University of Porto
Faculdade de Engenharia

FEUP

Services for Safety-Critical Applications on Dual-Scheduled TDMA Networks

By

Valerio Rosset

A dissertation submitted in partial fulfilment of the requirements for the degree of Doctor in Electrical and Computer Engineering

Supervisor: Pedro F. Souto
Co-supervisor: Francisco Vasques

July 2009

Abstract

The development of safety-critical embedded applications in domains such as automotive or avionics is an exceedingly challenging intellectual task. However, this task can be significantly simplified through the use of middleware that offers specialized fault-tolerant services. This thesis develops and studies a set of protocols intended to provide reliable group communications services (Group Membership and Reliable Broadcast).

These protocols were specially designed to take advantage of dual scheduling TDMA (DuST) protocols, which support not only static scheduling, as in classic TDMA protocols, but also dynamic scheduling. This class of DuST protocols is likely to become rather widespread in embedded applications, because it was adopted by FlexRay, a specification that is expected to become the de-facto standard for the next generation automotive networks. The basic idea of our group communication protocols is to improve their performance by scheduling the non-periodic traffic they generate in the part of the TDMA-cycle that is dynamically scheduled. By exploring this idea, our group membership protocol (GMP) imposes an overhead of two bits per processor per communication cycle, when the system is in a quiescent state, and is able to tolerate benign failures of up to half of the group members between consecutive executions. Additionally, it removes a faulty processor within two communication cycles in the worst case and reintegrates a processor the latest two communication cycles after it recovers. Compared with protocols developed for similar systems, it is as tolerant as the most robust protocol with a traffic overhead slightly higher than the most efficient protocol, which is much less robust. The family of reliable broadcast protocols (RB-DuST) are a complement to FlexRay's native communication service, which does not provide fault-tolerance levels required by safety-critical applications.

Because these protocols are intended to be used as building blocks in the de-

sign of fault-tolerant applications, their dependability is of paramount importance. Therefore, we have applied formal methods, more specifically model checking, to verify their correctness and also to evaluate their reliability. The reliability models are discrete-time Markov chains, and consider an extensive range of fault scenarios, including permanent and transient faults, affecting both communication channels and nodes. Furthermore, we performed a sensitivity analysis, to assess the influence of different parameters on the protocols reliability. The results show that both the GMP and the RB-DuST protocols, when properly configured, can achieve reliability levels in the range required for safety critical applications, for parameters values of the models typical of the automotive domain.

Resumo

O desenvolvimento de aplicações-críticas para a segurança (safety-critical) nos domínios de automóveis e aviões representa um desafio a ser superado com grande esforço intelectual. Porém essa tarefa pode ser significativamente simplificada através da utilização de um middleware capaz de fornecer serviços tolerantes à falhas. Esta tese define e investiga um conjunto de protocolos destinados a prover serviços de comunicação fiável em grupo (Filiação em Grupo e Difusão Fiável).

Esses protocolos são especialmente projectados para aproveitar as vantagens oferecidas pelo modelo de escalonamento duplo sobre TDMA (DuST), o qual suporta não apenas escalonamento estático, como protocolos TDMA clássicos, mas também escalonamento dinâmico. Essa classe de protocolos do tipo DuST muito provavelmente será largamente utilizada em sistemas embebidos, isso porque ela foi adoptada no FlexRay, uma especificação que possuí grandes possibilidades de tornar-se "de facto" o padrão para a próxima geração de redes em automóveis. A ideia básica de nossos protocolos de comunicação fiável em grupo é a de melhorar o desempenho através do escalonamento do tráfego não periódico na parte dinamicamente escalonada do ciclo TDMA. Através dessa ideia, nosso protocolo de filiação em grupo (GMP) impõe uma carga extra de apenas dois bits por processo em cada ciclo de comunicação quando o sistema se encontra no estado passivo (sem falhas) e é capaz de tolerar falhas benignas de até a metade dos membros dum grupo durante execuções consecutivas. Além disso, o GMP remove os membros falhados no pior dos casos em dois ciclos e também reintegra membros ao grupo em menos de dois ciclos após sua recuperação. Comparado aos outros protocolos similares o GMP mostra-se tão tolerante à falhas quanto os mais robustos protocolos. Porém seu desempenho (em termos de sobrecarga de mensagens) é muito próximo aos dos protocolos de melhor desempenho, que são muito menos robustos. A família de protocolos para difusão

fiável de mensagens (RB-DuST) são um complemento ao sistema de comunicação nativo presente no protocolo de comunicação FlexRay, cujo não apresenta nível de tolerância à falhas requerido pelas aplicações-críticas.

Nossos protocolos são destinados a serem utilizados como serviços base (building blocks) necessários para o desenvolvimento de aplicações tolerantes à falhas, por isso sua fiabilidade é de extrema importância. Em virtude disso, nós aplicamos métodos formais, mais especificamente através da técnica denominada Model Checking, para verificar a correcção e fiabilidade dos protocolos. Os modelos foram desenvolvidos através de Cadeias de Markov discretas, e consideram uma larga gama de cenários de falhas incluindo falhas transitórias e permanentes, afectando ao mesmo tempo nós (nodes) e canais. Além disso, realizamos também uma análise sensitiva para estimar a influência de diferentes parâmetros sobre a fiabilidade dos protocolos. Os resultados mostraram que tanto os protocolos GMP e RB-DuST, quando configurados correctamente, podem atingir os níveis de fiabilidade requeridos por aplicações-critícas.

Résumé

Le développement d'applications de sécurité critique (safety-critical) dans les domaines de l'automobile et de l'aviation est un défi qui doit être surmonté avec beaucoup d'effort intellectuel. Néanmoins cette tâche peut être considérablement simplifiée par l'utilisation d'un middleware capable de fournir des services tolérants aux pannes. Cette thèse étudie et définit un ensemble de protocoles destinés à l'établissement de services de communication fiable (Filiation en Groupe et Diffusion Fiable).

Ces protocoles ont été spécialement conçus pour exploiter les avantages offerts par le modèle d'ordonnancement double sur TDMA (DuST), qui supporte non seulement un ordonnancement statique, comme les protocoles TDMA classiques, mais aussi un ordonnancement dynamique. Cette classe de protocoles du type DuST sera probablement largement utilisée dans les systèmes embarqués. Cela est dû au fait qu'elle a été adoptée dans le protocole FlexRay, une spécification qui présente un grand potentiel pour devenir la norme de la prochaine génération de réseaux dans les automobiles. L'idée fondamentale de l'ensemble de protocoles de communication fiable est d'améliorer la performance à travers de l'ordonnancement du trafic non périodique dans la partie dynamiquement ordonnancé du cycle TDMA. Ainsi notre protocole de gestion de groupe (GMP) impose une charge supplémentaire de seulement deux bits par processus dans chaque cycle de communication lorsque le système se trouve dans l'état passif (sans failles). Il est aussi capable de tolérer des pannes bénignes de jusqu'à la moitié des composants d'un groupe au cours des exécutions consécutives. En outre, le GMP supprime les composants défaillants dans le pire des cas dans deux cycles et aussi il réintègre les composants au groupe dans moins de deux cycles après leur récupération. Par rapport aux autres protocoles similaires au GMP, celui-ci se montre aussi tolérant aux pannes que les protocoles les plus robustes.

Néanmoins, sa performance (en termes de surcharge de messages) est très proche à celle des protocoles d'excellente performance, mais qui sont beaucoup moins robustes. La famille des protocoles pour la diffusion fiable de messages (RB-DuST) est un complément au système de communication initialement présent dans le protocole de communication FlexRay, lequel ne présente pas le niveau de tolérance aux pannes exigé par les systèmes critiques.

Nos protocoles sont destinés à être utilisés comme services base (building blocks) nécessaires au développement des applications tolérantes aux pannes. Alors la fiabilité de ces protocoles est d'extrême importance. En vertu de cela, nous appliquons des méthodes formelles, plus spécifiquement Model Checking, pour vérifier la correction et la fiabilité des protocoles. Les modèles ont été développés à travers des Chaînes de Markov discrètes, et ils prennent en considération un large gamma de scénarios d'imperfections y compris des pannes transitoires et permanentes, en affectant au même temps noeuds et canaux. En plus, nous réalisons aussi une analyse sensitive pour estimer l'influence de différents paramètres sur la fiabilité des protocoles. Les résultats ont montré que les protocoles GMP et RB-DuST, dès que correctement configurés, peuvent atteindre les niveaux de fiabilité exigés par les systèmes critiques.

To all my family.

Agradecimentos

"A imaginação é mais importante que o conhecimento." (Albert Einstein)

"Aos que possuem poder, o presente!
Aos que possuem conhecimento, o futuro!
Aos que possuem a sabedoria, a imortalidade!"
(Valerio Rosset)

Primeiramente gostaria de agradecer profundamente a Deus e à minha família, especialmente a minha mãe Izolete, meu pai Aldo, meus irmãos Ricardo e Marcelo, minha cunhada Graziela e meu sobrinho e afiliado Gabriel, pelo apoio que sempre me foi dado e sem o qual eu poderia ter seguido em frente na minha jornada.

Da mesma forma gostaria de agradecer também:

ao Professor Pedro, pela oportunidade que tive de trabalhar e aprender com ele. A relização desse trabalho e muito que aprendi só foi possível através da sua especial contribuição.

ao Professor Francisco, especialmente pelo apoio nos momentos mais críticos e pela oportunidade dada de poder realizar esse trabalho de doutorado em Portugal.

ao Professor Paulo Portugal, pelas importantes sugestões e colaborações durante a realização desse trabalho.

aos colegas e amigos, que de uma forma ou de outra contribuíram e compartilharam dessa minha experiência em Portugal. Especialmente gostaria de agradecer aqui a: Silvio, Ricardo, Andréia, Reinaldo, Adilson, Marcela, Digão, Luiz, Gabi, Amanda, Raphael, Kathrine, Mônica, Robson, Negretti, Bruno, Daniel, Paulo Renato, Miranda, Rodrigo, Marlos, Madruga, Gustavo, Rafael, Nelson, Valdir,

Dna. Angela, Dna. Linda, Dna. Margarida, Maria de Lourdes e outros que por ventura tenha esquecido de mencionar.

à Faculdade de Engenharia da Universidade do Porto, pelos recursos disponibilizados para que esse trabalho pudesse ser realizado.

e finalmente à Fundação para Ciência e Tecnologia - FCT, DEMEGI e IDMEC, pelo sempre essencial suporte financeiro.

Porto, 20 de Julho de 2009.

Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviation	Description
ACK	Positive Acknowledgement
BER	Bit Error Rate
BERPG	Bit Error Rate per Gigabit
CAN	Controller Area Network
CIR	Common-mode to Independent Fault Ratio
CRC	Cyclic Redundant Code
CTL	Computation Tree Logic
CTMC	Continuous-Time Markov Chains
DD	Distributed Diagnosis
DP	Diagnosis Period
DTMC	Discrete-Time Markov Chains
DusT	Dual Scheduled Time-Trigger
EMI	Electromagnetic Interference
EoE	End of Execution
EoRBR	End of Reliable Broadcast Round
FD-phase	Failure Detection Phase
FTDMA	Flexible TDMA
FTT	Flexible Time-Triggered
FTT-CAN	Flexible Time-Triggered on CAN
GMP	Group Membership Protocol
GM-phase	Group Membership Phase
Mbps	Mega bits per second
MFA	Maximum Fault Assumption
M-SET	Majority Set
MSV	Membership Status Vector
NACK	Negative Acknowledgment

Abbreviation	Description
OT	Other Traffic
PCTL	Probabilistic Computation Tree Logic
PLP	Processor-link-Processor protocol
PP	Processor-Processor protocol
RB	Reliable Broadcast
RB-DuST	Reliable Broadcast on DuST
RBRD	Reliable Broadcast Receiver & Deliver Protocol
RBRD(a)	RBRD protocol with positive acknowledgments
RBRD(n)	RBRD protocol with negative acknowledgments
RBTD	Reliable Broadcast Threshold Delivery Protocol
RBTD(a)	RBTD protocol with positive acknowledgments
RBTD(n)	RBTD protocol with negative acknowledgments
RBV	Traditional RB Validity property
ROBUS	Reliable Optical Bus
RT	Real-Time
SOA	Strictly Omission Asymmetric Fault
SSR	SOA to Symmetric Faults Ratio
TDMA	Time Division Multiple Access
TTCAN	Time-Triggered Controller Area Network
TTP	Time-Triggered Protocol

Chapter 1

Overview

The main focus of this thesis is on fault-tolerant protocols for safety-critical applications which communicate on top of dual scheduled communication protocols (DuST). The nature of this next generation communication protocols permits more flexible scheduling of messages transmissions supporting both event and time triggered communications. The protocols proposed in this thesis were designed to take advantage of this flexibility in order to improve performance while affording reliability levels required by safety-critical applications in the automotive domain.

1.1 Introduction

The demands for automotive and avionics communications systems have been increasing in recent years. More specifically in the automotive domain the growing performance and reliability of hardware components in combination with the software technologies allows to implement complex functions that improve the comfort and safety of the vehicle's occupants [1]. Such functions include control of powertrain and chassis, such as engine control, transmission, steering, brakes and stability control.

Indeed, due to replacement of mechanical and hydraulic systems by fully electronic ones, the control systems designed to control chassis functions, such as brakes and steering, require ultra-reliability levels on communications, including bounded response times and bounded jitters, in order to make them safety.

In fact such dependability requirements can be fulfilled only by networks which combine higher speed rates, predictability and fault-tolerant services [1]. Networks of class C and D, such as high-speed CAN, TTP/C [2] and FlexRay [3, 4], can comprise speed rates from 125 kb/s to 10 Mb/s. However, the predictability and composability (i.e., ability to integrate individually developed components) of time-triggered approach make the TTP/C and FlexRay more adequate to safety critical applications.

Nevertheless purely time-triggered protocols such TTP/C are not efficient enough to handle aperiodic traffic generated by non-safety critical functions as windows, seats, mirrors, and climate control. For this reason, due to the vast number of functions and the need of optimization on integration between different system components the emerging Dual Scheduled (DuST) protocols (i.e., that can support a combination of both time-triggered and event-triggered transmissions), such as FlexRay [3, 4] and FTT-CAN [5], have large possibility to become the next generation of de facto standards for communication in automotive domain.

1.2 Motivation & Objectives

The support of ultra-reliable communications is one of the major requirements in safety-critical application domain [6]. Emergent DuST protocols can provide both higher speed rates and predictability but, it is notable that they do not provide by themselves all the mechanisms necessary to meet fault-tolerance requirements. Specifically both FlexRay and FTT-CAN provide a basic set of fault-tolerant services, such as clock synchronization, but no group membership. Moreover, the FlexRay do not provide mechanisms for reliable message delivery.

In fact the absence of such fault-tolerant services have provided great motivation for the development of this work. In this thesis, we propose protocols to fill this omission, and the major objective consists in to design protocols that take advantage of the dual scheduling capability of DuST networks. Furthermore the major objective supports the fundamental assertion of this thesis, that is: ”its is possible to combine the dual scheduled scheme capability to achieve more efficient fault-tolerant services without compromising reliability levels required by safety-critical applications”.

1.3 Research Context

This thesis describes group membership and reliable broadcast services for real-time safety-critical applications. The protocols we describe in this thesis are specially designed to take advantage of the dual scheduling scheme (static and dynamic) provided by FlexRay protocol. First we present the Group Membership Protocol (GMP). The GMP uses the dual schedule to achieve an overhead of only two bits per processor per cycle in absence of faults and at the same time it can tolerate benign failures of up to half of the group members between consecutive executions. Moreover, even when there are group membership changes, the GMP overhead is lower than that of other protocols that provide the same level of fault tolerance. This thesis presents also a family of reliable broadcast protocols, so called RB-DuST protocols, that are specified to provide efficient reliable message delivery, which is not originally supported by FlexRay protocol.

Because these protocols are intended to be used as building blocks in the design of fault-tolerant applications, their dependability is of paramount importance. Therefore, we have applied formal methods to verify their correctness and also to evaluate their reliability. More specifically we have used model checking, a technique for verifying properties of a system through exhaustive and automatic exploration of all the system states.

Moreover a reliability analysis of proposed protocols is presented allowing to estimate the reliability levels of such protocols on realistic fault scenarios. The reliability models are discrete-time Markov chains, and consider an extensive range of fault scenarios, including permanent and transient faults, affecting both communication channels and nodes. Furthermore, we performed a sensitivity analysis, to assess the influence of different parameters on the protocols reliability.

1.4 Main Contributions of this Thesis

The main contributions of this thesis are:

1. A specification of a Group Membership Protocol designed to take advantage of scheduling scheme provided by DuST networks, such as FlexRay. The GMP provides a optimal performance in absence of faults saving bandwidth for other network traffic (Rosset *et al.* [7]).

2. A verification of GMP protocol correctness through the Model Checking approach. The formal verification provided include alternative modeling technics which allow faster and reliable models. The formal verification also have provided means to evidence protocol limitations under different faulty scenarios (Rosset *et al.* [8, 9]).

3. A reliability analysis of GMP protocol fault assumptions including scenarios subject to strictly omission asymmetric (SOA) faults and communication error-bursts. This analysis has shown that GMP can present acceptable reliability levels even if fault assumptions are not respected, e.g. on the presence o SOA faults (Rosset *et al.* [10, 11]).

4. A specification of family of reliable broadcast protocols designed for DuST networks, including implementation issues.

5. A reliability analysis of designed RB protocols under different realistic scenarios assuming strictly omission faults (SOA) on channels (Rosset *et al.* [12]). The results of reliability analysis of RB-DuST protocols are a valuable information and can be used by designers to improve the reliability of safety-critical applications.

6. The models developed for reliability analysis are reusable and can be considered a source of information about how to model similar reliable broadcast and group membership algorithms designed for DuST networks.

1.5 Thesis Outline

The remainder of this thesis is organized as follows:

In **Chapter 2** we present a survey of the state of the art on ultra-reliable communications services specially designed for safety critical applications domain, such as reliable broadcast and group membership communication protocols.

In **Chapter 3** is presented the specification of GMP. The chapter includes the GMP algorithm description, its correctness proofs and some issues about GMP implementation on top of FlexRay network.

Chapter 4 Chapter 4 presents the formal verification of the GMP. This chapter includes a detailed description of models developed for the UPPAAL model checker and an evaluation of the results obtained.

Chapter 5 presents a reliability analysis of GMP protocol. We present detailed description of designed discrete-time Markov Chains, PRISM models and discussion of results obtained in the reliability evaluation of GMP protocol.

A family of reliable broadcast protocols (RB-DuST) is presented in **Chapter 6**. This chapter also includes the proofs of correctness of these RB protocols. In addition some issues about the implementation of a middleware layer to provide reliable broadcast services on top of FlexRay network are discussed.

In **Chapter 7**, a reliability analysis of RB-DuST protocols is presented. It includes detailed description of designed discrete-time Markov Chains, PRISM models and fault case analysis used in the reliability evaluation of RB-DuST protocols.

Finally in **Chapter 8** we present the conclusions of this thesis, highlighting how the contributions have fulfilled the original research objectives. In addition we also discuss some future research directions that may emerge from the work reported in this document.

Chapter 2

State of the art

This chapter surveys the state of the art of the two communications services that are the focus of this thesis: group membership and reliable broadcast. These services are considered core fault-tolerant services and the bibliography is rather extensive. Therefore, we focus on the work on synchronous systems that is more closely related to the work in this thesis.

2.1 Introduction

This chapter reviews the state of the art on group membership protocols designed to synchronous communication systems. The most relevant research works are presented and compared with GMP approach. In the same way this chapter presents the most relevant studies on reliability broadcast protocols specially designed to time-triggered communication systems on automotive domain. The chapter also present the review of methodologies used to proceed with formal verification and reliability analysis of such protocols.

The remainder of this chapter is organized as follows. In the next Section is described the review of state-of-the-art group membership protocols. In Section 2.3 is presented the state of the art on reliable broadcast. Finally, in Section 2.4, a synthesis of the state of the art of formal verification and reliability analysis is made, where modeling and evaluation methodologies are compared to.

2.2 Group Membership

The group membership problem has been the focus of a lot of research on both asynchronous and synchronous distributed systems. This section is concentrated on previous work for synchronous systems. For a discussion on the work for asynchronous systems we suggest the reading of the comprehensive survey [13] by Chockler, Keidar and Vitenberg.

In [14], Cristian provides a specification of the group membership problem for synchronous systems very similar to the one we use. Furthermore it proposes three protocols that solve that specification. There are however two important model assumptions that make the protocols described by Cristian and ours not directly comparable. On one hand, Cristian assumes a general communication system, whereas we consider a broadcast network with a TDMA medium access protocol where each processor broadcasts at least once every cycle. Our assumption makes it much easier to detect processor failures in a timely fashion, whereas Cristian put a lot of effort in solving this problem in an efficient way and in analyzing the timing properties of the protocols proposed. On the other hand, the fault assumptions made by Cristian are stronger than the ones we make: Cristian assumes that processors can fail only by crashing, and that the communication system may experience omission and performance failures. However, it is also assumed that the communication system has enough redundancy so as to make it possible to implement atomic broadcast. Thus Cristian's fault model is such that if a processor does not receive a message that another processor was supposed to broadcast, it must because the broadcaster crashed. On the other hand, in our model, although we also assume that the communication system provides atomic broadcast, processor faults are not limited to crashes – we assume that processors can have also both send and receive faults.

RTCAST is a group communication system proposed by T. Abdelzaher *et al* in [15] that includes a group membership protocol. Like in Cristian's work, the communication system is assumed to have an arbitrary topology, and therefore RTCAST is based on a timed-token that rotates along a logical ring composed of the group members. The difference in the network topology and on the goals of the system, the emphasis of RTCAST is on flexibility in generic real-time applications, lead to a group membership protocol with very different properties. E.g. in RTCAST a faulty processor may cause the removal from the group of

a non-faulty member. In addition, processor reintegration is slower since the RTCAST protocol does not support the reintegration of more than one processor per token rotation, and its worst case behavior is worse than ours. As in our model processors can fail by crashing and may experience receive and send faults, although the latter are assumed to be detectable by the faulty processor, whereas we make no such assumption. Communications system faults are interpreted as processor receive faults, as we do. In contrast, the communications system is assumed to have an arbitrary topology, and therefore rather than using a TDMA protocol, RTCAST is based on a timed-token that rotates along a logical ring composed of the group members. To reduce the cost of group membership in such a general setting, membership is not based on voting but on the perception of individual processors. For example, if a processor does not receive the token by the corresponding deadline, it will broadcast a message removing from the group its predecessor in the ring. As a result, in contrast with what happens in our protocol, faulty processors may remove from the group non-faulty processors, a feature that is undesirable in safety critical applications. Recovery from such situations is achieved by having the erroneously removed processor rejoining the group. Adding processors to the group is somewhat heavier and succeeds only if all group members are non-faulty, as differences in the group membership will lead to the crashing of the joining processor. Furthermore, at most one processor can (re)join the group per token rotation, although this might be easily modified by adjusting the token rotation time. If more than one processor attempts to join, those that do not succeed will wait a random number of token rotations before retrying again. In our protocol there are no bounds on the number of processors that may rejoin the group in a cycle, and it tolerates the failure of up to half of the group members, even in cycles when processors try to join the group.

Much closer to our research is the extensive work on group membership that has been developed in the scope of the Time-Triggered Architecture and its protocol, TTP/C. The group membership protocol used in TTP/C is essentially the one described by Kopetz and Gruensteidl in [16], and an optimized version of this protocol with respect to failure detection latency is proposed in [17]. The system model used in these works is very similar to ours. In all cases, the network supports broadcast and uses a TDMA-based medium access protocol. The fault model assumed is also identical, in particular the network is supposed to be reliable, and processors are assumed to fail by crashing, or by failing to receive

or to send messages. A key assumption of the TTP/C protocol is that there is at most one fault within an interval of two TDMA cycles. The protocol explores this assumption to ensure agreement on group membership among non-faulty processors without any communication overhead. This is achieved by sending with every message a CRC that covers not only the message but also part of the processor state that includes the group membership.

In [2] Kopetz and Gruensteidl argue for their fault arrival assumption based on a very simple reliability assessment. We suspect that these numbers are somewhat optimistic because the protocol interprets transient communication faults as processor faults. The study carried out by Latronico, Miner and Koopman [18] for the SPIDER group membership protocol shows that transient faults significantly affect the probability that the fault assumptions will hold. Although the support of a duplicated network medium by TTP/C reduces the likelihood of communication faults, the occurrence of more than one communication fault in one TDMA cycle caused by electromagnetic interference, e.g., cannot be ruled out. Because of this, in TTP/C, processors will switch to the blackout operating mode if they detect that the fault assumption is violated. Operation in this mode is severely degraded. By contrast, our protocol supports the failure of up to half of the group members in one TDMA cycle, although at the cost of some communication overhead and by assuming an extended TDMA protocol.

More recently, with the adoption of a redundant star topology [19], the TTP/C group membership protocol is able to tolerate an arbitrary fault per TDMA cycle. It should be noted that this is possible because the bus guardian is now placed at the star coupler and it transforms an arbitrary fault at the processor into a fault that is tolerated by the group membership protocol. Failure of one star coupler can be masked by the other star coupler, but faults in this component cannot be arbitrary.

Ezhilchelvan and Lemos [20] proposed a group membership protocol also designed for broadcast networks using a TDMA medium access protocol. The fault model is similar to the one assumed in our GMP specification: processors can experience send and receive faults and can fail by crashing, whereas the communications system is assumed to provide a reliable broadcast service. This protocol has some resemblance to ours, but there are significant differences that lead to very different performance. In [20], every processor maintains information on the

group membership using a Membership Status Vector (MSV vector), which it broadcasts in every cycle. In contrast, in our protocol a processor only broadcasts group membership information whenever it detects a group membership change event, because the communication system supports both static and dynamic scheduling. With more classical TDMA schemes that support only static scheduling, there is no real advantage in not broadcasting the membership information. In addition, the group membership information sent by each processor in our protocol is just half of the information sent in the protocol of [20]. Indeed, whereas in our protocol, the perceived state of a processor can be coded with 1 bit, each element of the MSV vector can have 3 values and therefore requires at least two bits. This higher efficiency has a cost in terms of the fault tolerance. Whereas the protocol of [20] is able to detect every processor fault type of the failure model assumed, in our protocol receive faults may be masked by send faults. On the other hand, our protocol tolerates the failure of up to half of the total number of non-faulty group members between two consecutive executions, whereas the protocol of [20] tolerates that many failures but in three consecutive cycles.

Walter, Lincoln and Suri [21] proposed a sequence of protocols for distributed on-line diagnosis, which is equivalent to the group membership problem, that are tolerant to increasing weaker fault assumptions. The problem of fault-diagnosis is actually equivalent to the problem of group membership and our work shares with theirs some mechanisms and design principles. First, we base agreement on the majority function also used in [21], second we strive to present the protocols in such away that they are devoid of implementation details. These similarities are very clear as the communication systems are assumed to have the same topology and the access control protocol is TDMA-based, and our base protocol could be placed in the sequence of protocols in [21] between the processor-processor (PP) and the processor-link-processor (PLP) protocols. Although such similarities, there are also some major differences. First, [21] focus on fault diagnosis and is not concerned with processor reintegration. Second, their protocols require the exchange of diagnosis information in every cycle, whereas our protocol does not. This is possible because we consider only faults that can be locally detected by a non-faulty processor. On the other hand, [21] presents protocols that tolerate Byzantine, or arbitrary, faults, which cannot be locally detected. The use of a stronger fault model has the additional advantage of requiring the exchange of

less diagnosis information. There is clearly a trade-off between fault tolerance and efficiency that can be solved only by considering the requirements of the application.

The SPIDER group membership protocol [22] is an optimized version of the Distributed Diagnosis (DD) protocol presented in [21] that is used by the Reliable Optical Bus (ROBUS) communication system, which has a redundant active star topology. Again there are some resemblance between both protocols, but there are also major differences, as they were designed with two different communication systems in mind, each of which with its own emphasis. Whereas SPIDER was designed to improve the reliability of ROBUS and is designed to tolerate hybrid faults [23], including asymmetric or arbitrary faults, our protocol was designed to reduce the traffic required by group membership in communication protocols with both static and dynamic scheduling. Therefore, the SPIDER group membership protocol tolerates more severe failures, but it depends strongly on the ROBUS communication system and requires that the number of communication channels be at least three, i.e. it requires that each processor be connected to at least three hubs. Furthermore, it requires that the presumed state of the components of ROBUS be exchanged periodically. On the other hand, our protocol makes much stronger fault assumptions, but it does not require such a high degree of replication of the communication system. On the other hand, whereas the SPIDER protocol requires each ROBUS processor, including the star hubs, to broadcast the presumed state of all other processors periodically.

2.3 Reliable Broadcast

Group communication services are designed to achieve different degrees of consistency. Reliable message dissemination protocols based on broadcast/multicast approaches have been designed for different types of networks and are exhaustively discussed in the literature [13, 24, 25, 26].

This thesis present a family of reliable communication protocols that are the core of group communication services designed for DuST networks, and that provide strong guarantees with respect to message delivery consistency, even in the face of faults. Although several works follow a similar approach to achieve reliable communication in the context of CAN, which poses specific problems

inherent to its specification, e.g. [27, 28], as far as we know, only [29] attempts to address this problem in the context of time-triggered networks, such as TTP/C [2], FlexRay [4] and TTCAN [30].

In [29], the authors propose and adaptation of the Unified protocol specified in [31] to provide interactive consistency, i.e. reliable broadcast, service to automotive time-triggered networks protocols. The Unified protocol was originally specified for a network consisting of several fully connected stages, and uses flooding in this network to provide strong consistency properties even in the presence of Byzantine faults. The flooding approach used in the Unified have first appeared in [25], where Babaoglu and Drummond have presented Byzantine-tolerant reliable broadcast protocols designed for redundant shared medium networks, which terminate in two rounds. The protocols proposed in [25] are based on a two-step flooding algorithm in which all nodes are required to re-broadcast the messages they have received. Differently, the message retransmission arrangement provided by the adapted Unified protocol [29] requires that only a subset of nodes execute the retransmission. The solution proposed in [29] preserves the consistency guarantees, but is practical only in networks where the number of transmitters is very small. Because even in the adapted Unified protocol nodes have re-broadcast all the messages they receive, possibly leading to a polynomial increase of the network traffic. Whereas a fully connected point-to-point network can withstand this traffic increase because its bandwidth increases with the number of links, a shared medium network like FlexRay has a fixed bandwidth, imposing strict limits on the amount of information that can be transmitted with the Unified protocol.

To increase the scalability of reliable broadcast and make it affordable in networks with tenths of nodes, our protocols rely on acknowledgements and repeated broadcast by the original broadcaster. Although the use of acknowledgements do not represent the better performance choice, for example, if compared with the implicit acknowledgment approach presented in the TTP/C protocol [2] [1]. We show that our protocols can be accommodate on DuST communication segments in order to provide minimum network usage to achieve better execution perfor-

[1] In the TTP/C the interactive consistence can be achieved by the combination of checksummed transmissions and clique avoidance services [6]. However, such TTP/C services specifications are based on strongest fault assumptions in comparison to that assumed to our reliable broadcast protocols.

mances without reliability loss.

2.4 Analysis and Validation

Safety critical domain applications require high reliability levels on message delivery, and such applications are extremely depended of group communication services. However protocols' specifications are prone to design errors generally induced by fault assumptions made. Therefore the analysis and discussion about how protocols can fail to provide the expected safety guarantees when implemented is necessary [32]. The family of reliable group communication protocols presented in this thesis were analyzed in order to verify their correctness and safety guarantees. We first provide a formal verification of GMP protocol by using the model checking approach, an alternative formal method than mathematical proofs. Model checking is a technique for verifying properties of a system through exhaustive and automatic exploration of all the system states. The use of model checking is a crucial tool to detect design errors that cannot be detected by mathematical proofs.

A second step to formal verification is reliability analysis. We evaluate the reliability of both GMP and RB protocols through the computation of the reliability of their fault assumptions. To do that we have used a Probabilistic Model Checker tool to automatic calculate the probability of safety and liveness properties be violated on fault scenarios where the original fault assumptions do not hold [2].

To the best of our knowledge, this methodology was first proposed by Latronico, Miner and Koopman in [18]. In [18] is used a Group Membership Protocol (of the SPIDER architecture) as a case study. Although the methodology used is generally the same, there are some major differences between both works. First, whereas [18] uses a continuous-time Markov chain (CTMC) model for modeling the protocol we have chosen to use a discrete-time model. The use of a discrete-time model is more appropriate for round-based protocols that are executed repeatedly, such as GMP and RB protocols. In particular, it allows to model the rounds themselves, facilitating the evaluation of assumptions on the

[2] Actually, the probabilistic model checker can also be used as a pure model checker, for example, the models used to evaluate the reliability of presented RB protocols are also used to verify their correctness

number of faults in each round. This is important, because the fault tolerance of most round-based protocols relies on this kind of assumption. On the contrary, CTMC models make it virtually impossible to accurately model the rounds. This requires the analyst to use rough approximations in the evaluation of properties that depend on rounds. E.g. in [33] Latronico and Koopman also use CTMC models to evaluate the reliability of the group membership protocol of TTP/C. However, because of the limitation mentioned above, rather than evaluating the reliability of TTP/C's fault assumption – that exactly one fault may occur within two-rounds, – they had to state a different fault assumption. Such an approach may raise some doubts regarding the outcome of the evaluation. Second, whereas in [18] the authors consider only independent fault models, in this thesis we consider also common-mode transient communication faults such as those caused by localized EMI or EMI bursts that affect several communication rounds.

Finally, whereas [18] considers arbitrary, i.e. Byzantine, faults we do not. The reason for this is that whereas the SPIDER protocol tolerates these faults, both the GMP and RB protocols we analyzed do not. Therefore, a single arbitrary fault will lead to violation of its fault assumptions. As pointed out in [18], faults outside the MFA are addressed by the concept of assumption coverage, as defined by Powell in [34]. However although presented protocols are not tolerant to Byzantine faults, they are nevertheless tolerant to strictly omissive asymmetric communication faults. Furthermore, our reliability evaluation shows that they provide acceptable reliability levels for most safety-critical applications.

Chapter 3

Group Membership Protocol (GMP)

This chapter presents the specification of the Group Membership Protocol (GMP), a protocol specially designed for safety critical applications on DuST networks. Furthermore, this chapter presents some arguments on the correctness of the GMP. The content of this chapter essentially comprises the text published in [7].

3.1 Introduction

Group membership is considered an important abstraction to facilitate the provision of fault tolerance in systems in general [14] and in safety-critical applications in particular [6]. In this chapter, we describe a group membership protocol for real-time safety-critical applications, specially designed for communication systems that support both static and dynamic communication scheduling in a communication cycle such as FlexRay [3] and FTT-CAN [35]. Both FlexRay and FTT-CAN provide a basic set of fault-tolerant communication services, but no group membership.

A group membership protocol comprises two fundamental operations: failure detection and agreement. Failure detection is performed locally by a processor, by monitoring the messages it receives, or it does not receive, from other processors.

Agreement is achieved through the exchange among the different processors of the perceived operating state of processors in the system.

Currently, many communications systems for safety critical real-time applications [6] are synchronous with only static communications scheduling implemented on top of a protocol based on Time-Division Multiple Access (TDMA). I.e., in those protocols it is assumed that in each TDMA cycle, each processor can send a fixed amount of traffic, which must be sufficient to satisfy the worst case traffic requirements of all the applications in the processor. Therefore, virtually all group membership protocols for this class of systems described in the literature use the messages sent in a TDMA cycle to detect failure of a remote processor. Furthermore, to achieve agreement, they exchange processor state information in every TDMA cycle and strive to minimize this information. Essentially, the differences among the published protocols depend on the failure assumptions; usually, the stronger these assumptions, the more efficient the protocol. For example, the group membership protocol [16, 17] executed in every cycle of the TTP/C [36], requires the sending of only one additional bit per cycle per processor [37]. However, it can tolerate at most one failure within any interval of two TDMA cycles. Actually, the implementation of the protocol takes advantage of this property, and achieves group membership without sending any additional bit. Given that the protocol interprets a communication failure as a processor failure, this assumption is too strong for the operation environment of many safety-critical applications. Therefore, in the TTP/C, if a processor detects a potential violation of this assumption, it switches to the blackout operating mode, in which system operation is severely degraded. On the other hand, the protocol of Ezhilchelvan and Lemos [20] tolerates the failure of up to half of non-faulty processors in three consecutive TDMA cycles, but requires that every processor broadcast a vector with the state of every other processor in every TDMA cycle. Given that buses for safety-critical applications may have several tens of processors [38], this protocol may lead to TDMA cycles with a duration longer than required by the real-time applications that execute in the system.

The proposed GMP protocol relies on the observation that group membership does not change in every TDMA cycle, and takes advantage of next generation communication protocols for safety-critical applications such as FTT-CAN and FlexRay. In these communication protocols the TDMA cycle is divided in two segments: a statically scheduled segment essentially to support periodic traffic,

and a dynamically scheduled segment essentially to support aperiodic traffic or traffic required for non-safety-critical applications. Therefore, the basic idea of the protocol is to exchange information on the processors state in the dynamically scheduled segment only and when there is a change to the membership. In a quiescent state, when there are no membership changes, the protocol requires only an overhead of two bits per processor per cycle that are sent in the static segment. Note that even in the case of membership changes, the protocol requires the exchange of less state information than the protocol proposed by Ezhilchelvan and Lemos, while tolerating a higher fault arrival rate than their protocol.

The remainder of the chapter is organized as follows. In the next section we present the system model, including the fault assumptions. In Section 3.3 we provide the specification of the group membership protocol. A protocol satisfying this specification is presented in Section 3.4. The protocol is developed gradually and we provide informal arguments for the correctness of each version. In Section 3.5 we outline its implementation on top of FlexRay. Finally we summarize the chapter in Section 3.6.

3.2 Model

We assume a synchronous system that is composed of a fixed set of processors P that are connected via a broadcast network, in which a processor receives every message it broadcasts.

The execution model is based on the one presented in [39]. Each processor begins its execution in some start state and then repeatedly executes, in lock-step with the other processors, a *phase* that has two steps:

Communication step, in which each processor generates a *message*, if any, that depends on the processor's *state*, broadcasts it to the network, and receives messages broadcasted in this step;

Processing step, in which each processor generates the new state, by processing the messages received in the communication step.

Note that a state is comprised of the values of a set of state-variables and the set of states may not be finite. Some states are *halting states*, i.e. a processor in such a state will not send any message or modify its state.

By using this model, we abstract away some details that are not essential, thus simplifying the presentation of the protocol. For example, we do not consider the exact time when a processor broadcasts its message. We assume that some protocol ensures that every processor is granted access to the network to broadcast its message and that every processor knows when no more messages broadcasted in the communication step will be received, so that the processing step can be executed.

Processors can fail by experiencing one of three types of faults: a *crash fault*, i.e. a processor enters an halting state and takes no further action; a *receive fault*, i.e. a fault on reception, that prevents a processor from receiving a message broadcasted by another processor in that step; a *send fault*, i.e. a fault on broadcasting, that prevents a processor from broadcasting a message to the network. Note that receive and send faults do not need to be persistent, e.g. a processor may have a receive fault in a phase, but be able to receive a message in a later phase. We say that a processor is non-faulty if it has not experienced any fault since the beginning of the execution.

Finally, we assume that the network provides a reliable broadcast service. I.e. a message broadcasted by a non-faulty processor will be correctly received by all other non-faulty processors.

3.3 Group Membership Specification

We state the Group Membership problem in terms of the set of group members (M-SET) maintained by every processor.

We consider essentially two properties:

Agreement: All non-faulty group members compute the same M-SET.

Validity:

1. A faulty processor will be removed from the M-SET of a non-faulty group member in a bounded time interval;
2. A non-faulty processor attempting to be reintegrated will be added to the M-SET of a non-faulty group member in a bounded time interval.

In addition to these properties, Cristian [14] defines stability properties of the group membership to ensure that a group's membership does not change for no reason. The proposed protocols satisfy such properties, but we will not provide the arguments for that, because of lack of space.

3.4 Group Membership Protocol

The protocol has two phases that are repeated in every cycle: Failure Detection phase (FD-phase) and the Group Membership phase (GM-phase). Every group member is required to broadcast one message in the FD-phase, so that failures can be detected with bounded delay. In the GM-phase a protocol may be executed to achieve agreement on the group membership.

We present first a very basic protocol that does not support processor reintegration and that requires the sending of messages in both phases of every cycle. Then we add support for processor reintegration and finally present a version that not only supports processor reintegration, but also does not require sending messages in the GM-phase of every cycle. Note that the goal of presenting several versions with increasing complexity is to facilitate the understanding of the final protocol. The intermediate protocols are not intended to be the most efficient of their kind.

All three versions use the same majority function, majSet, that computes the set of elements that are members of at least a majority of n sets. A special value, *undefined*, is returned when no majority is found. Figure 3.1 shows the pseudo-code for the majSet function. Note that undefined is returned only when the number of sets in R is smaller than n and there is no agreement among the sets in R. E.g., consider the invocation $majSet(\{a, b\}, \{\{a, b\}, \{a\}\}, 3)$. In this case, although a is a member of two sets in R, a majority for $n = 3$, there is no majority of sets agreeing on the membership of b. Thus this invocation would return undefined.

3.4.1 Base Protocol

In addition to the M-SET, in the base protocol every processor maintains the M-set, the set of candidate members, and an integer, u, an upper bound on the

```
Set majSet(Set S, SetofSet R, int n)
begin
   Set M to the ∅
   for every p in S do
      if p is an element of ⌈n/2⌉ or more sets in R
      then add p to M
      else if p is not an element of ⌈n/2⌉ or more sets in R
      then continue
      else return undefined
      end
   end
   return M
end
```

Figure 3.1: majSet function used by all versions of the protocol.

size of the group. Initially, both sets are set to P, the set of processors, and u is set to the size of P.

During the FD-phase, every group member broadcasts a heartbeat message and receives the heartbeats broadcasted by group members in this phase. Then it removes from its M-set the processors from which no heartbeat message was received in this phase.

During the GM-phase a group member broadcasts the M-set it computed in the FD-phase, and receives the corresponding sets broadcasted by group members. It then computes the set of candidate members proposed by a majority of all the group members, the Maj-set, by applying the majSet to the M-sets it received from group members, i.e. processors in the M-SET. If this set is *undefined* or different from the processor's M-set or the processor is not in the Maj-set, then it halts. Otherwise, it uses the Maj-set and the set of all M-sets to compute the new group membership, i.e. the M-SET.

3.4.1.1 Properties of the Base Protocol

We now argue that this protocol satisfies the Agreement and the first of the Validity properties stated above.

Agreement A rigorous proof of Agreement can be done by induction on the number of cycles. Here we provide informally the arguments that can be used in such a proof. First, note that given the assumptions on the communication subsystem, every non-faulty processor receives the same set of messages in every phase. Because the computation of the M-SET is based on a deterministic algorithm and, assuming that every non-faulty group member computed the same

Group Membership Protocol (Base Version)

State

P the set of all processors

M-SET the set of group members, initially set to P

M-set the set of candidate group members, initially set to P

u upper bound of the group's size, initially set to $|P|$

FD-phase

Communication step: Broadcast heartbeat message.

Processing step: Remove from the M-set every processor from which no heartbeat message was received.

GM-phase

Communication step: Broadcast the M-set.

Processing step:

1. Let Maj-set be the result of applying the majSet function to P, the set of all the M-set's received from processors in the M-SET and u.
2. If the Maj-set
   a) is undefined, or
   b) is different from the M-set the processor broadcasted, or
   c) does not contain the processor;
   then halt.
3. Remove from the M-set every processor from which an M-set different from the Maj-set was received.
4. Set u to the size of the M-set.
5. Remove from the M-set every processor from which no message was received in this phase.
6. Set the M-SET to the M-set.

Figure 3.2: Base version of Group Membership Protocol-GMP

values for u and for M-SET in the previous cycle, every non-faulty group member uses the same inputs, therefore every non-faulty group member will compute the same values for u and for the M-SET.

Note that the base protocol can be seen as yet another instance of the state machine approach to fault-tolerance [40, 41], and the arguments provided in the previous paragraph are the standard arguments of such an approach and independent of the problem at hand. The assumptions on the communication subsystem make the arguments easier, because they imply interactive consistency with the fault model considered, i.e. the communication subsystem provides a reliable broadcast service.

Validity It can be shown that the base protocol ensures that a faulty group member will be removed from the M-SET of a non-faulty group member in no later than two cycles. I.e., if a group member has a fault in a cycle, then it will be removed from the M-SET of non-faulty group members by the end of the following cycle, in the worst case, as long as a majority of group members remain non-faulty.

The proof can be done by case analysis, considering each of the three possible processor faults in each of the two protocol phases. Here we just outline the main arguments used in that case analysis. Send faults are easily detected by a non-faulty group member because it will not receive the message the faulty processor was supposed to send. If the fault occurs in the FD-phase, then the processor will be removed from the M-set of non-faulty group members, and because they are a majority it will not be a member of the Maj-set and consequently of the M-SET by the end of that cycle. If the fault occurs in the GM-phase, then non-faulty group members will not receive the M-set from the faulty processor and will remove it from their M-SET by the end of that cycle. Processor crashes are detected by other processors when the crashed processor does not send a message it was supposed to send. Therefore this kind of fault is similar to a send fault, except that there may be an extra delay between the occurrence of the fault and its detection. This delay can be as long as one phase when a processor crashes immediately after sending the message in a phase. In any case, a crashed processor will be removed from the M-SET of non-faulty group members at the latest by the end of the following cycle. Receive faults are detected by comparing the M-set received from the faulty processor with the Maj-set computed in the GM-phase. If a processor has a receive fault in the FD-phase, then it will erroneously remove the sender of the missing message from its M-set whereas non-faulty group members will not. Therefore, the M-set broadcasted in the GM-phase by the faulty processor will differ from the Maj-set computed by non-faulty group members in at least the sender of the missed message, and the faulty processor will be removed from the M-SET of non-faulty group members. Receive faults in the GM-phase will result in the sender of the corresponding message being removed from the M-SET of the faulty processor because it did not receive the former's M-set. This will be detected in the following cycle, because the M-set broadcasted by the faulty processor will not include the sender of the missed message, whereas that of non-faulty group members will.

Note that we do not claim that we have just given a proof of correctness of the base protocol. Actually, the last sentence of the previous paragraph will be true only if the processor that sent the missed message does not fail to send its message in the FD-phase of the following cycle. Therefore, with rigor, the base protocol satisfies Validity only if we strengthen the fault assumptions by excluding send faults in the FD-phase by the senders of messages on which other processors had a receive fault in the GM-phase of the previous cycle. It is possible to modify the protocol to eliminate this exception, but it requires sending extra information in the GM-phase message.

3.4.1.2 Faulty Processor Behavior

Usually, it is easier to build fault-tolerant systems if faulty processors fail silently. Because of that, the protocol requires a processor to halt whenever its state indicates that the processor may be faulty, given the assumptions made. As a result:

Proposition A faulty processor will halt at the latest by the end of the cycle after which it has a fault.

Again, this proposition can be proven by induction and by case analysis using essentially the arguments used above to argue for Validity; the caveat regarding receive faults in the GM-phase still applies. But the key to the proof is that the value u computed is an upper bound of the group size. As a result, under our fault assumptions, faulty processors will never be able to compute a Maj-set different from that computed by non-faulty processors. Note that if u were computed at the end of the processing step of the GM-phase rather than in point 4, a faulty processor might compute a value lower than the actual group size. Therefore, it could compute a different non-undetermined Maj-set, even though there were a majority of non-faulty group members, and do not halt.

Halting faulty processors complicates slightly the protocol, but it is important for the final version of the protocol, therefore we decided to introduce this feature early on.

3.4.2 Processor Reintegration

The base protocol does not handle processor reintegration. From a practical point of view processor reintegration is very important as it allows reintegrating not only repaired processors but also non-faulty processors that were excluded because of transient communication faults.

Handling processor reintegration requires a very simple modification to the base protocol. Essentially, a processor that wishes to join (or rejoin) the group needs to send a *join request* message instead of the heartbeat message in the FD-phase and then it has to participate in the GM-phase that follows, as if it were a member of the group. However, because its state may not be in agreement with that of group members, its messages have to be handled in a special way in the GM-phase. Furthermore, every group member has to broadcast not only its M-set but also its group size upper-bound, so that a joining processor can compute the Maj-set.

The new version of the protocol is presented next and changes with respect to the base protocol are highlighted in bold.

3.4.2.1 Properties of the Protocol

It can be shown that this protocol satisfies both the Agreement and the Validity properties, with the same strengthening of the fault assumption as for the base protocol, as long as more than half of the group members remain non-faulty from one group to the next. The delay for removing faulty processors or to add joining processors is, in the worst case, two cycles.

As mentioned above, Agreement is straightforward as the protocol uses the state machine approach to fault-tolerance. Note that although the state of joining processors at the beginning of a cycle may not be equal to that of group members, the computation of the M-SET does not use directly that state, but rather information derived from the state that is broadcasted to all processors, including the sender itself.

With respect to Validity, the reasoning that faulty processors are removed from the M-SET of non-faulty group members still applies. However, correctness also depends on the upper bound of non-faulty group members being the

Group Membership Protocol (Reintegration Version)

State

P the set of all processors

M-SET the set of group members, initially set to P

M-set the set of candidate group members, initially set to P

u upper bound of the group's size, initially set to $|P|$

FD-phase

Communication step:
If processor is group member
Then broadcast hearbeat message
Else if wishing to join group
Then set the M-set and the M-SET to P, u to the size of P, and broadcast join-req message.

Processing step:

1. Remove from the M-set every processor from which no heartbeat message was received.
2. **For every join-req message received add its sender to the M-set.**

GM-phase

Communication step:
Broadcast message with the M-set **and the group's size upperbound, u.**

Processing step:

1. Let Maj-set be the result of applying the majSet function to P, the set of all the M-set's received from processors in the M-SET **and the minimum of all u's received in the same messages**.
2. If the Maj-set
   a) is undefined, or
   b) is different from the M-set the processor broadcasted **and the processor is a member of the group,** or
   c) **is not a subset of the M-set the processor broadcasted and the processor is joining the group, or**
   d) does not contain the processor
   then halt.
3. Remove from the M-set
   a) every group member from which an M-set different from the Maj-set was received;
   b) **every joining processor whose M-set is not a superset of the Maj-set;**
4. Set u to the size of the M-set.
5. Remove from the M-set every processor from which no message was received in this phase.
6. Set the M-SET to the M-set.

Figure 3.3: Reintegration version of Group Membership Protocol-GMP

minimum of all the upper bounds. As already mentioned, this can be shown by induction on the number of cycles, and relies on faulty processors halting no later than the cycle when they have a fault, which is assured by the protocol. The same arguments can be used to show that faulty joining processors either will not be added to the M-SET or will be removed from the M-SET in the cycle immediately after being added.

The addition of non-faulty joining processors to the M-SET of non-faulty group members can be argued based on the fact that Join request messages are received by all the non-faulty group members. Therefore every non-faulty joining processor is added to the M-set of every non-faulty group member by the end of the FD-phase, and the M-set broadcasted by non-faulty group members in the GM-phase will include every non-faulty joining processor. Furthermore, even though a joining processor may not send an M-set equal to the Maj-set, because it initializes its M-set to the set of all processors, P, its M-set will be a superset of the Maj-set, and therefore a non-faulty joining processor will be a member of the M-SET of non-faulty group members by the end of the cycle. The delay for joining is in the best case one cycle and in the worst case two cycles, because a joining processor must execute a full FD-phase.

3.4.3 Protocol without GM-phase Messages

The versions of the group membership protocol described so far require sending messages in every GM-phase, and therefore are not particularly advantageous with respect to other protocols that were developed for communications systems with a static schedule only, e.g. the protocol by Ezhilchelvan and Lemos [20].

However, the decomposition of the protocol in two-phases, FD-phase and GM-phase, with failure detection occurring in the FD-phase and agreement in the second phase, allows for the non-execution of the latter, if there are no events that trigger the change of the group membership. This way, group membership is maintained with virtually no messages when the system is in a quiescent state, and the dynamically scheduled segment will be available for other traffic.

In the new and final version of the protocol, a processor sends a GM-phase message when it detects a fault that may lead to modifying the group membership. Furthermore, in the FD-phase, processors may explicitly request the sending of GM-phase messages. This is used after a GM-phase execution in which a

processor modified its M-SET in a way that other processors may not be aware of.

Because the GM-phase does not have to be executed in every cycle, there is the possibility that a faulty group member will not detect changes in the group membership, and later create havoc. To prevent that, every group member keeps a group id, a monotonically increasing integer variable, that is incremented by one every time there is a group membership change. The group id is sent together with the M-set and the group size upper bound in the GM-phase. The correctness of the final version relies on the fact that the group id of non-faulty group members is the maximum of all the group ids of all processors.

The final version of the GM protocol is presented next and changes with respect to the version supporting reintegration are highlighted in bold.

Note that it is essential to execute a GM-phase after another GM-phase in which a processor removes from the M-set processors from which it did not receive their M-set. This will allow detection of faults in the GM-phase. These faults are detected locally by each receiving processor, but by itself the receiving processor is unable to determine whether it had a receive fault or the sending processor had a send fault. To resolve this dilemma, a processor needs to know what is the perception of other group members.

3.4.3.1 Protocol Properties

Again, it can be shown that the final version of the protocol supports the Agreement and the Validity properties with the same fault assumption strengthening as for the previous versions, i.e. no send fault in the FD-phase by a processor that sent a GM-phase message in the previous cycle on which another processor had a receive fault, and assuming that at least a majority of group members remain non-faulty. As in the other versions of the protocol, with these assumptions, a faulty processor will be removed from the M-SET of non-faulty group members at the latest by the end of the cycle following the one in which it has a fault. Likewise, a non-faulty joining processor will be added to the M-SET of the non-faulty group members at the latest by the end of the cycle following the one in which it decides to join the group. It can also be shown, that under these

Group Membership Protocol (Final Version)

State

P the set of all processors

M-SET the set of group members, initially set to P

M-set the set of candidate group members, initially set to P

u upper bound of the group's size, initially set to $|P|$

group-id integer with group id, initially set to 0

GM-request boolean indicating whether execution of the GM-phase should be performed, initially set to false

FD-phase

Communication step:
If processor is group member
Then broadcast hearbeat message **with GM-request as determined in the previous GM-phase**
Else if wishing to join group
Then set the M-set and the M-SET to P,
u to the size of P, **the group-id to zero**
and broadcast a join-req message.

Processing step:

1. Remove from the M-set every processor from which no heartbeat message was received.
2. For every join-req message received
add its sender to the M-set.
3. **If received a message with GM-request**
or modified the M-set in 1 or 2
Then set GM-request.

Figure 3.4: Final version of Group Membership Protocol-GMP (part 1)

fault assumptions, a faulty processor will halt the latest by the end of the cycle following the one in which it has a fault.

The arguments presented for the version of the protocol supporting processor reintegration still apply, as long as the group id of non-faulty group members is the maximum of all the ids. Again, a rigorous proof can be done by induction on the number of cycles. Informally, note that if, in a GM-phase execution, a faulty group-member increments its group id, then it must have been able to compute a Maj-set with a value different from *undefined.* Because, faulty group members are a minority, then it must be the case that non-faulty group members also executed the GM-phase and will, therefore, increase their group id.

Group Membership Protocol (Final Version)

GM-phase

If GM-request is set, then

Communication step:

Broadcast message with the M-set, the group's size upperbound, u, **and the group-id**.

Processing step:

1. **Let max-id be the maximum of the group ids received.**
2. **If the processor is joining**
   Then set the group-id to max-id
   Else if its group-id is different from max-id
   Then halt
3. Let Maj-set be the result of applying the majSet function to P, the set of all the M-set's received from processors **with a group-id equal to max-id,** and to the minimum of all u's received in the same messages.
4. If the Maj-set
   a) is undefined, or
   b) is different from the M-set the processor broadcasted and the processor is a member of the group, or
   c) is not a subset of the M-set the processor broadcasted and the processor is joining the group, or
   d) does not contain the processor

   then halt.
5. Remove from the M-set
   a) every group member from which an M-set different from the Maj-set was received;
   b) every joining processor whose M-set is not a superset of the Maj-set;
6. Set u to the size of the M-set.
7. Remove from the M-set every processor from which no message was received in this phase.
8. **If removed some processor from M-set in 7**
   Then set the GM-request
   Else reset the GM-request.
9. Set the M-SET to the M-set **and increment the group-id.**

Figure 3.5: Final version of Group Membership Protocol-GMP (part 2)

Regarding the upper bounds of delays of the Validity property, note that faults in the FD-phase will be detected as before in the same cycle, although in the case of a receive fault the faulty processor may be removed from the M-SET only in the following cycle, as a result of the processor halting in the GM-phase of that cycle. Faults in the GM-phase are detected as before, indeed if a processor

has a receive fault it will request the execution of the GM-phase in the following cycle. Crash faults will originate send faults at the latest in the FD-phase of the following cycle, therefore the faulty processor will be removed from the M-SET at the latest the following cycle. With respect to the upper bound of the delay for a processor to join a group, the protocol execution is essentially identical to that of the version supporting processor reintegration only, and the arguments used there still apply. Thus, a non-faulty processor wishing to join the group will be added to the M-SET of non-faulty group members the latest by the end of the following cycle.

Note that just as non-faulty processors detect failure of a processor no later than the end of the following cycle, faulty processors also detect their own faults by the same time. Indeed, it can be shown by case analysis that for any fault different from a crash, a faulty processor will execute the GM-phase protocol either in the same cycle or in the following cycle, and in both cases it will halt.

3.5 Implementation Outline

The GMP was designed for take advantage of a new class of dual scheduled time division multiple access (TDMA) control protocols (DuST protocols), of which FlexRay is the best known example. The FD-phase, which must be run in every cycle to ensure timely fault detection, can be scheduled statically, whereas the GM-phase, which may not need to be executed in every cycle, can be scheduled dynamically. In this section, we provide further details on a possible implementation of this protocol on top of the FlexRay protocol.

3.5.1 FlexRay

FlexRay [3] is a TDMA-based medium access control protocol that supports both static (TDMA) and dynamic (FTDMA) communications scheduling. FlexRay is an evolution of the Byteflight [42], a FTDMA based protocol, which also incorporate concepts imported from the time-triggered approach. The FlexRay protocol is rather complex and below we omit some details to simplify its description. These omissions make the protocol less flexible than it really is, but they do not make it behave in a way that violates its specifications.

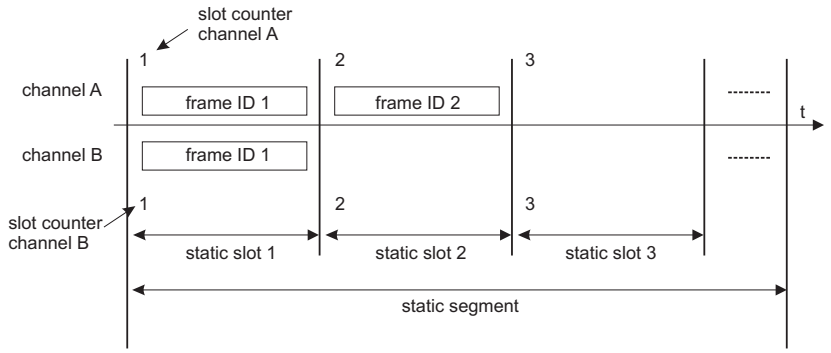


Figure 3.6: Structure of static segment of FlexRay protocol [3].

In FlexRay every TDMA cycle has a fixed length and it comprises two segments, a static segment and a dynamic segment, each of which has a fixed length. Both segments are divided in slots, however, whereas in the static segment (see figure 3.6) all slots in a cycle have the same fixed length, which does not change between cycles, in the dynamic segment (see figure 3.7), the length of the slots may vary both within a cycle and between cycles. More precisely, in the dynamic segment, time is divided in mini-slots and each slot has an integer number of mini-slots that may vary, depending on the amount of data to transmit.

Each slot is assigned in every cycle to the same processor, implicitly determining the processor that has access to the medium. However, whereas in the static segment, every processor transmits a frame in its slots, even if it has no data to send, in the dynamic segment, a processor may not transmit a frame in a slot assigned to it, in which case the slot takes only one mini-slot.

Medium access control is implemented using a slot counter per processor, which identifies the frame that is transmitted in that slot, if any, and that is reset to one in the beginning of every TDMA cycle. The FlexRay protocol ensures that the slot counters of all non-faulty processors are synchronized. In the dynamic segment, the slot number effectively determines the priority of the frames: depending on the configuration, it may be possible for a set of frames to prevent other frames with a higher id from being sent to the medium.

3.5.2 Implementation Issues

The group membership protocol puts some requirements on the communication system. First, it requires that every processor broadcast a heartbeat message in

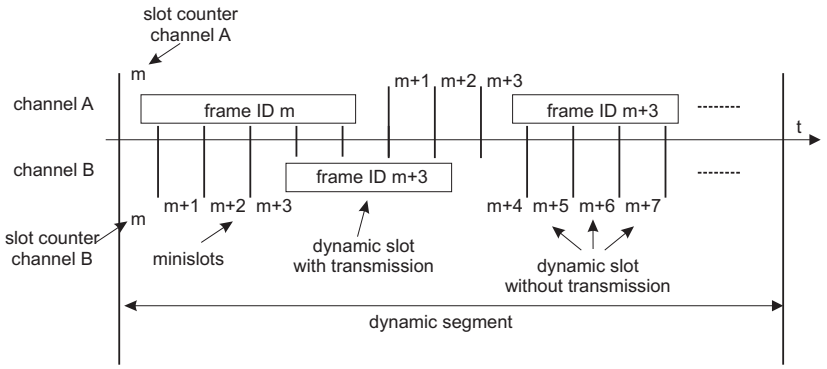


Figure 3.7: Structure of dynamic segment of FlexRay protocol [3].

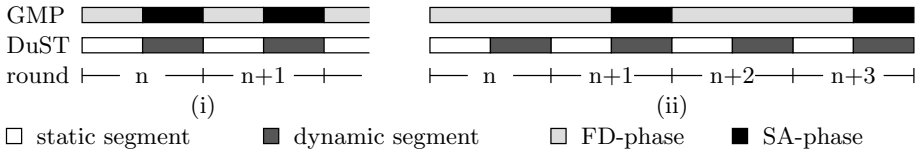


Figure 3.8: GMP phases *vs.* DuST phases. Single-round (i) and multiple-round diagnosis periods (ii).

the FD-phase and that the communication system be able to detect the absence of messages. Second, it requires that, if necessary, GM-phase messages be sent in the next GM-phase. Third, it requires that processors be able to signal their wish to join the group or to request the execution of the GM-phase. All these requirements can be satisfied by FlexRay as we argue next.

Regarding the first requirement, we note that in FlexRay's static segment a processor will always transmit in its slots, even if it has no application data to send. Therefore, by assigning a slot of the static segment to each processor, we can use the frames broadcasted in the static segment as heartbeat messages of the FD-phase (Figure 3.8(i) illustrates the relationship between GMP phases and the segments of DuST protocols, FlexRay in this case). The loss of one of these messages is detected by FlexRay, which provides several status bits that can be used to distinguish a send omission, caused by a crash of the sender, for example, from a message loss caused by communication faults, for example.

With respect to the GM-phase messages, they are broadcasted in the dynamic segment, because we expect that events that may lead to group membership changes do not occur that often. But when they do occur, it is important to

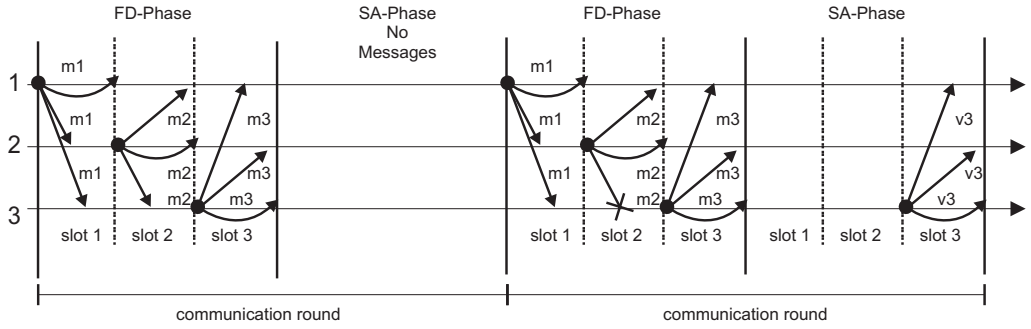


Figure 3.9: Possible execution of the GMP, illustrating the execution of the SA-phase only when membership-related events are observed.

broadcast these messages as soon as possible. For Example, figure 3.9 illustrates one possible execution of the GMP for a configuration with 3 nodes, which are assumed to be members of the group at the beginning of the first TDMA cycle, c. In that cycle, the 3 nodes execute only the FD-phase, i.e. only send their heartbeat messages. As no event that might lead to a change in the membership is observed by any of the 3 nodes, there is no execution of the SA-phase (equivalently, we say that the SA-phase has no messages). However, in the following TDMA cycle, node 3 has a receive fault on the heartbeat message sent by node 2. As a result, it sends a vote message in the following round (round n+3). The other nodes do not observe any event that might lead to a change in the membership and therefore do not send any vote.

An implementation satisfying this requirement is to reserve the first slots of the dynamic segment for the exchange of the GM-phase messages, by assigning one of these slots to each of the processors and binding the corresponding frame to the GM-phase message. This way, it is ensured that GM messages will not be preempted by other messages of the dynamic segment. If a processor has no GM-phase message to send, the corresponding slot will take only one mini-slot. All the mini-slots remaining in the dynamic segment after the broadcast of GM-messages can be used to exchange application data that presumably does not have hard deadlines for transmission. Note that, if the maximum time available for the dynamically scheduled traffic is not enough for a complete execution of the GM-phase protocol, the GM-phase may have to be spread over several dynamic segments, i.e. TDMA cycles, leading to higher latency in the group membership change. Fault-rate assumptions will be affected accordingly.

Finally, there is a need for some mechanism allowing processors to signal their wish to join the group or to request the execution of the GM-phase. In our protocol, this is done in the FD-phase, i.e. in the static segment. Thus a possibility is to reserve two bits in the message sent by each processor in the static segment, a join-bit and a group-bit that, if set, indicate a join request and a GM execution request, respectively. The main problem with this implementation is that these bits will use some of the space reserved for application data. An alternative that avoids this overhead is to use the Network Management (NM) Vector, which is an optional field of the payload of a frame in the static segment. This solution has the problem that the NM Vector is an optional feature, and as such may not be implemented by every FlexRay communications controller.

3.5.3 Assumptions

In addition to these requirements on the communication system, we made some assumptions regarding the capabilities of the communication system and the types of faults.

One important assumption is that the communication system supports reliable broadcast. This is certainly not true for FlexRay and, occasionally, a message broadcasted by a non-faulty processor may be corrupted by communication faults. We can make such scenarios less likely, by using FlexRay's support of duplicated communication channels: as long as a message is received correctly in one of the channels, it will be considered correct. Note that whereas FlexRay provides an abstraction of a single channel in the static segment, it does not provide such an abstraction in the dynamic segment. Thus, it is up to the application to build such an abstraction on top of FlexRay.

Whether or not a duplicated communication channel is used, messages will still be lost. In most cases, this will be detected by all non-faulty processors, but in some rare cases, an asymmetrical fault may result and some processors will discard the message whereas others will not. By assuming that the communication system is reliable, we assign the responsibility for these faults to processors. Essentially, we convert communication faults into processor faults. The symmetric case is not very problematic, if the group has several members, because it leads to the exclusion of only one processor. On the other hand, the case of asymmetric faults is not very well tolerated, as a single communication fault may lead to

the exclusion of several processors. In an extreme case, if other faults occur, all processors may halt because there is no majority.

To make the protocol more tolerant to communication faults we propose an extension of the GMP, in which the FD-phase rather than comprising the static segment of one communication round, comprises several static segments in consecutive communication rounds (Figure 3.8(ii)). This extension makes the GMP more resilient to transient communications faults and provides higher reliability in environments prone to these faults. We call this extension multiple-round diagnosis period GMP whereas we use the expression single-round diagnosis period GMP for the original protocol.

Another type of fault that may seriously affect the reliability of the protocol is the *babbling idiot*, a processor that transmits in slots other than its own. Our assumptions ruled out this type of fault, and therefore we should show that FlexRay prevents or masks these faults. Indeed, FlexRay provides a bus guardian to protect the communication channels in the static segment, by cutting off of the network the offending processor. However, communication in the dynamic segment is not similarly protected, and therefore such a processor might prevent the exchange of GM-phase messages. This is a limitation of the implementation of our protocol in FlexRay.

3.5.4 GM-phase Messages and Overhead

A key issue regarding GM-phase messages is the encoding of the group id. In the protocol, the group id is an integer that is incremented in every execution, and therefore it will increase without bounds. In order to minimize the traffic overhead caused by the protocol, the group id must be encoded with the minimum number of bits. Because the protocol ensures that a faulty processor halts no later than the TDMA cycle after which it has a fault, two bits are enough for encoding the group id. The encoding of the M-set and of the group size upper bound present no difficulty. The M-set can be encoded as a bit-vector and the group size upper bound as an integer. Thus, assuming a system with 64 processors, each GM-phase message could be 9 bytes long, only: 8 bytes for the M-set and one byte packing the group size upper bound and the group id.

3.6 Conclusion

We presented a new group membership protocol specially designed for next generation communication systems intended to support real-time safety-critical applications. By taking advantage of static and dynamic communication scheduling supported by these TDMA-based protocols, it has an overhead of only two bits per processor per cycle in a quiescent state, i.e. when there are no group membership changes. In addition, it tolerates benign failures of up to half of the group members between consecutive executions. Even when there are group membership changes, the overhead of the protocol is lower than that of other protocols that provide the same level of fault tolerance. The protocol is also very responsive, removing faulty processors no later than two TDMA cycles after they fail, and (re)integrating them no later than two TDMA cycles after they decide to join. These numbers compare favorably with protocols designed for systems based on TDMA with static scheduling only: the protocol is as fault tolerant as the most fault tolerant protocols [20], and has a slightly higher overhead than the most efficient [17], which assumes only one fault per TDMA cycle.

To show the feasibility of the group membership protocol, we have also outlined an implementation on top of the FlexRay protocol. We have found no major difficulty. However, the protection against the *babbling idiot* fault in the dynamic segment might be useful.

Chapter 4

Formal Verification of GMP

This chapter presents the formal verification of Group Membership Protocol (GMP) by means of model checking. It includes a detailed description of the models developed for the UPPAAL model checker and a discussion of the result obtained. The content of this chapter essentially comprises the text published in [9].

4.1 Introduction

In the previous chapter, we have presented a new group membership protocol (GMP), that takes advantage of the dual scheduling ability of the class of TDMA protocols used by FlexRay and argued informally its correctness. However, fault-tolerant distributed protocols are very subtle and informal arguments are prone to error making them clearly insufficient for safety-critical applications, which require a high level of assurance that they operate correctly. This holds especially for middleware that is supposed to be used in the development of safety critical applications. Ideally, mathematical proofs, either manual or automatic, of its correctness should be provided. An alternative formal method is model checking.

Model checking is a technique for verifying properties of a system through exhaustive and automatic exploration of all the system states. One problem with model checking is the state space explosion, i.e. the exponential growth of the number of system states when the number of components or the number of vari-

ables and their possible values increases. A well known technique to address this problem is symmetry reduction [43], which tries to explore the structural symmetry of the model. This technique is particularly effective in models composed by identical components, such as in distributed protocols.

In this chapter we focus on the use of symmetry reduction in verifying the GMP. The GMP is particularly challenging in this respect, because its behavior is somewhat "irregular". This is compounded by our desire in keeping the model close to the GMP, in order to ensure a high confidence level on the verification results. Therefore, the model we have developed includes an implementation of the GMP that could be used almost verbatim in an executable implementation of the protocol. This allowed us to detect an error in the outline of an implementation of the GMP protocol that we previously proposed.

The remainder of this chapter is organized as follows. In the next section we provide a very quick review of Uppaal, the model checker we use. In Section 4.3, we describe an Uppaal model of the GMP. The techniques used to reduce the state space size are described in Section 4.4. Section 4.5 presents the correctness properties, and in Section 4.6 we present and discuss the verification results. Finally, we conclude in Section 4.7.

4.2 Background

4.2.1 UPPAAL

The Uppaal model checker [44] is a toolbox for the verification of real-time systems modeled as non-deterministic timed automata. A timed automata [45] is a finite-state machine containing a set of clocks that advance synchronously. Uppaal supports a number of extensions to timed automata such as integer variables, structured data-types and channel synchronization, that make it suitable to model more than just the temporal behavior of a system.

Uppaal models comprise a set of timed-automata that execute concurrently and that may synchronize with each other through broadcast or binary channels. Figure 4.1 shows a simple model with two automata, **P** and **Q**, of three and two locations respectively, i.e. **P0** to **P2** and **Q0** and **Q1**. **P0** and **Q0** are the initial locations of the respective automata and are represented as a double circle.

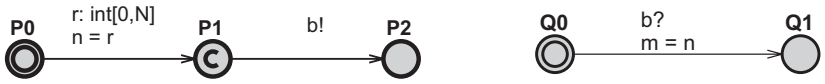


Figure 4.1: Simple Uppaal model composed of two timed-automata.

Location **P1**, represented as a circle with a C inside, is a committed location, which means that time is not allowed to advance while such a location is active. The model includes channel `b` and integer variables `m`, `n` and `r`.

Initially, the two automata are in their respective initial location, i.e. **P0** and **Q0**, and all integer variables values are zero. In this state, automaton **Q** cannot take the transition from **Q0** to **Q1** because it is blocked waiting on channel `b`. Therefore, progress is possible only by automaton **P** taking the transition from location **P0** to location **P1**. The edge from location **P0** to location **P1** has a *select*, `r:int[0,N]`, and an *assignment*, `n=r`, labels. The select label binds identifier `r` to a random value in the range `[0,N]`. This value is then assigned to variable `n` in the assignment label. Therefore, when **P** takes transition **P0** to **P1**, variable `n` is assigned a random value in the range `[0,N]`. Because location **P1** is a *committed* location, automaton **P** takes transition **P1** to **P2** immediately after. Simultaneously, automaton **Q** takes transition **Q0** to **Q1**, because both transitions have matching *synchronization* labels on channel `b`. Note that Uppaal also supports the synchronization of multiple automata on a single *broadcast channel*, i.e. if multiple automata are waiting on a broadcast channel, they will all be unblocked if another automata signals that channel.

A more detailed, and formal, description of Uppaal can be found, for example, in [46].

4.3 Verification Model

The Uppaal model of the GMP comprises two types of automata, or templates: `Node` and `Scheduler`. The `Scheduler` automaton controls the evolution of the protocol by initiating each of the GMP phases. The `Node` automaton models the behavior of one node and is the core of the model. A GMP Uppaal model comprises one `Scheduler` automaton and N `Node` automata, where N is the number of nodes in the system.

4.3.1 Basic Model

In order to simplify the presentation of the model we first present a model that does not consider the occurrence of faults.

4.3.1.1 Global Variables and Synchronization Channels

Variables in Uppaal may be either local or global. Local variables are private to a particular automaton. Global variables in Uppaal can be accessed by all the automata in the model, i.e. they are shared, and they play an important role in the communication between automata. This is because synchronization channels in Uppaal are strictly for synchronization; it is not possible to pass data through a channel in Uppaal.

The following table shows some global variables used in the model:

```
typedef struct{                          // Info sent in heartbeat messages
        bool el[N];                      Set SAreqH;
} Set;                                   // Info sent in the vote messages
// GLOBAL State                          meta Set Mset[N];
Set MSETo, Joinable;                     meta int[0,N] gsubV[N];
// Schedule for GM events: joins         meta int[0,MaxGId] gidV[N];
Set Joining;
```

In the GMP model there are two classes of global variables. The first class comprises variables that are updated only by the `Scheduler` and that are intended to control the behavior of the `Node` automata. Two variables of this class are the sets `MSETo` and `Joining`. The former keeps track of the group membership as determined by an omniscient observer, whereas the latter keeps track of the nodes that try to join the group. The second class comprises variables that contain information that a node sends in the messages of the GMP. Two variables of this class are the set `SAreqH` and the array of sets `Mset`. The former contains the nodes that have requested to execute the SA-phase, i.e. each element of this array represents the SA-req bit of the heartbeat message sent by the corresponding node, whereas each element of the latter contains the group membership sent by each node in their vote message.

In addition to global variables the model comprises four broadcast synchronization channels: `join`, `startPhase`, `startProc` and `terminateCycle`. The latter two are not strictly necessary, but are used to eliminate intermediate states

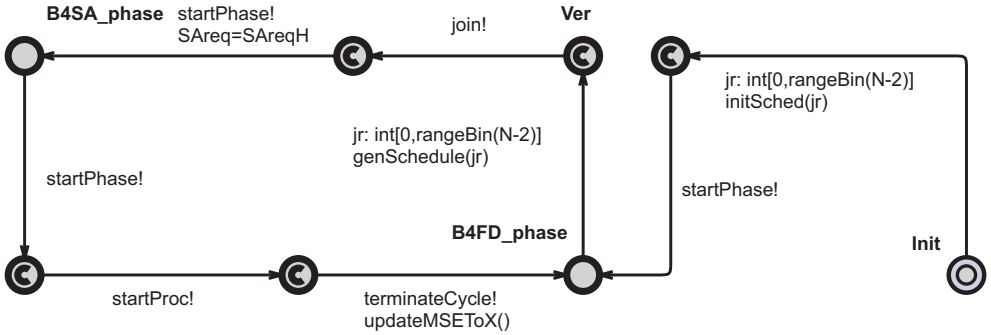


Figure 4.2: Scheduler automaton for the model without faults.

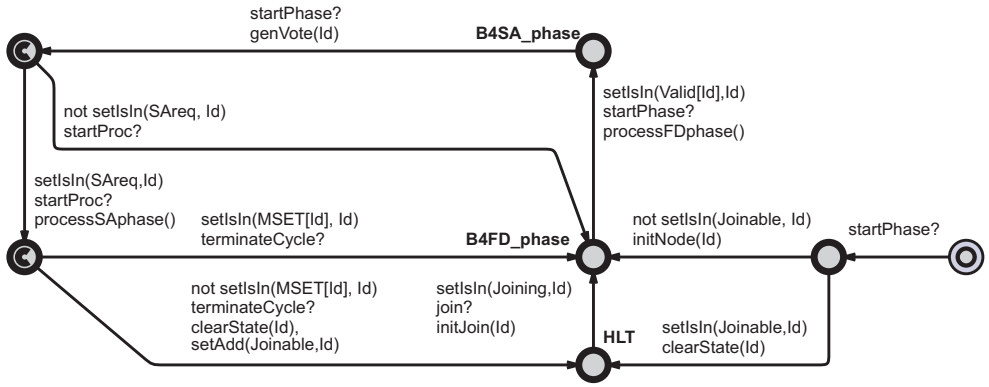


Figure 4.3: Node automaton for the model without faults.

that are not relevant, thus reducing the size of the model's state space. (A detailed discussion on the use of synchronization is presented in Subsection 4.4.3.)

4.3.1.2 Automata

Figures 4.2 and 4.3 show the Scheduler and the Node automata for the basic model. In addition to the two phases of the GMP that are repeated one after the other indefinitely, the model includes an initialization phase. We briefly describe the base model considering each phase in turn. As stated above the Scheduler automaton controls the model by initiating the phases.

Model Initialization: This phase comprises the initialization of the variables of the model. It simplifies the verification of configurations with different number of nodes. In this phase, the Scheduler initializes itself and determines which nodes start as group members and those that do not.

Failure Detection Phase: Just before the FD-phase begins the `Scheduler` automaton is in location `B4FD_phase` and the `Node` automata are either in location `B4FD_phase` or in location `HLT`. Initiation of the FD-phase by the `Node` automata is controlled by the `Scheduler` automata.

At the beginning of the FD-phase, the `Scheduler` determines, with the help of a selection label, which of the nodes that can join the group will attempt it and initializes the `Joining` set with these nodes. After that it signals the selected nodes on broadcast channel `join`, so that they move from `HLT` to the `B4FD_phase` and therefore become ready to initiate the FD-phase. At this point, the nodes that will execute the FD-phase in this execution of the GMP are in the `B4FD_phase` state waiting on the `startPhase` broadcast channel

Immediately after, the `Scheduler` signals on that channel, triggering the execution of the FD-phase by the nodes. The *actions* taken in this transition are specified inside function `processFDphase()`, which is executed by the `Node` automata and does the processing of the FD-phase of the GMP. In this processing, each `Node` automaton updates its `Mset` variable and the `SAreq` variable, as described in GMP specification (chapter 2). In addition to its local state, each `Node` uses all messages it received in the FD-phase as input to `processFDphase()`. This information can be found by looking up sets `MSETo`, `Joining` and `SAreqH`.

Set Agreement Phase The pattern of the set agreement phase (SA-phase) is very similar to that of the other phases.

Before executing the SA-phase the `Scheduler` and the `Node` automata that execute the GMP are all in their `B4SA_phase` location. Execution of the SA-phase by the `Node` automata is driven by the `Scheduler`.

When the `Scheduler` takes the transition out of location `B4SA_phase`, it signals on the `startPhase` broadcast channel, unblocking all the `Node` automata executing the GMP in this cycle. At this point each `Node` sends its vote, if any, by executing function `genVote()`.

Sending of a vote consists in updating global meta variables `gsubV` and `gidV`, with the values of the corresponding local variables, as described in the GMP specification shown in chapter 2. The value of the `Mset` sent in the vote message, can be found directly in meta array variable `Mset`.

After that step, the `Scheduler` signals on broadcast channel `startProc` triggering the processing of the SA-phase. If a node is not in `SAreq`, i.e. the node has not observed any event that might lead to the change of the group membership, then it does not send any message in this phase and ignores all messages sent by other nodes, moving directly to location `B4FD_phase`. On the other hand, nodes in `SAreq`, must process the votes they receive. This is done in function `processSAphase()`, which implements the processing of the SA-phase of the GMP. In this processing, a `Node` uses its own state variables, such as the `MSet` and the `MSET` sets, the meta variables with the votes sent, the `SAreq` set, which indicates which votes were actually sent, and the global variable `Joining`, which indicates which of those votes were sent by nodes joining the group.

In the absence of faults, this function is executed only when a node requests to join the group, and the outcome is the update of the state variables associated with the group, namely `MSET, gid` and `gsub`. However, in the presence of faults, a node may find out that its view of the group membership is different from that of the majority, or even that it is not able to determine the view of the majority. Under our fault assumptions, both cases indicate the occurrence of a fault in the `Node` and the protocol determines that the node must halt. To allow testing of this outcome, a node removes itself from its `MSET` if it must halt.

Thus, when the Scheduler signals on the `terminateCycle` broadcast channel, if a node is not a member of its `MSET` it moves to location `HLT` and is added to the set of nodes that can join the group(`Joinable`). Otherwise, the node moves to state `B4FD_phase` and becomes ready to execute the FD-phase again.

4.3.2 Modeling of Faults

In the previous section we have presented an Uppaal model for the GMP in the absence of faults. In this subsection we describe how we model faults. The basic idea is to use *fault schedules* for each phase of the GMP execution. These fault schedules specify the fault events, i.e. send faults and receive faults, that each node will experience in the corresponding phase.

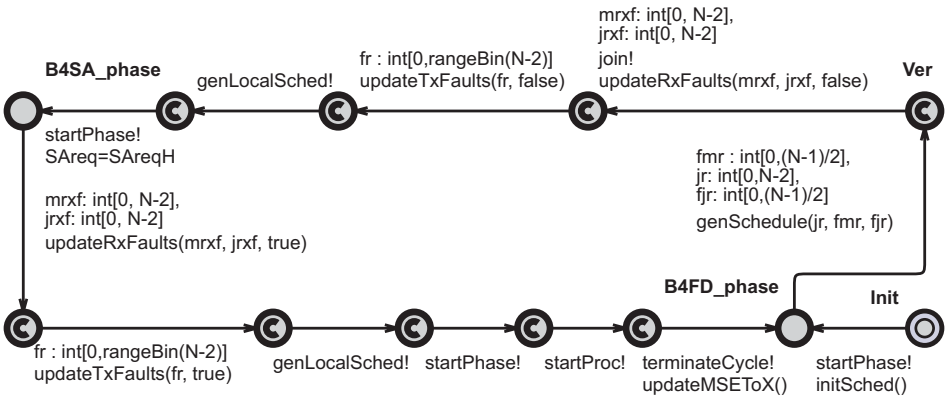


Figure 4.4: Scheduler automaton for the model with faults.

Generation of the fault schedules is done at two levels. At a system-wide level, the `Scheduler` determines which nodes have send faults and which nodes have receive faults. At a local level, each `Node` designated to have receive faults generates its own receive faults, i.e. determines on which messages it will experience a receive fault. Generation of the global fault schedule by the `Scheduler` makes it easier to ensure that the GMP fault assumptions are not violated. On the other hand, the generation of local receive fault schedule by nodes leads to a more structured approach and makes it easier to change the receive fault assignment policy.

Fault schedules are implemented as sets. The following state variables were added with that purpose:

```
// Global state variables
Set Faulty;
Set TxFaults;
Set RxFaults;
// Per node state variables - Scheduler needs to access them
Set NFaultsFD[N];     // Faults in the FD-phase
Set NFaultsSA[N];     //Faults in the SA-phase
```

In order to generate all fault schedules of interest in a compact way, we use select labels. The random integers generated by these labels are used either as the number of nodes that fail, or as an encoding, with one bit per element, of a set of nodes that fail.

Figure 4.4 shows the `Scheduler` automaton that generates the fault schedules as described above. To generate the global fault schedule, the `Scheduler` automaton determines which nodes fail in a GMP execution, at the beginning

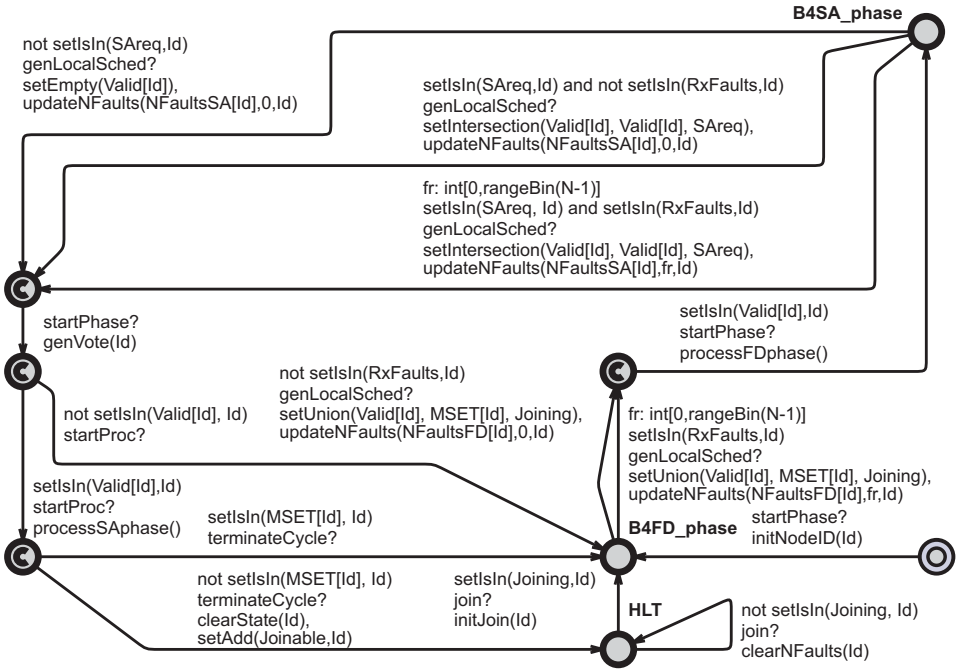


Figure 4.5: Node automaton for the model with faults.

of each execution. I.e., now function `genSchedule()` not only determines which nodes will attempt to join the group, but also which nodes may fail. Then, before starting each phase, the `Scheduler` selects which nodes experience send faults and which nodes may experience receive faults.

Figure 4.5 shows the new `Node` automaton. Like in the `Scheduler` automaton, the structural changes concern only the generation of local schedules at the beginning of each phase. In addition, we had to make some changes to both `processFDphase()` and `processSAphase()`, because faults will affect which messages are received, and consequently processed, by each node.

We terminate our description of the modeling of faults with a reference to crash-faults, a kind of fault the GMP is supposed to tolerate but that we have ignored so far. It turns out that the model we have developed for receive and send faults subsumes the case of crash-faults. A crash-fault is a fault in which a node enters a halting state and takes no further action. To the other nodes an execution with such a fault is equivalent to an execution in which a node does not send any message, from some instant onwards. This behavior can be exhibited by this model, indeed a node that has send and receive faults from some point of its execution onward, moves to the `HLT` state and stays there indefinitely behaves

like a crashed node.

4.4 Limiting the Size of the State Space

Modeling of faults makes the model inherently more complex. For example, in our fault model we consider that a node may fail in one of three ways: by crashing, by omitting to send a message or by omitting to receive a message. Given that each GMP execution has 2 phases that are not identical, each node may fail in 25 different ways. (Actually, this number is a lower bound as it considers only whether or not a node experiences at least one receive fault in a phase, disregarding the number of receive faults and on which messages these faults occur.)

In principle, one might argue that the number of receive faults in each phase is irrelevant and, in addition, that it does not matter in which phase of the GMP execution one node has a given fault. It turns out that none of these observations hold for the GMP, as some subtle fault scenarios that we described in chapter 2 illustrate. We call these scenarios *masked faults*, as they correspond to cases in which a receive fault of one node is masked by another fault in the same or in the subsequent cycle. We have identified the following 3 cases:

1. SF_n in FD-phase; $RF_{m,n}$ in SA-phase.
2. $RF_{m,n}$ in SA-phase; SF_n in FD-phase.
3. $RF_{n,o}$ in FD-phase; $RF_{m,n}$ in SA-phase.

where SF_n means a send fault in node n, and $RF_{m,n}$ means the receive fault in node m on a message sent by node n. Thus, in the first two cases, the receive fault in m is effectively masked by a send fault in n, and therefore node m is not removed from the group. In the third case, the receive fault by m is masked by n's receive fault, and therefore node m is not removed from the group.

It is clear that if, e.g. in case 1 or 2, node m had a receive fault on all the messages sent, no masked fault would occur. This is because, for that to happen, all senders would have to fail, but such a fault scenario violates the fault assumptions of the GMP. It is also clear that if, e.g. in case 3, node m had its fault in the same phase as node n, then it would detected as faulty by the good

nodes. These examples show that general principles [47] for model checking fault tolerant systems must be applied with care.

Still we can apply some general techniques to reduce the size of the state space. This is particularly important for model-checking the GMP because the state kept by each node is relatively large and we want to verify the protocol for configurations with a sufficiently large number of nodes to exhibit interesting behavior. We have found the following three techniques particularly useful in reducing the size of the state space of the model: 1) symmetry reduction; 2) priorities; 3) synchronization.

4.4.1 Symmetry Reduction

This technique is particularly effective for distributed algorithms, such as the GMP, where a set of identical components executes the same algorithm. Essentially, the idea is to take advantage of the fact that, for the GMP, it is not relevant which nodes are members or which of those are faulty, but rather how many nodes are group members or how many of those are faulty.

Uppaal itself provides support for symmetry reduction through *scalarset types*. They provide a way to tell the model checker about symmetries. Scalarset types can be seen as a bounded integer type with restricted operations, namely assignment and equality testing. Scalars may also be used as indices of arrays. Because of these restrictions, we found no clean way to model the GMP without using arrays indexed by scalarsets and whose elements contain scalarsets. However, for models with arrays indexed by scalarsets that contain elements of scalarsets the algorithm used by Uppaal for symmetry reduction is unlikely to provide any benefit [46]. Some preliminary experiments with simplified models with patterns of usage of scalarsets that would allow to model the GMP confirmed that. We have therefore implemented symmetry reduction directly in the model.

As stated above, for the GMP what is important is the number of nodes that fail, and not which nodes fail. Therefore, to eliminate "redundant states", the fault schedules are generated such that faults are assigned to nodes with higher identifiers. For example, in a configuration of 5 nodes, N0 to N4, in an execution where N4 is in location HLT and the remaining nodes are members of the group, the GMP tolerates one additional fault. In that event, which is generated randomly, the fault will be always assigned to node N3. This eliminates

states where each of the remaining members fail instead of N3. Note that this technique does not eliminate all the redundant states. E.g., if instead of node N4 the node in HLT were node N3, in the event of a fault, that fault will be assigned to node N4. Although, such a state is equivalent to the state above, basically it can obtained by swapping the states of N3 and N4, our model is not able to eliminate it.

However, the number of these redundant states can be reduced, by adopting a consistent policy to select the nodes that join: the model generates randomly the number of nodes that will join, and then selects those that can join with lower identifiers. This policy, together with the one described in the previous paragraph, makes it highly probable that the group is composed by the members with lower identifiers, and that only nodes with higher identifiers will fail. In particular, it ensures that nodes N0 and N1 will never fail, whatever the number of nodes in the system, because the GMP requires at least two nodes, and the generation of faults in the model is such that it does not violate the GMP fault assumptions.

Selection of the faulty nodes that experience receive faults follows the same approach as that of the assignment of faults. For example, if nodes N4 and N5 are both selected as faulty, and the model determines randomly that one of them will have a receive fault in the FD-phase, then node N5 will be selected. On the other hand, selection of faulty nodes that experience transmission faults is done in a completely random way using select labels with a range from 0 to $2^{(N-2)} - 1$. The number selected is then used as an encoding of a set of N-2 elements and the latter is intersected with the set of faulty nodes. The reason for generating transmission faults in a completely random way is to allow all relevant combinations of send and receive faults. This approach is particularly effective for N smaller than 7, i.e. for at most 2 faulty nodes, in that it generates only 2 redundant pairs of receive faulty and send faulty node sets, in a total of 17, but the effectiveness of this policy decreases as the N increases.

Finally, we have also tried to explore symmetry reduction in the local receive fault schedules of nodes that are supposed to experience receive faults. Rather than generate completely random fault schedules, the receive fault schedules are only random with respect to messages sent by faulty nodes. With respect to messages sent by non-faulty nodes, we ensure that nodes will loose only the mes-

sage sent by N0, which is guaranteed to be always a group member as explained above. This policy has two additional benefits. First, it ensures that all faulty nodes "collude" to remove a non-faulty node. Second, it does not eliminate fault schedules with multiple and reciprocal faults that may lead to subtle protocol behaviors. Again, this approach is particularly effective for N smaller than 7, in that it prevents redundant states, but for larger values of N redundant states will be generated.

4.4.2 Priorities

Another well-known technique to reduce the state space size of the model is to remove uninteresting interleavings. For example, in the GMP the order in which nodes execute the processing pertaining to each phase is not relevant. I.e., it does not matter whether node 0 executes before node 1 or the other way around. Uppaal allows reducing these interleavings by means of *process priorities*. Using this feature, one can specify the order by which automata will take transitions when more than one transition is enabled at the same time, essentially inhibiting the transitions of automata with lower priority.

4.4.3 Synchronization

However, the use of priorities does not remove all the intermediate states. For example, considering that the higher the `Id` of a node the higher its priority, although node 1 will always take a transition before node 0, if both of them have enabled transitions, the intermediate state that occurs after node 1 taking its transition and before node 0 takes its transition will still be considered. One technique to remove these uninteresting states is to add synchronization, as we have done with the `startProc` and the `terminateCycle` broadcast channels. By adding the additional synchronization, all nodes take the transition simultaneously, and none of the otherwise intermediate states will be considered (unless it occurs in some other way).

It should be noted that although removing intermediate states is interesting for the sake of reducing the size of the state space, it may have adverse effects on the time for model checking. For example, we might reduce the size of the state space for about 30% for 5 nodes, by generating the schedules for send faults

and receive faults on the same transition in `Scheduler`. However, verification of the properties described in the next section with such a model takes more than twice the time. The reason is that although the number of states is smaller, the number of transitions in the model is much larger, and therefore Uppaal spends a lot of time testing transitions that in the end lead to the same state.

4.5 Correctness Properties

In GMP specification, we have stated the Group Membership Problem in terms of the set of group members (M-SET) maintained by every node, and specified two properties:

Agreement: All non-faulty group members compute the same M-SET.

Validity:

1. A faulty node will be removed from the M-SET of a non-faulty group member in a bounded time interval;
2. A non-faulty node attempting to be reintegrated will be added to the M-SET of a non-faulty group member in a bounded time interval.

And we have also stated that the GMP ensured a bound of two TDMA cycles for removing a faulty member and one TDMA cycle for a non-faulty node to be reintegrated. The latter bound considers that the delay is measured starting on the instant the node sends a joining request.

Uppaal allows the specification of the properties that a model must satisfy in a simplified version of CTL [44]. In particular it allows to specify safety properties like Agreement and Validity, using the $A\Box$ modal operator as follows:

Agreement: `A[] Sched.B4FD_phase imply Agreement()`

Validity1: `A[] Sched.B4FD_phase forall(i: int[0,N-1]) FDdelay[i]<3`

Validity2: `A[] Sched.B4FD_phase imply Validity2()`

where `Agreement()`, `Validity2()` are predicates that check the corresponding properties, and are as follows:

```
bool Agreement() {
    return
        forall (i: int[0,N-1])
            ((setIsIn(MSETo, i)
              and not setIsIn(MSEToF,i))
                imply MSET[i]==MSETo);
}
```

```
bool Validity2() {
    return
        forall(i: int[0,N-1])
            setIsIn(Joining, i) imply
                ( setIsIn(MSETo, i)
                  or not setIsEmpty(NFaultsFD[i])
                  or not setIsEmpty(NFaultsSA[i]));
}
```

Essentially, these expressions state that the corresponding properties hold after every execution of the GMP.

Actually, both Validity properties are bounded liveness properties and could have been checked using the leads to operator ($\rightsquigarrow$), also supported by Uppaal. However, we found it more efficient to augment the model with some state variables and with the appropriate code. This augmentation concerned only Validity1. In particular, we added array `FDdelay` of integer variables that counts the number of GMP executions it takes for good members to remove faulty members from the group.

4.6 Verification Results

We verified both Agreement and Validity for configurations with three, four and five nodes. Table 4.1 shows the number of states stored and visited, as well as the time taken in checking each of the properties presented in the previous paragraph. For the case of 5 nodes, we present also the results we have obtained using an option provided by Uppaal that reduces the memory requirements by not storing committed states, i.e. states in which at least one automaton is in a committed location. For the latter case, the table shows both the number of states stored and the number of states explored. When no memory reduction technique is used, only one value is shown because both numbers are equal.

The figures clearly show that the state space size increases exponentially with the number of nodes in the system, in spite of our efforts to explore symmetry at the level of the model. Although, the use of Uppaal's memory reduction option allowed us to reduce the memory requirements for about one order of magnitude,

Table 4.1: State Space Size (in thousand states) and approximate time execution for the different models and properties verified.

Model		Agreement		Validity1		Validity2	
No.Nodes	Mem.Red.	No.States	Time(s)	No.States	Time(s)	No.States	Time(s)
3	N	5.3	0.5	5.3	0.5	5.6	0.6
4	N	220	56	220	56	221	57
5	N	14,237	28,920	14,232	29,640	14,870	30,060
5	Y (stored)	1,367	51,780	1,367	56,200	1,389	54,000
	Y (explored)	67,324		67,276		69,094	

the verification of these properties for configurations with more than 5 nodes leads to an exhaustion of memory resources.

Nevertheless, to be able to check the GMP for 5 nodes gives us a high confidence level in its correctness, because with 5 nodes we are able to generate rather subtle fault scenarios, such as the masked faults, that arise with the simultaneous fault of two nodes, which may be either members of the group or attempting to join. Although checking the correctness of the GMP for 7 nodes would provide an even higher confidence, because with that many nodes we could consider scenarios with 3 simultaneous faults, we believe that the change from 2 to 3 nodes does not lead to very different fault scenarios. Furthermore, to be able to verify the correctness of the GMP for a higher number of nodes in Uppaal is likely to require the use of *abstraction*, another well known technique of addressing the state space explosion problem.

However, the use of abstraction usually leads to models that are significantly different from the system being checked and consequently the level of confidence will be lower than if a model like the one we have developed were used. Indeed, our model includes an implementation of the GMP, except for the use of communication primitives such as `send` or `receive`, i.e. we abstract the communications layer. Given that Uppaal uses a syntax very close to C, it is straightforward to convert that model to a C implementation of the protocol.

Including an implementation of the GMP in the model allowed us to find a *bug* in the implementation outlined in Chapter 3 that is related to the fact that the number of group identifiers in an implementation must be bounded. In the GMP, shown in chapter 3 (Figure 3.5), the group id is incremented in step 9 of the SA-phase. At the level of abstraction of the specification, we considered that

this variable is unbounded. However, in an implementation, as well as in model-checking, this variable has to be bounded. In GMP specification, we have argued that an integer with a range from 0 to 3 is enough, and stated that the GMP did not require any other change. Although we were right with respect to the minimal range of group ids, we were wrong with respect to the need to change the GMP. The problem is in steps 1 to 3 of the SA-phase, where the maximum group id is determined and is then used to compute the majority set. With bounded group ids, these must be recycled, and therefore an id of 0 may be larger than an id of 3. Thus determining the maximum id is not straightforward, especially because joining nodes always send votes with a group id of 0, and faulty nodes may send any value, if they do not execute the SA-phase a number of times. The group ids of joining nodes can be easily fixed by ignoring the group id sent in their votes. The group ids sent by faulty nodes can be filtered by group members, taking into account the state of the GMP. However, joining nodes lack this state, and may compute a wrong group id. This may lead to an erroneous computation of the majority set in step 3. One way to fix this problem is to change the GMP so that joining nodes check that the majority they compute is consistent with the votes received (every node must consider itself a group member, and non-joining group members must agree on the group id). If it is not, they will cycle through all the group ids until a consistent majority is found, or they have tried all the ids. In the latter case, the joining node will consider itself faulty, and will halt.

4.7 Conclusion

We presented a formal verification of GMP, a protocol designed to provide a Group Membership Service for FlexRay, a minimalist middleware for the development of safety critical applications, that is likely to become the *de facto* standard bus for automotive applications.

The results obtained show that the GMP satisfies its specification for configurations of up to 5 nodes, providing us further assurance on its correctness. The fact that the model developed includes an implementation of the GMP contributes significantly to our confidence in its correctness, but also limits the number of nodes of the configurations that we are able to check. However, we strongly believe that we did the right choice, as it allowed us to detect a bug in the outline of an implementation we have proposed. The alternative would be to use abstrac-

tion, which might lead to a model far removed from the GMP, and a doubt of whether the abstraction used was correct would always linger.

Chapter 5

Reliability Analysis of GMP

This chapter presents a reliability analysis of the Group Membership Protocol (GMP) by means of probabilistic model checking. It includes a detailed description of the discrete-time Markov chains and PRISM models used in that analysis. Furthermore, it describes the experiments carried out and discusses the results obtained. The content of this chapter essentially comprises the text published in [11].

5.1 Introduction

The GMP is intended to be used as a general service by safety-critical applications, because of that it is important to ensure its correctness, ideally through a proof. However, any proof relies on fault assumptions and it ensures that the protocol behaves correctly only as long as the fault assumptions hold true. Therefore the reliability evaluation of the protocol is of paramount importance.

In this chapter we evaluate the reliability of the GMP by equating it to the reliability of the assumptions made in its proof, i.e. the probability of these assumptions being true, an approach that, was first proposed by Latronico, Miner and Koopman in [18]. The fault model used in this study includes both permanent and transient faults affecting both nodes and channels. In addition, we consider two classes of common-mode transient communication faults, faults that partition the network for the duration of the one message and error bursts.

In order to carry out this study we have developed several discrete-time Markov chain (DTMC) models, which were evaluated using PRISM [48], a probabilistic symbolic model checker. The results show that the GMP protocol can achieve reliability levels required by safety-critical applications, even for configurations of a small number of nodes and common-mode faults, by introducing a small modification to the original protocol.

The remaining of this chapter is structured as follows. In Section 5.2, we describe the GMP maximum fault assumption (MFA). The fault model and the use of discrete-time Markov chains with PRISM to evaluate the reliability under this fault model are shortly described in Section 5.3. Then in Section 5.4 we present the experiments we carried out in this study, and analyze their results. . In Section 5.5 we conclude with a summary of the main results.

5.2 Maximum Fault Assumption

The majority set function is the core of the GMP. To mask faults and to ensure agreement among non-faulty group members, no more than half of the group members may fail between consecutive executions of the GMP. This condition is the maximum fault assumption (MFA). If it is violated, there is no guarantee that the GMP will satisfy its specifications.

5.2.1 Goals of the Reliability Evaluation

The reliability evaluation reported here started out with the general goal of quantifying the reliability of the GMP. The methodology we follow is to determine the reliability of the assumptions made in its proof, i.e. the reliability of the MFA.

From this general goal, more specific questions arose as the study progressed. The following, is a summary of the questions to which we try to answer:

1. What are the main factors that affect the GMP reliability?
2. What is the effect of using diagnosis period multiple of the communication round?
3. How do common-mode faults affect the GMP reliability?

5.3 Model

5.3.1 Fault Model

In arguing the correctness of the GMP in [7] we assumed that a node might fail by crashing or by omitting either to send or to receive messages. However, because of well-known results on the impossibility of agreement in the presence of communication faults [39], we had to assume that the communication channels were reliable. For reliability evaluation studies, like this one, these results are not important. Thus, we now consider a more realistic fault model, by assuming that communication channels may also fail by crashing or by omitting to deliver messages, e.g. as a result of noise. Furthermore, we assume that these faults may be either permanent or transient. E.g., a node might fail to receive a message because it has not enough buffer space, but recover later.

Finally, in addition to single message faults, we consider two other types of transient fault in channels: common-mode faults and error bursts. Common-mode faults partition the network for the duration of one message, leading to what some authors [49] call strictly omissive asymmetric faults, i.e. a scenario in which some nodes receive a message correctly, whereas other nodes receive no messages. (We assume that the error detection codes used by communication protocols are strong enough to detect virtually all communication errors.) Communication error bursts are caused by long duration electromagnetic interference (EMI) bursts that make communication all but impossible while it lasts.

5.3.2 Models

In this section, we present the different models we have developed to evaluate the reliability of the GMP. Because the GMP is round-based and the probabilistic evolution of the system depends on the current state of the nodes and of the network, we chose to use discrete-time Markov chains (DTMC) to model its reliability. Indeed, the use of DTMCs allows us to associate the passing of time intervals such as rounds or slots of a TDMA cycle with state transitions, making it easier to model the reliability of our fault assumptions.

All the models comprise one DTMC per node, and one DTMC for the communications network. So, a system composed by n nodes needs at least $n + 1$

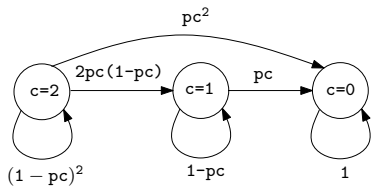
DTMCs. Each of the models for SOA communication faults and communication error-burst faults includes one additional DTMC to generate the corresponding faults.

One of the main problems when evaluating this type of systems is the state-space explosion [50], i.e. the size of the model grows exponentially with the size of the modeled system. (E.g. n concurrent DTMCs, each one with m states, can lead to a model with m^n states.) Consequently, the amount of memory necessary to store the model and the time necessary to compute the reliability increases dramatically, raising severe computational problems. To overcome these problems it is necessary to select an evaluation tool that is able to cope efficiently with them. We chose the PRISM tool [48], because it implements symmetry reduction techniques [51], which enable a significant reduction of the state-space when models have symmetry. This is the case for the GMP models given that all nodes have the same behavior.

5.3.2.1 PRISM

PRISM [48] is a probabilistic model checker that supports Discrete-Time Markov Chains (DTMC), Continuous-Time Markov Chains (CTMC) and Markov Decision Processes (MDP). The system to be analyzed is defined using a high-level state-based language and from this description the tool constructs a probabilistic model. Model evaluation requires the specification of the property to evaluate. PRISM allows the specification of properties in PCTL (Probabilistic Computation Tree Logic), a probabilistic extension of CTL (Computation Tree Logic), a temporal logic.

Therefore, in the subsequent subsections, we focus on modeling of DTMCs with PRISM by means of examples. We begin with the presentation of the DTMC for the communication network. This DTMC is used, except for minor and obvious changes, in all models of the GMP. Next we describe the model for the GMP with single round diagnosis period (DP), which can be seen as the base model. After that, we describe the model for the GMP with multiple round DP. Finally, we present the models for SOA communication faults and communication error-burst faults, in that order. Because PRISM uses a somewhat unusual language to define the models, we complement the model's description with conventional



(i)

```
 1: dtmc
 2:   // Parameters to initialize at invocation time
 3:   const int NC;    // Number of channels
 4:   const double pc; // Channel failure probability
 5: // Network model: supports single and duplicated
 6: //      channels, depending on input parameter NC
 7: module Net
 8:   c: [0..2] init NC;
 9:   [] c=2 -> 2*pc*(1-pc):(c'=1) + (pc*pc): (c'=0)
                + (1-pc)*(1-pc):(c'=2);
10:   [] c=1 -> pc:(c'=0) + (1-pc):(c'=1);
11:   [] c=0 -> true;

endmodule
```

(ii)

Figure 5.1: DTMC of a communications network comprised of a duplicated channel. It assumes that each channel has only permanent faults and that they fail independently. (i) Transition diagram. (ii) PRISM model.

transitions diagram in some cases. This approach was adopted for the sake of ease of understanding of the models.

5.3.2.2 Communication network DTMC

The communication network interconnects all nodes and is composed of replicated broadcast channels. All the channels have the same characteristics. For practical reasons we limit the maximum number of channels to 2. This corresponds to the most common scenarios found in real applications.

Since the GMP maps the loss of a message due to a transient fault on either the sender or the receivers of that message, we chose to represent this event in the node DTMC (described below in this subsection). Therefore, the network DTMC presented here only includes permanent faults in the channels.

Figure 5.1 (i) shows the DTMC for the communication network. It is assumed that each channel fails independently. Each DTMC state is characterized by the value of state variable `c`, the number of operational channels, i.e. channels that have not failed permanently.

Thus, transitions model the permanent failure of the channels and are labeled with the probability of occurrence of this event during a time-step. We use `pc` to

denote this probability. The model for a network that does not have replicated channels can be easily obtained from this one by removing state `c=2` and the corresponding transitions.

Figure 5.1 (ii) shows the PRISM model of this DTMC. The `dtmc` keyword in line 1 indicates that this is a DTMC model. A PRISM model comprises one or more *modules*. In this case, we consider only one module named `Net`, which has local state variable `c`, which counts the number of operational channels. The behavior of the `Net` module is described by three guarded commands in lines 9 to 11, each of which starts with the string "`[]`". The guard, a predicate to the left of the string "`->`", specifies the state of the system to which the corresponding command applies. The right hand side of a command specifies the possible transitions in this state, each transition being separated by the character '`+`'. In the case of a DTMC, a transition is specified by its occurrence probability in a time-step, represented to the left of the character '`:`', and the new state, represented as a set of updates of model variables to the right of that character. The character '`'`' is used to denote the value of the state variable after the transition is taken. For example,

```
[] c=1 -> pc:(c'=0) + (1-pc) : (c'=1)
```

is interpreted as follows: if `c=1`, then with probability `pc` the next state will be `c=0`, and with probability `(1-pc)` the next state will be `c=1`. As already mentioned, the property to evaluate is specified in PCTL. For example, the following property:

```
P=? [true U<=N c==0]
```

can be used to compute the probability of failure of both channels of the network in a time interval of `N` time-steps.

5.3.2.3 Single-round diagnosis period model

This model is composed of the communication network DTMC described above and of one DTMC per node that we now describe. All node faults in our fault model have the same effect on the protocol: the faulty node will be removed from the group membership. However, it is important to distinguish between permanent and transient faults. In the former case, the node will be permanently removed from the group, whereas in the latter case it may rejoin, depending on whether or not further faults occur.

To represent this behavior, we use a DTMC with six states (Figure 5.2):

Good (Gs) A node in this state is a group member.

Permanent Faulty (Ps) A node in this state has failed permanently and is not a group member any more.

Transient Faulty (Ts) A node in this state has failed temporarily and is not a group member.

New (Ns) A node in this state is a newly joined member; it was not a member in the previous DP.

Convicted (Cs) A node in this state has been permanently faulty for more than one DP.

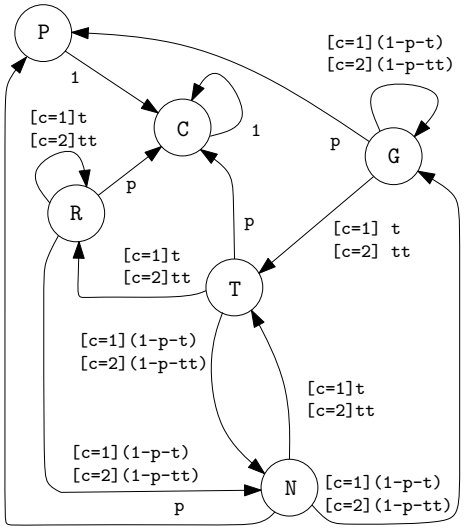
Recidivist Transient Faulty (Rs) A node in this state has experienced transient fault in the last DP, when it was not a group member any longer.

The time-step chosen for this model is the GMP execution, which is the same as the communications round.

In summary, group members are either in state `Gs` or `Ns`. Given that transitions occur only at the end of each cycle the following properties can be easily established:

- Nodes in states `Ps` or `Ts` were members of the group in the previous time step, when they failed.
- Nodes do not remain in states `Ps` or `Ts` for more than one time step.
- Nodes in states `Cs` or `Rs` have not been group members for more than one time step.

As usual, each transition is labeled with the probability of occurrence of the transition in a time-step. E.g. the transition from state `Gs` to state `Ps` is labeled with the probability p of a node failing permanently during a diagnosis period. In some transitions, we use labels of the form $[pred]prob$, where $pred$ is a predicate on the state of the model and $prob$ is a probability. If $pred$ is true, the probability of the transition being taken is $prob$, otherwise it is zero. Note that this notation



(i)

```
 1: // Node model - states
 2: const int Gs = 0; // Good node (member)
 3: const int Ps = 1; // Perm. faulty node
 4: const int Ts = 2; // Transiently faulty
 5: const int Ns = 3; // Newly joined node
 6: const int Rs = 4; // Recidivist transient
 7: const int Cs = 5; // Convicted node
 8: module N1
 9:   s1: [0..5] init Gs; // Initially Good
10:   // Variables for verifying properties
11:   g1: [0..1] init 1;
12:   p1: [0..1] init 0;
13:   t1: [0..1] init 0;
14:   [round] s1=Gs & c=2 ->
                 p:(s1'=Ps)&(g1'=0)&(p1'=1)
                 + tt:(s1'=Ts)&(g1'=0)&(t1'=1)
                 + (1-p-tt):(s1'=Gs);
15:   [round] s1=Gs & c=1 ->
                 p:(s1'=Ps)&(g1'=0)&(p1'=1)
                 + t:(s1'=Ts)&(g1'=0)&(t1'=1)
                 + (1-p-t):(s1'=Gs);
16:   ...
17: endmodule
18: // Use module renaming for other modules
19: module N2 = N1 [s1=s2, g1=g2, t1=t2, p1=p2]
20: endmodule ...
```

(ii)

Figure 5.2: DTMC of a node.(i) Transition Diagram. (ii) Segment of the PRISM model, comprising part of one of the node modules, `module N1`.

allows only a more compact representation of a DTMC, i.e. allows it to be represented with fewer states and transitions.

As already stated, both transient communications faults and transient node faults are modeled in the node-DTMC. In the case of a single round diagnosis period, a node will be detected as faulty by the other nodes, if either its heartbeat or its vote are lost.

A message will be lost, if none of the network channels delivers it. Therefore, for a message not to be delivered in the event of a transient communication fault, all operational channels must have a transient communications fault affecting that message. We model this dependence on the network state by labeling transitions with predicates. For example, from state `Gs` to state `Ts` the node-DTMC includes

two transitions, each of which with a predicate on the state of the network-DTMC, i.e. on the number of operational channels. The probabilities associated with each transition, `t` and `tt`, denote the probability of at least one of the heartbeat or of the vote messages being lost in a network with one and two operational channels respectively.

In the computation of these probabilities (`t` and `tt`), we assume that transient communication faults in different channels are independent events and that a transient communications fault in one channel is characterized by its bit-error ratio (BER) and leads to the loss of only one message. I.e., we assume that if a message is affected by at least one bit-error, then it will be lost and that a single communication fault does not affect more than one message; we consider the possibility of burst errors in a model below.

Figure 5.2 (ii) shows a segment of the PRISM model, with part of one of the node modules, `N1`. This model can be easily derived from the transition diagram in Figure 5.2 (i). Nevertheless, we would like to make three remarks. First, variables `g1`, `p1` and `t1` defined in lines 11, 12 and 13, are used, together with corresponding variables of other nodes, in the property for reliability computation as explained below. Second, synchronization label `round` is used in all guarded commands of all modules of the model. This ensures that all modules will change their state synchronously as time advances, and allows for a simple specification of the behavior of the system in terms of the behavior of the individual modules. Furthermore, it reduces the number of states of the models, by removing states that would arise from different interleavings of the nodes' transitions. Third, module `N2` is defined in lines 19 and 20 in terms of module `N1`, by renaming the variables used in the latter. (Other node modules can be defined similarly.) This is particularly useful in modeling systems such as the GMP that comprise identical components.

The reliability of the GMP after the execution of `N` time-steps (diagnosis periods) can be computed using the following PCTL property:

$$1 - (P=?[\text{true } U<=N\ (+\ n_G <= n_T + n_P\ ) \mid (c=0)]\ )$$

where n_G, n_T and n_P are the number of nodes in state `Gs`, `Ts` and `Ps`, respectively, and are computed using variable states `gi`, `ti` and `pi`, where `i` is an integer that ranges between 1 and the number of nodes in the model. E.g. for a 3 node

configuration n_G=g1+g2+g3. The proposition c=0 reflects the fact that the GMP fails, if all the communication channels have failed permanently.

5.3.2.4 Multiple-round diagnosis period model

This is a model for the extension of the GMP that uses a diagnosis period longer than one round. The purpose of this extension is to make the GMP less sensitive to transient faults. Of course, improving the reliability by extending the diagnosis period is possible only if the GMP "forgives" nodes that are affected by transient communication faults during the extended diagnosis period. Thus, the GMP cannot detect a node as faulty as long as in a diagnosis period the number of frames it does not receive from that node is smaller or equal than a maximum value that we call *conviction threshold*.

This change implies that the loss of a heartbeat is not equivalent to the loss of a vote: whereas the loss of a vote may affect the outcome of majority voting, the loss of a number of heartbeats up to the conviction threshold does not. To model these different effects of transient faults in the GMP, it is more appropriate to use a DTMC whose time-step is the GMP phase, rather than the GMP execution.

The DTMC for the network is essentially the one described above, although it now has to take into account the phases of the protocol. In the new network DTMC, permanent channel faults always occur in the FD-phase, with probability p_c adjusted to the length of the GMP execution. This leads to a slightly pessimistic estimation of the failure probability than if permanent faults in the SA-phase were not included in the FD-phase, but it simplifies the model, as it reduces the number of transitions.

On the contrary, the DTMC for the nodes requires more extensive changes. The model for the FD-phase is essentially that for a single-round diagnosis period, with all fault probabilities adjusted. Permanent node faults have only to take into account the longer time-step, i.e. they are modeled in a way similar to that of permanent channel faults. The transient fault probabilities, t and tt, have to take into account the number of heartbeats, i.e. the duration of the diagnosis periods, and the conviction thresholds. Rather than computing these values using a mathematical expression, we have developed an auxiliary model that is described at the end of this subsection.

Because node permanent faults and transient communication faults affecting heartbeats are modeled in the FD-phase, modeling of the SA-phase needs only to consider transient communication faults affecting votes, as shown by the following segment of module `N1`:

```
[round] s1=Gs & c=2 & phi=1 -> fl*fl:(v1'=1) + (1-fl*fl):(v1'=0);
[round] s1=Gs & c=1 & phi=1 -> fl:(v1'=1) + (1-fl):(v1'=0);
[round] phi=1 & c!=0 & s1!=Gs -> (v1'=0);
```

Where `phi` indicates the protocol phase, i.e. is 0 for the FD-phase and 1 for the SA-phase, `fl` is the probability of a channel losing a frame because of a transient communication error, and `v1` memorizes the loss of a vote and is used, together with the corresponding variables of other node modules, to compute the number of lost votes n_V.

The expression used to compute the reliability of the GMP can now be expressed as follows:

```
1 - (P=? [true U<=N (phi = 0) & (n_G − n_V <= n_T + n_P)|(c=0))])
```

I.e., the GMP fails either if all communication channels fail or if in an execution the number of votes by good members that were not lost is insufficient to ensure a majority, i.e. is not larger than the number of members that have failed either transiently or permanently.

Computation of `t` and `tt` To compute `t` and `tt` we developed another DTMC that models the loss of frames in a network as a result of transient communication faults and used PRISM to solve that model. Like the other models, this one is also parametric. However, it is executed off-line and the results of its executions, i.e. the values of `t` and `tt`, are used as input parameters to the node DTMCs.

Figure 5.3 shows the transition diagram of that DTMC. Variable `nt` represents the number of transient faults that occur in a diagnosis period, and `NC` is a parameter whose value is the number of communication channels of the network. Variable `fl` is the probability of a channel losing a frame because of a transient communication error, and is computed based on the bit-error ratio (BER) and the frame size. This model computes `t` or `tt`, depending on the value of parameter `NC` being 1 or 2, respectively, using the PCTL property:

```
P=? [ true U<=DP nt > Thr ]
```

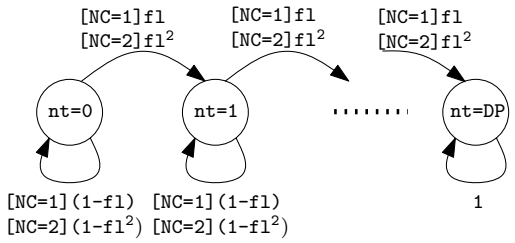


Figure 5.3: Auxiliary DTMC used for the computation of `t` and `tt` of the node models for multiple round diagnosis period.

I.e., the probability of a node being diagnosed as faulty as a result of communication faults, is the probability of the number of messages it sends during a diagnosis period that are lost to exceed the selected threshold, Thr.

By changing parameters `DP` and `Thr`, it is possible to compute the values of `t` or `tt` for different configurations of the GMP, and, by plugging these values into the node DTMC described above in this section, it is possible to model the GMP for these configurations. The whole process can be easily automated through some scripts.

5.3.2.5 Strictly omission asymmetric faults

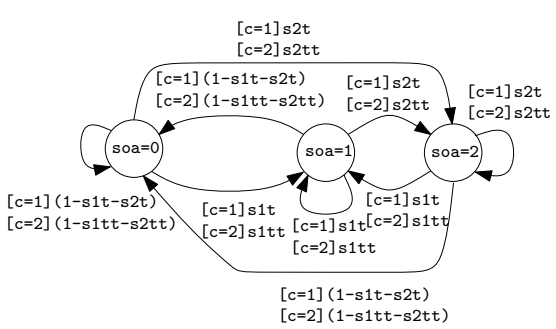
Strictly omission asymmetric (SOA) transient communication faults are faults in which some nodes receive a message correctly, whereas other nodes receive no message, thus partitioning the network for the duration of one message. This type of faults corresponds to noise that is spatially localized and affects only some of the nodes rather than all nodes, as we considered in the models presented above.

The main challenge in designing a model for SOA faults is the size of the design space and the computational practicality of the models. The two key issues that we had to consider were:

1. The time-step of the model;

2. The effect of a SOA fault on the behavior of the nodes.

For efficiency reasons, in terms of both execution time and memory, we have chosen to use in this case the diagnosis period (DP) as the time-step of the DTMC model. A consequence of this choice is that it is not possible to model SOA faults



(i)

```
module SOA
  soa: [0..2] init 0; // SOA faults in DP
  [round] c=2 -> s2tt:(soa'=2)
                 + s1tt:(soa'=1)
                 + (1-s2tt-s1tt):(soa'=0);
  [round] c=1 -> s2t:(soa'=2)
                 + s1t:(soa'=1)
                 + (1-s2t-s1t):(soa'=0);
  [round] c=0 -> true;
endmodule
```

(ii)

Figure 5.4: DTMC used for generating strictly omissive (SOA) faults in a diagnosis period (DP). (i) Transition diagram. (ii) PRISM code.

on individual frames; instead, we need to consider its effect on the entire DP. The effect of a SOA fault is that it may cause nodes to disagree on the operational state of other nodes. This may lead to a temporary drop of the membership to half its size. If the model were to reflect this behavior accurately it would have its symmetry broken, making it impractical except for configurations with only a few nodes[1]. Therefore, we used an approximate model to evaluate the reliability of the GMP in the presence of SOA faults. In this model the occurrence of a SOA fault in one round affects all nodes, i.e. it is as if all nodes had experienced a communication fault. This is somewhat conservative, but it helps keeping the model practical. Basically, this means that if the number of SOA faults in a DP is `s`, then the effective conviction threshold for that DP will be `Thr-s` rather than `Thr`.

To take into account SOA faults, we augmented the model with an additional DTMC that "generates" SOA faults. Figure 5.4 (i) represents its state diagram and Figure 5.4 (ii) shows the PRISM code. Variable `soa` represents the number of SOA faults in a diagnosis period (DP) and `c` the number of operating channels. Variable `s2tt` is the probability of occurrence of two SOA faults in a DP in a network with duplicated channel, `s1tt` the probability of occurrence of a single SOA fault in a DP in a network with duplicated channel. Variables `s2t` and `s1t` denote the corresponding probabilities for a network with a single channel. In this model, we consider at most 2 SOA faults in a DP, but it would be straightforward

[1] We were able to solve such a model only up to 5 nodes

to extend the model for an arbitrary number of SOA faults.

The node DTMCs are based on those developed for multiple round per DP, but because the time-step is the entire DP instead of the GMP-phase, we consider the effect of a lost vote similar to that of lost heartbeats. Furthermore, the model is modified to take into account the number of SOA faults that occur in a DP. The following segment shows one guarded command for a DP with a single SOA fault:

```
[round] s1=Gs & c=2 & soa=1 -> p:(s1'=Ps)&(g1'=0)&(p1'=1)
                             + tt1s:(s1'=Ts)&(g1'=0)&(t1'=1)
                             + (1-p-tt1s):(s1'=Gs);
```

where `tt1s` is the probability of a node being detected as faulty in a DP with one SOA fault in a network with two operating channels.

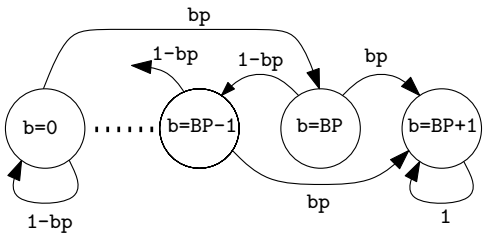
Auxiliary Models The computation of the probabilities of occurrence of one and two SOA faults (`s1t`, `s1tt`, `s2t`, `s2tt`) used in the SOA DTMC, and of the probabilities of a node being detected faulty because of transient communication faults (`t1s`, `tt1s`, `t2s`, `tt2s`) used in the node DTMC, was done with the help of two auxiliary DTMC-based models. These models were run off-line, with different input parameters, and their outputs were used as inputs to the DTMCs of the SOA models.

The DTMC for the computation of `t1s`, `tt1s`, `t2s` and `tt2s` is the auxiliary model developed for multiple round DP (cf. Figure 5.3). However, the PCTL property used is:

```
P=? [ true U<=(DP+1) nt>(Thr-s) ]
```

This property differs from the one used for the auxiliary model for multiple round DP in two points: 1) the number of steps considered is `DP+1` instead of `DP` because in this model fault detection does not distinguish between vote messages and heartbeat messages; 2) the effective conviction threshold depends on the number of SOA faults that occur in a diagnosis period.

The DTMC for the computation of `s1t`, `s1tt`, `s2t` and `s2tt` is structurally similar to the other auxiliary DTMC. Indeed both DTMCs are used to count the number of faults: in the former case, that of transient faults, and in the current case the number of SOA faults. The main difference is on the probabilities



(i)

```
module Burst
  b: [0..BDP+1] init 0; // Consecutive DP
                        //  with error burst;
  [round] b>0 -> bp:(b'= BDP+1)
               + (1-bp):(b'=b-1);
  [round] b=0 -> bp:(b'= BDP)
               + (1-bp):(b'=0);
endmodule
```

(ii)

Figure 5.5: DTMC used for generating burst faults. (i) Transition diagram. (ii) PRISM code.

associated with the transitions. Because we did not find operational data on the rate of SOA faults, we assume that this type of fault is a fraction of the other transient communication faults, that we call the *SOA to symmetric fault ratio (SSR)*. Thus, all the terms with `fl` are modified to include `SSR` as a factor. I.e., the term `fl` is replaced with the `term fl*SSR` and the term $\texttt{fl}^2$ is replaced with the term $\texttt{fl}^2$`*SSR`. Note that this means that `SSR` is a ratio for the network as a whole and not for one channel in particular, or alternatively that SOA faults in different channels are correlated. This is a conservative assumption in that it leads to a lower estimate of the GMP reliability.

5.3.2.6 Communication error bursts

The fourth and last model considers communication error-bursts, which are caused by long duration electromagnetic interference (EMI) bursts that make communication all but impossible while they last.

EMI bursts may have a duration of several tens of milliseconds, and consequently may affect multiple communication rounds or even DPs of the GMP, assuming a typical FlexRay network. This would require the GMP to bootstrap once the burst terminates. In order to tolerate faults that long without bootstrapping the GMP, we propose to change the GMP so that it allows failure of the protocol execution, i.e. failure in computing a majority, in as many consecutive diagnosis periods as can be affected by the error-burst.

To model the reliability of this proposed variation of the GMP, we added a new DTMC that "generates" error-bursts. Figure 5.5 (i) shows that DTMC,

and Figure 5.5 (ii) shows the PRISM code. Variable `b` represents the number of consecutive DP affected by error-bursts, `BDP` is the duration of the error-bursts in DP and `bp` their probability of occurrence in a DP. Although we assume that all error-bursts have the same duration, this is a parameter of the model and therefore different configurations can use different durations. Furthermore, it is assumed that if an error-burst occurs before the previous one has terminated the GMP will fail. This allows us to bound the value of `b` to `BDP+1` and therefore bound the size of the model.

While an error burst lasts, we propose that the GMP do not change its state, therefore we freeze the evolution of the node modules, as shown in the following segment of a node module:

```
[round] s1=Gs & c=2 & b=0 -> p:(s1'=Ps)&(g1'=0)&(p1'=1)
                           + tt:(s1'= Ts)&(g1'=0)&(t1'=1)
                           + (1-p-tt):(s1'=Gs);
... [round] c=0 | b!=0 -> true;
```

In addition, we added the predicate $b = BDP + 1$ to the PCTL property to be verified by PRISM:

`P=?[true U<=TS` $(n_G <= n_P + n_T)$ `|(c=0)|(b=BDP+1)]`

I.e., the protocol will fail if one of the following events occurs: the good members are not a majority, all communication channels fail simultaneously, and the number of DP affected by error-bursts exceeds `BDP`.

This model is rather simple and assumes that error-bursts lead to no communication in the network, and that the GMP is able to detect such a condition, via the underlying communication services. Nevertheless, it permits to roughly and quickly predict the expected reliability of a proposed modification to the protocol to tolerate burst errors. This allows to better assess if more work is warranted, towards fully fleshing the changes to the protocol and verifying its correctness, a not so trivial task.

5.3.3 Reliability Evaluation

We have implemented the reliability models mentioned using PRISM [48], a probabilistic model checker, and determined the reliability of the GMP MFA, which can be expressed as a conjunction of two predicates:

1. The number of good channels, i.e. channels without permanent faults, which we denote n_c must be larger than 0, otherwise no communication will be possible.

2. In every protocol execution, the number of members that may fail, either transiently or permanently, which we denote n_t and n_p respectively, must not exceed the number of good members, i.e. that do not fail, which we denote n_g. This condition is necessary so that the majority function can mask faults.

Thus, the reliability of the GMP is given by the probability of the predicate

$$(n_c > 0) \wedge (n_g > n_p + n_t)$$

holding true in a time interval of a given duration.

5.4 Experiments Design

In order to answer each of the questions stated in Section 5.2.1, we have designed one set of experiments except for the case of common-mode faults, for which we considered two sets of experiments: one for faults that affect a single message and another for communication error bursts.

For each experiment, all frames have the same size. Furthermore, we assume that each node transmits 2 frames per communication round, and that the communication round is as short as possible. I.e. we assume that as soon as the last node in a round finishes the transmission of its second frame the first node starts the transmission of its first frame of the next round. Therefore, the duration of a round depends on the number of nodes, the frame size and the bit-rate. An alternative is to assume that a communication round has a fixed duration. We decided to model the protocol as described, because most automotive applications are real-time requiring high responsiveness.

Also, in all these experiments we consider that the fault inter-arrival times have an exponential distribution. This is a common assumption for faults with a physical cause such as electromagnetic interference (EMI). Furthermore, for the model parameters we use values from the automotive domain, the main application domain of DuST networks in the near future. These values are based

Table 5.1: Parameter values used in all GMP experiments.

Parameter	Values (units)
Number of nodes (N)	3, 4, 5, 6, 7, 8, 9, 10
Number of channels	1, 2
Frame Size (FS)	32, 128 (bytes)
Bit Rate	1, 10 (Mbps)
Bit-error rate (BER)	1E-6, 1E-7, 1E-8
Node Permanent Fault Rate (PHw)	1E-5 (faults/hour)
Channel Perm. Fault Rate (PCh)	1E-6 (faults/hour)

on figures provided both in [18] and in [52]. Table 7.2 shows the values of the relevant parameters that we have used in virtually all experiments. Other parameters used in specific sets of experiments are presented when we describe those experiments in more detail.

The number of nodes range from 3 to 10. The reason for this is threefold. First, to limit the number of configurations. Second, because the time required to evaluate the models increases almost exponentially with the number of nodes. This is especially true for some models whose symmetry is limited. Third, and most importantly, the reliability of the protocol improves with the number of nodes, as we shall see, leveling off at the probability of failure of the communication channels.

We consider both single and duplicated channels. Indeed, whereas the use of duplicated channels affords higher reliability, its cost is higher. In most applications, a single channel will be used if its reliability is acceptable. Thus, it is important to evaluate the reliability of single channel configurations.

The frame size, the bit error rate (BER) and the bit-rate all affect the probability of transient communication faults. We consider frames of two sizes: 32 and 128 bytes. This is likely to cover most automotive applications. E.g. the header, trailer and synchronization bits in FlexRay lead to an overhead of around 16 bytes. Frames with size of 128 bytes are probably very rare in automotive applications, however we have chosen this value because longer frames have a higher probability of being affected by errors. For the BER we consider values in the range typical of copper medium. Although, optical fiber has a much lower BER, it is more expensive and therefore seldom used in the automotive domain. With

respect to the bit-rate we consider 2 values: 1 Mbps and 10 Mbps. Again, these are typical of the automotive domain. E.g., the timing parameters specified in FlexRay were determined for a 10 Mbps. However, it mentions the possibility of specifying values for lower bit-rates.

We assume that the only cause for transient faults in the GMP are communication errors. This is because for the parameter values considered, cf. Table 7.2, the probability of transient faults in nodes ([52] mentions a fault rate between 1E-3/hour and 1E-5/hour) is at least two orders of magnitude lower than the probability of transient communication faults and does not affect the reliability of the GMP.

For each of the permanent fault rates, hardware and channels, we used a single value, 1E-5/hour and 1E-6/hour, commonly used in the literature. However, we have performed some additional experiments to evaluate the effect of these parameters on the reliability of the protocol.

All the results presented in this section are the result of the evaluation of the probability of violation of the maximum fault assumption after one hour. This is a standard time interval used for reliability evaluation. Also, all the results were obtained starting from one initial state in which all components are working properly and all nodes are members of the group.

5.4.1 Single-Round Diagnosis Period

The goal of these experiments was to determine the main factors that affect the reliability of the GMP proposed in [7].

First, we carried out an experiment for all combinations of the values of the parameters shown in Table 7.2.

In addition, in order to evaluate the effect of the permanent fault rates, of both channels and nodes, and to keep the number of configurations evaluated manageable, we run two additional sets of experiments. In one of them we varied the channel permanent fault rate one order of magnitude above and below the values shown in Table 7.2 while maintaining the node permanent fault rate constant (and equal to the value shown in Table 7.2). In the second one, we varied the node permanent fault rate while maintaining the channel permanent fault rate

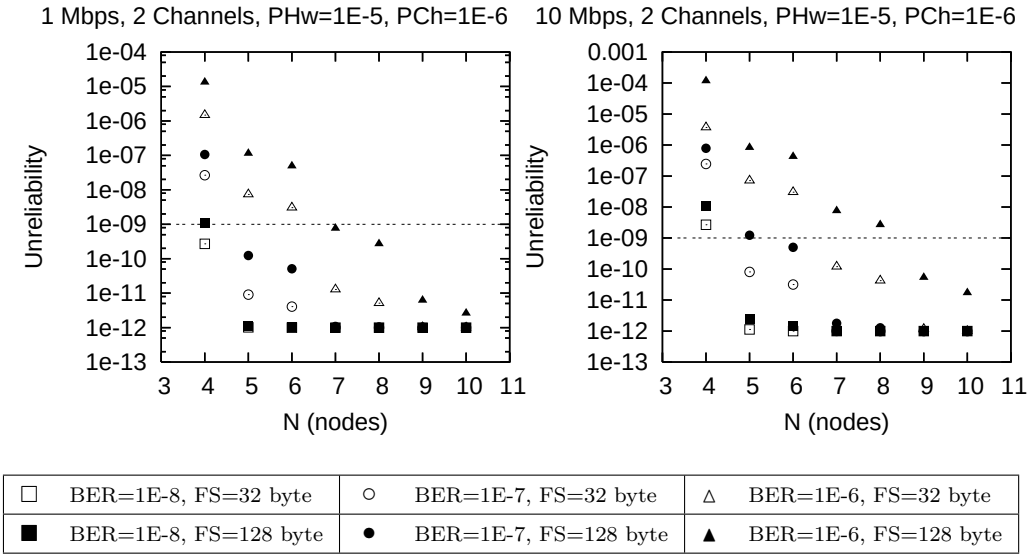


Figure 5.6: Assumption violation probability in the first hour for the single-round diagnosis period GMP [7].

constant. In each of these experiments we considered all possible combinations for the remaining parameters.

5.4.1.1 Data Analysis

Figure 5.6 shows the unreliability of the GMP for a communication network with two channels for bit-rates of both 1Mbps and 10 Mbps. The permanent hardware fault rate is 1E-5 per hour and the permanent channel fault rate is 1E-6 per hour.

The figure shows that the GMP reliability is highly sensitive to transient communication faults. In particular, it depends heavily on the BER. The effects of the frame size are also visible but not as large. Indeed, larger frames lead to a higher transient communication fault, but this is partially compensated in our models by a decrease in the number of executions in the reliability evaluation interval as a consequence of the increase in the duration of the communication round. The effect of the number of protocols executions can be observed comparing the results for 1 Mbps and 10 Mbps channels. All other parameters being equal, the difference between configurations with 1 Mbps and 10 Mbps channels is on the number of protocol executions, and this leads to a reliability almost one

order of magnitude lower for 10 Mbps channels, especially for configurations with a small number of nodes.

The GMP sensitiveness to transient communication faults is especially acute for configurations with a small number of nodes. This is because the GMP perceives transient communication faults as faults in nodes, and the smaller the number of nodes more likely is that the GMP will be unable to gather a majority. As we increase the number of nodes, the reliability improves fast. Note however that when we increase the number of nodes by one, the reliability improvement is larger if the number of nodes before the increment is even. This is because in that case the increment will allow for the failure (real or perceived) of one additional node per protocol execution without violating the MFA, whereas if the number of nodes before the increment is odd it will not. Ultimately, for the factor values considered, the system reliability is bounded by the probability of both communication channels failing permanently. For example, for a BER of 1E-6 and frames with the size of 32 bytes, this limit is reached for configurations of 9 nodes.

From this discussion, it is clear that the best we can hope with a single channel configuration is an unreliability of 1E-6 – this bound is determined by the probability of the single channel failing permanently. (This is confirmed by the experiments for single channel configurations whose results we do not show.) It is obvious then that single channel configurations are not appropriate for safety-critical applications, and we do not consider them further in this chapter.

These results hint that the probability of permanent channel faults affects significantly the reliability of the GMP. This is because, if a channel fails permanently, the protocol will operate with a single channel and therefore, the probability of a transient fault will be much higher. Indeed, the results we obtained from our experiments show that for networks with duplicated channels and the factor values shown in Table 7.2, a one order of magnitude variation in the probability of the permanent channel fault may lead to a variation up to two orders of magnitude in the GMP unreliability, for configurations with a large number of nodes. Conversely, for configurations with a small number of nodes and a high transient communication fault rate, e.g. 128 byte frames, the communication channel permanent fault rate has virtually no effect on the protocol reliability. On the other hand, our experiments have shown that for the values of all the

other factors that we have considered, one order of magnitude change above or below the permanent hardware fault probability has no effect on the reliability of the GMP.

5.4.2 Multiple-Round Diagnosis Periods

One approach to improve the resiliency of the GMP to transient communication faults is to make the diagnosis period a multiple of the communication round and to diagnose a node as faulty only if it is affected by transient communication faults in more than some number, that we call *diagnosis threshold*, of these rounds.

In order to assess the efficacy of this approach we carried out an experiment in which we varied both the diagnosis period from 2 to 5 communication rounds, and considered thresholds of 1 and 2 messages.

We considered configurations with 3 to 6 nodes, only. This is because most configurations for single round diagnosis periods and a larger number of nodes already present an acceptable reliability for safety-critical applications. Furthermore, as shown in Figure 5.7, below, the reliability increases with the number of nodes and configurations with 4 nodes already exhibit an unreliability below 1E-9.

For the BER we used only one value: 1E-6. This corresponds to the worst case for the values considered in Table 7.2.

5.4.2.1 Data Analysis

Figure 5.7 shows the reliability of the GMP for diagnosis periods multiple of the communication round, for communication network with two channels of 1 Mbps and 10 Mbps. The threshold used by the fault diagnosis algorithm is 1 message.

These results show clearly the efficacy in terms of reliability improvement of using longer diagnosis periods, even with a diagnosis threshold of only 1 message. Note that this approach is especially advantageous for configurations with a low number of nodes, which have a relatively low reliability for single round diagnosis periods. Nevertheless, configurations with 3 nodes are still too unreliable to be used in safety-critical applications, except for configurations of 1 Mbps and frame sizes smaller than 32 bytes.

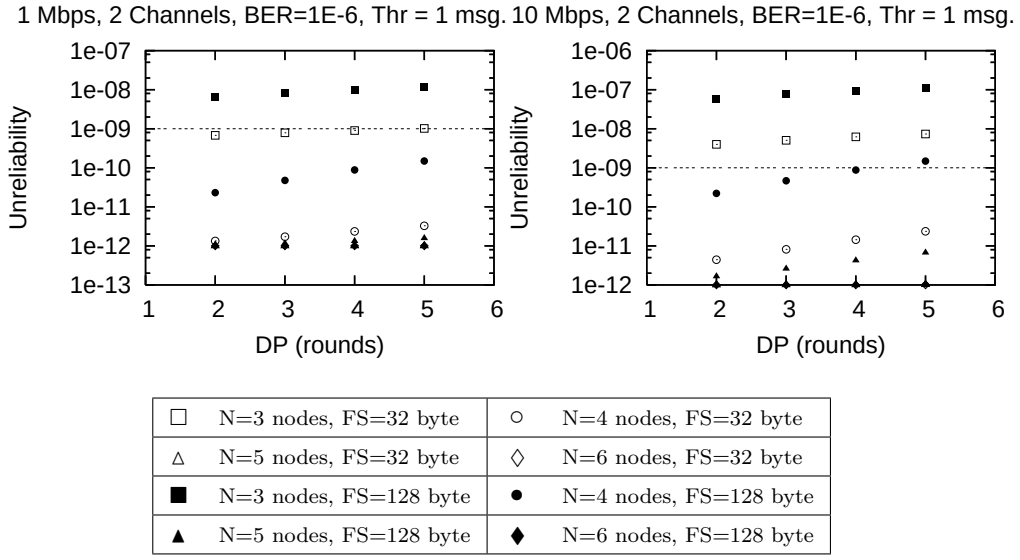


Figure 5.7: Effect of the length of the diagnosis period on the MFA violation probability.

We have also evaluated the reliability of the GMP for a diagnosis threshold of 2 messages. As expected, the reliability improved compared with that obtained with 1 message thresholds, but 10 Mbps configurations with 3 nodes and 128 bytes still exhibit insufficient reliability for safety-critical applications.

5.4.3 Common-Mode Faults

In all previous experiments we considered only independent faults, i.e. faults that affect only one node. However, virtually all transient communication faults are caused by EMI, which may be localized and therefore affect a subset of the nodes. Essentially, this corresponds to a partition of the network and may split the group members almost evenly.

In order to assess how this kind of fault affects the reliability of the GMP, we designed an experiment in which we modeled common-mode faults as a fault in every node. To keep the model tractable we made two simplifications. First, we considered that at most two common-mode faults may occur per diagnosis period. Second, we handled the loss of votes just like that of heart-beat messages: they are used to diagnose a node as faulty but are not otherwise taken into account in the gathering of a majority. This leads to a slight overestimation of the reliability.

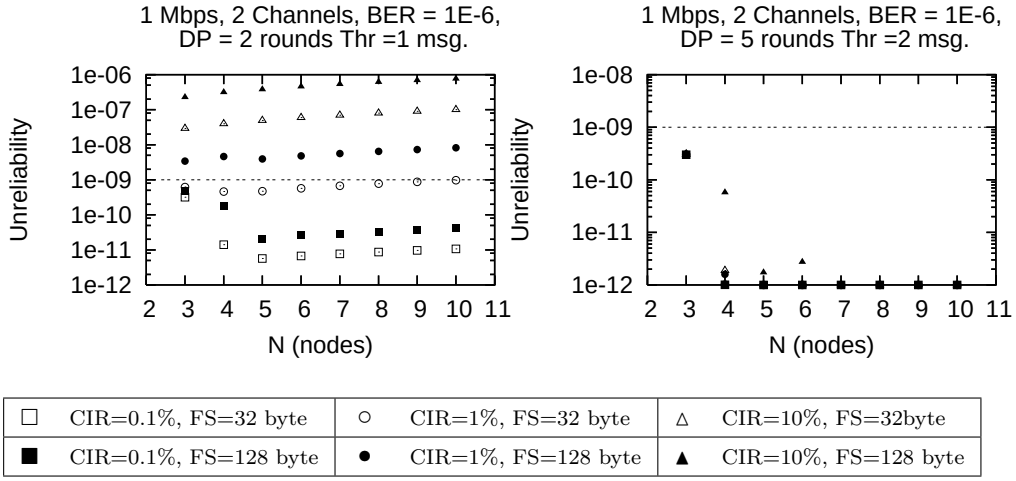


Figure 5.8: Effect of the common-mode to independent faults ration in the MFA violation probability.

Given that we did not find data on the rate of this kind of fault, we assumed that it is a fraction of the transient communication faults, that we call the *common-mode to independent fault ratio (cir)*. We considered a range between 0.1% and 10% for this parameter.

Furthermore, we have considered diagnosis periods of both 2 and 5 communication rounds, and thresholds of 1 and 2 messages, respectively.

Because of the execution time of the model is large, we run this experiment only for channels with a 1 Mpbs bit-rate, but nevertheless considered configurations with 3 to 10 nodes.

5.4.3.1 Data Analysis

Figure 5.8 shows the results of these experiments.

As we might expect the reliability of the GMP is badly affected by the common-mode faults, especially for common-mode to independent fault ratios (CIR) above 1%, and for diagnosis thresholds of 1.

In contrast to the results of the previous experiments, for the values used, the unreliability increases with the number of nodes, especially for thresholds of one. This is partially an artifact of the way we model common-mode faults: for a threshold of one, in diagnosis periods with common mode faults, one additional

independent fault causes the violation of the MFA, and the probability of such a fault occurring increases with the number of nodes. For configurations with a threshold of two messages, the occurrence of one additional independent fault is not enough to cause a violation of the MFA. This partially explains the much better results obtained in experiments with that threshold, in spite of using a higher diagnosis period.

Another factor that accounts for the much better results of configurations with a threshold of two is that we limit the number of common-mode faults per diagnosis period to two. For configurations with a threshold of one, this does not affect the results because the occurrence of two common-mode faults leads automatically to the violation of the MFA. However, for configurations with a threshold of two this is not the case, leading to an overestimation of the reliability.

5.4.4 Communication Error Bursts

The goal of this experiment was to determine whether it is possible to make the GMP resilient to long duration EMI bursts by using diagnosis periods multiple of the communication round.

The conventional approach to deal with this kind of fault is to switch to a special operating mode such as the black-out mode in the TTP/C architecture. The main problem with this approach is that it requires that the nodes re-synchronize. In the case of the GMP it would require the creation of a new group, usually a lengthy procedure.

In order to tolerate faults that long, we propose to change the GMP so that it allows failure of the protocol execution in as many consecutive diagnosis periods as can be affected by the error burst. An alternative that we considered was to lengthen the diagnosis period to a duration longer than the error burst and to define a threshold that would tolerate the loss of as many messages as can be affected by the error burst. However, this alternative makes the GMP less responsive to permanent faults.

We characterize error burst faults with two parameters: the burst duration and the burst rate. In our experiment we used values of 50 ms for the burst duration, and varied the error burst rate (BR) one order of magnitude below and above 1E-4 per hour. These values are based on the figures provided in [52] for transient hardware faults in nodes caused by EMI.

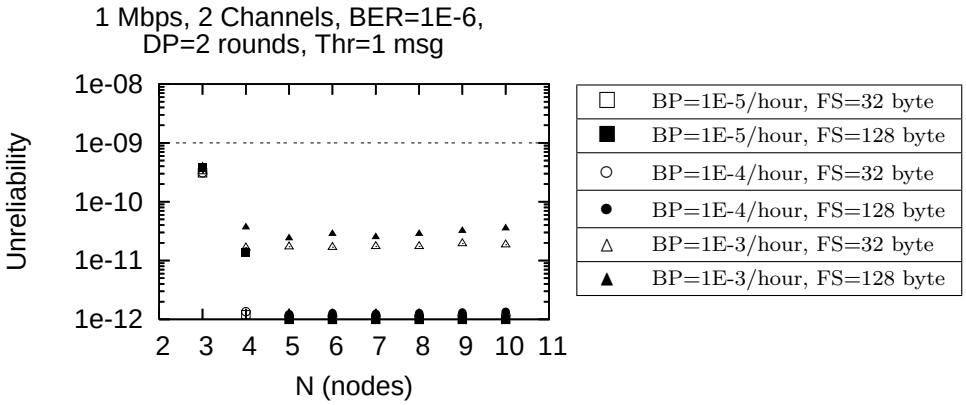


Figure 5.9: Effect of the error burst rate on the MFA violation probability.

To keep the model tractable we handle the loss of votes just like that of heart-beat messages, as described in the previous subsection.

We run this experiment for configurations of both 1 Mbps and 10 Mbps channels, diagnosis periods of 2 communication rounds and thresholds of 1 message. We chose these values because the results obtained in Section 5.4.2 show that they provide adequate reliability for almost all configurations.

5.4.4.1 Data Analysis

Figure 5.9 shows the results of these experiments.

These results indicate that the approach proposed is effective to tolerate error bursts. Except for burst rates of 1E-3 per hour, the reliability is comparable to that for the corresponding configurations of the models without burst errors reported in Subsection 5.4.2. However, the reliability for configurations of 3 nodes is larger than that shown in Figure 5.7. This is because the way we model the effect of lost votes leads to an overestimation of the reliability.

Although at the time of writing we do not have values for 10 Mbps channels, we expect them to be in agreement with those obtained for 1 Mbps channels.

5.5 Conclusions

We have evaluated the reliability of the assumptions made in the proof of a group membership protocol (GMP) especially designed to take advantage of a

new TDMA protocol that is likely to become the de facto standard for next generation networks in the automotive domain.

In our study we considered several fault scenarios, including permanent, transient and common-mode faults, affecting both channels and nodes. Furthermore we performed a sensitivity analysis to assess the influence of different parameters on the protocol's reliability, using value ranges typical of the automotive domain.

The results obtained show that the protocol as originally proposed is highly sensitive to transient faults especially for configurations with a few nodes. However, reliability levels close to those required for safety critical applications can be achieved, through the use of duplicated channel and a diagnosis period multiple of the communication round. This same approach works also for common-mode transient communication faults, both those that partition the network for a frame duration and error-bursts that may affect several frames.

Chapter 6

Reliable Broadcast for Dust Networks

This chapter presents a family of reliable broadcast protocols (RB-DuST) designed for DuST protocols such as FlexRay that do not provide such protocols natively. In addition to the specification of the protocols, we also present some arguments on their correctness. We conclude the chapter with a discussion of the implementation of these protocols as a middleware layer on top of FlexRay.

6.1 Introduction

We have argued in previous chapters that safety critical applications domains have extremely demanding reliability requirements. In particular, FlexRay is a minimalist bus that provides only clock synchronization and basic communication services, which do not provide sufficient fault tolerance for message broadcasting. Although CRCs detect corrupted messages, limiting the effects of communication faults [53], they do not prevent messages from being lost due to these transient faults.

To fulfill this demand we present a family of reliable broadcast (RB) protocols that can be implemented as a middleware layer on top of FlexRay's basic communication services. These protocols take advantage of the dual scheduling

TDMA scheme used to achieve better performance than that achievable with vanilla-TDMA.

In the next section we provide a brief introduction on reliable communication principles. In Section 6.3, we specify a set of reliable broadcast protocols specially designed to take advantage of the DuST scheme. In Section 6.4 we present some implementation issues of related reliable broadcast protocols on FlexRay. Finally, we conclude in Section 6.5.

6.2 Reliable Broadcast Specification

Reliable communication services are designed to tolerate channel faults and recover corrupted messages, usually by means of repeated retransmissions. A Reliable Broadcast (RB) service must ensure that broadcast messages are delivered by all operational nodes even in spite of failures [26].

The properties of the RB services we present here are based on properties defined by Hadzilacos and Toueg in [24]. However we have adjusted these properties to better represent the requirements imposed by safety critical systems. The RB properties we consider in this work are the following:

RB1 (Flex-Validity): if a correct node broadcasts a message m, then all correct receiver eventually delivers m or ϕ (which is a special *null message*).

RB2 (Integrity): Every correct receiver delivers at most one message, and if it delivers $m!=\phi$ then the sender must have broadcast m.

RB3 (Agreement): If a correct receiver delivers a message m, then all correct receivers eventually deliver m.

RB4 (Timeliness): There is a known constant Δ such that if the broadcast is initiated at real-time t, then all correct receivers deliver a message (either m or ϕ) before $t + \Delta$[1].

Usually instead of RB1, the validity property used is:

RBV (Validity): If a correct node broadcasts a message m, then all correct receiver eventually delivers m.

[1]Note that RB4 implies also termination since nodes are obligated to deliver m or ϕ before the deadline.

However, it is clear that this property cannot be satisfied in the presence of faults on the communication channels, because even if the node that broadcasts m is correct, this does not imply that m will be received by other correct nodes. Thus, we have specified a weaker property, RB1. Although this property can be satisfied by a protocol that always delivers ϕ, our protocols do satisfy RBV, if there are no faults in the communication channels.

Note also that our reliable broadcast protocols are not designed to ensure order in message delivery. However, order properties can be implemented as a complement on top of the protocols described in this paper.

6.3 Reliable Broadcast Protocols

In this section, we specify four reliable broadcast protocols especially designed for DuST communication networks like FlexRay. They all share the same system model and fault assumptions. The protocols differ from each other on the strategy used to deliver messages and on the type of acknowledgment messages (ACK vs. NACK).

6.3.1 System Model

These reliable broadcast protocols were designed to work on a network composed of a fixed number of nodes that are connected via a broadcast channel, in which every node receives the messages sent by all nodes including itself. Each node receives its own broadcasts via its network interface, not via a loopback device. The network uses a medium access control protocol based on a dual scheduled TDMA scheme, e.g. FlexRay, and all nodes are aware of the slots assignment in both the static and the dynamic segments. To simplify the presentation, we assume that each node is associated only to one slot in each communication segment and uses these slots to execute the reliable broadcast protocol. Furthermore, we assume that the segments are long enough so that all nodes are able to send the messages required by the reliable broadcast protocol, except possibly in the presence of faults.

6.3.2 Fault Model

We assume that nodes can fail only by crashing. Channel faults can corrupt messages in transit preventing their reception. All channel faults affect all nodes in the same manner, i.e. we assume that all channel faults are symmetric. Furthermore, channel faults are not persistent, i.e. if the channel drops a message, it may deliver a subsequent frame.

6.3.3 Protocols Primitives

The reliable broadcast protocols have three primitives:

a)*broadcast*: transmits a message to all nodes in N (set of nodes) in every available channel in C (set of channels).

b)*receive*: returns either a message previously broadcast, or some special message such as ϕ indicating an error condition.

c)*deliver*: delivers a received message to the higher layer protocols or applications.

6.3.3.1 RB Execution

The execution of the RB protocols proceeds in rounds one after the other. Each *RB round* comprises three phases, and takes two DuST communication rounds (each of which comprises the static and dynamic segments). In the first phase, the *B-phase*, which takes a communication round, the transmitters broadcast a message which is received by the correct receivers in the same round, if no channel fault occurs. In the second phase, the *C-phase*, which takes another communication round, the receivers process the messages received in the previous communication round and acknowledge the reception of messages. Finally, in the last phase, the *D-phase*, which does not require any communication round, the nodes deliver the messages received in the previous phase.

6.3.4 Protocols

6.3.4.1 Receive & Deliver (RBRD) Protocols

In the RBRD protocols, each receiver delivers the broadcast message as soon as it receives it. The confirmation can be done using either negative or positive acknowledgments, leading to two different versions of the protocol. That we now describe.

6.3.4.2 Receive & Deliver with Positive Acknowledgments - RBRD(a)

In the RBRD(a) protocol the transmitter uses positive acknowledgments to decide whether or not it should retransmit a message broadcasted in the previous RB round. The RBRD(a) protocol algorithm is presented in Figure 6.1. The transmitter initiates a new RB round until either it has received acknowledgements from all receivers or the maximum number of retransmissions is reached. Receivers acknowledge every message they receive, and if the message has not been delivered yet they deliver it.

6.3.4.3 Receive & Deliver with Negative Acknowledgments - RBRD(n)

In the RBRD(n) protocol the transmitter uses negative acknowledgments to decide whether or not is should retransmit a message broadcasted in the previous RB round. RBRD(n) algorithm is showed in Figure 6.2. The transmitter initiates a new RB round until it does not receive a negative acknowledgement or the maximum number of rounds is reached. A receiver sends a negative acknowledgement for every message it misses. Otherwise, if the message has not been delivered yet it deliver it. If a receiver does not receive a correct message after the maximum number of RB-rounds, it delivers ϕ.

In principle, the use of negative acknowledgments has the advantage that they are sent only in the presence of faults. As faults are not that common, the traffic generated by negative acknowledgments is lower than that by positive acknowledgments. On the other hand, acknowledgment messages may also be lost and as consequence some faults may not be detected. However in the case of FlexRay we assume that all lost messages can be detectable.

RBRD(a) Protocol

State

m: D-Message;

a: C-Message;

$Rec - Set$: set of Receivers;

$ack - Set$: set of C-Message;

$ackbit$: positive acknowledgment bit initialized as false;

$delivery - Deadline$: a flag that indicates the end of delivering time;

***** TRANSMITTER *****

B-Phase

Communication step:

1. broadcast (m)

C-Phase

Communication step:

2. receive()

Processing step:

3. **for all** receive() == a **do**

3.1 add a to the $ack - Set$

4. **If** $ack - Set$ differs from the $Rec - Set$ **and not** $delivery - Deadline$ **then**

4.1. go to step 1 (retransmit m)

***** RECEIVER *****

B-Phase

Communication step:

1. receive()

Processing step:

2. **if** have received m **then**

2.1 set the $ackbit$ to true

C-Phase

Communication step:

3. **If** $ackbit$ is set to true **then**

3.1. set a with the value of $ackbit$

3.2. broadcast(a)

3.3. set $ackbit$ to false

D-Phase

4. **If** have received m correctly **and** did not deliver it yet **then**

4.1. deliver (m)

Figure 6.1: RBRD protocol with retransmission based on positive acknowledgments.

RBRD(n) Protocol

State

m: D-Message;
n: C-Message;
$nack - Set$: Set of C-Message;
$nackbit$: negative acknowledgment bit initialized as false;
$delivery - Deadline$: a flag that indicates the end of delivering time;

***** TRANSMITTER *****

B-Phase

Communication step:

1. broadcast (m)

C-Phase

Communication step:

2. receive()
2.1 **for all** receive() = n **do**
2.2 add n to the $nack - Set$

Processing step:

3. **If** $nack - Set$ is not null **and not** $delivery - Deadline$ **then**
3.1. go to step 1 (retransmit m)
3.2. reset $nack - Set$

***** RECEIVER *****

B-Phase

Communication step:

1. receive()

Processing step:

2. **if** have not received m **then**
2.1 set the $nackbit$ to true

C-Phase

Communication step:

3. **If** $nackbit$ is set to true **then**
3.1. set n with the value of $nackbit$
3.2. broadcast(n)
3.3. set $nackbit$ to false

D-Phase

4. **If** have received m **and** did not deliver it yet**then**
4.1. deliver (m)

Figure 6.2: RBRD protocol with retransmission based on negative acknowledgments.

6.3.4.4 RBRD's Correctness

Both versions of RBRD protocol (a and n) are very similar and so most correctness arguments apply equally to both. Whenever necessary, however, we provide separate arguments.

- *RB1(Flex-Validity):* if the transmitter fails then this property is trivially satisfied. On the other hand, if the transmitter does not fail, the message may not be received, depending on the occurrence of communications faults. In any case, if the message is received, it will be delivered. If not, ϕ will be delivered.

- *RB2(Integrity):* a receiver only delivers a message it received, if it has not delivered it previously. Furthermore, if it receives no message it delivers ϕ.

- *RB3(Agreement):* because communication faults are symmetric all correct receivers will receive the same messages. In addition, the protocol is deterministic, therefore all correct receivers will deliver the same message: either the message broadcasted by the transmitter or ϕ.

- *RB4(Timeliness):* the system is synchronous and the protocol has a maximum number of RB-rounds, r, each of which comprising two communication rounds of fixed duration. Therefore, after r RB-rounds, in the worst case, each receiver either delivers m or ϕ.

6.3.4.5 Threshold Delivery (RBTD) Protocols

In the threshold delivery protocols (RBTD) the messages are delivered based on the information provided by receivers. Nodes only deliver the message if they learn that number of receivers that received the messages reaches a specified threshold.

6.3.4.6 Threshold Delivery with Positive Acknowledgments - RBTD(a)

Like in RBRD(a), in the RBTD(a) protocol the transmitter uses positive acknowledgments to decide on message retransmission. However the delivery is conditioned to the number of nodes that signal the reception of the message.

Hence nodes that have received the message m correctly deliver it only if the necessary threshold is reached. Otherwise, they deliver ϕ.

Figure 6.3 shows the RBTD(a) protocol's algorithm. The operation of RBTD(a) differs from RBRD(a) as shown in bold.

- The transmitter keeps retransmitting the message until either it has received acknowledgments from all the receivers or the maximum number of rounds is reached.
- The receiver acknowledges every message it have received. The message is delivered if the it **has been acknowledged by enough receivers** and has not been delivered yet.

Note that RBRD(a) and RBTD(a) do not differ with respect to transmitters. The difference is with respect to the receivers, which only deliver a message if they have received acknowledgements from enough receivers.

The advantage of the RBTD(a) protocol is that it guarantees that a message is delivered only if it has been received by enough receivers, whereas RBRD(a) would deliver the message even if it was acknowledged only by a smaller fraction of the nodes. Thus, RBTD(a) can prevent correct nodes from delivering a message because too many nodes are faulty.

6.3.4.7 Threshold Delivery with Negative Acknowledgments - RBTD(n)

In the RBTD(n) protocol the transmitter uses negative acknowledgments to decide on the retransmission of messages. The retransmission occurs whenever the transmitter gets a negative acknowledgment and the maximum number of RB-rounds has not been reached. Receivers use negative acknowledgements to decide on delivery of messages. All receivers that have correctly received a message m deliver it, if the number of received negative acknowledgements is below a given threshold. Otherwise, they deliver ϕ.

The Figure 6.4 shows the RBTD(n) protocol's algorithm. The operation of RBTD(n) differs from RBTD(a) as shown in bold.

- The transmitter keeps retransmitting the message until **either the number of NACKs received in a round is zero** or the maximum number of rounds is reached.

RBTD(a) Protocol

State

m: D-Message;

a: C-Message;

$ackbit$: positive acknowledgment bit initialized as false;

$rec - Set$: set of Receivers;

$ack - Set$: received acknowledgment set;

$delivery - Deadline$: a flag that indicates the end of delivering time;

TSD: Constant that defines the threshold;

***** TRANSMITTER *****

B-Phase

Communication step:

1. broadcast (m)

C-Phase

Communication step:

2. receive()

Processing step:

3. **for all** receive()==a that **do**

3.1 add a to the $ack - Set$

4.**If** $ack - Set$ differs from the $rec - Set$ **and not** $delivery - Deadline$ **then**

4.1. go to step 1 (retransmit m)

***** RECEIVER *****

B-Phase

Communication step:

1. receive()

Processing step:

2. **if** have received m **then**

2.1 set the $ackbit$ to true

C-Phase

Communication step:

3. **If** $ackbit$ is set to true **then**

3.1. set a with the value of $ackbit$

3.2. broadcast(a)

3.3. set $ackbit$ to false

4. receive()

Processing step:

5. **for all** receive()==a **do**

5.1 add a to the $ack - Set$

D-Phase

6. **If** (have received m correctly) **and** $ack - Set$ reaches the TSD **then**

6.1. deliver (m)

Figure 6.3: RBTD protocol with positive acknowledgments.

- The receiver sends a **NACK for every message it did not receive** and delivers the message if **both the number of NACKs received in a C-phase does not exceed a given threshold** and it has not been delivered yet.

The main advantage of RBTD(n) over RBTD(a) is that acknowledgments are sent only in the presence of faults. On the other hand, acknowledgment message are subject to be lost and as consequence some faults may not be detected. Note that in the RBTD(n) nodes do not need to keep the information about received acknowledgment among rounds. Therefore if some receiver fails permanently all other will assume that it have received the message.

6.3.4.8 RBTD's Correctness

Like with RBRD, both versions of RBTD protocol (a and n) are very similar and so most arguments of their correctness apply equally to both versions. Whenever necessary, however, we provide separate arguments.

- *RB1 (Flex-Validity):* As in the case of RBRD, channel faults may prevent a receiver from delivering m. In RBTD this may happen even if the receiver receives m. However, in this case, the receiver will deliver ϕ.
- *RB3(Agreement):* Like in RBRD, the argument relies on the determinism of the algorithm and on the assumption of symmetry in channel faults. The latter ensures that all correct receivers receive the same messages. This together with the protocol's determinism ensures that all receivers make the same decisions, i.e. either ϕ or the message received.
- *RB2(Integrity) and RB4 (Timeliness):* The arguments presented for the RBRD protocols are also applicable to the RBTD protocols.

RBTD(n) Protocol

State

m: D-Message;

n: C-Message;

$nackbit$: positive acknowledgment bit initialized as false;

$nack - Set$: received negative acknowledgment set;

$delivery - Deadline$: a flag that indicates the end of delivering time;

TSD: Constant that defines the threshold;

***** TRANSMITTER *****

B-Phase

Communication step:

1. broadcast (m)

C-Phase

Communication step:

2. receive()

Processing step:

3. **for all** receive()==n **do**

3.1 add n to the $nack - Set$

4. **If** $nack - Set$ is not null **and not** $delivery - Deadline$ **then**

4.1. go to step 1 (retransmit m)

4.2. reset $nack - Set$

***** RECEIVER *****

B-Phase

Communication step:

1. receive(m)

Processing step:

2. **if** have not received m **then**

2.1 set the $nackbit$ to true

C-Phase

Communication step:

3. **If** $nackbit$ is set to true **then**

3.1. set n with the value of $nackbit$

3.2. broadcast(n)

3.3. set $nackbit$ to false

4. receive()

Processing step:

5. **for all** receive()==n **do**

5.1 add n to the $nack - Set$

D-Phase

6. **If** (have received m correctly) **and** $nack - Set$ does not surpass the TSD **then**

6.1. deliver (m)

Figure 6.4: RBTD protocol with negative acknowledgments.

6.4 Implementation Issues

The proposed RB protocols are intended to be used in DuST networks such as FlexRay. Due to the potential benefits of using such networks brings the possibility to explore the reliable broadcast of sporadic urgent/critical messages (e.g. in a car the engine temperature increase to a dangerous level, handbrakes on and etc) over the dynamic scheduled communication segment. This option is took because to provide reliable broadcast of sporadic messages in static segment seems not be profitable since message retransmissions must be scheduled a priori, possibly causing excessive use of bandwidth transmitting useless or repetitive information. Therefore the use of a "dynamic" reliable broadcast, through DuST networks, can reduce the pre-allocated bandwidth destined to those static sporadic messages and also to release bandwidth to other traffics.

In this section we present some implementation directions about a middleware Reliable Broadcast service layer on top of dynamic scheduled segment of DuST networks. Such middleware layer includes the implementation of RBTD and RBRD protocols and shall provide the basic services to handle information provided by network communication layer, such as RB protocols primitives (broadcast, receive and deliver). In addition the middleware layer must also provide means to detect the loss of messages. This "detection error" service can be done handling information provided by the medium access protocol. Corrupted messages are usually discarded by the network protocols unadvisedly, but in the FlexRay protocol it is possible to get some extra information about discarded messages. FlexRay provides state information about messages that it has received in it's reception buffers. The status information variables (namely $Valid\ Frame$, $Syntax\ Error$, $Content\ Error$ and $BViolation$) are provided to from FlexRay to the Controller Host Interface (CHI) and can be used to detect whether messages were broadcast during a specific slot or not. More specifically a "network silence state" is determined if these all four status variables's values are set to *false*. Thus, in absence of the "network silence state", it is possible to presume an attempt of message broadcast. This FlexRay feature are very important because it can directly influence on the increase the reliability of communications. More specifically the detection of lost messages is useful to enforces the protocol initialization on dynamic segment. Because without detection, the loss of any starting reliable broadcast message may not be perceived by receivers. Similar

effects can be observed in RBTD(n) and RBRD(n) protocols if all negative acknowledgments are lost by the transmitters (possibly due to a transient fault in channels). This may lead transmitters to skip the message retransmission by assuming that all nodes have received correctly the message.

Another issue concerns about the allocation of the RB protocols in the dynamic segments. Evidently it is necessary to ensure sufficient time to reliable broadcast each RB message in the dynamic scheduled segment. This can be done by pre-allocating the initial slots of dynamic segment exclusively to reliable broadcast messages. We assume that the initial portion of slots can be always dedicated to RB protocols execution. Note that, in absence of reliable broadcast transmissions, this strategy blocks only a small portion of the dynamic segment that is occupied by the unused mini-slots and releases the major part of dynamic segment to other traffic (OT). An example is illustrated in Figure 6.5 which shows a execution of RBTD(n) and RBTD(a) protocols assuming a fail on the first reliable broadcast message sent by a transmitter. Note that, independently of protocol type, if no RB traffic is took than there is always significant space available. Indeed this is only possible due to the flexible mini-slot approach used by FlexRay.

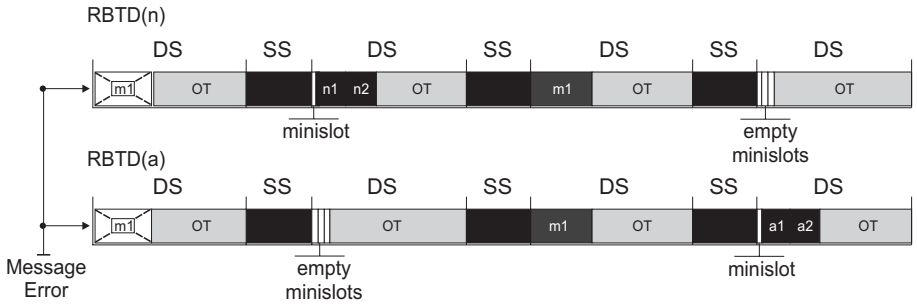


Figure 6.5: Example of RBTD protocol executions assuming priority slot allocation on dynamic segment of FlexRay protocol

6.5 Conclusions

In this chapter we have presented a family of reliable broadcast protocols designed to take advantage of DuST protocols. These protocols are considered to be simple and differ essentially on two aspects: the type of acknowledge messages used (positive vs. negative) and on the policy used to deliver a message. The

delivery of messages can be based on a threshold (RBTD) or at reception time (RBRD). The RBRD protocol can deliver messages faster than RBTD. However, as drawback, they do not provide order and consistence on message delivery that are inherent of RBTD protocols. We also have provided arguments for protocol's correctness. Finally we have presented some implementation issues concerning about the use of presented protocols as a middleware service layer which provide reliable communication on dynamic segment of FlexRay protocol.

Chapter 7

Reliability Analysis of RB-DuST Protocols

This chapter presents a reliability analysis of the Reliable Broadcast Protocols (RB-DuST) by means of probabilistic model checking. It includes a detailed description of the discrete-time Markov chains and of the PRISM models used in the analysis. Furthermore, it describes the experiments carried out and discusses the results obtained.

7.1 Introduction

Although the performance of the communication services is important in safety critical applications, the dependability of these services is crucial. Given that the automotive domain is the most likely application domain of DuST networks such as FlexRay, we evaluate the reliability of Reliable Broadcast protocols presented in the previous section. For this evaluation we have developed a set of discrete-time Markov chains (DTMCs) and solved them with the help of PRISM, a symbolic probabilistic model checker.

The probabilistic model checking approach can help to improve the confidence on the protocols' correctness and allows to estimate its reliability under more realistic fault assumptions than those considered in arguing their correctness.

More specifically, we use it to perform reliability evaluation of RB protocols in the presence of strictly omissive asymmetric (SOA) faults [49].

This chapter includes a channel fault case analysis showing the scenarios where RB-DuST protocols are prone to fail. This case analysis is used as the starting point in the development of the models that are used in our reliability analysis. We have designed a set of discrete-time Markov chains (DTMC) models for each variation of the RB-DuST protocols. The modeling, verification and reliability evaluation are made using the Probabilistic Model Checker PRISM [54, 48]. Our results show that the RB-DuST protocols proposed can provide reliability levels required by safety-critical applications. However, this is achieved at a bandwidth cost that depends mostly on the transient channel fault rate. Thus, to prevent wasting bandwidth, the RB-DuST protocols should be carefully tuned to provide only the required reliability levels.

In the next section we present the channel fault case scenarios used to elaborate the reliability models. In Section 7.3 we present the Markov models of the proposed protocols followed by the examples of codes used in the implementation. An analysis of results obtained by evaluating these models in the PRISM probabilistic model checker is presented in Section 7.4. Finally, we conclude in Section 7.5.

7.2 Channel Fault Scenarios

Assuming the RB-DuST fault and communication models we present here eight cases (Figure 7.1) where transient channel faults can induce the protocols to violate those reliable broadcast properties specified in the previous chapter. In addition we also show in which cases the traditional Validity property (RBV) can be violated.

Note that permanent faults on transmitters and channels, in conjunction with transient faults, can make the RB-DuST protocols to violate RB properties. For example, the transmitter crashes or channels become permanent faulty immediately after a broadcast. However this cases are upmost unlikely if compared to transient faults cases. Because of that transient channel faults demands special attention and must be correctly mapped in order to ensure more accuracy in models design. In the fault cases presented we assume that B-phase and C-

Phase are executed only in the dynamic segments (labeled as D in Figure 7.1) of communication protocol.

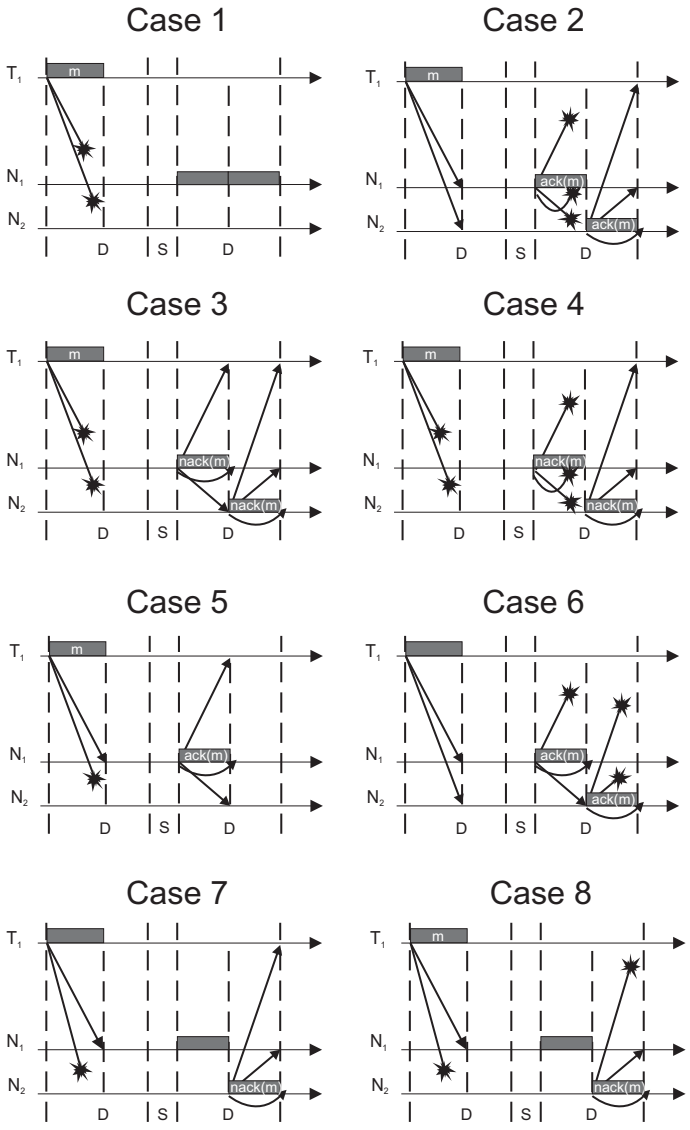


Figure 7.1: Channel fault cases (D: Dynamic Segment, S: static segment)

7.2.1 Effects of Symmetric Channel Faults on RBTD(a) and RBRD(a) Protocols

Case 1 : repeatedly loss of messages due to symmetric channel failures.

RBTD(a): if transmitter is also receiver property RBV is violated. Other-

wise no violation.

RBRD(a): if transmitter is also receiver then RB3, RBV and RB1 are violated. Otherwise no violation.

Case 2: acknowledgement messages are repeatedly lost due to symmetric channel failures.

RBTD(a): if the number of nodes whose ACKs are lost is a majority then property RBV is violated (it does not depend whether transmitter is a receiver or not). Otherwise there is no violation.

RBRD(a): no violation.

7.2.2 Effects of Symmetric Channel Faults on RBTD(n) and RBRD(n) Protocols

Case 3: messages are repeatedly lost due to symmetric channel failures.

RBTD(n): if transmitter is a receiver then RBV is violated. Otherwise, no violation.

RBRD(n): if transmitter is a receiver then RB3, RB1 and RBV are violated. Otherwise, no violation.

Case 4: messages and also negative acknowledgments are lost due to symmetric channels failures.

RBTD(n): if transmitter is a receiver and the number of lost NACKs are majority then RB1, RB3 and RBV are violated. Otherwise, if transmitter is a receiver and the number of lost NACKs are not majority then only RBV is violated.

RBRD(n): if transmitter is a receiver then RB3, RB1 and RBV are violated. Otherwise, no violation.

7.2.3 Effects of Asymmetric Channel Faults on RBTD(a) and RBRD(a) Protocols

Case 5: some receivers lose the message broadcast due to asymmetric channel failures.

RBTD(a): if there is a majority of nodes that lose the message then RBV is violated. Otherwise RB1, RB3 and RBV are violated.

RBRD(a): RB3,RB1 and RBV are violated

Case 6: acknowledge messages are lost, these case includes a arbitrary number of faults.

RBTD(a): if the number of acknowledgement messages lost by at least one node represents a majority then RB3, RB1 and RBV are violated. Otherwise, if all nodes do not receive acknowledgments enough to deliver the message then RBV is violated.

RBRD(a): no violation

Note that cases where messages are lost followed by fail in acknowledgements are not showed because these scenarios represent a variation of Case 5, and as consequence, are self included.

7.2.4 Effects of Asymmetric Channel Faults on RBTD(n) and RBRD(n) Protocols

Case 7: some receivers lose the message broadcast due to asymmetric channel failures. This case is similar to Case 5.

RBTD(n): if there is a majority of nodes that lose the message then RBV is violated. Otherwise RB3, RB1 and RBV are violated.

RBRD(n): RB3, RB1 and RBV are violated

Case 8: some receivers lose message broadcast and some negative acknowledgments due to asymmetric channel faults.

RBTD(n): if there is a majority in the number of nodes that lose the message and in the number of negative acknowledgments received by each node then only RBV is violated. Otherwise RB1, RB3 and RBV are violated.

RBRD(n): RB3, RB1 and RBV are violated

7.2.5 Correctness Issues

Symmetric channels vs protocol's correctness: as the case analysis have shown the impact of symmetric faults on the presented RB-DuST protocols is null when it is assumed that transmitters are not receivers of the messages they transmit. Indeed the results of experiments with the models that we describe in the next sections and that assume that transmitters are not receivers of their own messages have shown that in the presence of symmetric channel faults only the probabilities of *Agreement* (RB3) and *Flex-Validity* (RB1) being violated by any RB protocol is exactly zero.

RB1 violation assuming symmetric faults: the *Flex-Validity* (RB1) property can be violated if we assume that the transmitter is a direct receiver, i.e. it receives the messages it transmits using some loopback mechanism that does not comprise the physical layer. In that case the reliability is determined by the probability of a single message being lost due to symmetric channels faults. However only the RBRD variations of the RB-DuST protocols violate RB1 in these circumstances, as summarized in table 7.1. The RBTD protocols do not violate RB1 because no other node will ever receive the broadcast message and therefore the threshold condition will not be satisfied in any node, even on the transmitter.

Table 7.1: Violation of Flex-Validity (RB1), assuming that transmitters are direct receivers and symmetric channel faults only.

	RBRD	RBTD
ACKs	Yes	No
NACKs	Yes	No

7.3 Reliability Models

Model checking helps to improve the confidence on the correctness of the protocols and allows to estimate the reliability under a more comprehensive fault model than that considered in the arguing the protocols correctness in the previous chapter. In this section, we present the DTMC models of the RB-DuST protocols we developed to evaluate their reliability under a fault model that allows all channel fault scenarios presented in the previous section. For each variation of the RBRD and RBTD protocols, we have developed a different model and implemented it in PRISM.

7.3.1 Models

All models are discrete time Markov chains (DTMC). These DTMCs model a protocol execution and are used to determine the probability of violating the RB specification, more specifically the *agreement* property. The time step used is a TDMA slot. This allows us to model accurately the effect of channel faults on different protocol messages, and thus better estimate the protocol reliability. The set of faults considered is rather comprehensive, including permanent faults both on the nodes and on the channels, and transient channel faults. Furthermore, the transient channel faults considered are not only symmetric but also strictly omissive asymmetric (SOA), i.e. they also consider faults in which some nodes receive correctly a message, whereas other nodes receive no message at all.

The reliability model of each protocol comprises models for each of the system components: the communication channels and the nodes. For the latter there are actually different models depending on whether the node is a transmitter or a receiver. In addition, to model the effect of SOA faults on the protocols, we use different models for receivers that loose messages upon occurrence of a SOA fault and for receivers that are not affected by SOA faults.

Note that some of the models presented here are only approximate. This is because we abstract the differences among the different protocols with respect to the acknowledgment and the delivery of broadcast messages.

7.3.1.1 Channel model

A channel can be in one of four states: non-faulty (0), with a symmetric transient fault (1), with a SOA transient fault (2) and permanently faulty (3). In state 0, all messages transmitted on the channel are received by all correct nodes. In state 3, no message transmitted on the channel is received by any node. Furthermore, once in state 3 a channel will remain in that state henceforth. The difference between states 1 and 2 is that, in state 1, if a node transmits a message on the channel no node will receive that message, whereas in state 2, some nodes receive the message while other nodes will receive no message.

Figure 7.2 shows the DTMC for a channel. The labels t, pc and soa of the arcs are the probabilities of occurrence of transient, permanent and SOA channel faults respectively.

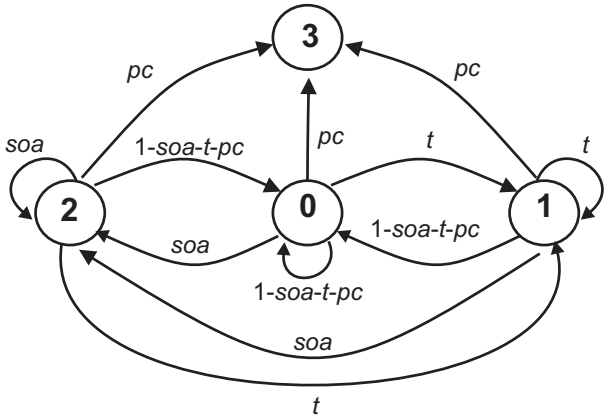


Figure 7.2: Channel Model.

7.3.1.2 Node model

Rather than using a single DTMC, as for the channel model, we model each node as a set of smaller DTMCs that model separately the basic primitives of the RB protocols as well as fault generation. This modularization leads not only to simpler models but also to more scalable models, since these DTMCs can be easily added or removed from the models, as necessary.

- **Transmitter Model:** Figure 7.3 shows the model for a transmitter node. In contrast with the model for a channel, this model has three DTMCs. The transmitter permanent fault (Tpf) DTMC models the operational state of

the node. The transmitter transmission (Ttx) DTMC is used to model the broadcast by a transmitter in a RB round. The SOA DTMC models the occurrence of a SOA channel fault when the transmitter broadcasts its message. The state of a transmitter node is determined by the state of these three DTMCs. E.g., the state (Tpf=1, Ttx=0, SOA=0) indicates that the transmitter failed permanently before it was able to successfully broadcast a message in the current RB round.

Before moving to an explanation the models, we need to explain the notation. In the model in Figure 7.3b. and in other figures below, labels of some transitions are of the form $[G] : p$, where the *guard* G is a predicate and p is a probability. The meaning is that if G is true then the probability of taking the transition is p, otherwise it is 0. Thus the standard notation where a transition is labeled with probability p corresponds to a label of the form $[true] : p$. It should be noted that this extended notation only allows for a more compact representation of the model, which is a standard DTMC.

The Tpf DTMC models the operational state of the node: when in state 0, the node is operating correctly, whereas when in state 1, the node has crashed. The variable p is the permanent fault probability for a node.

The Ttx DTMC models the broadcasting of a message in a RB round: state 1 is active if and only if the transmitter has successfully broadcasted a message in the current RB round. A broadcast is successful only if, at time of the broadcast ($phi = txid$), the channel is not transiently nor permanently faulty ($Channel! = (1, 3)$) and the transmitter is not permanently faulty (Tpf=0). At the end of each RB round (EoRBR) the Ttx DTMC is reset to its initial state.

Note that in the case of a SOA fault, the broadcast is considered successful. This is because receivers not affected by the fault will receive the broadcast message. Instead, the occurrence of the SOA fault is recorded by the SOA DTMC: state 0 means that no SOA fault occurred on the transmitter slot, whereas state 1 means otherwise. The transition from state 0 to state 1 occurs, if at the time of the broadcast($phi = txid$), the channel has a SOA transient fault ($Channel = 2$). Like in the Ttx DTMC, at the end of the RB round, the SOA DTMC is reset to its initial state.

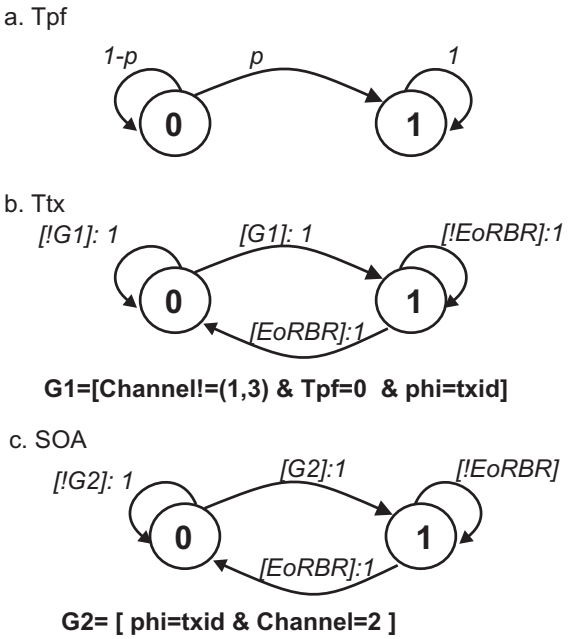


Figure 7.3: Transmitter DTMCs: a. Transmitter permanent fault. b. Transmitter tx. c. SOA fault.

- **Receiver Model:** the receiver model also comprises several DTMCs. However as receivers can be affected either by symmetric or by asymmetric channel faults we use a different model for each of these fault types.

- **Symmetric Receiver Model:** to model the effect of SOA faults, we also assume that SOA faults in a channel always affect the same nodes. This assumption corresponds to the worst case, and has the added benefit of leading to a simpler model: we can use different models depending on whether or not a receiver is affected by a SOA fault. In this section, we describe the model for *symmetric receivers* — receivers that are not affected by SOA faults, i.e. that receive a message transmitted in a slot that has a SOA fault. In the next section, we describe the model for *asymmetric receivers*.

  The model for symmetric receivers comprises 4 DTMCs, as shown in Figure 7.4: the Rpf, the Rrx, the Rtx, and the Delivery DTMCs. The Rpf DTMC models the operational state of the receiver node and is similar to the Tpf DTMC of the transmitter model. The Rrx DTMC is used to record whether the broadcast message was received at least once: state 1 indicates

that the receiver has correctly received the broadcast message in the current execution of the RB protocol. The transition from state 0 to state 1 occurs upon reception ($phi = rxid$) of a successfully transmitted broadcast message ($Ttx = 1$), and depends on the receiver still being operational ($Rpf = 0$).

The Rtx DTMC models the transmission of the acknowledgment message. State 1 indicates that the receiver has successfully transmitted the acknowledgment in the current execution of the RB protocol. The transition from state 0 to state 1 is taken by a non-faulty receiver ($Rpf = 0$) when it successfully transmits ($Channel! = (1, 3)$) an acknowledgment ($phi = rxid$) in a RB round when broadcast message was successfully transmitted ($Ttx = 1$).

Finally, the Delivery (DSR) DTMC models the delivery of a message by the receiver. In the model, the delivery always occurs in the last step ($phi = N$), only if at least one message was received ($Rrx = 1$), the receiver is not faulty ($Rpf = 0$) and the threshold condition of the protocol is satisfied (TSD).

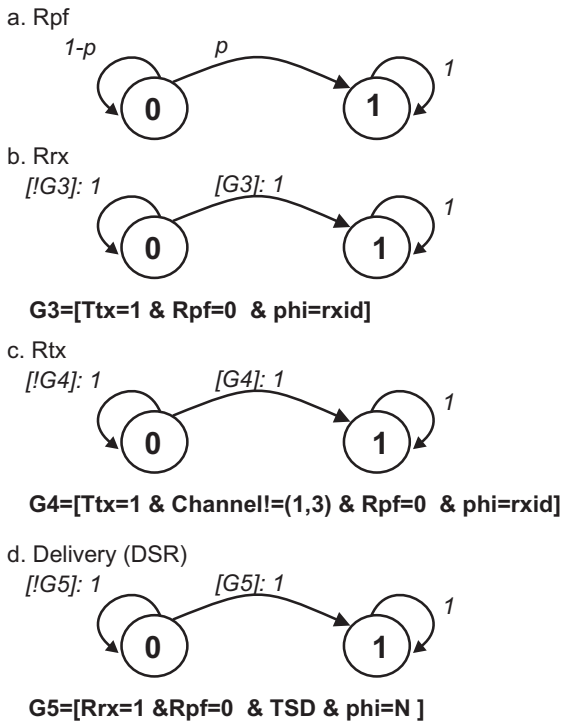


Figure 7.4: Symmetric Receiver DTMCs: a. Receiver permanent fault, b. Receiver rx, c.Receiver tx and d. Delivery (DSR).

Multiple Transmitters To prevent the state explosion problem typical of model checking, the symmetric receiver models for multiple transmitters do not keep track of the reception of individual messages, i.e., they use only a single Rrx DTMC. However, in this case, the guard $G3$ from state 0 to state 1 becomes:

$$Ttx1 = 1 \,\&\, \ldots \,\&\, Txtj = 1 \,\&\, Rpf = 0 \,\&\, phi = rxid$$

That is, the transition now depends on the successful transmission by all transmitters of their broadcast messages. Note that the major effect of this simplification is that if, in every RB round, there is a symmetric fault affecting always a message broadcasted by a different transmitter, the protocol will actually deliver all messages, whereas in our model no node will deliver any message. However, this does not affect the value of the reliability computed by the model.

- **Asymmetric Receiver Model:** The asymmetric receiver model, i.e. the model for receivers that do not receive the message broadcasted when a SOA occurs, is very similar to the one used for symmetric receivers. In particular, the Rpf, the Rtx and the Delivery DTMCs do not change. The only change occurs in the Rrx DTMC, which is now named ARrx and is shown Figure 7.5. Like the Rrx, the ARrx DTMC models the reception of messages, the difference is that the broadcast message will be received only if no SOA fault ($SOA = 0$) occurred on the channel while the message was broadcasted.

  Multiple transmitters Like in the symmetric receiver model, the asynchronous receiver model uses only one ARx DTMC, i.e. one DTMC to keep track of all the messages received from the transmitters, independently of the number of transmitters. To model the possibility of different transmitters being affected by SOA faults in the same RB round, in these models, the guard $G6$ from state 0 to state 1 becomes:

$$Ttx1 = 1 \,\&\, \ldots \,\&\, Txtj = 1 \,\&\, Rpf = 0 \,\&\, phi = rxid$$
$$\&\, SOA1 = 0 \,\&\, \ldots \,\&\, SOAt = 0$$

  That is, the transition now depends on the successful transmission by all

transmitters of their broadcast messages, and on those transmissions not being affected by SOA faults in models. This simplification, leads to an underestimation of the protocols reliability.

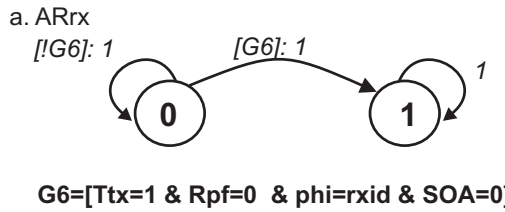


Figure 7.5: DTMC specific to asymmetric receivers: a. Asymmetric Receiver rx.

7.3.2 Model Implementation

In this section we show the implementation of models that we have defined in previous section. It is our intention to provide a light description of model implementation showing the main components used and their codification in PRISM tool. We choose the codification of RBTD(a) protocol to exemplify our implementation.

Basically we distinct the model implementation in a set of components such as modules, constants and formulas. Modules are used to implement the DTMCs directly and also other functionalities such as rounds controlling. The formulas and constants are essential and provide support to make short and plain models.

7.3.2.1 Constants

Constants do not vary during model evaluation and are typically used to calculate the fault probabilities as well as the number of steps necessary to evaluate the reliability of the protocol. Some constants are requested to be informed as input values whereas other have predefined values. We show here only the main constants used in the core of each implemented model such as follows(see Figure 7.6):

Input Value Constants : Frame Size (FS) which the value defines the frame size in bytes, Bit Rate (BR) that indicates the bit rate in Mbps, $BERPG$ that sets the value of BER(Bit Error Rate) in FIT (Failure In Time) [1], SSR which

[1]FIT is commonly used as the unit of reliability ($1FIT = 1failure/1X10^9$ device hour)

```
// These have to be initialized upon invocation
const int FS;                       // Frame size -- in bytes
const int BR;                       // Bit rate in Mbps
const int BERPG;                    // BER in FIT
const double SSR;                   // SOA faults Rate w.r.t Symmetric Channel Faults
// Predefined Constants
const NT=1;                         // Number of transmitters
const NR=3;                         // Number of receivers
const int N = NR + NT;              // Total Number of nodes
const NS=N+1;                       // Number of steps per RB execution
const double BitRate=BR*Mega;       // Bit rate
const double BER=BERPG/Giga;        // BER Rate
// Transient faults probabilities
const double t = BER * FS * 8;      // Frame loss probability for single channel
const double soa=t*(SSR/100);       // SOA occurrence probability for single channel
// Permanent faults probabilities
const double PHw = 0.00001;         // Permanent HW fault rate 1E-5 (faults/hour)
const double PCh = 0.000001;        // Permanent channel fault rate 1E-6 (faults/hour)
// Slot is stated as:
const double Slot = ((FS + 1) * 8)/BitRate;
const double p = PHw*Slot/3600;     // Probability of permanent fault in HW
const double pc = PCh*Slot/3600;    // Probability of permanent fault in a channel
const MaxRoundRetransmission;       // Number of message retransmissions
```

Figure 7.6: Constants

is the symmetric to SOA fault ratio and $MaxRoundRetransmission$ (Maximum number os rounds used to retransmit a message)

Predefined Constants are: NT (Number of Transmitters), NR (Total number of Receivers), NS (Number of Slots), soa (SOA fault rate), PHw (Permanent Hardware fault rate), PCh (Permanent Channel fault rate), $Slot$ (Slot length in bits), t (frame loss probability for single channel), p (Probability of permanent fault in Hardware) and pc (Probability of permanent fault in a channel).

7.3.2.2 Formulas

Formulas are used to simplify some extensive guards and verifications that often appear in models, making models simple and reading attractive. We classify formulas as: state; decision and auxiliary for verification; *State formulas* are helpful to determine some specific states of environment components, e.g. the state of channels. Figure 7.7 shows some formulas used to define the state of two channels ($Ch1$ and $Ch2$) and also formulas that identify the beginning and ending of the protocol execution.

Decision formulas are used to calculate values required to execute model state transitions, e.g. to calculate if the number of received positive acknowledgements reaches a specified threshold (Figure 7.8).

```
// Defines the general state of channels
formula gch = (Ch1=0 | Ch1=3) & (Ch2=0 | Ch2=3);      // No fail for symmetric nodes
formula gchSOA = (Ch1!=0 & Ch2=3) | (Ch1=3 & Ch2!=0); // SOA channel
// Flags
formula EndOfPExecution = RoundCounter=MaxRoundRetransmission;
formula BegOfPExecution = RoundCounter=0;
```

Figure 7.7: State Formulas

```
// Threshold for ACK messages
formula TSD= Rtx1 + Rtx2 + ARtx1 >= (NR)/2;

// True if some confirmation was lost
formula MissSomeAck = Rtx1 + Rtx2 + ARtx1 < NR;
```

Figure 7.8: Decision Formulas

```
formula Agreement = phi=NS+1 & EndOfPExecution & Agg;

formula Agg= (((D1 & Rpf1=0) | Rpf1=1) & ((AD1 & Rpf1=0)|ARpf1=1))|
             (((!D1 & Rpf1=0) | Rpf1=1) & ((!AD1 & ARpf1=0)| ARpf1=1));

formula FlexValidity = phi=NS+1 & EndOfRBExecution  & FlexVa;

formula FlexVa = ( TCtx1=true & (( ((D1 & Rpf1=0)|Rpf1=1)
                  & (( AD1 & ARpf1=0)|ARpf1=1) ) |
                 ( ((!D1 & Rpf1=0)|Rpf1=1) & ((!AD1 & ARpf1=0)|ARpf1=1) ))) | TCtx1=false;
```

Figure 7.9: Auxiliary Verification Formulas Example

Finally, the *auxiliary verification* formulas are used to evaluate the general model state verifying if it satisfies the reliable broadcast properties. Figure 7.9 presents formulas used to verify the both Agreement and Flex-Validity properties based in the delivery of message. The variables $D1$ and $AD1$ are used to indicate if a specific message were delivered or not by a receiver. In this example we assume a configuration of two receivers, one that can be affected only by symmetric faults ($D1$) and another by both symmetric and SOA faults ($AD1$). Note that agreement on delivery occurs only if ($D1$) and ($AD1$) have the same values.

7.3.2.3 Modules

Phases Module: the Phases module (Figure 7.10) is used to synchronize the modules's state transitions and operations during a RB-round. A variable denoted as *phi* is used to enumerates each step of the model. Therefore, modules can only transit to new states only if they are labeled with the current value of

```
module Phases
   phi:[0..NS+1] init 0;
   [step] true -> phi'= func(mod,phi+1,NS+2);
endmodule
```

Figure 7.10: Phases Module

```
module RetransmissionController

  RBRoundCounter:[0..MaxRoundRetransmission] init MaxRoundRetransmission;

  [step] phi=0 -> RoundCounter'= func(mod,RBRoundCounter+1,MaxRoundRetransmission+1);
  [step] phi!=0 -> true;

endmodule
```

Figure 7.11: Retransmission Control

phi. The maximum value of *phi* is denoted by the total number of nodes plus one ($NS + 1$). When *phi* reaches $NS + 1$ than the RB-round is finished.

Retransmission Controller Module: this module (Figure 7.11) is used to count the number of RB-rounds which each protocol execution takes. The number of RB-rounds is bounded by the $MaxRoundRetransmission$ constant.

Channel Module: as seem before the module Channel implements the behavior of a single channel, it includes the probabilities of the channel be transiently, permanently and SOA faulty. Figure 7.12 shows the module channel implementation. The $Ch1$ variable is used to determine the currently state of the channel. The initial state is defined as not faulty (0) and updates the channel state according to the model execution progress. The possible channel faulty states are transient(1), permanent(2) and SOA (3). All of these faults have their rates defined by constants such as the permanent channel fault probability *pc*, the frame loss probability for single channel t and the SOA frame loss probability *soa*. The *pc* is a constant value given by the product of permanent fault rate per hour (Pch) by the slot length. The value of t depends on the imputed frame size (FS) and the BER (Bit Error Rate) values. Due to lack of accurate data about SOA fault rates in the literature, in our models we define *soa* as a percentage of the t that must be externally provided as the constant SSR value. Note that multiple channels can be modeled by instantiation of multiple channel modules.

Transmitter Module: the transmitter module implements the Ttx and Tpf

```
module Channel1
  // 0: channel has no fails
  // 1: channel has a transient fault
  // 2: channel has a permanent fault
  // 3: channel has SOA fault
  Ch1: [0..3] init 0;

  [step] Ch1!=2 -> pc:(Ch1'=2) + t:(Ch1'=1) + soa:(Ch1'=3) + 1-soa-t-pc:(Ch1'=0);
  [step] Ch1=2  -> true;

endmodule
```

Figure 7.12: Channel Module

```
module Transmitter1
  Ttx1 : [0..1] init 0;
  TCtx1: bool init false;
  Tpf1 : bool init false;

  [step] phi=0 | Tpf1 -> (Ttx1'=0);
  [step] phi=t1 & gch & MissSomeAck & !Tpf1-> p:(Tpf1'=true) + 1-p:(Ttx1'=1)&(TCtx1'=true);
  [step] phi=t1 & (!gch | !MissSomeAck) & !Tpf1 -> p:(Tpf1'=true)&(Ttx1'=0) + 1-p:(Ttx1'=0);
  [step] !phi=0 & !(phi=t1 & gch & MissSomeAck & !Tpf1) & !(phi=t1 &
        (!gch | !MissSomeAck) &!Tpf1) & !Tpf1 -> true;

endmodule
```

Figure 7.13: Transmitter module

DTMCs (Figure 7.13) by using the follow set of variables. The transmitter broadcast variable (Ttx) indicates whether a message is broadcast correctly (set as 1) or not (set as 0). The Transmitter permanent fault (Tpf) variable is set to *true* whenever the transmitter experiences a permanent fault. Once the model is executed cyclically (in multiple RB-rounds), it is not possible verify previous values of modules variables. For this reason there is a need to keep information about if messages were correctly broadcast at most once through the multiples RB-rounds. Therefore, to do that we use the $TCtx$ flag and set it to *true* as an indication of message broadcast success.

SOA Module: this module implements the SOA DTMC and its state is stored in a specific variable, i.e $SOA1$ in Figure 7.14. The variable SOA1 is set to true whenever the state of the network is such that a message is affected by a SOA fault; this is represented by the formula $gchSOA$ shown in Figure 7.7. Note that a model needs a SOA module per transmitter, because this module memorizes the occurrence of a broadcast by a single node. Unfortunately formulas cannot be used in this case because they do not hold past values.

```
module MSOA1
 SOA1:bool init false;
      [step] phi=0 -> SOA1'=false;
      [step] phi=t1 & gchSOA -> SOA1'=true;
      [step] phi=t1 & !gchSOA -> SOA1'=false;
      [step] !phi=0 & !(phi=t1 & gchSOA) & !(phi=t1 & !gchSOA) -> true;
endmodule
```

Figure 7.14: SOA module

7.3.2.4 Receiver Modules

As seen in Section 7.3.1.2 receivers can be affected by SOA faults whilst other may not, for this reason we implement symmetric and asymmetric receivers (SOA) in different modules. Figure 7.15 presents the *Symmetric Receiver* DTMCs implementation. As defined in DTMCs the message reception is signaled by Rrx whilst Rtx determines the state of confirmation message.

It is important note that reception and confirmation are modeled in the same step ($phi{=}r1$) as an optimization for state space reduction purposes. Another important aspect is the approach used to model multiple transmitter through the formula gTS. Naturally multiple transmitters represent more messages. Hence more variables are necessary to represent reception of messages in a particular fashion. The increasing of variables makes the models bigger and memory spender due to state space explosion problem. To overcome this problem we compress all Ttx states in the gTS formula (e.g., $gTS = Ttx1\&Ttx2\&..Ttxn$) and allow symmetric receivers indicate the reception $Rrx = 1$ whenever all messages were correctly sent by transmitters. Actually what we do is to delay the delivery of received messages, waiting for a RB-round without failures. Note that this approach does not interfere on confidence of reliability results since we underestimate the probability of faults reaching a reliability smaller than it really is.

The modules for *asymmetric receiver* are similar to the ones used for symmetric receivers. However, in the asymmetric receiver module, the transitions depend on the occurrence of SOA faults, which is memorized by the SOA module (Figure 7.14). The codification used in the implementation of asymmetric receiver module is shown in Figure 7.16. The formula $gSOA$ (defined as $gSOA = SOA1|SOA2|...SOAn$) groups all information about SOA faults occurred during broadcasts. Like the gTS formula it is used to reduce the size of

```
module SR1

  Rrx1:[0..1] init 0;
  Rtx1:[0..1] init 0;
  Rpf1:bool init false;

  [step] phi=r1 & gch & gTS & !Rpf1 -> p:(Rpf1'=true)&(Rrx1'=1)&(Rtx1'=1)
         + 1-p:(Rrx1'=1)&(Rtx1'=1);
  [step] phi=r1 & !gch & gTS & !Rpf1 -> p:(Rpf1'=true)&(Rrx1'=1)
         + 1-p:(Rrx1'=1);
  [step] !(phi=r1 & gch & gTS & !Rpf1) &
         !(phi=r1 & !gch & gTS & !Rpf1) & !Rpf1 -> true;
  [step] Rpf1-> true;

endmodule
```

Figure 7.15: Symmetric Receiver module

```
module AsR1

  AsRrx1:[0..1] init 0;
  AsRtx1:[0..1] init 0;
  AsRpf1:bool init false;

  [step] phi=r2 &  gch & gTS & !gSOA & !AsRpf1 -> p:(AsRpf1'=true)&(AsRrx1'=1)
         &(AsRtx1'=1) + 1-p:(AsRrx1'=1)&(AsRtx1'=1);
  [step] phi=r2 &  gch & gTS &  gSOA & !AsRpf1  -> p:(AsRpf1'=true)
         + 1-p: true;
  [step] phi=r2 & !gch & gTS & !gSOA & !AsRpf1 -> p:(AsRpf1'=true)&(AsRrx1'=1)
         + 1-p:(AsRrx1'=1);
  [step] phi=r2 & !gch & gTS &  gSOA & !AsRpf1  -> p:(AsRpf1'=true) + 1-p: true;
  [step] !(phi=r2 & gch & gTS & !gSOA & !AsRpf1 )
         & !(phi=r2 & gch & gTS & gSOA & !AsRpf1)
         & !(phi=r2 & !gch & gTS & !gSOA & !AsRpf1)
         & !(phi=r2 & !gch & gTS & gSOA & !AsRpf1)
         & !AsRpf1 -> true;
  [step] AsRpf1 ->true;

endmodule
```

Figure 7.16: Asymmetric Receiver module

the model for configurations with multiple transmitters.

7.3.2.5 Delivery Model

The model for delivery of message is generic. It can be used by both asymmetric and symmetric receivers. Model delivery separately allows to easily modify the models according to the protocols' delivery policies. For example, Figure 7.17 presents the implementation of reception used in RBTD(a) protocol where the threshold is specified in the formula TSD. In the RBRD protocols, no extra

```
module Delivery1
  DSR1: bool init false;

  [step] phi=NS & Rrx1=1 & TSD-> DSR1'=true;
  [step] !(phi=NS & Rrx1=1 & TSD) -> true;

endmodule
```

Figure 7.17: Delivery module

information is needed to delivery messages than the TSD is not necessary and can be removed without any changes in the receiver's codification.

7.3.2.6 Model Evaluation

Although PRISM offers symmetry reduction techniques, which can reduce the time required used to evaluate models with symmetry, our models are not perfectly symmetric and therefore we cannot take advantage of these techniques. For this reason we have first proceeded with the evaluation of reliability of the protocols after a single execution instead of after one hour.

We have computed the reliability of one execution of the RB-DuST protocols by evaluating the following PCTL properties:

$$(1 - P = ? [\text{true U} <= N (!Agreement \mid !FlexValidity)])$$

where N, is the number of steps required to execute the RB protocol. $Agreement$ and $FlexValidity$ are auxiliary verification formulas shown in Figure 7.9.

To compute the reliability of the protocols for repeated executions, e.g. a mission of one hour, we use an "auxiliary model", which receives as input parameter the unreliability values for single protocol execution model described in the previous subsection. Figure 7.18 shows the PRISM modules of that auxiliary model. The protocol state is represented by the variable $State$, which can switch from an initial no-violation state ($State = 0$) to a violation state ($State = 1$). The probability of this state transition depends on the number of operational channels c (modeled in the $ChannelsState$ module). More specifically, the probabilities $p1$ and $p2$, are the probabilities of violating the RB specification in a

single execution assuming configurations with one and two operation channels, respectively.

The reliability after one hour of RB-DuST protocols was computed by the following PCTL property:

(1 - P = ? [true U <= N *State*=1])

where N is now the number of protocol executions in one hour. Note that each step of this model represents one execution of the evaluated protocol. The effect of the number of RB-rounds is reflected in the probabilities p1 and p2 and on the number of protocol executions in one hour, N: all other things being equal, the larger the number of RB-rounds in a protocol execution, the smaller N will be.

```
module ChannelsState
// 0: two channels have failed permanently
// 1: one channel has failed permanently
// 2: no channel has failed permanently
  c: [0..2] init 2;
  [step] c=2 -> 2*pc*(1-pc):(c'=1) + pc*pc:(c'=0) + (1-pc)*(1-pc):(c'=2);
  [step] c=1 -> pc:(c'=0) + 1-pc:(c'=1);
  [step] c=0 -> true;
endmodule

module Protocol
   State:[0..1] init 0;   // 0: no violation; 1: violation
   [step] State=0 & c=2 -> p2:(State'=1) + 1-p2:(State'=0);
   [step] State=0 & c=1 -> p1:(State'=1) + 1-p1:(State'=0);
   [step] State=0 & c=0 -> true;
   [step] State!=0 -> true;
endmodule
```

Figure 7.18: Auxiliary model

7.4 Reliability Analysis

In this section we present a reliability analysis of the RB-DuST protocols. This analysis is based on the results obtained from a set of experiments in which we used PRISM to evaluate the models presented in the previous section and to compute the unreliability of the corresponding RB-DuST protocol in one execution. In all these experiments we used parameters values from the automotive domain, the main application domain of DuST networks, extracted from [18] and [52]. Table 7.2 shows the values of the relevant parameters used in all experiments.

Table 7.2: Parameter values used in all RB-DuST experiments.

Parameter	Values (units)
Number of channels	1, 2
Frame Size (FS)	16, 32, 128 (bytes)
Bit Rate	1, 10 (Mbps)
Bit-error rate (BER)	1E-6, 1E-7, 1E-8
Node Permanent Fault Rate (PHw)	1E-5 (faults/hour)
Channel Perm. Fault Rate (PCh)	1E-6 (faults/hour)

7.4.1 Effect of Acknowledgment Policy

The two variants of RBRD protocols, like the two variants of RBTD protocols, differ on the type of acknowledgment message used: RBRD(a) and RBTD(a) use positive acknowledgments, and RBRD(n) and RBTD(n) use negative acknowledgments. There are two crucial differences between these two types of acknowledgment messages. First, with negative acknowledgments a receiver must know *a priori* when it should receive the messages being acknowledged. Second, when the sender of a message being acknowledged does not receive an acknowledgment it usually cannot tell whether that was because the message was correctly received or because the acknowledgment message was lost, and therefore assumes the former, which is more likely. However, FlexRay is able to distinguish silence in the network from the occurrence of a communications error or fault. In addition, this ability and the fact that in FlexRay slots are assigned to message ids even in the dynamic segment make it possible for receivers to detect whether they should have received a frame in a given slot. As a consequence, the reliability of the RBRD(a) and RBRD(n) protocols obtained in our experiments are very similar. The same applies to the reliability of the RBTD(a) and the RBTD(n) protocols. Therefore, in the remainder of this section we will concentrate on the protocols with positive acknowledgment.

7.4.2 Unreliability Contributors

In this subsection we discuss the main factors that affect the reliability of the RB-DuST protocols, and show some results of the experiments we carried out that support our conclusions. The reliability of protocols can be affected observing the follow contributors:

a) **Number of Asymmetric Receivers:** as argued in Section 6.3, the RB protocols satisfy their specification in the presence of symmetric faults only. Therefore, unless our models include asymmetric receivers, their reliability is 1. The number of asymmetric receivers can have different effects on the different protocols. For example in the RBRD(a) protocol, a SOA fault, possibly in conjunction with other faults, may lead to a violation of RB properties, independently of the number of receivers affected by the SOA fault. This is because in this protocol receivers deliver a frame they receive, independently of all other receivers, and, in the presence of SOA faults, it is possible that the messages received by correct receivers be different. On the other hand, in the RBTD(a) protocol, if the number of correct nodes not affected by the asymmetric faults is below the threshold, then no correct node will deliver the message. However, if the number of correct nodes not affected by asymmetric faults is above the threshold, these nodes will deliver the message, whereas nodes affected by SOA faults may not. Thus RB properties such as *Agreement* and *Flex-validity* may also be violated by the RBTD(a) protocol. I.e. a single SOA receiver is enough to induce all reliable protocols to violate their specification.

   In all our models, we assume that the number of receivers affected by SOA faults, i.e. asymmetric receivers, is one. For RBTD protocols, this value corresponds to the worst case, because the threshold becomes effective only when the number of correct nodes minus those affected by SOA faults is smaller than the delivery threshold. Indeed, only in this case will the occurrence of a SOA prevent all correct receivers from delivering the message, and thus ensure agreement between symmetric and asymmetric receivers. Thus, the larger the number of nodes affected by SOA faults less nodes need to fail permanently for the threshold to become effective. Because the probability of permanent faults on nodes is very small, the outcome is that the results obtained for the RBTD protocols are indistinguishable from those of the corresponding RBRD protocols. This is unfortunate, but we could not find field values on the occurrence of SOA faults. For this reason, we have estimated SOA faults rates on communication channels as a percentage of symmetric faults that is given by the *SOA to Symmetric Faults Ratio (SSR)* parameter.

b) **Frame Size:** as the frame size increases the reliability decreases, because

the probability of the messages being affected by transient channel faults increases. Figure 7.19 illustrates how the frame size can influence the reliability of RBTD(a) protocol.

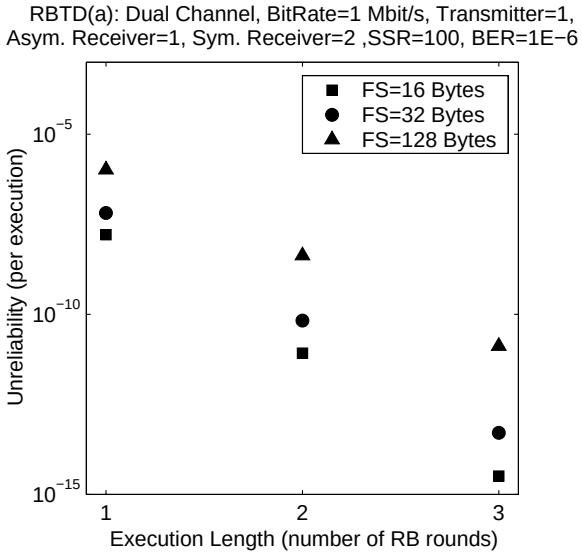


Figure 7.19: RBTD(a) unreliability as a function of the frame size (results per execution).

c) **Bit Rate:** higher bit rates also decrease the reliability, because the number of messages broadcast in a given time interval, and consequently the number of protocol execution increases, thus increasing the probability that at least one execution will violate the RB specification.

d) **BER:** the bit error ratio (BER) is the major contributor to unreliability of the studied communication protocols. As the reliability decreases when the BER values increase (Figure 7.20). In all experiments we have fixed the value of BER to 1e-6. Although some authors consider it an extreme value even for automotive protocols we chose this value because it represents our worst case.

e) **Number of RB-rounds:** the number of RB-rounds determines the number of times the broadcast message may be retransmitted. As expected, the reliability improves when the number of RB-rounds increases. For example, Figure 7.21 shows the unreliability results for the RBTD(a) protocol for different numbers of RB-rounds in the range between one and ten. These results show that, for the parameter values used, the reliability increases

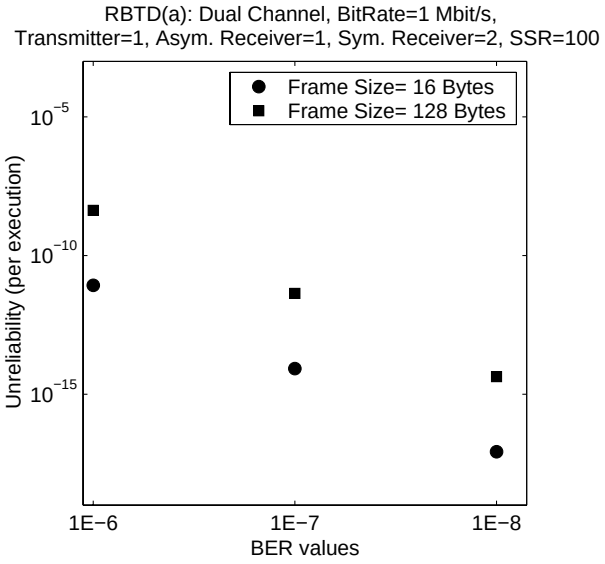


Figure 7.20: RBTD(a) unreliability as a function of the BER

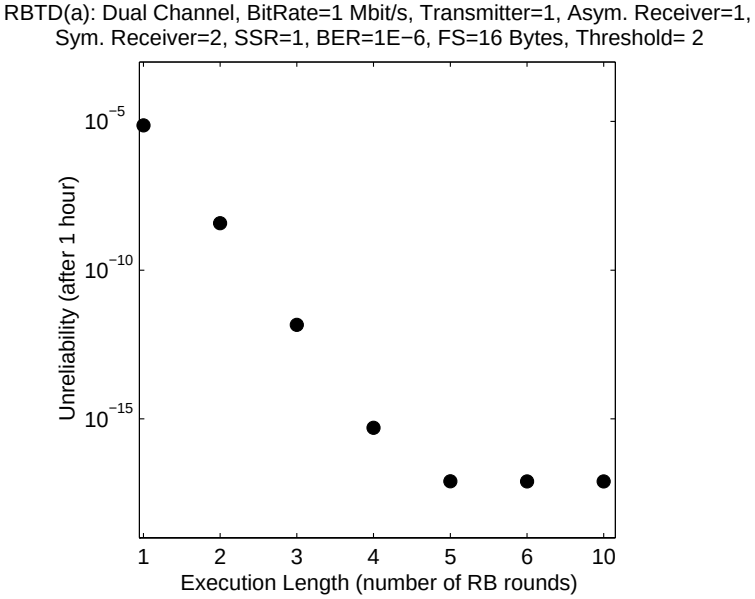


Figure 7.21: RBTD(a) unreliability as a function of the execution length, i.e. number of RB-rounds.

fast when the number of RB-rounds increases from 1 to 5. However, there is no improvement on the reliability when the number of RB-rounds increases above 5. This is because for executions with 5 or more rounds, the dominant factor of the reliability are the permanent faults, both on the channel, and most importantly on the transmitter node, which cannot be overcome by adding rounds, i.e. retransmissions. Thus, increasing the

number of rounds beyond a few, does not improve the reliability, but it may increase the protocol overhead and its latency. This applies not only to the RBTD(a) protocol, but also to the other 3 RB protocols.

f) **Number of Symmetric Receivers:** the reliability of the RB protocols is mostly independent of the number of receivers. This is in part a consequence of the way we model SOA faults. Indeed, because we assume that a SOA fault affects only one node, this means that for all RB protocols the occurrence of one SOA fault in one RB-round and transient communication faults, whether symmetric or asymmetric, in the remaining RB-rounds will lead to a violation of both *Agreement* and *Flex-validity*. In addition, the probabilities of the transient communication faults are essentially independent of the number of symmetric receivers. This is not to say, that the reliability of the RB protocols is completely independent of the number of symmetric receivers, only that they have a very small effect. For example, in the case of RBTD(a), the reliability decreases in an unperceptive way with the number of symmetric receivers. This may be justified with the effectiveness of the threshold mechanism. As already noted the use of thresholds, under our assumptions, is only effective to defeat SOA faults only when the number of operational receivers is equal to the threshold, or enough acknowledgment messages are lost in all RB-rounds. The probability of these conditions being met are very low, and they decrease as the number of receivers increases.

g) **SOA to Symmetric Faults Ratio (SSR):** Figure 7.22 shows, as expected, that the reliability of RBTD(a) increases with the SSR, for all execution lengths (Ex. L.). This is because the RB protocols specified fail only in the presence of asymmetric faults and, by increasing the SSR parameter, we increase the rate of SOA faults and hence the probability of the RB protocols violating the RB specification.

h) **Number of Transmitters:** to model the case where the number of nodes broadcasting in a given RB round, we assume that each receiver acknowledges all transmitters with a single confirmation frame that is sent in the C-phase of the RB round. As shown in Figure 7.23, the unreliability of the RB protocols increases with the number of transmitters. The reason for this variation is that, as the number of transmitters increases, the number of

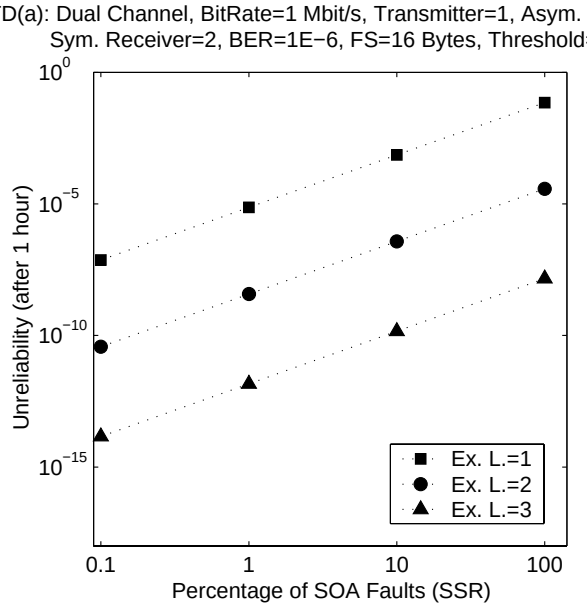


Figure 7.22: Comparison between protocols unreliability when increased SOA fault rates

messages broadcasted in a RB round increases and therefore the probability that at least one of them will be affected by a SOA fault increases. This increase is slightly compensated by the decrease in the number of executions in one hour, because the duration of an RB round increases with the number of transmitters. However, the effect of the increase on the number of messages in a RB round susceptible to SOA faults is clearly dominant. Again, for the parameter values used in these experiments, the differences among the RB protocol variants are not significant.

Effects of Packing acknowledgment in messages: the RB protocols pack acknowledgments to different transmitters in a single confirmation frame for efficiency reasons. However this solution raises some issues upon loss of the confirmation frame. These issues differ from protocol to protocol and depend essentially on whether the protocol uses positive or negative acknowledgments. The first drawback of packing acknowledgments stems from the message overload produced by the loss of any confirmation message (carrying ACKs or NACKs) which forces all transmitters to retransmit in the next round.

The second drawback, with respect to reliability, is the impossibility of nodes detecting the loss of confirmation messages. Protocols with posi-

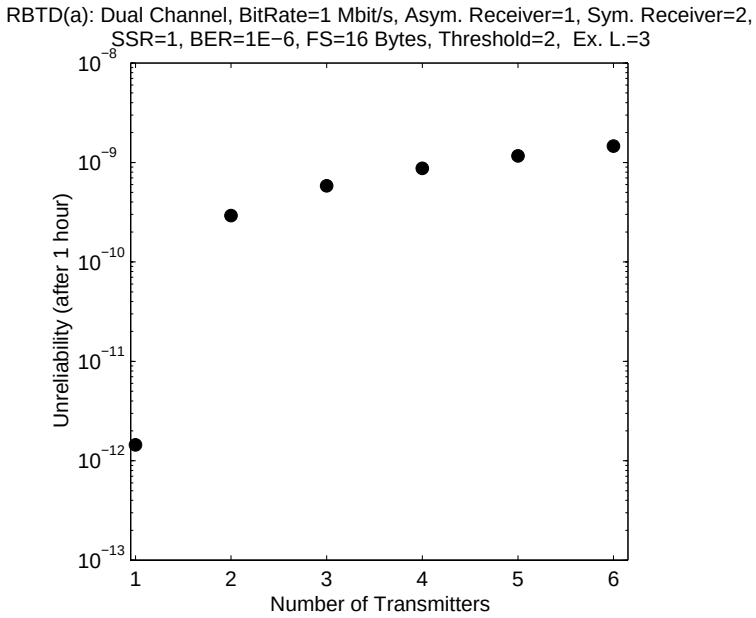


Figure 7.23: Effects of transmitter number on RB protocols reliability

tive acknowledgments are not affected by the loss of confirmation messages because the absence of expected ACKs is detectable [2]. Differently, Reliable broadcast protocols with negative acknowledgements are vulnerable to the loss of confirmation messages and may not present acceptable reliability levels when these losses are not detectable. One such scenario for the RBRD(n) protocol occurs, for example, when a receiver does not receive a message (because of SOA fault on the network) and the transmitter does not receive the NACK sent by that receiver (because of a symmetric fault on the network). Because of these undetectable losses, the transmitter may skip message retransmission thus reducing the reliability of the protocol.

Fortunately, some network protocols such as FlexRay provide means to detect loss of messages increasing the reliability levels of RB protocols that rely on negative acknowledgments. We have carried out some experiments in order to measure the reliability of the RB protocols under networks that are not able to detect message loss. Figure 7.24 shows some results illustrating the reliability reduction of RBTD(n) protocol in scenarios with

[2]However, the RBTD(a) can present a case where some receiver nodes do not receive sufficient acknowledgments to deliver a message. This can be caused by persistent SOA faults on channels. However these succession of faults have low probability occurrence compared with the loss probability of a single data message.

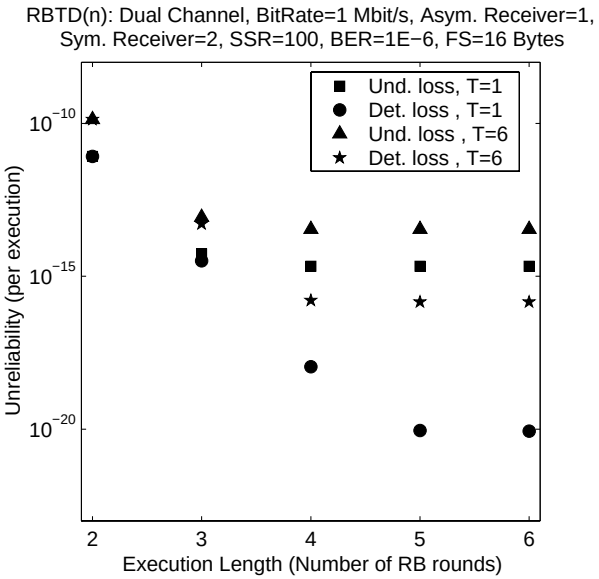


Figure 7.24: RBTD(n) reliability reduction in scenarios with undetectable message loss

(un)detectable loss of messages. As conclusion, there is a notable reliability advantage in being able to detect loss of messages. However, for the parameter values we used, this is true only for configurations with 4 or more RB-rounds.

7.5 Conclusions

In this chapter we have presented a reliability analysis of the family of reliable broadcast protocols designed to take advantage of DuST networks. The reliability analysis was carried out with the probabilistic model checker PRISM. The analysis allows to determine not only whether protocols satisfy the reliability requirements of safety-critical applications, but also the main factors that affect their reliability.

In this chapter we have presented a reliability analysis of the family of reliable broadcast protocols designed to take advantage of DuST networks. It was carried out with the help of the probabilistic model checker PRISM. This analysis has shown that the RB protocols are highly sensitive to asymmetric transient faults. But, by increasing the number of RB-rounds of the RB protocols, their reliability becomes adequate for safety-critical applications. However, this is achieved at the cost of additional bandwidth. Therefore designers have to look for the right

balance between the number of rounds and the desired reliability. Indeed even if the designers decide on adding more RB-rounds, the reliability will not improve indefinitely, because of limits imposed by permanent fault rates.

Finally, we have presented very detailed description of designed models, showing some techniques to make models simple and effective. Our models are reusable and can be used by other researchers as an important reference about how to model and evaluate similar protocols.

Chapter 8

Conclusions and Future Work

This chapter summarizes the research results achieved throughout this thesis, highlighting how the contributions have fulfilled the original research objectives. In addition we present some future research directions that may emerge from this work.

8.1 Conclusions

We have specified protocols for two core services that facilitate the development of safety-critical applications that communicate on top of dual scheduled TDMA communication protocols (DuST).

This new class of communication protocols is likely to become widely deployed, because FlexRay was specified by a consortium of major players in the automotive industry and is likely to become the next generation automotive communication network de facto standard. Furthermore, the flexibility and determinism of DuST protocols in general, and of FlexRay in particular, make it likely that these protocols will also be deployed in the industrial domain.

The protocols proposed in this thesis were designed to take advantage of the dual scheduling scheme in order to achieve better performance than existent solutions,which were not developed for DuST protocols, and at the same time to provide similar reliability levels. The GMP is very good example of how the dual schedule approach can bring benefits, more specifically with respect

to performance and bandwidth consumption, without compromising reliability, what we claim as to be the fundamental assertion of this thesis. We have shown that GMP has an overhead of only two bits per processor per cycle in absence of faults and can tolerates benign failures of up to half of the group members between consecutive executions. Even when there are group membership changes, the overhead of the protocol is lower than that of other protocols that provide the same level of fault tolerance. Moreover, both GMP and RB-DuST protocols were proved to be efficient even in realistic scenarios where generally the specified fault assumptions do not hold.

The validation of presented RB-DuST and GMP protocols comprised the modeling and verification of their safety and liveness properties. In the modeling context there are two possible directions to be followed. One of them leads to design models that can reproduce exactly, or much close, the implementation of the original verified protocol. That is the case of UPPAAL models developed for GMP, such models have brought confidence in correctness of GMP protocol specification but they have led to loss of scalability. More specifically we were able to verify the GMP protocol, especially some properties, only for 6 nodes. However, although this number seems rather small, it has provided a better understanding of the protocol, and it were assumed to be large enough to exhibit all the idiosyncracies of the GMP protocol. Therefore, this gives us a confidence that it works for a larger number of nodes. In the other hand, simple models are scalable and can provide means to verify models with a large number of components, but they provide also less confidence. In a balance, the choice between simple or sophisticated models must be based on the complexity of the protocols. For example, the proposed RB-DuST protocols are quite trivial in comparison to GMP, which have permitted to skip the construction of sophisticated models specifically to evaluate their correctness.

The models developed for reliability analysis are reusable and can be considered a source of information about how to model reliable broadcast and group membership algorithms designed for DuST networks.

The reliability analysis results obtained have shown that GMP and RB-DuST protocols are drastically affected by transient faults. The sensitive analysis has shown the relation between protocols reliability and the different types of faults and also that relation directly depends on the parameters used. We have pre-

sented reliability evaluation assuming what the literature admits to be the worst case values of involved parameters. In response to these aggressive scenarios both the GMP and RB-DuST protocols have presented acceptable reliability levels. These are very important results because, in particular, the GMP can tolerate a variety of faults without execution of a blackout operating mode. Moreover the results of reliability analysis of RB-DuST protocols can be used also by designers to improve the performance and reliability of safety-critical applications by choosing between the different delivery policies and the number of retransmissions necessary to each kind of message broadcast.

8.2 Future Work

The main focus of this thesis is on the specification, validation and reliability analysis of protocols for core safety-related services on top of DuST communications protocols. In spite of all the work we have reported, we believe that there is still a lot of work that can be done. In the following paragraphs we describe two possible directions that our work might take.

A performance analysis of proposed RB-Dust protocols can bring more clarification about the impact of acknowledgment messages. In addition it could be possible to estimate the benefits of a alternative allocation of acknowledgment messages. More specifically, acknowledgments could be scheduled to be sent only by a reduced portion of nodes. This could be interesting because this approach could release time in dynamic segment that can be utilized to broadcast data from other applications.

Another interesting future research consists in the integration between RB-Dust protocols and the GMP. More specifically, the information provided by each protocol could be combined to provide faster fault detection and fast group reconfiguration. This is possible because acknowledgments are important extra information. Indeed reliable broadcast and group membership services are quite different, but it is perceptible that faults detected by reliable broadcast protocols, e.g. those occurred in dynamic segment, can complement the system view provided by the GMP. Moreover, we believe that it could be possible to provide a faster and consistent views, based on the threshold information, in a system where all nodes execute the RBTD protocol.

Bibliography

[1] N. Navet, Y. Song, F. Simonot-lion, and C. Wilwert, "Trends in automotive communication systems," in *Proceedings of the IEEE Volume: 93, Issue:6*, 2005, pp. 1204 – 1223. [cited at p. 1, 2]

[2] Hermann Kopetz and Gunter Grünsteidl, "Ttp- a protocol for fault-tolerant real-time systems," *Computer*, vol. 27, no. 1, pp. 14 – 23, 1994. [cited at p. 2, 10, 13]

[3] *FlexRay Communications System Protocol Specification Version 2.1*, FlexRay Consortium, 2005. [cited at p. xviii, 2, 17, 32, 33, 34]

[4] Rainer Makowitz and Christopher Temple, "Flexray - a communication network for automotive control systems," in *In 6th IEEE International Workshop on Factory Communication Systems - WFCS*, 2006. [cited at p. 2, 13]

[5] J. Ferreira, L. Almeida, J. Fonseca, G. Rodriguez-Navas, and J. Proenza, "Enforcing consistency of communication requirements updates in ftt-can," in *Proceedings of Workshop on Dependable Embedded Systems*, Florence, Italy, October 2003. [cited at p. 2]

[6] John M. Rushby, "Bus architectures for safety-critical embedded systems," in *Proceedings of the 1st International Workshop on Embedded Software*, 2001, pp. 306–323. [cited at p. 2, 13, 17, 18]

[7] Valerio Rosset, Pedro Souto, and Francisco Vasques, "A group membership protocol for communication systems with both static and dynamic scheduling," in *In 6th IEEE International Workshop on Factory Communication Systems - WFCS*, Torino, Italy., June 28-30 2006. [cited at p. xix, 3, 17, 59, 75, 76]

[8] Valerio Rosset, Pedro Souto, and Francisco Vasques, "Formal verification of a group membership protocol using model checking," in *In Proceedings of OTM Conferences - The 9th International Symposium on Distributed Objects, Middleware, and Applications (DOA 2007)*, Vilamoura, Portugal, November 2007, pp. 471–488. [cited at p. 4]

[9] Valerio Rosset, Pedro Souto, and Francisco Vasques, "Formal verification of a group membership protocol using model checking," in *Lecture Notes in Computer Science, On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS*. 2007, vol. Vol. 4803, pp. 471–488, Springer Berlin / Heidelberg. [cited at p. 4, 39]

[10] Valerio Rosset, Pedro Souto, and Francisco Vasques, "A reliability evaluation of a group membership protocol," in *In Proceedings of The 26th International Conference on Computer Safety, Reliability and Security*, Nuremberg, Germany., September 2007, pp. 397–410. [cited at p. 4]

[11] Valerio Rosset, Pedro Souto, and Francisco Vasques, "A reliability evaluation of a group membership protocol," in *Lecture Notes in Computer Science, Computer Safety, Reliability, and Security*, 2008, vol. Vol. 4680, pp. 397–410. [cited at p. 4, 57]

[12] Valerio Rosset, Pedro Souto, and Francisco Vasques, "Reliable communication for dust networks," in *In proceedings of ETFA 2009 - 14th IEEE International Conference on Emerging Techonologies and Factory Automation (To Appear).*, Mallorca, Spain, September 2009. [cited at p. 4]

[13] Gregory V. Chockler, Idid Keidar, and Roman Vitenberg, "Group communication specifications: a comprehensive study," *ACM Comput. Surv.*, vol. 33, no. 4, pp. 427–469, 2001. [cited at p. 8, 12]

[14] F. Cristian, "Reaching agreement on processor-group membership in synchronous distributed systems," *Distributed Computing*, vol. 4, no. 4, pp. 175–188, 1991. [cited at p. 8, 17, 21]

[15] T. Abdelzaher, A. Shaikh, F. Jahanian, and K. Shin, "Rtcast: lightweight multicast for real-time process groups," in *Proceedings of the 2nd IEEE Real-Time Technology and Applications Symposium*, 1996, pp. 250–259. [cited at p. 8]

[16] H. Kopetz, G. Grünsteidl, and J. Reisinger, "Fault-tolerant membership service in a distributed real-time system," in *IFIP WG10.4 Int'l Working Conference on Dependable Computing for Critical Applications*, August 1989, pp. 167–174. [cited at p. 9, 18]

[17] K. H. Kim, H. Kopetz, K. Mori, E. H. Shokri, and G. Grünsteidl, "An efficient decentralized approach to processor-group membership maintenance in real-time lan systems: the prhb/ed scheme," in *Proceedings of the 11th Symposium on Reliable Distributed Systems*, 1992. [cited at p. 9, 18, 38]

[18] Elizabeth Latronico, Paul Miner, and Philip Koopman, "Quantifying the reliability of proven spider group membership service guarantees," in *DSN '04: Proceedings of the 2004 International Conference on Dependable Systems and Networks (DSN'04)*, Washington, DC, USA, 2004, p. 275, IEEE Computer Society. [cited at p. 10, 14, 15, 57, 74, 121]

[19] G. Bauer, H. Kopetz, and W. Steiner, "The central guardian approach to enforce fault isolation in the timed-triggered architecture," in *Proceedings of the 6th International Symposium on Autonomous Decentralized Systems*, 2003, pp. 37–44. [cited at p. 10]

[20] Paul D. Ezhilchelvan and Rogerio de Lemos, "A robust group membership algorithm for distributed real-time systems," *Proceedings of the 11th Real-Time Systems Symposium*, pp. 173 – 179, 1990. [cited at p. 10, 11, 18, 28, 38]

[21] Chris J. Walter, Patrick Lincoln, and Neeraj Suri, "Formally verified on-line diagnosis," *IEEE Transactions on Software Engineering*, vol. 23, no. 11, pp. 684–721, 1997. [cited at p. 11, 12]

[22] Afons Geser and Paul S. Miner, "A new on-line diagnosis protocol for the spider family of byzantine fault tolerant architectures," Tech. Rep. NIA 2003-07/ NASA TM-2004-212432, National Institute of Aerospace and NASA, 2004. [cited at p. 12]

[23] P. Thambidurai and Y.-K. Park, "Interactive consistency with multiple failure modes," in *Proceedings of the 7th Symposium on Reliable Distributed Systems*, 1988, pp. 93–100. [cited at p. 12]

[24] Vassos Hadzilacos and Sam Toueg, "A modular approach to fault-tolerant broadcasts and related problems," Tech. Rep. TR94-1425, 1994. [cited at p. 12, 86]

[25] O. Babaoglu and R. Drummond, "Streets of byzantium: Network architectures for fast reliable broadcasts," *IEEE Trans. Softw. Eng.*, vol. 11, no. 6, pp. 546–554, 1985. [cited at p. 12, 13]

[26] Pankaj Jalote, *Fault tolerance in distributed systems*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1994. [cited at p. 12, 86]

[27] Luís Miguel Pinho and Francisco Vasques, "Reliable real-time communication in can networks," *IEEE Trans. Comput.*, vol. 52, no. 12, pp. 1594–1607, 2003. [cited at p. 13]

[28] J. Rufino, P. Verissimo, G. Arroz, C. Almeida, and L. Rodrigues, "Fault-tolerant broadcasts in can," *Fault-Tolerant Computing, 1998. Digest of Papers. Twenty-Eighth Annual International Symposium on*, pp. 150–159, Jun 1998. [cited at p. 13]

[29] C. Ryan, D. Heffernan, and G. Leen, "Interactive consistency on a time-triggered real-time control network," *Industrial Informatics, IEEE Transactions on*, vol. 2, no. 4, pp. 242–254, Nov. 2006. [cited at p. 13]

[30] G. Leen and D. Heffernan, "Ttcan: a new time-triggered controller area network," *Microprocessors and Microsystems*, vol. 26, no. 2, pp. 77 – 94, 2002. [cited at p. 13]

[31] Paul Miner, Alfons Geser, Lee Pike, and Jeffery Maddalon, "A unified fault-tolerance protocol," in *International Conferences on Formal Techniques, Modeling and Analysis of Timed and Fault-Tolerant Systems - FORMATS 2004*, Yassine Lakhnech and Sergio Yovine, Eds. 2004, vol. 3253 of *Lecture Notes in Computer Science*, pp. 167–182, Springer, Available at url http://www.cs.indiana.edu/ lepike/pub_pages/unified.html. [cited at p. 13]

[32] Stefan Pleisch, Olivier Rütti, and André Schiper, "On the specification of partitionable group membership," pp. 37–45, 2008. [cited at p. 14]

[33] Elizabeth Latronico and Philip Koopman, "Design time reliability analysis of distributed fault tolerance algorithms," in *DSN '05: Proceedings*

of the 2005 International Conference on Dependable Systems and Networks (DSN'05), Washington, DC, USA, 2005, pp. 486–495, IEEE Computer Society. [cited at p. 15]

[34] David Powell, "Failure mode assumptions and assumption coverage," in *22nd Annual Intl. Symposium on Fault-Tolerant Computing (FTCS '92)*, July 1992, pp. 386–395. [cited at p. 15]

[35] J. Ferreira, P. Pedreiras, L. Almeida, and J. Fonseca, "The ftt-can protocol: improving flexibility in safety-critical systems," in *IEEE Micro (special issue on Critical Embedded Automotive Networks)*, July/August 2002, vol. 22, pp. 46–55. [cited at p. 17]

[36] Hermann Kopetz and Günther Bauer, "The time-triggered architecture," *Proceedings of the IEEE, Special Issue on Modeling and Design of Embedded Software*, Oct. 2001. [cited at p. 18]

[37] Shmuel Katz, Patrick Lincoln, and John M. Rushby, "Low-overhead time-triggered group membership," in *Proceedings of the 11th International Workshop on Distributed Algorithms*, 1997, pp. 155–169. [cited at p. 18]

[38] A. Albert, "Comparison of event-triggered and time-triggered concepts with regards to distributed control systems," in *Embedded World Conf*, Germany, 2004. [cited at p. 18]

[39] Nancy A. Lynch, *Distributed Algorithms*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996. [cited at p. 19, 59]

[40] Fred B. Schneider, "Implementing fault-tolerant services using the state machine approach: a tutorial," *ACM Comput. Surv.*, vol. 22, no. 4, pp. 299–319, 1990. [cited at p. 23]

[41] Leslie Lamport, "Using time instead of timeout for fault-tolerant distributed systems.," *ACM Trans. Program. Lang. Syst.*, vol. 6, no. 2, pp. 254–280, 1984. [cited at p. 23]

[42] G. Cena and A. Valenzano, "On the properties of the flexible time division multiple access technique," *Industrial Informatics, IEEE Transactions on*, vol. 2, no. 2, pp. 86–94, May 2006. [cited at p. 32]

[43] C. Norris Ip and David L. Dill, "Better verification through symmetry," *Form. Methods Syst. Des.*, vol. 9, no. 1-2, pp. 41–75, 1996. [cited at p. 40]

[44] Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi, "Uppaal – a tool suite for automatic verification of real–time systems," in *Proc. of Workshop on Verification and Control of Hybrid Systems III*. Oct. 1995, number 1066 in Lecture Notes in Computer Science, pp. 232–243, Springer–Verlag. [cited at p. 40, 52]

[45] Sergio Yovine, "Model checking timed automata," in *Lectures on Embedded Systems, European Educational Forum, School on Embedded Systems*, London, UK, 1998, pp. 114–152, Springer-Verlag. [cited at p. 40]

[46] Martijn Hendriks, Gerd Behrmann, Kim G. Larsen, and Frits W. Vaandrager, "Adding symmetry reduction to uppaal," in *Formal Modeling and Analysis of Timed Systems (FORMATS 2003)*. 2004, vol. 2791 of *LNCS*, pp. 46–59, Springer Berlin / Heidelberg. [cited at p. 41, 49]

[47] Cinzia Bernardeschi, Alessandro Fantechi, and Stefania Gnesi, "Model checking fault tolerant systems," *Software Testing, Verification and Reliability.*, vol. 12, no. 4, pp. 251–275, 2002. [cited at p. 49]

[48] M. Kwiatkowska, G. Norman, and D. Parker, "Quantitative analysis with the probabilistic model checker prism," 2005, vol. 153, pp. 5–31. [cited at p. 58, 60, 72, 102]

[49] Nancy A. Lynch, *Distributed Algorithms*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996. [cited at p. 59, 102]

[50] J. K. Muppala, R. Fricks, and K. S. Trivedi, *Computational Probability*, chapter Techniques for System Dependability Evaluation, pp. 445–479, Kluwer Academic Publishers, 2000. [cited at p. 60]

[51] M. Kwiatkowska, G. Norman, and D. Parker, "Symmetry reduction for probabilistic model checking," vol. 4114, pp. 234–248, 2006. [cited at p. 60]

[52] P. Peti, R. Obermaisser, A. Ademaj, and H. Kopetz, "A maintenance-oriented fault model for the decos integrated diagnostic architecture," in *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 2*, Washington, DC, USA, 2005, p. 128.2, IEEE Computer Society. [cited at p. 74, 75, 81, 121]

[53] Justin Ray and Philip Koopman, "Efficient high hamming distance crcs for embedded networks," pp. 3–12, 2006. [cited at p. 85]

[54] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM: Probabilistic symbolic model checker," in *Proc. Tools Session of Aachen 2001 International Multiconference on Measurement, Modelling and Evaluation of Computer-Communication Systems*, P. Kemper, Ed., September 2001, pp. 7–12, Available as Technical Report 760/2001, University of Dortmund. [cited at p. 102]