach shall provide diagnosis reliability as well. This can be achieved with the help of a redundant message stream monitoring which takes advantage of the fact that bus-attached resources can also detect messages which are not destined for them. As mentioned above, one goal of our decentralized security diagnosis approach is to evenly distribute the diagnosis effort to the monitoring resources while maintaining a predefined redundancy level λ_m for the message streams. This problem can be formulated as an ILP containing several constraints and one optimization objective. The ILP and its corresponding parameters are defined in the following.

$M_r \subseteq M$	subset of the message streams which are detectable by a resource $r \in R$
$\kappa(r) : R \to \{0, 1\}$	*implementation function* indicating if a resource r is capable of implementing a diagnosis task (1) or not (0)
$\lambda_m \in \mathbb{N}$	*redundancy level*, defining on how many different resources one particular message stream m is being monitored
$\overline{\lambda_r} \in \mathbb{R}$	average redundancy level on a resource $r \in R$
$\theta_r^l, \theta_r^u \in \mathbb{R}$	*tolerance limits*, define the lower and upper limit for the monitoring messages around the system average $\theta_r^l \leq \frac{\lvert M \rvert}{\sum_{\forall r \in R} \kappa(r)} \leq \theta_r^u$
$\mathbf{o}_{r,m} \in \{0, 1\}$	*observation switch*, determining if a message stream observable by resource r will be omitted (0) or monitored (1) during security diagnosis
$\mathbf{x}_r \in \mathbb{N}$	*monitoring variable*, determining the number of monitored message streams on resource $r \in R$
$\mathbf{y}_r \in \{0, 1\}$	*range switch*, determining whether the monitoring variable $\mathbf{x}_r$ resides in the desired range (1) or not (0)

[1]Reasons for a resource being incapable of diagnosis can range from insufficient computational performance to the lack of fundamental architectural capabilities, e.g., in case of a bus resource.

$$\underset{\mathbf{y}_r \in \{0,1\}}{\text{maximize}} \sum_{\forall r \in R} \mathbf{y}_r \tag{4.8a}$$

subject to:

$$\forall m \in M: \quad \sum_{\substack{\forall r \in R \\ \wedge m \in M_r}} (\kappa(r) \wedge \mathbf{o}_{r,m}) = \lambda_m \tag{4.8b}$$

$$\forall r \in R: \quad \sum_{\forall m \in M_r} (\kappa(r) \wedge \mathbf{o}_{r,m}) = \mathbf{x}_r \tag{4.8c}$$

$$\forall r \in R: \quad \mathbf{x}_r \leq |M_r| \tag{4.8d}$$

$$\forall r \in R: \quad \mathbf{x}_r \leq \theta_r^u \cdot \overline{\lambda_r} \cdot \mathbf{y}_r + |M_r| \cdot (1 - \mathbf{y}_r) \tag{4.8e}$$

$$\forall r \in R: \quad \mathbf{x}_r \geq \theta_r^l \cdot \overline{\lambda_r} \cdot \mathbf{y}_r \tag{4.8f}$$

Constraint (4.8b) uses the *redundancy level* λ_m to determine the number of observations of one particular message stream monitored on different resources. For instance, a value of $\lambda_m = 1$ means no redundancy and the message stream will be monitored by only one resource. At the same time, Constraint (4.8c) uses a *monitoring variable* $\mathbf{x}_r$ to determine the number of message streams on each resource r which are simultaneously monitored during the security diagnosis. In both equations, the implementation function $\kappa(r)$ is used to indicate the diagnosis capability of a resource. As defined by Constraint (4.8d), the monitoring variable is limited by the cardinality of M_r, the set of all streams observable by r. Constraints (4.8e) and (4.8f) are used to define the desired range for the monitoring variable and, thus, the number of observations on each resource. For this, the *tolerance limits* θ_r are used which state how much the observation number can deviate from the system-wide average, where the latter is calculated by the ratio of all monitored message streams to all monitoring resources. To include the influence of message streams with a redundancy level other than one, the tolerance limits are multiplied with the average redundancy level value $\overline{\lambda_r}$ for each regarded resource. The *range switch* $\mathbf{y}_r$ is used to decide if the predefined range has been met. Finally, the objective of the optimization problem is to maximize the sum of range switches over all resources, as defined in Equation (4.8a). This ideally should lead to a balanced distribution of message stream monitors in the investigated automotive E/E architecture.

4.6 Diagnosis Evaluation

In this section, the presented framework is subjected to a number of experiments which are evaluated accordingly. In the scope of this work, we are especially interested in investigating the general feasibility and efficiency of our diagnosis method and, by doing so, providing a convincing proof of concept for the security diagnosis in automotive E/E architectures. After a brief description of the test environment, we use a test case evaluation to demonstrate both the fast detection of manipulated message streams and the automated distribution of diagnosis task. Furthermore, a runtime analysis shows the efficiency and scalability of our framework. Finally, a case study illustrates our approach in more detail using a large E/E architecture with more than 100 ECUs.

Test environment. Altogether, 460 synthetic test cases have been generated for the experimental results. Following our system model, as it was introduced in Section 4.5.2, each of these test cases comprises the specification for an automotive in-vehicle network, including the graph-based descriptions of the system architecture, the application and the corresponding mappings between them. In an architecture graph the vertices represent ECUs, gateways and buses while the edges stand for the links connecting them (compare Figure 4.5). In an application graph, on the other hand, vertices represent tasks and messages and their mutual dependences are depicted by edges. Essential parameters of the test cases are listed in Table 4.1 together with their minimum and maximum values. As indicated, the test cases encompass E/E architecture sizes of up-to-date vehicles with more than 100 ECUs.

All experiments were carried out on an Intel Core i7 with 3.4 GHz and 16 GB RAM. The system specifications were generated with the help of OPENDSE [87] and the ILP-based computations used GUROBI version 6.0 [39] as solver.

4.6.1 Test Cases Evaluation

In the following, all test cases are evaluated with respect to the duration of diagnosis, the efficiency of its distribution as well as the corresponding computation times.

4.6.1.1 Diagnosis Time

At first, we want to demonstrate the general feasibility of the proposed security diagnosis algorithm. For this, Figure 4.8 illustrates the periods and diagnosis times averaged over all message streams in each test case. More precisely, the graph depicts the predefined nominal periods (ring marks), the generated compromised periods (cross marks) as well as the corresponding

Table 4.1: *Reference values of essential parameters for 460 test cases.*

	ECUs	*Buses*	*GWs*	*Tasks*	*Message streams*	*Nominal period* *[ms]*	*Compromised period* *[period ratio]*
MIN	2	1	0	6	3	5	5%
MAX	117	4	1	366	438	100	90%

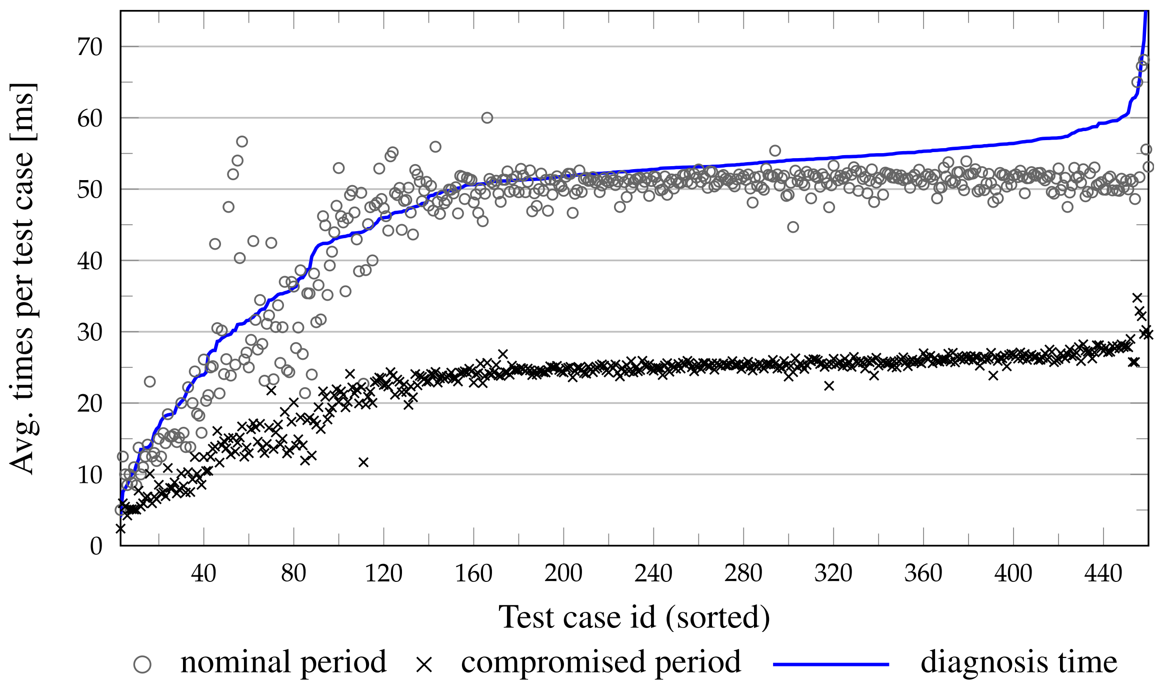


Figure 4.8: *Average nominal (predefined) periods (O) and compromised (randomly generated) periods (X) of the message streams. The solid curve illustrates the average diagnosis times needed to detect a potential attack. To improve legibility, the results for the test cases are ordered by increasing average diagnosis times.*

diagnosis times (solid curve), respectively. For each message stream, the compromised (i.e., reduced) period has been randomly generated and lies in a range between 5% and 90% of the nominal period. Due to the normal distribution of the random variable, the associated average values of the compromised period are located roughly at 50% of the nominal period curve and show a similar trend in the graph. Here, the test cases are sorted by increasing average diagnosis times which leads to a smooth diagnosis curve but results in a slightly dispersed distribution of the period points.

A general inspection of the graph indicates a clear dependency between the the diagnosis time and the message stream periods. This is in accordance with the diagnosis algorithm which uses the nominal period for the parameter calculation necessary for a correct violation detection. At the same time, it is visible that, on average, the diagnosis times themselves are in the range of the nominal message stream periods, lying only slightly above their average towards higher periods. Consequently, the overall outcome in Figure 4.8 shows that a real-time diagnosis of a compromised message stream using the introduced method is feasible. Moreover, the numeric results of the diagnosis times are not notably exceeding the nominal periods and indicate a promising application of the security diagnosis algorithm in practice.

4.6.1.2 Diagnosis Distribution

Next, the ability of our approach to distribute the diagnosis task shall be investigated. Figure 4.9 shows three graphs depicting the average amount of monitored message streams per resource for each test case (curve with error bars), which are calculated in accordance with the ILP defined in Section 4.5.2. The error bars represent the appropriate standard deviation from these average values and, hence, are an indicator for how evenly the monitored messages are distributed among the observing resources. It holds true that lower standard deviations, i.e. shorter error bars, correspond to a more even distribution and vice versa. A second curve in each graph shows the percentage of the results with a correct adherence to the predefined monitoring range (i.e., cases where $\mathbf{y}_r = 1$). To evaluate the effect of the tolerance limits θ_r, the curves are depicted for three desired ranges around the average message per resource number, namely $\pm 5\%$ (top), $\pm 20\%$ (center), and $\pm 50\%$ (bottom).

When comparing the compliance curve with the message stream number curve, it can be seen that a lower percentage of hits for the monitoring range is met by a higher standard deviation. This correct and presumed correlation verifies the ILP formulation and allows us to estimate an appropriate tolerance level. Obviously, while a too small θ_r value in many cases prevents the solver from finding an optimal solution (compare upper graph), a too large value results in an overall high standard deviation (compare lower graph). From the three selected values, the middle one ($\pm 20\%$) seems to lead to the smallest standard deviation among all test cases and, hence, to a most balanced distribution of the monitored message streams among all resources. Although for particular architectures the lowest tested tolerance level can indeed lead to an optimal diagnosis distribution (e.g., test cases around 110 and 250), a level of $\pm 20\%$ provides satisfying results for a broad range of system specifications and could be used by the

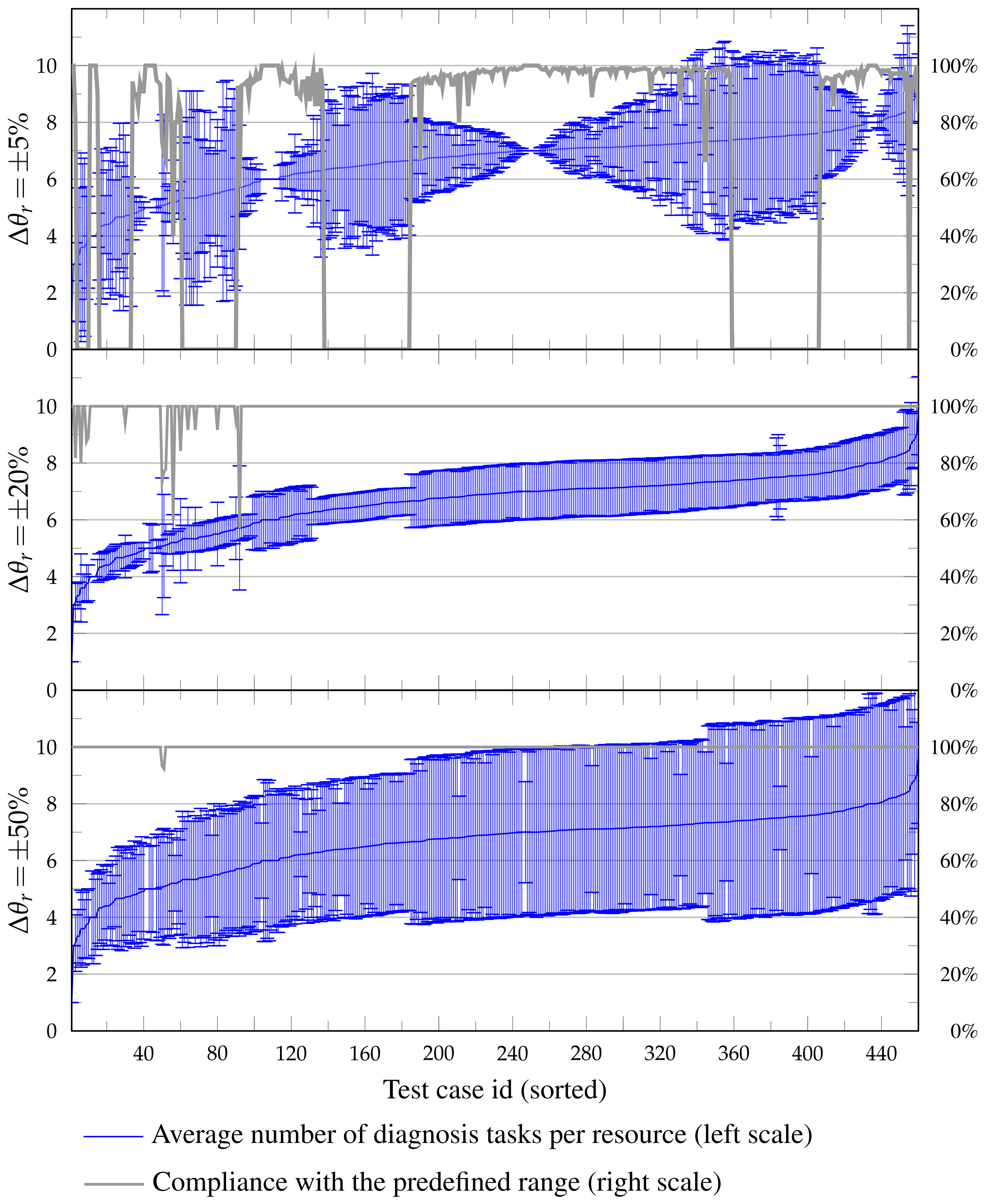


Figure 4.9: *Three graphs showing the average amount of diagnosis tasks (i.e., number of monitored message streams) per resource together with their standard deviation for each test case. Additionally, a second curve indicates the compliance with the predefined monitoring range (right y-axis). The curves are plotted for three different tolerance values. The results are shown for a redundancy level* $\lambda_m = 2$.

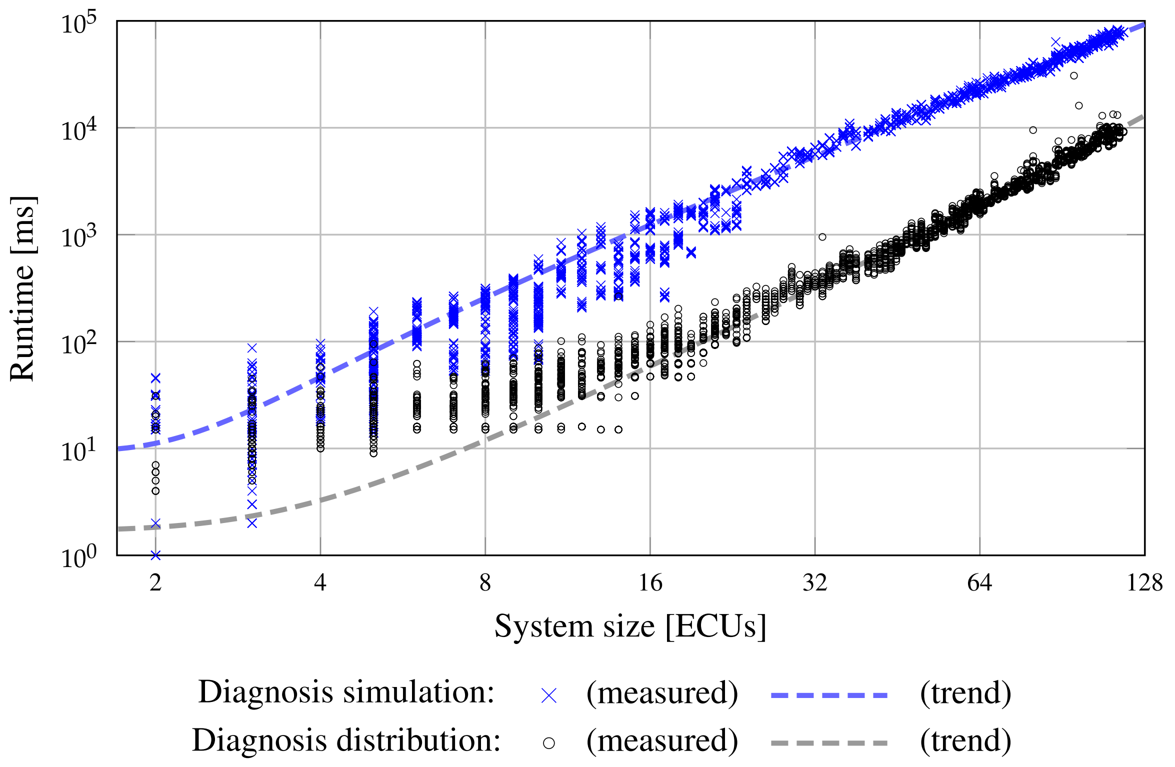


Figure 4.10: *Runtime performance of the security diagnosis framework for different system sizes. The graph shows the computation times for each test case (including different configurations) for the ILP-based distribution of monitored message streams (O). It also illustrates the detection algorithm-based diagnosis simulation (X). Additionally, two trend curves indicate a polynomial computational complexity.*

system designer as a starting point for a fine-grained adjustment. Considering the number of redundantly monitored message streams, all results in Figure 4.9 are calculated for a redundancy level $\lambda_m = 2$. A lower ($\lambda_m = 1$) and a higher ($\lambda_m = 3$) level have been evaluated as well, and in both cases the overall tendency discussed above remained similar. The case study investigated below provides more detailed results concerning the redundancy levels.

4.6.1.3 Runtime

Regarding the timing behavior, a short detection time is of utmost relevance for the security diagnosis method and has been discussed by means of Figure 4.8. However, within our diagnosis framework, both the algorithm-based diagnosis simulation and the ILP-based distribution of diagnosis tasks are performed at design-time, and, hence, do not have such strict demands on the runtime. Nevertheless, like most processes in the automotive industry, also the system design is must be cost-efficient and, thus, low computation times are beneficial. To investigate this, a runtime analysis has been performed and its results are shown in Figure 4.10. Here, the computation times of the simulation and distribution are regarded separately and plotted as a function of the system size depicted by the number of ECUs in the architecture. Each measured point represents the runtime for the simulation and distribution part, respectively, of one test case with a particular parameter configuration where the largest analyzed architecture is composed of 117 ECUs. In order to get a better visualization of the distribution and behavior of the different runtimes, two dashed curves illustrate trends for both the simulation and distribution, respectively[2]. The calculated trends are polynomial curves indicating a quadratic and cubic complexity in the number of ECUs for the simulation and distribution, respectively.

At the same time, it can be seen that, in general, the duration of the computation does not exceed 100 seconds even for the largest analyzed architectures. In detail, the longest simulation takes 83.1 seconds and the longest distribution 30.7 seconds. Considering that today's modern automotive networks consist of a lower three-digit number of ECUs, the results illustrate practical computation times of our diagnosis framework and indicate a good scalability for even larger architectures in the future. Note, that both axes in Figure 4.10 are in logarithmic scale and the numbers of test cases for different system sizes may vary, causing noticeable changes in the distribution of the single measurement points.

4.6.2 Case Study

An automotive case study shall illustrate the functionality of the proposed distributed security diagnosis in detail. The study is based on a modern state-of-the art automotive E/E architecture which consists of 144 ECUs[3], five buses and one automotive gateway. On the application side, the system uses 618 tasks which communicate via 321 messages with message periods ranging from 5 to 100 ms and message jitters determined according to the delay calculation for event-triggered systems proposed in [88].

The results of the case study are shown in Figure 4.11 where for each of the four graphs, monitoring resources are depicted on the x-axis and the number of diagnosis tasks (i.e., the number of monitored message streams per resource) on the y-axis. The bars in each graph stand for a different tolerance level, namely 5% (top graph) 10% (second graph), 20% (third graph), and 50% (bottom graph), with the different shades representing three redundancy levels, $\lambda_m = 3$ (plain bars), $\lambda_m = 2$ (north-east striped bars) and $\lambda_m = 1$ (north-west striped bars), for which each message stream is monitored by three resources, two resources, and one resource, respectively. Consequently, the increasing height of each of the shaded stacks towards the bottom clearly indicates the growing number of monitoring tasks with each higher redundancy level. It can be seen that for the two middle graphs with tolerance levels of 10% and 20%, the diagnosis tasks for the different redundancy levels are quite evenly distributed among the resources, with approximately 7, 5 and 3 tasks per resource, respectively. However, for $\lambda_m = 1$, the second graph shows a slightly larger variance leaving a tolerance level of 20% as the most optimal option. By contrast, the graphs illustrating a tolerance level of 5% and 50% show much

[2]The approximation has been determined by the method of least squares.

larger irregularities with up to 14 tasks difference between the resources. Summing up, this result shows how a proper adjustment of the optimization parameters during the system design can help to properly distribute the diagnosis tasks to monitoring resources.

Computational overhead. The presented results are based on the analysis and simulation of synthetic test cases. That is why, at this time, we can only give an estimation of the computational overhead which will be caused by the implementation of the security diagnosis method on real hardware. For the estimation, we orientate ourselves on the implementation results of the distributed diagnosis of permanent faults described in Section 3.5.3. Although the fault diagnosis algorithm is different in its basic functionality, it is relatively comparable regarding its length and complexity as well as the way it deals with multiple message streams. On the basis of those measurements, the proposed security diagnosis, monitoring about 15 message streams per resource, would utilize a processor by not more than 5% and use less than 10 KB of memory. Keeping the computational overhead in such a low range will facilitate its implementation and support the efficiency of our security diagnosis approach. The corresponding prototypical implementation is part of the future work.

[3] As of the 144 resources 24 have no monitoring capability, for clarity, only the results for the remaining 120 resources are depicted in Figure 4.11.

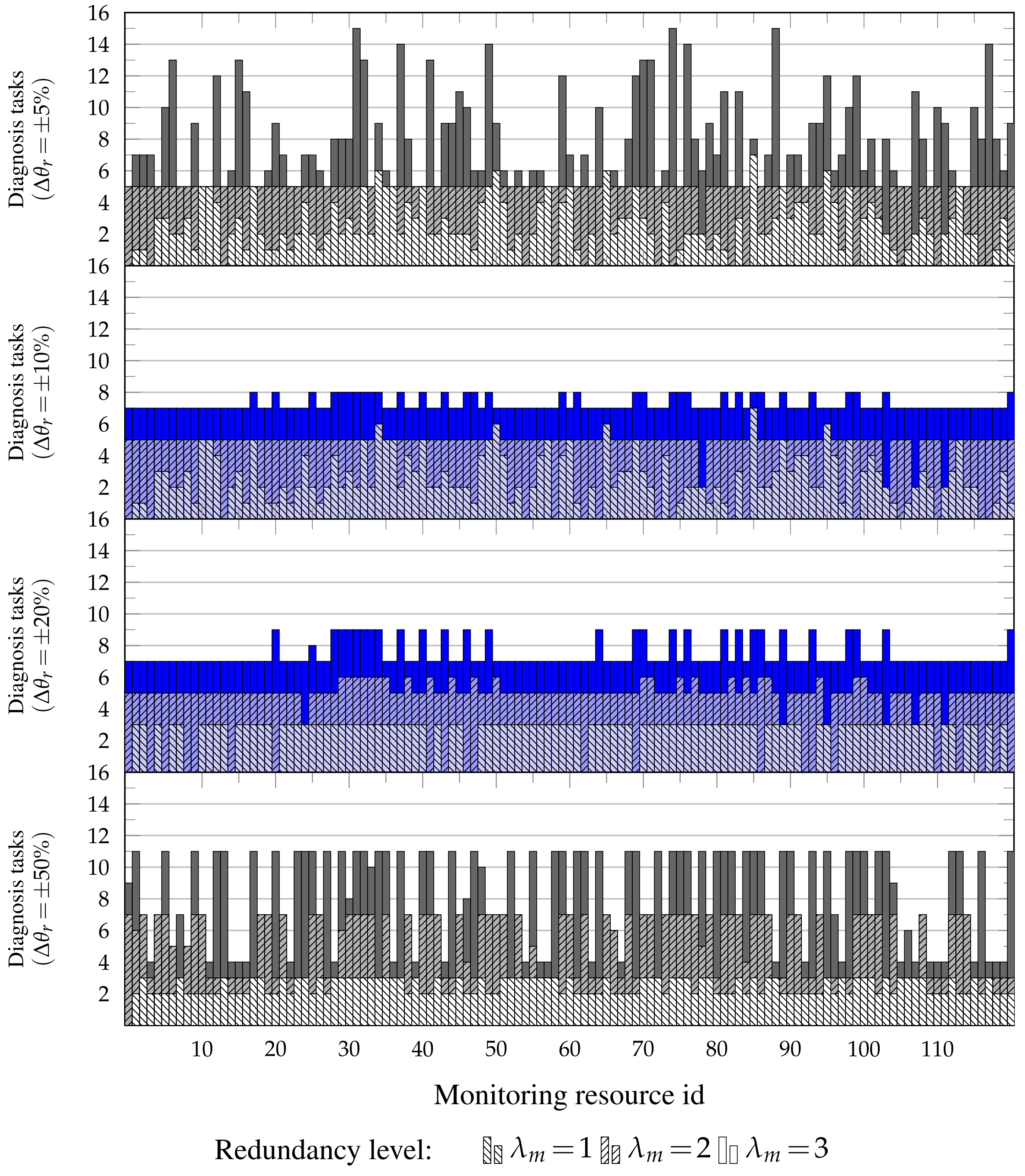


Figure 4.11: *Amount of diagnosis tasks (i.e., number of monitored message streams) allocated to different resources for an exemplary automotive E/E architecture with 144 ECUs. The distribution is most balanced for the tolerance levels ±10% and ±20%, as depicted by the two middle graphs. It varies largely for ±5% and ±50%.*

4.7 Conclusion

In this chapter, we presented a novel approach to diagnose security attacks on the communication in safety-critical distributed systems, such as automotive E/E architectures. The overall goal lies in improving the security in a decentralized and implicit manner, which allows both enhancing the reliability of the diagnosis and reducing the costs at design time. Motivated by a realistic scenario describing the intrusion into an electric vehicle, we have emphasized the importance of an early, distributed and reliable diagnosis which can be assured by our approach. The proposed framework consists of two stages comprising the analysis of a given system specification and the actual diagnosis functionality. The first stage defines the necessary message arrival curves and provides additional architecture-related parameters which are used in the second stage for both the component-based real-time detection of altered traffic patters and the distribution of the corresponding diagnosis tasks among suitable resources. The implicit and decentralized nature of the proposed approach allows the security diagnosis to be efficiently implemented in an existing system architecture without the risk of a single point of failure, hence, increasing the overall diagnosis reliability. We demonstrate the general feasibility and efficiency of our approach based on the evaluation of a number of test-cases including 460 synthetically generated system specifications with sizes up to 117 ECUs and, hence, applicable to modern automotive E/E architectures. Additionally, a case study with 144 ECUs provides a more detailed discussion of the security diagnosis including a realistic estimation of the computational overhead below 5%.

Future work. As part of future work, we want to extend the optimization-based diagnosis task distribution by allowing the designer to define more specific constraints, such as different criticality levels or specifically vulnerable system resources. Furthermore, we are looking into other security threats which might result in an alteration of the traffic patterns, such as the removal or overwriting of lower priority messages or the manipulation of the message content causing a potential delay of the sending instance. Finally, the security diagnosis method shall be implemented on an existing research platform for automotive E/E architectures to further investigate its applicability as well as the actual computational overhead.

CHAPTER 5

Concluding Remarks

The rapid development of automotive electronics is a double-edged sword. On the one hand, sophisticated engine or drive train control applications enhance the vehicle's efficiency, ABS, Electronic Stability Program (ESP) or ADAS greatly contribute to the safety of all road participants and HMI or infotainment improve the overall driving experience. On the other hand, however, the growing number of VLSI IC-components and interfaces to the outside world increase the susceptibility for faults and security attacks, respectively.

The main research goal of this thesis was to design and investigate approaches to system-level diagnoses which can improve the safety, security and overall reliability of modern automotive E/E architectures.

5.1 Summary

In particular three system level diagnoses for distributed automotive architectures have been proposed, addressing intermittent faults, permanent faults and security attacks.

The intermittent fault diagnosis uses expected transient fault rates to identify cases where resources are affected by an increased (unexpected) occurrence of non-permanent faults. Considering possible data dependencies between tasks on different resources, the proposed approach is able to diagnose faulty components that do not have integrated fault detectors or monitoring functionality. For this, the outcomes of existing distributed plausibility tests are collected, analyzed and compared based on special predefined expectation matrices. To gain a better insight as well as a broader scope for a future practical use, four different evaluation strategies, consisting of two vector-based and two probabilistic methods, have been proposed and compared. Additionally, it has been shown, that the presented diagnosis methods can be applied to modern many-core architectures as well, as they are becoming more and more important in the automotive domain due to an increased demand for computational power. We have demonstrated the overall feasibility as well as an encouraging diagnosability and efficiency for the proposed approach (e.g., up to 95 % and 75 % correct test outcomes for the automotive and many-core test cases, respectively).

In contrast to intermittent faults where observation times can be relatively long, it is imperative that the diagnosis of permanent faults happens instantly in order to apply countermeasures as fast as possible. The proposed permanent fault diagnosis uses special diagnosis functions which define distinct traffic patterns for different fault scenarios from the perspective of each observing system resource. At runtime, monitoring all message streams and evaluating the diagnosis functions accordingly on different resources provides a reliable real-time diagnosis of the entire system. Here, to improve the flexibility and efficiency of the approach, a trade-off analysis allows to optimize the diagnosis functions towards a better observation time or a lower number of monitored message streams. Furthermore, it has been presented, how a diagnosis-aware system design can further optimize the permanent fault diagnosis and even extend the number of detectable resources. Complementary to test cases and a case study proving the method's feasibility, an implementation on a research platform demonstrated its low computational and memory overhead.

Finally, the decentralized monitoring of existing message streams can be also applied for the detection of malicious attacks on the automotive E/E architecture. It has been shown how an algorithm for the verification of the runtime conformity can be used in the context of a message-based security diagnosis, for instance, to detect DoS or message flooding attacks. For this, a strategy has been proposed, to distribute the corresponding diagnosis tasks evenly among the system resources, considering user-specified redundancy levels and preferences for observing components. Experimental results consisting of several hundred test cases and an automotive case study have demonstrated the applicability and efficiency of the proposed security diagnosis method for in-vehicle architectures with more than 100 ECUs.

Summing up, we have presented how an implicit and decentralized methodology that takes into account system design knowledge about the corresponding architectures can offer a good and reliable diagnosis while ensuring a minimal overhead and system intrusion. Consequently, regarding the overall outcome summarized above, it can be concluded that the research goal of this thesis has been reached.

5.2 Limitations

One of the main aspects for the presented diagnosis strategies is their lightweight implementation necessitating none or only minimal adaptations to existing specifications of distributed systems. However, one consequence of this approach is that it cannot cover all possible fault and attack scenarios. For instance, the intermittent fault diagnosis is depending on the propagation of soft-errors towards devices executing specific plausibility test tasks in order to properly identify an increased fault rate. Stand-alone devices or faults that are masked and do not manifest themselves as errors need to be diagnosed with different, possibly more hardware-oriented detection methods. In terms of permanent faults and security attacks, the proposed methods are especially analysing message transmission times without considering their content. In cases where the contents of messages are of interest for the diagnosis, different detection strategies must be applied. The solutions in this thesis are considered fully orthogonal to other existing and future fault and attack diagnosis methods.

5.3 Future Work

The proposed system-level diagnoses can be further improved and extended in multiple aspects, some of which are discussed in the following.

Generally, the presented decentralized diagnosis methods were designed with the single-fault assumption in mind which implies that only one independent fault occurs at a time. Although this is a common approach for automotive architectures design and testing [77], the growing system complexity and shrinking components also increase the risk of multiple independent faults at a time or at least withing a short time interval. In this context, the ILP-based diagnosis methods for *intermittent faults* already innately support multiple faulty resources and for the vector-based methods the expectation matrices could be adapted accordingly. In order to diagnose more than one *permanent fault* simultaneously, an additional set of specific fault scenarios and corresponding diagnosis functions targeting several resource failures could be generated at design time and executed in parallel to the single fault functions. Regarding the *security threats*, a diagnosis of multiple attacks on different message streams is already covered by the appropriate distribution of monitors among the observing resources.

An important property of the overall diagnosis methodology is the non-intrusiveness, meaning that the detection methods merely utilize the existing communication or built-in plausibility tests and the only overhead comes from the storage and execution of necessary fault and intrusion monitors. Nevertheless, in the context of the permanent fault diagnosis, it has been shown how already small adaptations and extensions of a system specification during the design phase can be beneficial, for instance, by improving a subsequent real-time diagnosis or enhancing its range. Such a diagnosis-aware system design could be also applied to improve the diagnosis of intermittent faults. For example, a deliberate and smart placement of plausibility tests might increase the overall positive detection rate and reduce the number of necessary tests. In cases where the execution of particular tasks is not strictly bound to specific resources, a diagnosis-aware mapping of tasks could improve the detection algorithms and reduce the observation times for all presented diagnosis approaches.

A related strategy could be also applied in the area of a diagnosis-aware control and architecture co-design. Here, a joint controller and schedule synthesis could be used to maximize the performance during both fault-free and faulty periods. For example, from two schedules, one with a shorter end-to-end delays but longer diagnosis times and another where this relations are reversed (i.e., longer end-to-end delays and shorter diagnosis times), a multi-objective optimization problem could determine the best solution in terms of the system performance.

Finally, the experimental and evaluation phase of our methods could be extended as well. The presented implementation of the permanent fault diagnosis on a research platform stressed the relevance of applying the theoretical diagnosis concepts on real hardware. On the one hand, this is necessary to investigate their practical use and efficiency but, more importantly, it helps to gain an early insight into potential difficulties and future challenges that were not evident during the conceptual design of the approaches. The aforementioned research platform for automotive E/E architectures or a similar experimental setup could also be used for hardware implementations of the diagnoses of intermittent faults and security attacks. Basically, it is essential that the testing environment resembles a final target system in as many aspects as possible, for example, regarding the number of resources which is an important aspect for us, given the distributed nature of the presented methods. As it could be too expensive to build test architectures with several tens or hundreds of ECUs, an extension of the presented test cases and case studies into a comprehensive and flexible test and simulation platform might improve the further development of the diagnosis strategies as well.

5.4 Outlook

The trend to more complex architectures with more ECUs will most probably slow down and finally revert in the future, leaving a small number of powerful multi-core ECUs, only a fraction of which will be assigned dedicated functionality and the rest will be multi-purpose computing devices [15]. However, this change will not happen overnight. For instance, traditionally, the car development is carried out in an evolutionary rather then revolutionary way meaning that a radical change of E/E architectures will not happen from one vehicle generation to the other. Moreover, even though the number of ECUs might decrease, the corresponding sensors and actuators will remain necessary. Their number will most probably even increase and with novel functionalities they will become smarter and, thus, more communication-intensive. Consequently, it can be expected that distributed diagnosis solutions, such as the ones proposed in the scope of this thesis, will still be relevant in the future.

Bibliography

[1] U. Abelein, H. Lochner, D. Hahn, and S. Straube. Complexity, quality and robustness-the challenges of tomorrow's automotive electronics. In *Proceedings of the Design, Automation & Test in Europe Conference (DATE)*, pages 870–871, 2012.

[2] A. Albert. Comparison of event-triggered and time-triggered concepts with regard to distributed control systems. In *Proceedings of the embedded world Conference*, pages 235–252, 2004.

[3] A. Amory, C. Marcon, F. Moraes, and M. Lubaszewski. Task mapping on NoC-based MPSoCs with faulty tiles: Evaluating the energy consumption and the application execution time. In *Proceedings of the International Symposium on Rapid System Prototyping (RSP)*, pages 164–170, 2011.

[4] E. Armengaud and A. Steininger. Remote measurement of local oscillator drifts in FlexRay networks. In *Proceedings of the Design, Automation & Test in Europe Conference (DATE)*, pages 1082–1087, 2009.

[5] Automotive Open System Architecture. AUTOSAR specification - release 4.2. [Online]. Available: http://www.autosar.org, 2015.

[6] Automotive Open System Architecture. AUTOSAR technical overview. [Online]. Available: http://www.autosar.org/about/technical-overview/, 2015.

[7] M. Barborak, A. Dahbura, and M. Malek. The consensus problem in fault-tolerant computing. *ACM Computing Surveys (CSur)*, 25(2):171–220, 1993.

[8] S. Baruah, G. Buttazzo, S. Gorinsky, and G. Lipari. Scheduling periodic task systems to minimize output jitter. In *Proceedings of the International Conference on Real-Time Computing Systems and Applications (RTCSA)*, pages 62–69, 1999.

[9] L. Ben Othmane, R. Fernando, R. Ranchal, B. Bhargava, and E. Bodden. Likelihood of threats to connected vehicles. *International Journal of Next-Generation Computing (IJNGC)*, 5(3):1–14, 2014.

[10] C. Bolchini and A. Miele. Reliability-driven system-level synthesis for mixed-critical embedded systems. *IEEE Transactions on Computers*, 62(12):2489–2502, 2013.

[11] S. Bono, M. Green, A. Stubblefield, A. Juels, A. D. Rubin, and M. Szydlo. Security analysis of a cryptographically-enabled rfid device. In *Proceedings of the USENIX Security Symposium*, volume 5, pages 1–16, 2005.

[12] K. Borgeest. *Elektronik in der Fahrzeugtechnik*. Springer, 2010.

[13] S. Borkar. Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. *Micro, IEEE*, 25(6):10–16, 2005.

[14] Broadcom. BroadR-Reach physical layer transceiver specification for automotive applications. [Online]. Available: http://www.ieee802.org/, 2014.

[15] M. Broy. Challenges in automotive software engineering. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 33–42. ACM, 2006.

[16] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 100(8):677–691, 1986.

[17] G. Calandriello, P. Papadimitratos, J.-P. Hubaux, and A. Lioy. On the performance of secure vehicular communication systems. *IEEE Transactions on Dependable and Secure Computing*, 8(6):898–912, 2011.

[18] G. Cena, I. C. Bertolotti, T. Hu, and A. Valenzano. Improving compatibility between CAN FD and legacy CAN devices. In *Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI), 2015 IEEE 1st International Forum on*, pages 419–426, 2015.

[19] S. Chakraborty, S. Kunzli, and L. Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 190–195, 2003.

[20] R. N. Charette. This car runs on code. *IEEE Spectrum*, 46(3):1–3, 2009.

[21] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, T. Kohno, et al. Comprehensive experimental analyses of automotive attack surfaces. In *Proceedings of the USENIX Security Symposium*, 2011.

[22] A. Y. Chong and C. S. Chua. *Driving Asia: As Automotive Electronic Transforms a Region*. Infineon Technologies Asia Pacific Pte Limited, 2011.

[23] C. Constantinescu. Trends and challenges in VLSI circuit reliability. *Micro, IEEE*, 23(4):14–19, 2003.

[24] C. B. Cristina and A. Miele. Reliability-driven system-level synthesis of embedded systems. In *Proceedings of the International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT)*, pages 35–43, 2010.

[25] A. Das and A. Kumar. Fault-aware task re-mapping for throughput constrained multimedia applications on NoC-based MPSoCs. In *Proceedings of the International Symposium on Rapid System Prototyping (RSP)*, pages 149–155, 2012.

[26] L. De Moura and N. Bj0.2rner. Z3: An efficient smt solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[27] V. Desmet, Y. Sazeides, and C. Vrioni. Performance implications of hard-faults in non-architectural structures. In *Proceedings of the Reconfigurable and Adaptive Architecture Workshop (RAAW)*, 2007.

[28] M. Di Natale and A. Sangiovanni-Vincentelli. Moving from federated to integrated architectures in automotive: The role of standards, methods and tools. *Proceedings of the IEEE*, 98(4):603–620, 2010.

[29] J. Dickie and M. Rutter. ETAS' AUTOSAR-OS and -RTE reach maturity. *RealTimes - The ETAS Group Magazine*, pages 34–35, 2009.

[30] Elektrobit. EB automotive ECU solutions-AUTOSAR basic software, tooling, functional safety, customization services (brochure). [Online]. Available: http://www.automotive.elektrobit.com, 2014.

[31] P. Emberson and I. Bate. Extending a task allocation algorithm for graceful degradation of real-time distributed embedded systems. In *Proceedings of the International Real-Time Systems Symposium (RTSS)*, pages 270–279, 2008.

[32] P. Fernandez, J. Santa, F. Bernal, and A. Gomez-Skarmeta. Securing vehicular IPv6 communications. *IEEE Transactions on Dependable and Secure Computing*, 13(1):1–14, 2015.

[33] I. Fliss and M. Tagina. Multiple fault diagnosis of discrete event systems using petri nets. In *Proceedings of the International Conference on Communications, Computing and Control Applications (CCCA)*, pages 1–6, 2011.

[34] A. Girault, C. Lavarenne, M. Sighireanu, and Y. Sorel. Fault-Tolerant Static Scheduling for Real-Time Distributed Embedded Systems. Rapport de recherche RR-4006, INRIA, 2000.

[35] M. Glaß, M. Lukasiewycz, C. Haubelt, and J. Teich. Incorporating graceful degradation into embedded system design. In *Proceedings of the Design, Automation & Test in Europe Conference (DATE)*, pages 320–323, 2009.

[36] M. Glaß, M. Lukasiewycz, T. Streichert, C. Haubelt, and J. Teich. Reliability-aware system synthesis. In *Proceedings of the Design, Automation & Test in Europe Conference (DATE)*, pages 409–414, 2007.

[37] R. B. GmbH, editor. *Bosch Automotive Electrics and Automotive Electronics: Systems and Components, Networking and Hybrid Drive*. Springer, 2014.

[38] A. Gothard, R. Kreifeldt, and C. Turner. AVB for automotive use. *AVnu Alliance White Paper*, 2014.

[39] Gurobi Optimization, Inc. Gurobi Optimizer Reference Manual. [Online]. Available: http://www.gurobi.com/, 2015.

[40] G. Han, H. Zeng, Y. Li, and W. Dou. SAFE: Security-Aware FlexRay scheduling engine. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 1–4, 2014.

[41] A. Herkersdorf, M. Engel, M. Glaß, J. Henkel, V. B. Kleeberger, et al. Cross-layer dependability modeling and abstraction in system on chip. In *Proceedings of the Workshop on Silicon Errors in Logic - System Effects (SELSE)*, 2013.

[42] O. Héron, J. Guilhemsang, N. Ventroux, and A. Giulieri. Analysis of on-line self-testing policies for real-time embedded multiprocessors in DSM technologies. In *Proceedings of the International On-Line Testing Symposium (IOLTS)*, pages 49–55, 2010.

[43] T. Hoppe and J. Dittman. Sniffing/replay attacks on can buses: A simulated attack on the electric window lift classified using an adapted cert taxonomy. In *Proceedings of the 2nd workshop on embedded systems security (WESS)*, pages 1–6, 2007.

[44] T. Hoppe, S. Kiltz, and J. Dittmann. Adaptive dynamic reaction to automotive IT security incidents using multimedia car environment. In *Proceedings of the International Conference on Information Assurance and Security (IAS)*, pages 295–298, 2008.

[45] T. Hoppe, S. Kiltz, and J. Dittmann. Security threats to automotive CAN networks–practical examples and selected short-term countermeasures. In *Computer Safety, Reliability, and Security*, pages 235–248. Springer, 2008.

[46] T. Hoppe, S. Kiltz, A. Lang, and J. Dittmann. Exemplary automotive attack scenarios: Trojan horses for electronic throttle control system (etc) and replay attacks on the power window system. In *In Automotive Security - VDI-Berichte Nr. 2016, Proceedings of the VDI/VW Gemeinschaftstagung Automotive Security*, pages 165–183. 2007.

[47] J. Huang, K. Huang, A. Raabe, C. Buckl, and A. Knoll. Towards fault-tolerant embedded systems with imperfect fault detection. In *Proceedings of the Design Automation Conference (DAC)*, pages 188–196, 2012.

[48] L. Huang and Q. Xu. On modeling the lifetime reliability of homogeneous manycore systems. In *Proceedings of the IEEE Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 87–94. IEEE, 2008.

[49] X. Huang, J. Wu, W. Li, Z. Zhang, F. Zhu, and M. Wu. Historical spectrum sensing data mining for cognitive radio enabled vehicular ad-hoc networks. *IEEE Transactions on Dependable and Secure Computing*, 13(1):1–13, 2015.

[50] K. Huh, K. Han, D. Hong, J. Kim, H. Kang, and P. Yoon. A model-based fault diagnosis system for electro-hydraulic brake. Technical report, SAE Technical Paper, 2008.

[51] IABG - Industrieanlagen-Betriebsgesellschaft. V-Model XT, version 1.4. [Online]. Available: http://v-modell.iabg.de, 2012.

[52] Infineon. Scalable and highly integrated - 16/32-bit microcontrollers for automotive applications (brochure). [Online]. Available: http://www.infineon.com/automotive, 2014.

[53] Infineon. Highly integrated and performance optimized - 32-bit microcontrollers for automotive and industrial applications (brochure). [Online]. Available: http://www.infineon.com/TriCore, 2015.

[54] Institute of Electrical and Electronics Engineers. IEEE standard glossary of software engineering terminology. *Std 610.12-1990(R2002)*, pages 1–94, 2002.

[55] International Electrotechnical Commission. IEC 61508:2010 CMV – functional safety of electrical/electronic/programmable electronic safety-related systems, 2010.

[56] International Organization for Standardization. ISO 11898: Road vehicles – controller area network (CAN) – parts 1-5, 2003.

[57] International Organization for Standardization. ISO 26262: Road vehicles – functional safety – parts 1-10, 2011.

[58] International Organization for Standardization. ISO 17458: Road vehicles – FlexRay communications system – parts 1-5, 2013.

[59] International Organization for Standardization. ISO 17987: Road vehicles – local interconnect network (LIN) – part 6: Protocol conformance test specification, 2015.

[60] R. Isermann. Model-based fault-detection and diagnosis–status and applications. *Annual Reviews in control*, 29(1):71–85, 2005.

[61] Ixia. Automotive ethernet: An overview (white paper). [Online]. Available: https://www.ixiacom.com/, 2014.

[62] J. Kaida, T. Hieda, I. Taniguchi, H. Tomiyama, Y. Hara-Azumi, and K. Inoue. Task mapping techniques for embedded many-core socs. In *Proceedings of the International SoC Design Conference (ISOCC)*, pages 204–207, 2012.

[63] N. R. Kandimala and M. Sojka. Safety and security features in AUTOSAR. Technical report, Chech Technical University in Prague, 2012.

[64] S. Kelkar and R. Kamal. Adaptive fault diagnosis algorithm for controller area network. *IEEE Transactions on Industrial Electronics*, 61(10):5527–5537, 2014.

[65] H. Kellermann, G. Nemet, J. Kostelezky, K. Barbehön, A. El-Dwaik, and L. Hochmuth. Fahrzeugarchitektur. In *Der neue BMW 7er: Entwicklung und Technik*, pages 28–35. Springer Science+Business Media, 2009.

[66] U. Klehmet, T. Herpel, K.-S. Hielscher, and R. German. Delay bounds for CAN communication in automotive applications. In *Proceedings of the Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems (MMB)*, pages 1–15, 2008.

[67] T. Klier and J. Rubenstein. Making cars smarter: The growing role of electronics in automobiles. *Chicago Fed Letter*, 2011.

[68] H. Kopetz. Event-triggered versus time-triggered real-time systems. In *Operating Systems of the 90s and Beyond*, pages 86–101. Springer, 1991.

[69] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, et al. Experimental security analysis of a modern automobile. In *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, pages 447–462, 2010.

[70] A. Krüger, B. Hardung, and T. Kölzow. Reuse of software in automotive electronics. In N. Navet and F. Simonot-Lion, editors, *Automotive Embedded Systems Handbook*, Industrial Information Technology Series, pages 8.1–8.19. Taylor & Francis / CRC Press, 2009.

[71] S. Kuenzli and L. Thiele. Generating event traces based on arrival curves. In *Proceedings of the Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems (MMB)*, pages 1–18, 2006.

[72] C. Kühnel and M. Spichkova. Upcoming automotive standards for fault-tolerant communication: FlexRay and OSEKtime FTCom. In *EFTS 2006 International Workshop on Engineering of Fault Tolerant Systems. Universite du Luxembourg, CSC: Computer Science and Communication*, 2006.

[73] G. Kurian, J. E. Miller, J. Psota, J. Eastep, J. Liu, et al. Atac: a 1000-core cache-coherent processor with on-chip optical network. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 477–488, 2010.

[74] J. J. Labrosse. *uC/OS-III: The Real-Time Kernel for the STM32 ARM Cortex-M3*. Micrium Press, 2011.

[75] K. Lampka, S. Perathoner, and L. Thiele. Analytic real-time analysis and timed automata: A hybrid methodology for the performance analysis of embedded real-time systems. *Design Automation for Embedded Systems*, 14(3):193–227, 2010.

[76] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

[77] P. E. Lanigan, S. Kavulya, P. Narasimhan, T. E. Fuhrman, and M. A. Salman. Diagnosis in automotive systems: A survey. Technical report, Tech. Rep. CMU-PDL-11-110, Carnegie Mellon University Parallel Data Lab, 2011.

[78] J.-Y. Le Boudec and P. Thiran. *Network calculus: a theory of deterministic queuing systems for the Internet*, volume 2050. Springer Science & Business Media, 2001.

[79] G. Leen and D. Heffernan. Expanding automotive electronic systems. *Computer*, 35(1):88–93, 2002.

[80] G. Leen and D. Heffernan. TTCAN: a new time-triggered controller area network. *Microprocessors and Microsystems*, 26(2):77–94, 2002.

[81] H.-T. Lim, B. Krebs, L. Volker, and P. Zahrer. Performance evaluation of the inter-domain communication in a switched ethernet based in-car network. In *Proceedings of the Conference on Local Computer Networks (LCN)*, pages 101–108, 2011.

[82] C. Lin, Q. Zhu, C. Phung, and A. Sangiovanni-Vincentelli. Security-aware mapping for CAN-based real-time distributed automotive systems. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, pages 115–121, 2013.

[83] C. Lin, Q. Zhu, and A. Sangiovanni-Vincentelli. Security-aware modeling and efficient mapping for CAN-based real-time distributed automotive systems. *Embedded Systems Letters, IEEE*, 7(1):11–14, 2015.

[84] F. Lombardi and M. Sami. *Testing and Diagnosis of VLSI and ULSI*, volume 151. Springer Science & Business Media, 1988.

[85] M. Lukasiewycz and S. Chakraborty. Concurrent architecture and schedule optimization of time-triggered automotive systems. In *Proceedings of the International Conference on Hardware/Software Co-Design and System Synthesis (CODES)*, pages 383–392, 2012.

[86] M. Lukasiewycz, M. Glaß, C. Haubelt, and J. Teich. Efficient symbolic multi-objective design space exploration. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 691–696, 2008.

[87] M. Lukasiewycz and F. Reimann. OpenDSE - open design space exploration framework, 2015. http://opendse.sf.net/.

[88] M. Lukasiewycz, S. Steinhorst, and S. Chakraborty. Priority assignment for event-triggered systems using mathematical programming. In *Proceedings of the Design, Automation & Test in Europe Conference (DATE)*, pages 982–987, 2013.

[89] J. Luo, K. Choi, K. R. Pattipati, L. Qiao, and S. Chigusa. Distributed fault diagnosis for networked, embedded automotive systems. In *Proceeding of the International Conference on Systems, Man and Cybernetics (SMC)*, pages 1226–1232, 2006.

[90] C. Lyu, D. Gu, Y. Zeng, and P. Mohapatra. Pba: Prediction-based authentication for vehicle-to-vehicle communications. *IEEE Transactions on Dependable and Secure Computing*, 13(1):1–14, 2015.

[91] C. Mahulea, C. Seatzu, M. Cabasino, and M. Silva. Fault diagnosis of discrete-event systems using continuous petri nets. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 42(4):970–984, 2012.

[92] M. Manik. System level diagnostics over the PMC model. *Informattio Sciences and Technologies Bulletin of the ACM Slovakia*, 2(2):24–29, 2010.

[93] M. Manik and E. Gramatova. Diagnosis of faulty units in regular graphs under the PMC model. In *Proceedings of the International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, pages 202–205, 2009.

[94] E. Martin. *Oxford Concise Medical Dictionary*. Oxford University Press, 2015.

[95] E. Mayer. Serial bus systems in the automobile. *Vector Informatik GmbH*, 2006.

[96] N. McDowell, G. McCullough, X. Wang, U. Kruger, and G. Irwin. Fault diagnostics for internal combustion engines-current and future techniques. Technical report, SAE Technical Paper, 2007.

[97] McKinsey Automotive & Assembly Extranet. The future of the north american automotive supplier industry: Evolution of component costs, penetration, and value creation potential through 2020. Technical report, McKinsey & Company, 2012.

[98] MOST Cooperation. Media Oriented Systems Transport (MOST) Specification Rev. 3.0. [Online]. Available: http://www.mostcooperation.com/, 2010.

[99] T. Mueller. ISO 26262 in automotive IC development: is it just a tickbox exercise, or does it induce manufactur-ers to make safer components? *ams AG*, pages 1–7, 2013.

[100] P. Mundhenk, S. Steinhorst, M. Lukasiewycz, S. A. Fahmy, and S. Chakraborty. Security analysis of automotive architectures using probabilistic model checking. In *Proceedings of the Design Automation Conference (DAC)*, pages 38:1–38:6, 2015.

[101] M. Müter and N. Asaj. Entropy-based anomaly detection for in-vehicle networks. In *Proceedings of the Intelligent Vehicles Symposium (IV)*, pages 1110–1115, 2011.

[102] M. Müter, A. Groll, and F. Freiling. A structured approach to anomaly detection for in-vehicle networks. In *Proceedings of the International Conference on Information Assurance and Security (IAS)*, pages 92–98, 2010.

[103] M. Neukirchner, P. Axer, T. Michaels, and R. Ernst. Monitoring of workload arrival functions for mixed-criticality systems. In *Proceedings of the International Real-Time Systems Symposium (RTSS)*, pages 88–96, 2013.

[104] Nissan Motor Company Ltd. Nissan LEAF owner's manual - section LAN systems. [Online]. Available: https://carmanuals2.com/brand/nissan/leaf-2014-374, 2013.

[105] T. Nolte, H. Hansson, M. Nolin, and S. Punnekkat. Timing analysis of CAN-based automotive communication systems. In N. Navet and F. Simonot-Lion, editors, *Automotive Embedded Systems Handbook*, Industrial Information Technology Series, pages 13.1–13.30. Taylor & Francis / CRC Press, 2009.

[106] T. Nolte, H. Hansson, and C. Norstrom. Minimizing CAN response-time jitter by message manipulation. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 197–206, 2002.

[107] M. Nyberg. Model-based diagnosis of an automotive engine using several types of fault models. *IEEE Transactions on Control Systems Technology*, 10(5):679–689, 2002.

[108] Olimex Ltd. STM32-P407 development board - user's manual. [Online]. Available: https://www.olimex.com/Products/ARM/ST/STM32-P407, 2014.

[109] OPEN Alliance SIG. The OPEN Alliance (One-Pair Ether-Net) Special Interest Group (SIG). [Online]. Available: http://www.opensig.org/, 2015.

[110] OSEK/VDX. Fault-tolerant communication specification 1.0. [Online]. Available: http://portal.osek-vdx.org/, 2001.

[111] OSEK/VDX. Time-triggered operating system specification 1.0. [Online]. Available: http://portal.osek-vdx.org/, 2001.

[112] OSEK/VDX. Operating system specification 2.2.3. [Online]. Available: http://portal.osek-vdx.org/, 2005.

[113] V. Prasad. Algorithm development for the extended pmc model for fault detection. In *Proceedings of the IEEE South East Conference (SoutheastCon)*, pages 114–118, 1991.

[114] F. P. Preparata, G. Metze, and R. T. Chien. On the connection assignment problem of diagnosable systems. *IEEE Transactions on Electronic Computers*, (6):848–854, 1967.

[115] R. Regler, J. Schlinkheider, M. Maier, R. Prechler, E. Berger, and L. Pröll. Intelligent electrics/electronics architecture. *ATZextra worldwide*, 11(15):246–251, 2011.

[116] K. Reif. *Automobilelektronik - Eine Einführung für Ingenieure*. Springer, 2014.

[117] R. Roderick. A look inside battery-management systems. [Online]. Available:http://electronicdesign.com/power/look-inside-battery-management-systems, 2015.

[118] F. Sagstetter, S. Andalam, P. Waszecki, M. Lukasiewycz, H. Stähle, S. Chakraborty, and A. Knoll. Schedule integration framework for time-triggered automotive architectures. In *Proceedings of the Design Automation Conference (DAC)*, pages 1–6, 2014.

[119] F. Sagstetter, M. Lukasiewycz, and S. Chakraborty. Schedule integration for time-triggered systems. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 53–58, 2013.

[120] F. Sagstetter, P. Waszecki, S. Steinhorst, M. Lukasiewycz, and S. Chakraborty. Multi-schedule synthesis for variant management in automotive time-triggered systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(4):1–15, 2015.

[121] A. Sangiovanni-Vincentelli and M. Di Natale. Embedded system design for automotive applications. *IEEE Computer*, 40(10):42–51, 2007.

[122] F. Sanglietti. Software verfication and validation (lecture notes). [Online]. Available: http://www11.informatik.uni-erlangen.de/Lehre/WS1314/SWE-VV/, 2013. Friedrich-Alexander-University Erlangen-Nuernberg, Chair of Software Engineering.

[123] F. Schade. NXP semiconductors – TJA1055T-fault-tolerant CAN transceiver. [Online]. Available: http://www.nxp.com/documents/data_sheet/TJA1055.pdf, 2011.

[124] B. Schätz, C. Kühnel, and M. Gonschorek. FlexRay protocol. In N. Navet and F. Simonot-Lion, editors, *Automotive Embedded Systems Handbook*, Industrial Information Technology Series, pages 5.1–5.22. Taylor & Francis / CRC Press, 2009.

[125] M. Schmid. Automotive bus systems. *Atmel Applications Journal*, 2006.

[126] T. Sellke, M. Bayarri, and J. O. Berger. Calibration of ρ values for testing precise null hypotheses. *The American Statistician*, 55(1):62–71, 2001.

[127] M. Serafini, P. Bokor, N. Suri, J. Vinter, A. Ademaj, W. Brandstätter, F. Tagliabo, and J. Koch. Application-level diagnostic and membership protocols for generic time-triggered systems. *IEEE Transactions on Dependable and Secure Computing*, 8(2):177–193, 2011.

[128] M. Serafini, A. Bondavalli, and N. Suri. On-line diagnosis and recovery: On the choice and impact of tuning parameters. *IEEE Transactions on Dependable and Secure Computing*, 4(4):295–312, 2007.

[129] M. Serafini, N. Suri, J. Vinter, A. Ademaj, W. Brandstatter, F. Tagliabo, and J. Koch. A tunable add-on diagnostic protocol for time-triggered systems. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, pages 164–174, 2007.

[130] M. Short and M. Pont. Fault-tolerant time-triggered communication using CAN. *IEEE Transactions on Industrial Informatics*, 3(2):131–142, 2007.

[131] F. N. Sibai. Detecting matrix multiplication faults in many-core systems. In *Proceedings of the International Conference on Innovations in Information Technology (IIT)*, pages 330–335, 2011.

[132] D. Silva, L. Bolzani, and F. Vargas. An intellectual property core to detect task schedulling-related faults in RTOS-based embedded systems. In *Proceedings of the International On-Line Testing Symposium (IOLTS)*, pages 19–24, 2011.

[133] F. Simonot-Lion. The design of safe automotive electronic systems: some problems, solutions and open issues. In *Proceedings of the IEEE Symposium on Industrial Embedded Systems (IES)*, 2006.

[134] A. K. Singh, M. Shafique, A. Kumar, J. Henkel, A. Das, et al. Mapping on multi/many-core systems: survey of current and emerging trends. In *Proceedings of the Design Automation Conference (DAC)*, 2013.

[135] I. Stierand, S. Malipatlolla, S. Froschle, A. Stuhring, and S. Henkler. Integrating the security aspect into design space exploration of embedded systems. In *Proceedings of the International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 371–376, 2014.

[136] T. Streichert and M. Traub. *Elektrik/Elektronik-Architekturen im Kraftfahrzeug: Modellierung und Bewertung von Echtzeitsystemen*. Springer, 2012.

[137] B. Suiphon, Z. Simeu-Abazi, and E. Gascard. Implementation of a fault diagnosis method for timed discrete-event systems. In *Proceedings of the International Conference on Industrial Engineering and Systems Management (IESM)*, pages 1–8, 2013.

[138] B. Sun. Design and implementation of a permanent fault diagnosis method for automotive E/E architectures. Master's thesis, Technische Universität München, 2015.

[139] U. Suppiger, S. Perathoner, K. Lampka, and L. Thiele. A simple approximation method for reducing the complexity of modular performance analysis. Technical Report 329, ETH Zurich, 2010.

[140] Texas Instruments. Automotive and transportation solutions guide (brochure). [Online]. Available: http://www.ti.com/automotive, 2014.

[141] Toshiba Semiconductor & Storage Products. Automotive solutions (system catalog). [Online]. Available: http://toshiba.semicon-storage.com, 2015.

[142] S. Tosun, N. Mansouri, E. Arvas, M. Kandemir, and Y. Xie. Reliability-centric high-level synthesis. In *Proceedings of the Design, Automation & Test in Europe Conference (DATE)*, pages 1258–1263, 2005.

[143] M. Toulouse, B. Q. Minh, and P. Curtis. A consensus based network intrusion detection system. In *International Conference on IT Convergence and Security (ICITCS)*, pages 1–6. IEEE, 2015.

[144] S. Tripakis. A combined on-line/off-line framework for black-box fault diagnosis. In *Runtime Verification*, pages 152–167. Springer, 2009.

[145] Vector Informatik. Product information embedded operating systems (brochure). [Online]. Available: http://www.vector.com/vi_embedded_software_en.html, 2015.

[146] P. Wallin and J. Axelsson. A case study of issues related to automotive E/E system architecture development. *Proceedings of the Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS)*, pages 87–95, 2008.

[147] P. Waszecki, M. Kauer, M. Lukasiewycz, and S. Chakraborty. Implicit intermittent fault detection in distributed systems. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 646–651, 2014.

[148] P. Waszecki, M. Lukasiewycz, and S. Chakraborty. Multi-objective diagnosis of non-permanent faults in many-core systems. In *Workshop Proceedings of the International Conference on Architecture of Computing Systems (ARCS)*, pages 1–8, 2014.

[149] P. Waszecki, M. Lukasiewycz, and S. Chakraborty. Decentralized diagnosis of permanent faults in automotive E/E architectures. In *Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, pages 189–196, 2015.

[150] P. Waszecki, M. Lukasiewycz, A. Masrur, and S. Chakraborty. How to engineer tool-chains for automotive E/E architectures? *ACM SIGBED Review*, 10(4):6–15, 2013.

[151] P. Waszecki, F. Sagstetter, M. Lukasiewycz, and S. Chakraborty. Diagnosis-aware system design for automotive E/E architectures. In *Proceedings of the International Symposium on Integrated Circuits (ISIC)*, pages 456–459, 2014.

[152] Y. Xie, L. Li, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin. Reliability-aware co-synthesis for embedded systems. In *Proceedings of the International Conference on Application-Specific Systems, Architectures and Processors (ASAP)*, pages 41–50, 2004.

[153] X. Zhu, J. Zhu, M. Ma, and D. Qiu. Qaft: A qos-aware fault-tolerant scheduling algorithm for real-time tasks in heterogeneous systems. In *Proceedings of the International Conference on Computational Science and Engineering (CSE)*, pages 80–87, 2010.

[154] W. Zimmermann and R. Schmidgall. *Bussysteme in der Fahrzeugtechnik*. Springer, 2011.

List of Figures

List of Tables

List of Acronyms

ABS	Anti-lock Braking System
ACC	Adaptive Cruise Control
ADAS	Advanced Driver Assistance System
ADC	Analog/Digital Converter
ASIL	Automotive Safety Integrity Level
AUTOSAR	Automotive Open System Architecture
AVB	Audio Video Bridging
BDD	Binary Decision Diagram
C2C	Car-to-Car
C2I	Car-to-Infrastructure
CAN	Controller Area Network
CAN FD	CAN with Flexible Data-Rate
CDF	Cumulative Distribution Function
CNF	Conjunctive Normal Form
CPS	Cyber-Physical System
CPU	Central Processing Unit
CSMA/BA	Carrier Sence Multiple Access with Bitwise Arbitration
DES	Discrete Event System
DFG	Data Flow Graph

DFS	Depth-First Search
DFT	Design for Testability
DNF	Disjunctive Normal Form
DoS	Denial of Service
DSE	Design Space Exploration
E/E	Electrical/Electronic
ECU	Electronic Control Unit
EEPROM	Electrically Erasable Programmable ROM
EMC	Electromagnetic Compatibility
EMI	Electromagnetic Interference
ESD	Electrostatic Discharge
ESP	Electronic Stability Program
ETC	Electronic Throttle Control
HMI	Human-Machine Inteface
IC	Integrated Circuit
ICE	Internal Combustion Engine
IDS	Intrusion Detection System
ILP	Integer Linear Programming
IPAS	Intelligent Parking Assist System
ISS	Instruction Set Simulator
LDWS	Lane Departure Warning System
LIN	Local Interconnect Network
MAC	Message Authentication Code
MILP	Mixed Integer Linear Programming
MOST	Media Oriented Systems Transport
MPSoC	Multiprocessor System-on-Chip
NMR	n-Modular Redundancy

NoC	Network-on-Chip
OEM	Original Equipment Manufacturer
OS	Operating System
OSEK/VDX	Offene Systeme und deren Schnittstellen für die Elektronik in Kraftfahrzeugen / Vehicle Distributed Executive
OTA	Over-the-Air
QoS	Quality-of-Service
RAM	Random Access Memory
RAP	Resilience Articulation Point
ROM	Read-Only Memory
RTE	Runtime Environment
RTOS	Real-Time Operating System
SMT	Satisfiability Modulo Theories
SoCs	Systems-on-Chip
SVD	Singular Value Decomposition
TDMA	Time Division Multiple Access
TSN	Time-Sensitive Networking
TTCAN	Time-Triggered CAN
UART	Universal Asynchronous Receiver/Transmitter
V2I	Vehicle-to-Infrastructure
V2V	Vehicle-to-Vehicle
VLSI	Very Large Scale Integration