The University of Osaka
Institutional Knowledge Archive

Title	An Autonomous Decentralized Architecture with Agreement Protocols for Safety-Critical Embedded Distributed Control Systems
Author(s)	櫻井, 康平
Citation	大阪大学, 2014, 博士論文
Version Type	VoR
URL	https://doi.org/10.18910/34575
rights	
Note	

The University of Osaka Institutional Knowledge Archive : OUKA

https://ir.library.osaka-u.ac.jp/

The University of Osaka

An Autonomous Decentralized Architecture with Agreement Protocols for Safety-Critical Embedded Distributed Control Systems

Submitted to

Graduate School of Information Science and Technology

Osaka University

January 2014

Kohei SAKURAI

ABSTRACT

Embedded control systems are widely equipped in current industrial products such as home appliances, automobiles, trains, and power plants. They need to have hard real-time and mission-critical capabilities, and to be more severely restricted in cost and available hardware resources than computer systems in the information technology industry.

Automotive control systems, out of the various embedded control systems, impose particularly severe restrictions on the cost because of the scale of mass production, while recent advances in electronic control using embedded controllers should enable more sophisticated vehicle control systems that are aimed toward autonomous driving. One of the emerging systems is an X-by-Wire system, where driving, steering, and braking are electrically and electronically controlled synthetically, that further enhances vehicle driving performance and safety. Since control of acceleration, steering, and braking has a great influence on the safe operation of vehicles, X-by-Wire systems need to be extremely dependable. Various controllers, sensors, and actuators in this system cooperate with one another through a communication network. We take X-by-Wire systems as an example application of safety-critical embedded distributed control systems in this dissertation.

A technical challenge to X-by-Wire systems is that they are restricted by limited costs in mass production to achieve fault-tolerance. Therefore, the goal of this dissertation is to propose suitable solutions that can satisfy not only high dependability but also cost-effectiveness for automotive safety-critical distributed control systems.

We first propose a novel architecture that incorporates the concept of autonomous decentralized systems to accomplish this goal. This architecture allows all nodes in the system including sensor and actuator nodes to obtain the shared information required for vehicle control through the communication network and to autonomously execute backup control if some node in the system fails. Therefore, the proposed architecture can be fail-operational even though it does not have expensive fail-operational nodes with redundant hardware. This approach to dependability through reduced-redundancy is also applied to the node level by taking into consideration the node function. We propose a validity check method instead of dual redundancy to detect faults in actuator nodes. We demonstrate that the proposed system and node level architectures can be applied to actual automotive brake and steering control systems and that they satisfy both requirements of cost-effectiveness and dependability. Our estimation reveals that the system cost can be reduced by approximately from 20 to 30% due to the proposed autonomous decentralized architecture and optimal node hardware architecture, which contributes to a substantial cost reduction for automotive control systems.

Although the autonomous decentralized architecture satisfies competing demands, we point out that some coordination scheme in this architecture, i.e., an agreement protocol, is required to accurately identify failed nodes so that disagreements in the control mode can be avoided. To provide the coordination scheme, we propose a membership protocol as an agreement protocol for safety-critical distributed systems. In contrast to related work, our membership protocol tolerates simultaneous and non-fail-silent (Byzantine) faults and can flexibly be implemented in time-triggered systems as a middleware component. Important properties such as correctness, completeness, and consistency are defined for the proposed membership protocol and are proved by hand.

With a widely used time-triggered communication network in the automotive industry, we developed a prototype Brake-by-Wire system incorporating the proposed autonomous decentralized architecture and membership middleware in a realistic hardware and software environment for automotive control systems. Although we found that the prototype system could persevere in practical use, the results from evaluating the performance of this prototype system indicated that the computation overhead for the membership middleware was prohibitively large and that the execution time required for the voting process increased along with the number of nodes in the system.

To resolve these problems, we further propose novel lightweight membership protocols, which are based on what we call voting sharing and clustering approaches. These approaches can reduce the computation overhead and the communication bandwidth for the membership protocol. Our experiments revealed that the execution time for the voting process in the voting sharing approach was reduced by approximately 60% compared with the original protocol for eight nodes. Following proofs of the same properties as the original protocol, we investigate advantages and disadvantages of the three proposed membership protocols in terms of computation and communication overhead, diagnosis latency, and fault tolerance. Our analysis shows a tradeoff between the overhead and fault tolerance. The lightweight protocols incur degradation in diagnosis accuracy in exchange for the reduction of the computational overhead. We provide additional mechanisms such as rotating voters and self-accusation to mitigate this problem.

Finally, we propose a customizable formal model of generic time-triggered systems to support key system design processes such as task scheduling, test case generation, and verification. Because the proposed formal model has a modular architecture, it can be reused and easily customized, which can reduce the model development costs for industrial practitioners. We demonstrate a prototype implementation of the formal model with the SAL (Symbolic Analysis Laboratory) language and present some use cases using the SAL tool suite. The proposed membership protocols were model-checked in a use case of verification, and the design correctness of the protocols was guaranteed.

LIST OF MAJOR PUBLICATIONS

(1) Kohei Sakurai, Nobuyasu Kanekawa, Kunihiko Tsunedomi, Shoji Sasaki, Katsuya Oyama, Takanori Yokoyama, and Mitsuru Watabe, High performance and cost-effective electronic controller architecture for powertrain systems, In *Proceedings of SAE 2004 World Congress: In-Vehicle Network Session*, Paper Number: 2004-01-0209, March 2004.

(2) Kohei Sakurai, Yuichiro Morita, Kentaro Yoshimura, Nobuyasu Kanekawa, Kotaro Shimamura, Kenichi Kurosawa, and Yoshiaki Takahashi, Cost-effective and fault tolerant vehicle control architecture for X-by-Wire systems (Part 2: Implementation design), In *Proceedings of SAE 2005 World Congress: In-Vehicle Network Session*, Paper Number: 2005-01-1543, April 2005.

(3) Kentaro Yoshimura, Kohei Sakurai, Yuichiro Morita, Nobuyasu Kanekawa, Kenichi Kurosawa, Yoshiaki Takahashi, Shigetoshi Sameshima, and Akitoshi Shimura, A dependable and cost-effective vehicle control architecture for X-by-wire systems based on autonomous decentralized concept, In *Supplemental Volume of the 2005 International Conference on Dependable Systems and Networks (DSN 2005)*, pp. 130–138, June 2005.

(4) Kentaro Yoshimura, Kohei Sakurai, Yuichiro Morita, Kenichi Kurosawa, Yoshiaki Takahashi, Shigetoshi Sameshima, and Akitoshi Shimura, A dependable E/E ar-

chitecture for X-By-Wire systems based on autonomous decentralized concept, In *Proceedings of the 12th International Congress on Electronic Systems for Vehicles*, VDI Berichte 1907, pp. 523–534, October 2005.

(5) Kohei Sakurai, Masatoshi Hoshino, Yuichiro Morita, and Yoshiaki Takahashi, Design and implementation of middleware for network centric X-by-Wire systems, In *Proceedings of* SAE 2006 World Congress: In-Vehicle Software Session, Paper Number: 2006-01-1326, April 2006.

(6) Kohei Sakurai, Masahiro Matsubara, Marco Serafini, and Neeraj Suri, Dependable and cost-effective architecture for X-by-Wire systems with membership middleware, In *Proceedings of FISITA 2008 World Automotive Congress*, Paper Number: F2008-05-048, September 2008.

(7) Kohei Sakurai, Péter Bokor, and Neeraj Suri, Aiding modular design and verification of safety-critical time-triggered systems by use of executable formal specifications, In *Proceedings of the 11th IEEE High Assurance Systems Engineering Symposium (HASE 2008)*, pp. 261–270, December 2008.

(8) Masahiro Matsubara, Takao Kojima, Kotaro Shimamura, Nobuyasu Kanekawa, and Kohei Sakurai, Node status monitoring and state transition mechanism for network centric X-by-Wire systems, In *Proceedings of the 9th IEEE International Symposium on Autonomous Decentralized Systems (ISADS 2009)*, pp. 1–6, March 2009.

(9) Kohei Sakurai, Masahiro Matsubara, and Masatoshi Hoshino, Membership middleware for dependable and cost-effective X-by-wire systems, *SAE International Journal of Passenger Cars - Electronic and Electrical Systems*, vol. 1, no. 1, pp. 180–186, April 2009.

(10) Kohei Sakurai, Marco Serafini, Péter Bokor, and Neeraj Suri, Design and formal

verification of membership middleware for dependable automotive network systems, In *Proceedings of the 14th International Congress on Electronic Systems for Vehicles*, VDI Berichte 2075, pp. 399–410, October 2009.

(11) Masahiro Matsubara, Kohei Sakurai, Fumio Narisawa, Masushi Enshoiwa, Yoshio Yamane, and Hisamitsu Yamanaka, Model checking with program slicing based on variable dependence graphs, In *Proceedings of the 1st International Workshop on Formal Techniques for Safety-Critical Systems (FTSCS 2012)*, pp. 56–68, November 2012.

(12) Masahiro Matsubara, Kohei Sakurai, Fumio Narisawa, Masushi Enshoiwa, Yoshio Yamane, and Hisamitsu Yamanaka, Application of model checking to automotive control software with slicing technique, In *Proceedings of SAE 2013 World Congress: Model-Based Design and In-Vehicle Software Session*, Paper Number: 2013-01-0436, April 2013.

(13) Kohei Sakurai, Masahiro Matsubara, and Tatsuhiro Tsuchiya, Voting sharing: An approach to reducing computation time for fault diagnosis in time-triggered systems, *IEICE Transactions on Information and Systems*, vol. E97-D, no. 2, February 2014 (to be published).

ACKNOWLEDGMENTS

During the course of this study, I have been fortunate to receive invaluable assistance from many individuals.

I deeply appreciate my supervisor Professor Tatsuhiro Tsuchiya, who has continuously inspired me and provided me with valuable insights and advice throughout this work. I am also grateful to Emeritus Professor Tohru Kikuno for his encouragement before I enrolled in the doctoral course at Osaka University.

I would also like to thank the members of my dissertation review committee, particularly Professor Masaharu Imai, Professor Toshimitsu Masuzawa, and Associate Professor Masanori Hashimoto for their invaluable comments and constructive criticism of this dissertation.

Furthermore, my gratitude goes to the members of Tsuchiya Laboratory, especially to Assistant Professor Hideharu Kojima for the productive discussions on this work, and Mr. Hirofumi Terada, who is also a colleague at Hitachi, Ltd., for his encouragement.

A significant part of this work has also been done with researchers of the DEEDS (Dependable Embedded Systems and Software) group in Technische Universität Darmstadt in Germany. I would like to express my sincere gratitude to Professor Dr. Neeraj Suri, Dr. Péter Bokor, and Dr. Marco Serafini for all the valuable discussions I had with them on dependable distributed systems and formal methods.

Some research work I have engaged in Hitachi, Ltd. has been helpful to me in completing this dissertation. I would therefore like to express my thanks to Dr. Takashi Hotta, Mr. Atsushi Kawabata, Dr. Toshiharu Nogi, Mr. Takaomi Nishigaito, Dr. Masahiko Amano, Mr. Yoshiaki Takahashi, and Mr. Kenichi Kurosawa for giving me the opportunity of doing research on automotive safety-critical control systems. I would also like to acknowledge Dr. Hiroyuki Mikami and Mr. Takuya Shiraishi who encouraged me while I was compiling this thesis.

This work has been done collaboratively with many colleagues at Hitachi Research Laboratory, especially Mr. Masahiro Matsubara, Dr. Kentaro Yoshimura, Dr. Nobuyasu Kanekawa, Mr. Kotaro Shimamura, Mr. Masatoshi Hoshino, and Dr. Yuichiro Morita (the last two researchers are currently with Hitachi Construction Machinery). I would like to express my deep appreciation to all of them for the valuable insights they gave me and the productive discussions from the perspectives of industry.

Finally, my sincere appreciation also goes to my family who always supported me throughout difficult times.

CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Scope and Background

1.1.1 Embedded Control Systems

First, we define an embedded control system, which is the scope of this dissertation. An embedded control system is a computer system integrated into some equipment and it achieves the required functionality by executing the specific control operation to the equipment. A model of embedded control systems is shown in Figure 1.1. The system consists of a controller, a controlled object that is often called a plant, sensors, and actuators. The controller is equipped with microcontrollers, on which the embedded software runs. The controller calculates control target values using information from sensors that measure the status of the plant, and drives the actuators based on the calculated control target values.

Embedded control systems are widely used in industrial products such as home appliances, automobiles, trains, and power plants. For instance, the automotive industry recently recognized that electronics and embedded software developments would represent 90% of all new-vehicle innovations. The increasing functional requirements of embedded

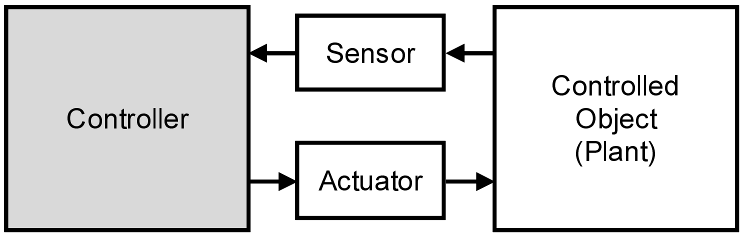


Figure 1.1: Embedded control systems

control systems have led to an enormous increase in software complexity and size.

Embedded control systems are required to have hard real-time and mission-critical capabilities, while they are severely restricted in cost and available hardware resources such as CPU performance, memory size, and communication bandwidth, compared with computer systems like servers and personal computers in the information technology industry.

This dissertation focuses on automotive control systems which impose one of the tightest technical restrictions in the industry due to the scale of mass production. In this dissertation, controllers will be called ECUs, or Electronic Control Units, within the context of automotive control systems.

1.1.2 Trend in Automotive Electronic Control Systems

Automotive control systems have been evolving to improve environmental friendliness (i.e., better fuel efficiency, lower exhaust gas emissions), safety, and passenger comfort. Figure 1.2 outlines the trend in automotive electronic control systems. Electronic control using embedded controllers started in the 1980s for powertrain systems such as engines and transmissions. After that, electronic control was also applied to chassis systems such as brakes, suspensions, and steering in the 1990s. In addition to electronics, electric inverters and motor technologies have been used in electric powertrain and chassis systems.

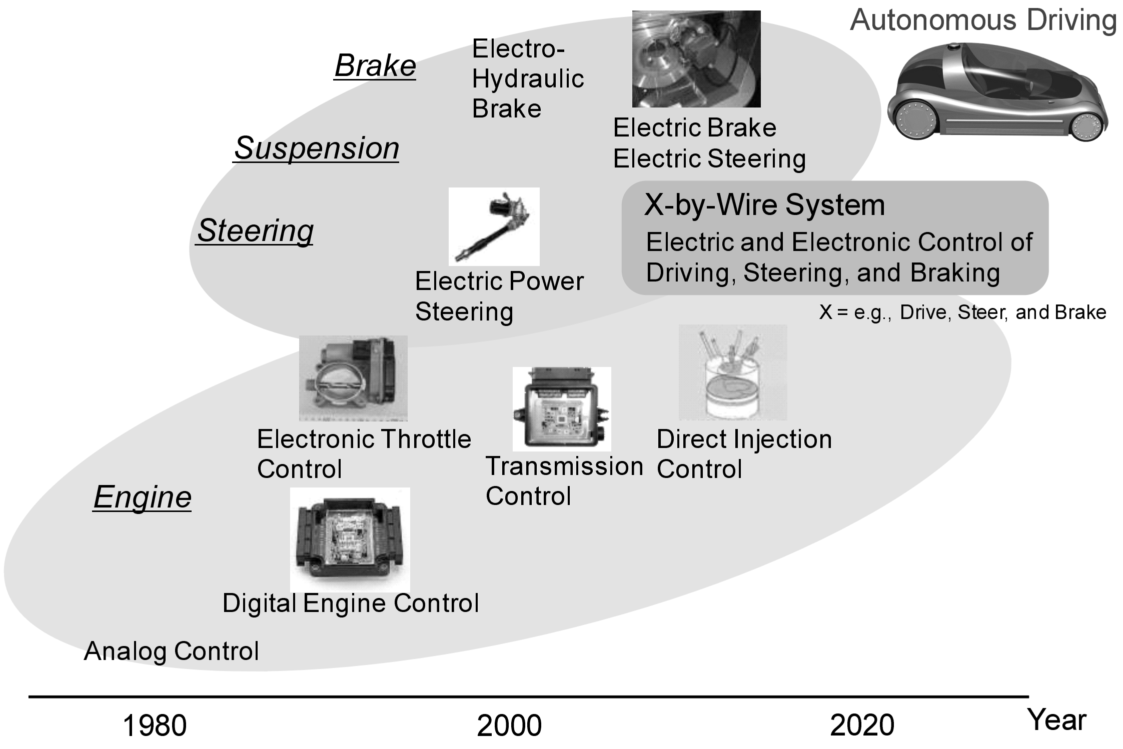


Figure 1.2: Trend in automotive electronic control systems

Recent advances in automotive electronic control systems should enable more sophisticated vehicle control systems toward autonomous driving [1]. Among these systems, *X-by-Wire* systems, where driving, steering, and braking are electrically and electronically controlled synthetically are expected to further enhance vehicle driving performance and safety. The term "X-by-Wire" was derived from "Fly-by-Wire" in aircraft control systems, where "by-Wire" means that systems are controlled by wire, i.e., by electricity, instead of conventional mechanical devices. "X" corresponds to such as Drive, Brake, and Steer. X-by-Wire systems are expected to reduce vehicle weight and increase cabin space due to the absence of mechanical links, as well as enhance vehicle controllability.

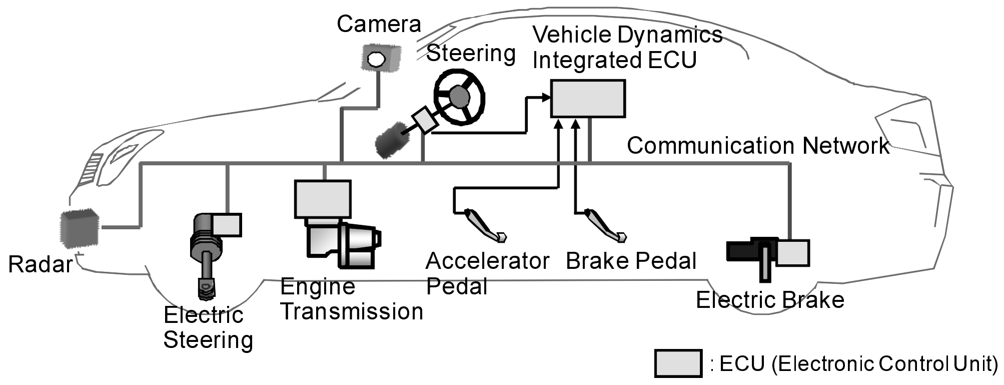


Figure 1.3: Automotive X-by-Wire system

1.1.3 Automotive X-by-Wire Systems

An example of the X-by-Wire system architecture is described in Figure 1.3. Various controllers (ECUs), sensors such as a camera, radar, and actuators including electric brakes and steering motors cooperate with one another through a communication network, which attains integrated control of vehicle dynamics. The vehicle dynamics integrated ECU executes integrated control using information on driver operations and the external environment, and sends control target values to the actuators. Braking and steering are electrically and electronically controlled and thus system dependability is no longer guaranteed by conventional mechanical links such as hydraulic brake hoses and steering columns to transmit driver operations to braking and steering mechanisms. Therefore, we can regard X-by-Wire systems as *safety-critical distributed control systems*. An example application in this dissertation is automotive X-by-Wire systems.

1.2 Motivation and Objective

X-by-Wire systems need to be highly dependable since acceleration, steering, and braking control have a great influence on the safe operation of vehicles. Several studies on the reliability of X-by-Wire systems have been done [2, 3]. However, the cost to implement fault-tolerance is limited in mass produced X-by-Wire systems in the automotive industry. Furthermore, available hardware resources such as CPUs, memories and communication bandwidths are severely restricted. These constraints differ from those in transportation or industrial systems domains like aviation, trains, and power plant systems, where relatively expensive systems with high redundancy and sufficient hardware resources are acceptable. Therefore, the goal of this dissertation is to propose suitable solutions that can fulfill not only high levels of dependability but also cost-effectiveness for automotive safety-critical systems.

1.3 Contributions

1.3.1 Autonomous Decentralized Architecture

Because acceleration, steering, and braking control have a great influence on the safe operation of vehicles, X-by-Wire systems are required to be highly dependable. The conventional approach to improving dependability is to have component level redundancy, where nodes in the system are designed to be fail-operational with such as triple or more redundancy and to keep on operating when faults occur. Several studies on the reliability of X-by-Wire systems with this approach have been done [2, 3]. On the other hand, to mass-produce systems for various vehicle segments, the cost cannot be excessive to implement fault-tolerance.

We take a reduced-redundancy approach with system level redundancy instead of component level redundancy to balance these competing requirements of cost-effectiveness and dependability in automotive control systems. We propose a novel architecture that incorporates the concept of autonomous decentralized systems [4]. Because no general methods of applying the concept of autonomous decentralized systems have yet been established, it is necessary to develop a suitable architecture for individual domains of application.

In the conventional architecture, all the control functions and information such as the sensing data from the driver's acceleration, braking, and steering operations are centralized in the controller node. In contrast, in the proposed architecture, all nodes related to vehicle control, including sensor and actuator nodes, share various data required for control through the communication network, and each node autonomously obtains or broadcasts the necessary data from or to the network. If a certain node stops operating because of some fault, the remaining normal nodes autonomously execute a backup control function to maintain at least the minimum functionality necessary for the system using the shared data. Therefore, the proposed architecture can tolerate the existence of failed nodes and thus does not need expensive fail-operational nodes with triple or more redundant architectures, which satisfies the requirements of cost-effectiveness as well as dependability. We demonstrate that the proposed architecture can be applied to actual automotive brake and steering control systems.

We apply this approach of reduced-redundancy dependability to the node level. Nodes in distributed systems need to be fail-silent so that a failed node does not interfere with communication between fully functional nodes. Conventional fail-silent nodes are designed with dual redundant architecture to detect faults. To further reduce redundancy, the concept of an output validity check is proposed for actuator control nodes. Low performance inexpensive sub-microcontrollers can be used in this concept to diagnose the main microcontrollers. The sub-microcontrollers only compare target control values and actual actuator output instead of rigidly checking the main microcontroller's execution as is done in a dual redundant architecture. The hardware architecture of individual controllers is optimized in this way according to their functions.

We estimate the system cost reduction by the proposed autonomous decentralized architecture and optimal node hardware architecture, and show that the system cost can be reduced by approximately from 20 to 30%, which contributes to a substantial cost reduction for automotive control systems.

1.3.2 Membership Protocol

Autonomous backup control in the autonomous decentralized architecture is based on the accurate identification of failed nodes. However, since there is no master node to monitor the status of nodes in the system, node status monitoring or diagnosis function in each node plays a key role for fault-tolerance. Therefore, some coordination scheme, i.e., an agreement protocol, is required to accurately identify the failed nodes and ensure consistency in views on available nodes for all the normally functioning nodes so that disagreements in the control mode can be avoided. We propose a *membership protocol* as an agreement protocol for safety-critical distributed systems to resolve this issue.

The membership protocol is a functionality that provides a consistent view of active nodes to each node. Time-triggered (TT) communication platforms such as FlexRay [5], TTP/C [6], TT-Ethernet [7], and SAFEbus [8] are increasingly being applied to safety-critical distributed control systems. Although FlexRay has widely been applied to automotive control systems, it does not specify a membership protocol in its standard specifications and this protocol remains as a user dependent functionality. The TT-Ethernet does not provide a standardized protocol either. The AUTOSAR (AUTomotive Open System ARchitecture) [9], which is a worldwide de facto standard specification for automotive electric/electronic systems, neither defines membership services. TTP/C, on the other hand, which is used in aerospace systems, has a membership protocol. However, the protocol is implemented in hardware and has been designed for dedicated applications. Furthermore, several membership protocols have been proposed [10–12], or formally verified [13, 14].

Many previous membership protocols for TT systems only assumed fail-silent nodes and single fault. We also assume fail-silence on the communication protocol level, as was discussed earlier. However, application programs in reality might send semantically erroneous messages because of, e.g., corrupted memory or failure in processing units, even though the messages conform to the communication protocol specifications. Several faults might also occur simultaneously. In contrast, Serafini et al. have proposed protocols that do not rely on the single-fault assumption and that can also tolerate non-fail-silent (Byzantine) faults [15]. The protocols can be added to generic TT communication protocols.

Our proposed membership protocol in this dissertation also tolerates simultaneous and Byzantine faults and can be flexibly implemented in TT systems as a middleware component [16]. We further enhance its real-time capabilities and aim at implementing the protocol in realistic automotive control systems. Each node in the membership protocol locally evaluates the status of other nodes in the system and exchanges a local view, which we call a local syndrome, with all nodes. Then, every node identifies the failed node by voting on the exchanged local syndromes. We propose a pipeline-like method of executing the protocol to improve its real-time capabilities, where a fault detected in a certain TT communication round can be identified in the next round. The membership middleware in practical X-by-Wire systems should coexist with real-time critical application programs such as motor control on microcontrollers with restricted resources. We developed a prototype Brake-by-Wire system that incorporated the proposed autonomous decentralized architecture and membership protocol, and clarified that the prototype system could persevere in practical use.

The results we obtained from evaluating the performance of this prototype system, however, revealed that the computational overhead incurred by membership functionality was unacceptably large and it increased along with the number of nodes in the system.

The membership overhead has to be small enough so that vehicle control applications can use sufficient CPU resources. Therefore, we propose novel lightweight membership protocols, which we call *voting sharing* and *clustering* in this dissertation.

The main idea behind voting sharing is to have each node vote for only one respective node and to share the voting results with all nodes. In the clustering concept, n nodes are logically divided into n/c clusters, where each cluster consists of c nodes. Node p sends a local syndrome only with respect to nodes within the cluster to which node p belongs to all the other nodes in the system. Both approaches can reduce the computation overhead for membership, and the clustering protocol can also decrease the communication bandwidth, compared with the original protocol. The results from our experiments revealed that the execution time for the voting process in the voting sharing protocol was reduced by approximately 60% compared with the original membership protocol for eight nodes.

We investigate advantages and drawbacks of the three proposed membership protocols in terms of computation and communication overhead, diagnosis latency, and fault tolerance. Our analysis shows that there is a tradeoff between the overhead and fault tolerance. The lightweight protocols incur degradation in diagnosis accuracy in exchange for the reduction of the computational overhead. However, it can be mitigated with additional mechanisms such as rotating voters, counter update algorithm, and self-accusation. Despite the drawbacks, we point out that the clustering protocol is a well-balanced protocol among these three protocols when the system consists of large number of nodes and the fault condition is not so severe.

1.3.3 Generic Formal Model of Time-Triggered Systems

The use of *formal methods* is increasingly being advocated to verify general safety-critical systems, e.g., [17]. However, previous work [18, 19] has demonstrated that the correctness of high-level applications in TT systems does not directly imply the correctness of implementation. Consequently, formal techniques dedicated to time-triggered systems are required, especially given their increasing deployment.

Results on the successful formal analysis of TT systems do exist; however, they present specific solutions (e.g., [18–20]), where modeling patterns can only partially be re-used in new projects. It is generally difficult in industry to apply formal methods to the development processes of mass production. Software engineers rarely design formal models of their systems or software from scratch. Therefore, we propose a generalized formal model of TT applications that can be customized, which is not restricted to any dedicated implementation and that can also easily be used by engineers who are not specialists in formal methods.

Furthermore, we seek a unified and formal treatment of general TT systems to guide key system design tasks such as task scheduling, test case generation, and verification. Although deductive reasoning (e.g., theorem proving [17]) is a powerful tool to even verify the complex properties of infinite systems, it cannot directly be used to simulate systems for finding certain execution paths (e.g., counterexamples and test cases). Consequently, we propose executable system specifications to provide further features besides verification by using model checking.

Because the proposed formal model has a modular architecture, it can be reused and easily customized, which can reduce the model development costs for practitioners in industry. Users only need to tailor the corresponding modules to customize the general model.

A prototype formal model was implemented with the SAL (Symbolic Analysis Laboratory) language [21]. We also demonstrate the usability of our prototype with the SAL tool suite by presenting use cases of verification, task scheduling, and test case generation based on an identical model. The proposed membership protocols were model-checked in a use case of verification, and we confirmed the design correctness of the protocols.

1.4 Overview of Dissertation

This dissertation is organized as follows:

Chapter 2 proposes an architecture that incorporates the concept of autonomous decentralized systems to satisfy both requirements of cost-effectiveness and dependability, which is in contrast to the conventional architecture. Fail-silent node architectures according to node functionalities are also discussed. We give the estimation results on the system cost reduction by the proposed autonomous decentralized architecture and optimal node hardware architecture.

Chapter 3 describes a membership protocol as an agreement protocol in distributed systems. We clarify a model of time-triggered systems including fault behaviors, following a discussion on the importance of membership services in the autonomous decentralized architecture. Then, we explain the proposed membership protocol using some examples and pseudo-codes. Important properties for distributed systems such as correctness, completeness, and consistency are defined and proved by hand.

Chapter 4 presents a prototype Brake-by-Wire system that employs the proposed autonomous decentralized architecture and membership protocol. We explain how the membership middleware is implemented on a resource-restricted microcontroller with realistic control application programs. Results obtained from evaluating the performance of the membership middleware are also provided in this chapter.

Taking into account the performance evaluation results, in Chapter 5, we propose lightweight membership protocols, voting sharing and clustering, which can reduce the computation overhead and communication bandwidth. We explain both protocols in detail by using pseudo-codes and prove the same properties as the original protocol. This chapter also compares three types of membership protocols, i.e., original, voting sharing, and clustering, in terms of various aspects and discusses a tradeoff between the computational overhead and fault tolerance.

Chapter 6 proposes a modular formal model of generic time-triggered systems. We demonstrate a prototype implementation of the formal model with the SAL language and provide example use cases of system design such as task scheduling, verification, and test case generation based on the same model in the SAL tool suite environment. The proposed membership protocols are also model-checked in a verification use case.

Finally, we conclude this dissertation in Chapter 7 with achievements and directions for future work.

CHAPTER 2

AUTONOMOUS DECENTRALIZED ARCHITECTURE

2.1 Reduced-Redundancy Approach

We take a reduced-redundancy approach to satisfy both dependability and cost-effectiveness requirements for automotive control systems.

In systems where system fault-tolerance is achieved by improving fault-tolerance of each component that makes up the system, components are designed to be fail-operational, e.g., triple redundant, which will increase the cost of the components. Our basic concept to balance dependability and cost-effectiveness is *reduced-redundancy dependability*. As can be seen in Figure 2.1, this approach tries to reduce redundancy as much as possible and to accomplish equivalent dependability with lower additional cost than the conventional solution. Because we cannot rely on the redundancy, we design the system such that it can keep on operating without the functions of failed components if some components should fail. Consequently, the component cost can be reduced as each component does not necessarily need to be fail-operational.

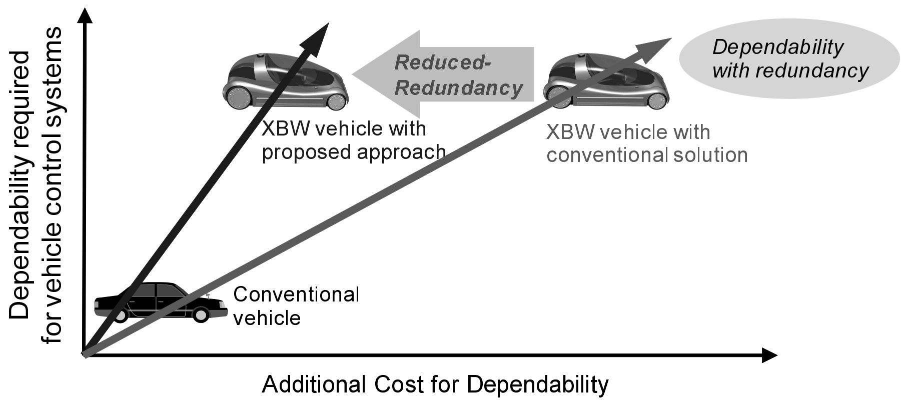

Dependability required for vehicle control systems
Reduced-Redundancy
Dependability with redundancy
XBW vehicle with proposed approach
XBW vehicle with conventional solution
Conventional vehicle
Additional Cost for Dependability

Figure 2.1: Reduced-redundancy approach

We have developed technologies for reduced-redundancy dependability in several layers at the system level, node (ECU) level, and chip (microcontroller) level. This dissertation focuses on the system and the node levels. We propose an autonomous decentralized architecture for the system level, and optimal hardware architectures in fail-silent nodes appropriate to their functions for the node level, which will be discussed in Sections 2.2 and 2.3 respectively.

2.2 Autonomous Decentralized Architecture

2.2.1 Conventional Control Systems

Figure 2.2 outlines the architecture for a conventional control system. The system consists of sensor nodes, controller nodes, and actuator nodes. The controller nodes execute the control functions based on sensor signals received from the sensor nodes, and send control commands to the actuator nodes. The controller nodes also monitor the status of the sensor/actuator nodes and if failure in a certain node is detected, the controller nodes change the normal control function to a backup control function. The actuator nodes

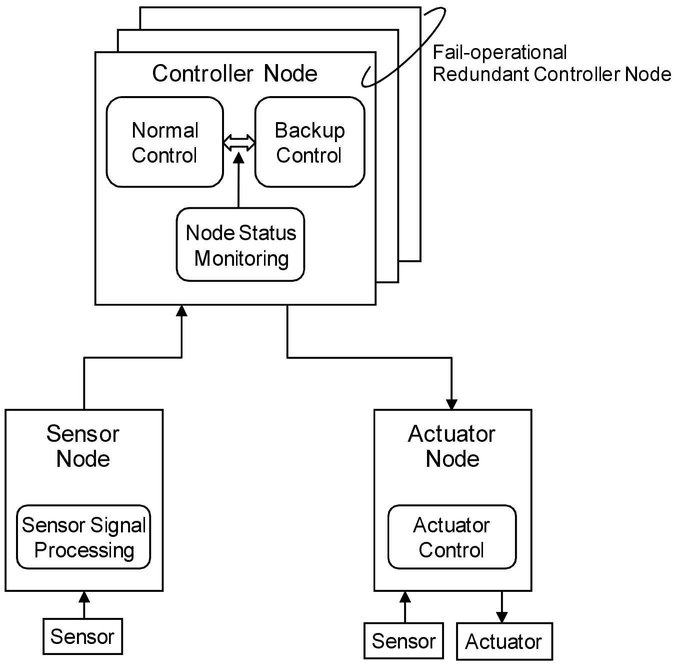


Figure 2.2: Conventional control system architecture

receive the control command for backup control*.

However, all the control functions in the conventional architecture are centralized in the controller node, which means that this architecture is essentially equivalent to a master-slave architecture. The actuator node (slave) only executes actuator control as the controller node (master) orders. It follows that a failure in the controller node will easily lead to system failure due to centralization of the control functions. To avoid this problem, the controller node should be fail-operational, i.e., it should keep operating even if one or possibly multiple faults have occurred in the node. A triple or more redundant architecture is commonly used for fail-operational nodes, but this solution tends to increase node costs, and consequently system costs.

*Each node actually has a self-diagnosis function and a node-level backup function based on the self-diagnosis results, which are not specifically described in Figure 2.2.

2.2.2 Autonomous Decentralized Control Systems

Basic Idea

We propose an architecture based on the concept of *autonomous decentralized systems* [4] to balance the competing requirements of cost-effectiveness and dependability for automotive control systems.

Autonomous decentralized systems represent one type of distributed control systems, that are used in industrial systems such as factory and train control systems required to be highly efficient and dependable. For example, the ATOS (Autonomous decentralized Transport Operation Control System) has been developed for the train traffic control system in the Tokyo metropolitan area [22].

The concept is derived from an analogy of living organisms that consist of *autonomous* and *decentralized* cells. Elements called *nodes* are loosely connected through a *data field*, where the data required for control are shared. We can achieve a fault-tolerant and scalable system with this concept, where every node is autonomous and independent. The system will not fail when nodes malfunction and improve scalability because of communication based on a standardized data interface. It is necessary to develop a suitable architecture for each application domain to apply the concept based on a biological model to real systems because no general methods of application have yet been established.

We propose the autonomous decentralized architecture for automotive control systems outlined in Figure 2.3. Although the normal vehicle control function is centralized in the controller node in this architecture, the backup control function and the node status monitoring function are decentralized in all nodes [†]. All nodes share various data required to control the vehicle through the data field, equivalent to a virtual shared memory. Each node autonomously *gets* or *puts* data from or to the data field and executes its functions,

[†]Sensor nodes do not have backup control functions.

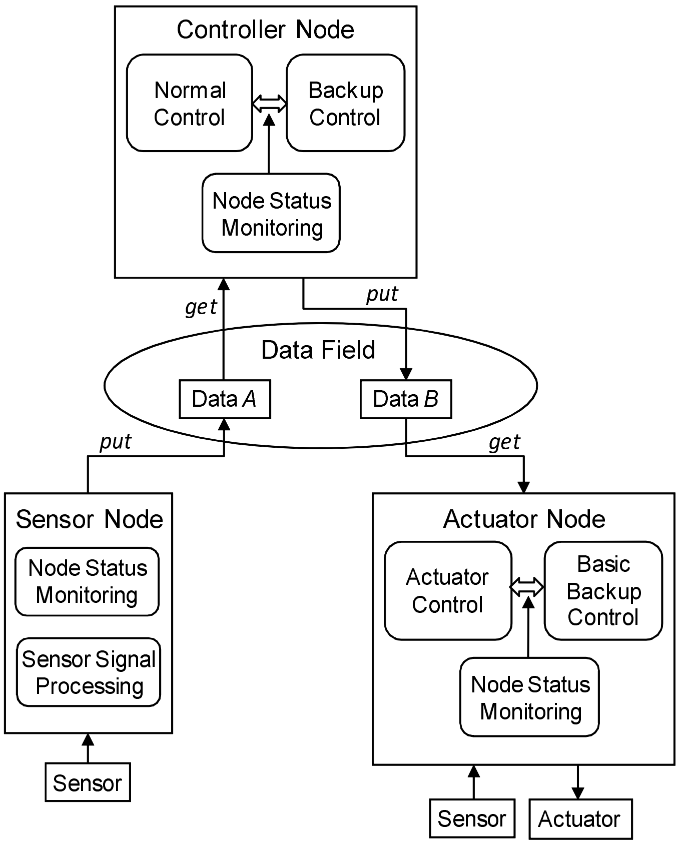


Figure 2.3: Architecture for autonomous decentralized control system

which are triggered by conditions in the time and state transitions of the node without receiving processing demands from the control nodes.

Every node, including the sensor/actuator nodes, can monitor the status of the other nodes in the system to ensure fault-tolerance. Figure 2.4 shows how the system operates when the controller node has failed. If the actuator node has diagnosed the controller node as faulty, it autonomously gets shared Data A that is put by the sensor node because the control target, Data B, from the controller node can no longer be used for actuator control. Then, the actuator node executes the backup control function by using shared Data A.

Figure 2.5 summarizes a process flow in actuator nodes. The actuator node periodi-

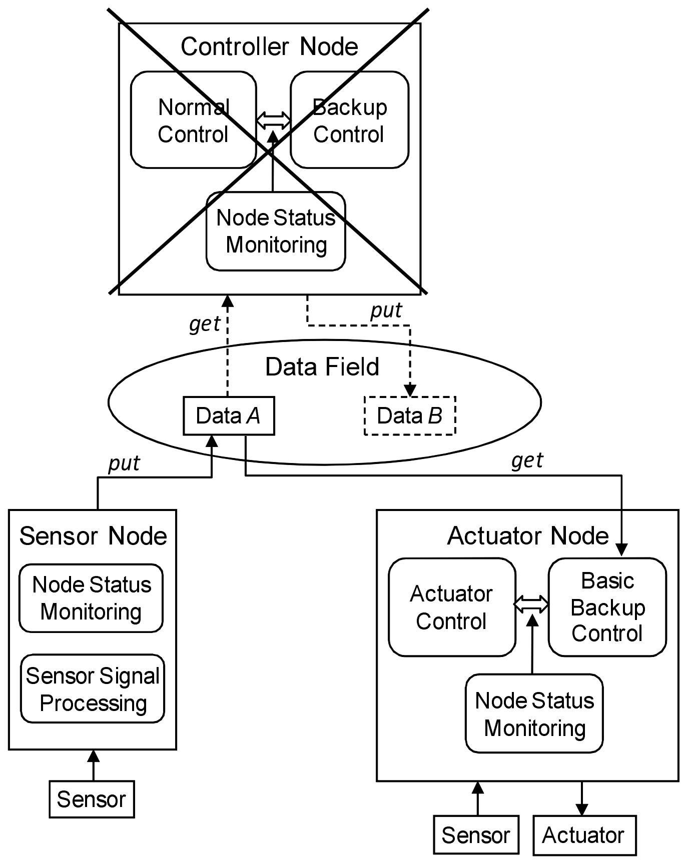


Figure 2.4: Autonomous backup control when controller node has failed

cally monitors the status of the controller node. If the actuator node diagnoses that the controller node normally functions, it uses Data B as the control target. Otherwise, the actuator node itself calculates the control target by $f(\mathrm{Data}A)$ and controls actuators with this target value. Although sensor/actuator nodes in the autonomous decentralized architecture are required to be more intelligent than those in the conventional architecture, the increase in computational overhead in actuator nodes can be suppressed by limiting the backup control function to a minimum necessary function for safe vehicle operation, which we call a *basic backup control* function.

Therefore, as the proposed architecture can tolerate the existence of failed nodes and does not require expensive redundant fail-operational nodes, we can reduce system costs.

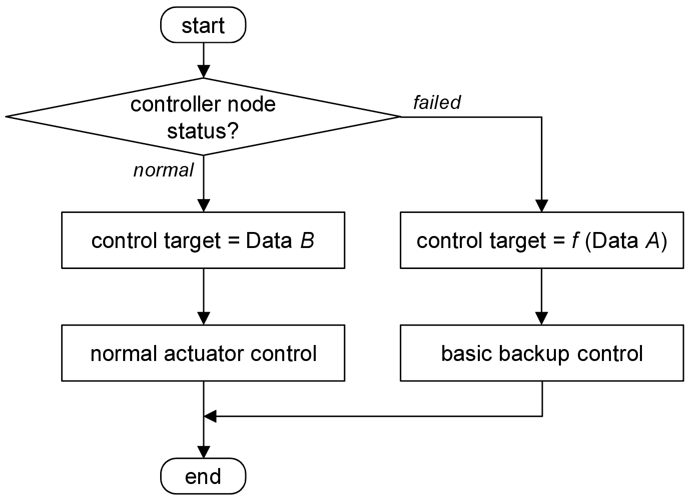


Figure 2.5: Process flow in actuator nodes

In other words, the system with the proposed autonomous decentralized architecture can be fail-operational although the system consists of only inexpensive fail-silent components.

Actual Vehicle Control Systems

Figure 2.6 shows a brake-by-wire system with the conventional control architecture. The system consists of a vehicle dynamics integrated ECU (Electronic Control Unit), four brake ECUs that actuate braking motors, and a brake pedal sensor. The brake pedal position signal, $S1$, from the brake pedal sensor is directly input only to the integrated ECU. The integrated ECU calculates the target braking force values, $B1$ to $B4$, for the four brake ECUs, using $S1$ as well as the signals from vehicle dynamics sensors such as yaw rate and acceleration sensors. Each brake ECU receives the target braking force values and controls a braking motor so that the actual braking force becomes the target value.

In this architecture, however, if the integrated ECU fails, it becomes impossible to control the vehicle dynamics because each brake ECU cannot receive the target braking

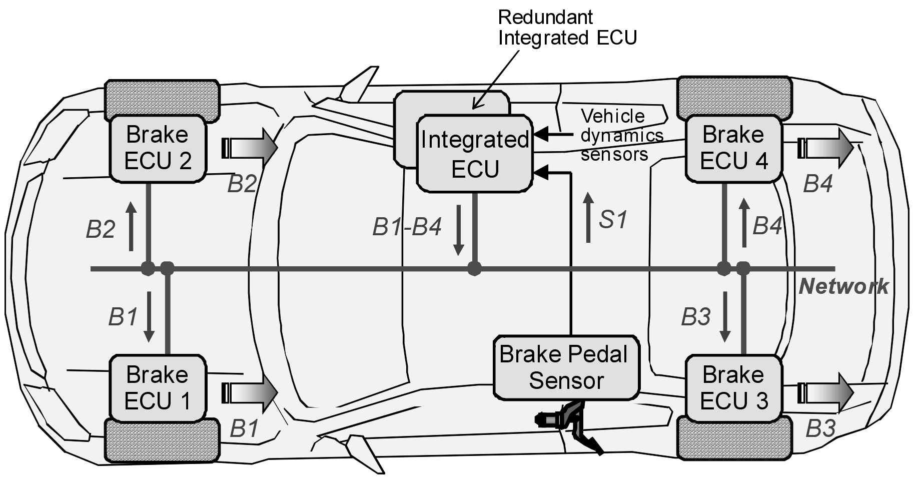
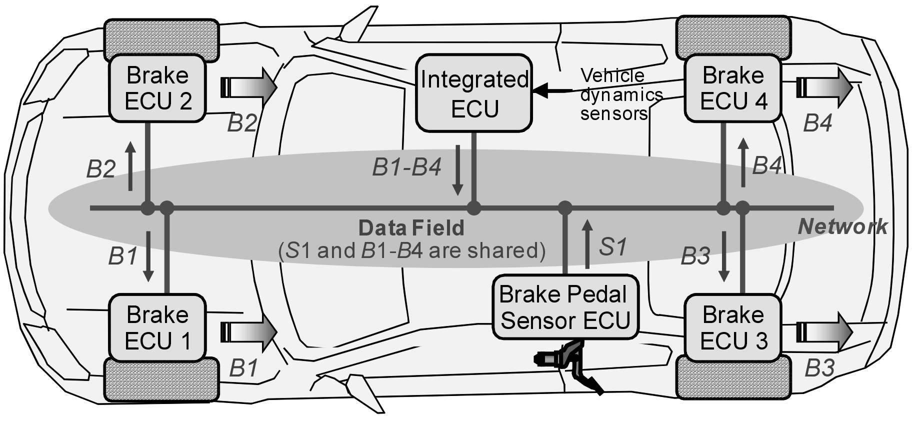


Figure 2.6: Brake-by-wire system with conventional control architecture

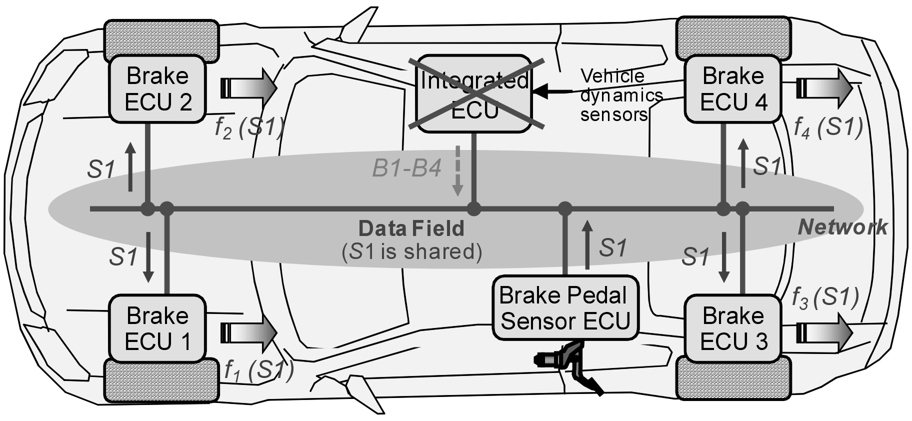
force values. Therefore, the integrated ECU should be redundant to be fail-operational, which increases the system cost.

In contrast to the conventional architecture, all the ECUs share the information required for brake control throughout the data field in the autonomous decentralized brake-by-wire system. The data field is implemented with a communication network. As shown in Figure 2.7, the brake pedal sensor is connected to the network and thus becomes more intelligent node, i.e., brake pedal sensor ECU, so that driver demand can be shared.

When the system normally operates, as indicated in Figure 2.7 (a), the control operation is logically the same as one in the conventional architecture. If the integrated ECU should fail, its sophisticated braking control function would be suspended. However, the vehicle can maintain the minimal necessary braking functionality for safe vehicle operation thanks to the autonomous backup control mechanism shown in Figure 2.7 (b). After recognizing the integrated ECU has failed, each brake ECU gets shared brake pedal position signal $S1$ in autonomous backup control. The brake ECU independently calculates the target braking force value, $f_i(S1)$ $(1 \leq i \leq 4)$, with the basic backup control function implemented in each brake ECU by using data $S1$. Therefore, we can eliminate



(a) Normal operation



(b) When integrated ECU has failed

Figure 2.7: Autonomous decentralized brake-by-wire system

redundancy from the integrated ECU because it does not need to be fail-operational.

Figure 2.8 outlines the entire architecture for vehicle dynamics control with the concept of autonomous decentralized systems. This figure focuses on the components that are related to vehicle fundamental functions of driving, steering, and braking. The data field is implemented by the communication network, which we call a vehicle control network. The integrated ECUs, motor driver ECUs, and sensor nodes communicate with one another through the vehicle control network. For example, FlexRay is utilized for the vehicle control network due to its high bandwidth and deterministic features.

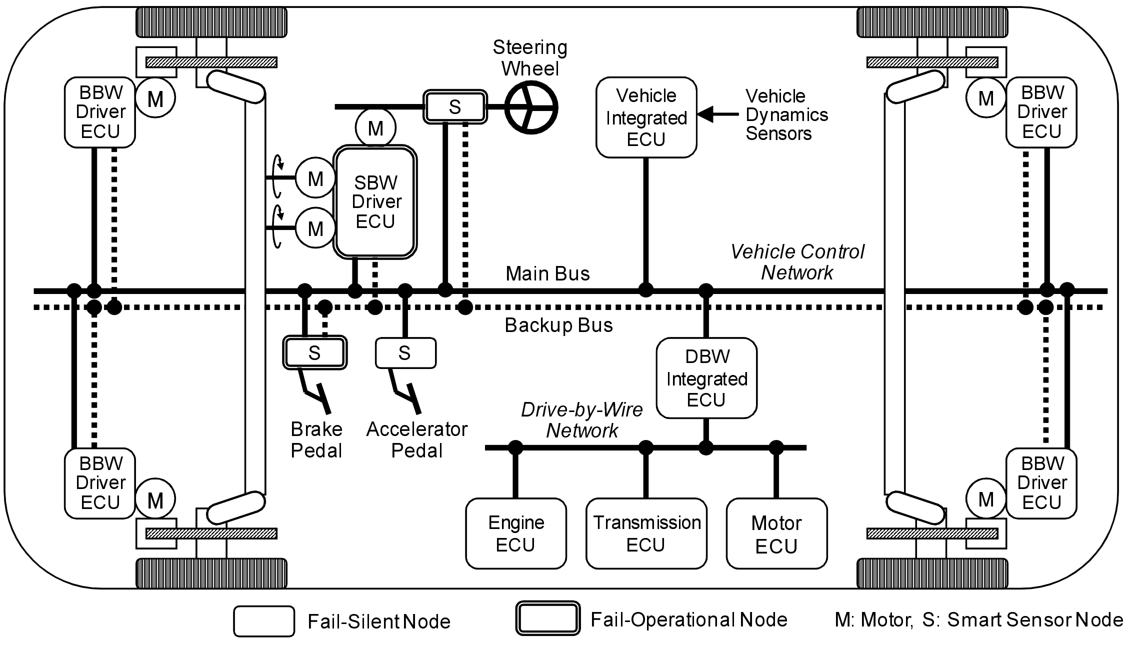


Figure 2.8: Vehicle dynamics control system with autonomous decentralized architecture

The vehicle integrated ECU synthetically controls the vehicle dynamics by interpreting driver demand from the signals received from the accelerator pedal sensor, brake pedal sensor, and steering wheel angle sensor, and also by recognizing the vehicle's motion status from the acceleration sensor, yaw rate sensor, and wheel rotation sensor. It is essential in the autonomous decentralized architecture to connect sensors to measure driver demand, i.e., the accelerator pedal sensor, brake pedal sensor, and steering wheel angle sensor, to the vehicle control network so that this information can be shared among all ECUs. On the other hand, these sensors are not connected to the network in the conventional architecture, but directly to the vehicle integrated ECU.

The vehicle integrated ECU calculates the target values for each actuator, such as those for the engine, steering, and braking, and transmits these target values to the vehicle control network. The Drive-by-Wire (DBW) integrated ECU receives the target driving force value, the Steer-by-Wire (SBW) driver ECU receives the target steering angle value, and the Brake-by-Wire (BBW) driver ECU receives the target braking force value. The SBW/BBW driver ECU controls a motor by calculating the motor torque required to achieve these target values. The SBW driver ECU also controls a motor for actuating a variable gear ratio (VGR) mechanism that is installed on the steering column to generate a virtual reactive force for vehicle drivers. The DBW integrated ECU is a master controller for the powertrain system. This ECU executes the driving force distribution to the engine and motor to improve energy efficiency. The calculated target driving torque and gear ratio values are transmitted to the engine ECU, motor ECU, and transmission ECU through the drive-by-wire network, e.g., CAN (Controller Area Network).

The vehicle control network has redundant buses for fault-tolerance, i.e., main and backup buses. The vehicle integrated ECU, DBW integrated ECU, SBW driver ECU, BBW driver ECUs, and three sensor nodes are connected to the main bus. In contrast, only the minimum necessary nodes for safe vehicle operation, viz., the SBW driver ECU, BBW driver ECUs, brake pedal sensor, and steering wheel angle sensor, are connected to the backup bus to decrease network costs. Since a loss of function to generate driving force does not cause fatal accidents, the accelerator pedal sensor is not connected to the backup bus and the drive-by-wire network is not redundant. If the main bus of the vehicle control network should fail, as described in the autonomous backup function, the BBW Driver ECU autonomously obtains data from the brake pedal sensor and the SBW driver ECU autonomously obtains those from the steering wheel angle sensor, and they control the motors with the control target values calculated in the basic braking/steering backup control functions.

Furthermore, the SBW driver ECU, brake pedal sensor, and steering wheel angle sensor should be fail-operational nodes as shown in Figure 2.8 to maintain the functions for safe vehicle operation. It is also necessary to make the steering motor dual-redundant. A fail-operational SBW driver ECU consists of two fail-silent nodes and each fail-silent node independently controls one steering motor. Although some nodes and motors have to be fail-operational, the number of fail-operational nodes can be minimized and thus the system cost can be reduced due to the autonomous decentralized architecture.

The autonomous decentralized architecture also improves system scalability and thus simplifies the development process because the data on the vehicle control network have a high degree of abstraction. For example, the target values for each actuator are defined so that the vehicle control logic in the vehicle integrated ECU can be developed without any knowledge of actuator variety or characteristics. Sensor nodes broadcast physical values that are meaningful to control logic after processing of sensor signal filtering and conversion of voltage to a physical value.

Various functions can easily be extended in the proposed architecture due to high levels of scalability by connecting the required components to the vehicle control network. Moreover, integrating a gateway function into the vehicle integrated ECU enables cooperative control with the components connected to other networks, such as information, body, and safety networks.

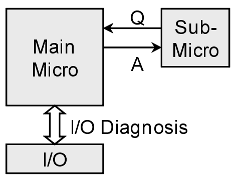
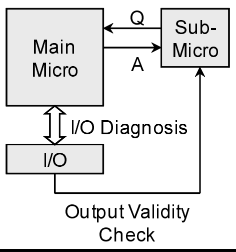
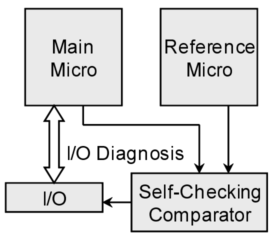
2.3 Fail-Silent Node Architecture

2.3.1 Comparison of Fail-Silent Nodes

The proposed X-by-Wire systems can be mostly implemented with inexpensive fail-silent ECUs, as shown in Figure 2.8, i.e., they do not interfere with communication between other ECUs even if they have failed. Although the SBW driver ECU, brake pedal sensor, and steering wheel angle sensor have to be fail-operational, a fail-silent node is essential for safety-critical distributed systems since a fail-operational node can be composed of two fail-silent nodes.

The fail-silent nodes of three redundancy types are compared in Table 2.1 in terms of

Table 2.1: Comparison of fail-silent nodes

Redundancy Type	Q&A	Validity Check	Dual Redundancy
Node Architecture		 Main Micro Q A Sub-Micro I/O Diagnosis I/O Output Validity Check	 Main Micro Reference Micro I/O Diagnosis I/O Self-Checking Comparator
Fault Detection Coverage	Low	Middle-High*	High
Cost	Low	Low	Low**-Middle
ECU Type	Engine ECU	BBW/SBW Driver ECU	Integrated ECU Sensor Node

* Coverage for fatal fault, ** Where dual CPU LSI is applied

cost and coverage of fault detection. The fault detection coverage has a great influence on system reliability, although it is difficult to precisely estimate its value. A node is not guaranteed to be fail-silent if a fault is not detected, and this fatal event occurs with a rate of $(1 - C) \times \lambda$, where C is the fault detection coverage and λ is the failure rate of the node. The fatal event rate decreases as C approaches one. However, because there is a tradeoff between node cost and coverage, we have to apply a suitable architecture to each node depending on node function to optimize costs.

A node in the *question and answer (Q&A)* method consists of a main microcontroller and a sub-microcontroller (a microcontroller is called a micro after this). The sub-micro transmits an appropriate calculation problem to the main micro, and the main micro calculates the answer and returns it to the sub-micro. The sub-micro compares the returned answer with a predetermined answer. If the two values differ, the sub-micro determines that the main micro has failed, and stops the node function. Conversely, the state of health of the sub-micro is monitored by the main micro. The main micro also diagnoses I/O circuits and related sensors/actuators. Although this structure is inexpensive, its fault detection coverage is the lowest at around 90% [23] among the three types because it is unsure whether the calculation problems are designed such that all faults can be completely detected. This architecture has been applied to conventional ECUs, such as engine and transmission ECUs.

We found that while the hardware architecture was similar to the question and answer method, i.e., without cost increases, a method of *validity check* could improve the fault detection coverage. The sub-micro monitors not only the returned answer from the main micro, but also the actuation output result. The sub-micro compares the actuator output with the target value for actuation and determines the validity of the actuation. Although a rigid check of the main microcontroller's execution result is not carried out, a fatal fault generated in the path from the main micro to the actuator can reliably be detected by monitoring the final actuation status. This architecture is suitable for the motor driver ECU. It is essential to design the node such that the execution load to compare these two values in the sub-micro can be reduced to utilize inexpensive sub-micros.

Rigid checks of the microcontroller execution result are required in the node that calculates the control target value based on control logic and outputs the calculated value to the other nodes. Therefore, not the validity check method but a *dual redundancy* architecture is applied to nodes such as vehicle integrated ECUs. The nodes are composed of two equivalent microcontrollers and a comparator. The comparator confirms whether the execution results of the main and reference micros coincide. A self-checking type comparator has to be used to detect faults in the comparator itself, which will be discussed later. The coverage in this architecture exceeds 99% [23]. The cost of dual redundancy is higher than that of the other two types, but it is still lower than that of a fail-operational architecture like triple redundancy. Moreover, the cost can be reduced where the two micros and the self-checking comparator are integrated into one LSI chip.

2.3.2 Node Hardware Architecture

Motor Driver ECU

The hardware architecture for the fail-silent SBW driver ECU[‡] is shown in Figure 2.9 as an example of the method of validity check. The SBW driver ECU obtains the target steering angle value calculated by the vehicle integrated ECU. The main micro calculates the required target motor torque and current to attain this target steering angle, and performs vector control of the three phase motor. The sub-micro compares the steering angle target value with the actual steering angle that is measured by a steering angle sensor. When the sub-micro detects that these values are different, it disables access to the communication network and shuts down the power supply for the steering motor.

It is important to determine the timing in this architecture to compare the target and the actual steering angle by taking into account the response delay time of the mechanical system. Because the sub-micro is not connected to the network to reduce the cost of the communication interface, it cannot receive the target value directly from the network. Consequently, if the main micro fails and sends an incorrect target value to the sub-micro, failure in the main micro may not be able to be detected since the main micro controls the motor and the sub-micro executes the validity check based on this incorrect target value. The vehicle integrated ECU adds a data check code to the target value data to prevent this problem. After the frame from the vehicle integrated ECU is received, the main micro sends it to the sub-micro without processing. The sub-micro can determine whether the data are correct or not by checking this data check code.

[‡]Two fail-silent ECUs form fail-operational SBW driver ECU described in Subsection 2.2.2.

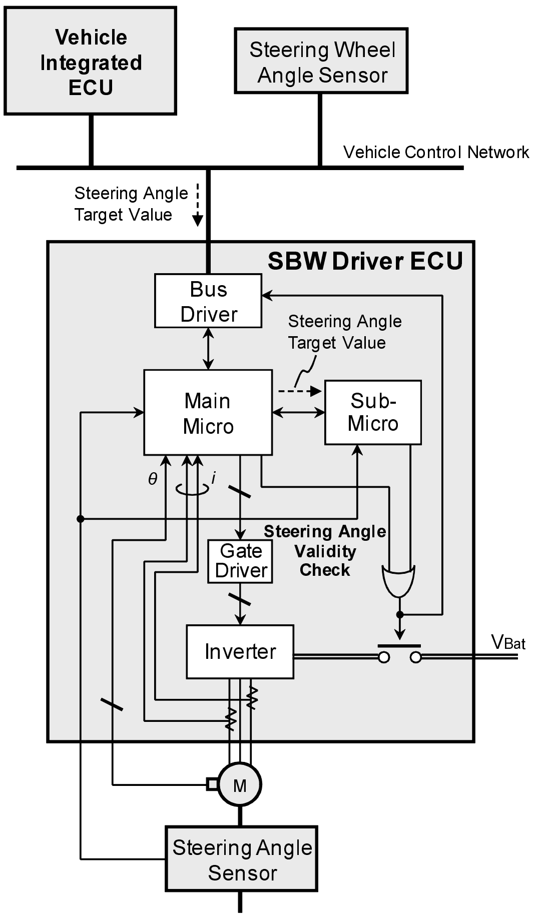


Figure 2.9: Validity check architecture for SBW driver ECU

Integrated ECU

Figure 2.10 indicates the hardware architecture for the vehicle integrated ECU to which dual redundancy architecture is applied. Signals from vehicle dynamics sensors, such as the acceleration and yaw rate sensors are input to both the main and the reference micros. The main and reference micros communicate with each other via the serial communication to synchronize the analog-to-digital (A/D) conversion and make the conversion values coincide. The calculated values in both microcontrollers are compared by self-checking comparators. If the self-checking comparators detect disagreements in both values, they disable the bus driver to stop network access.

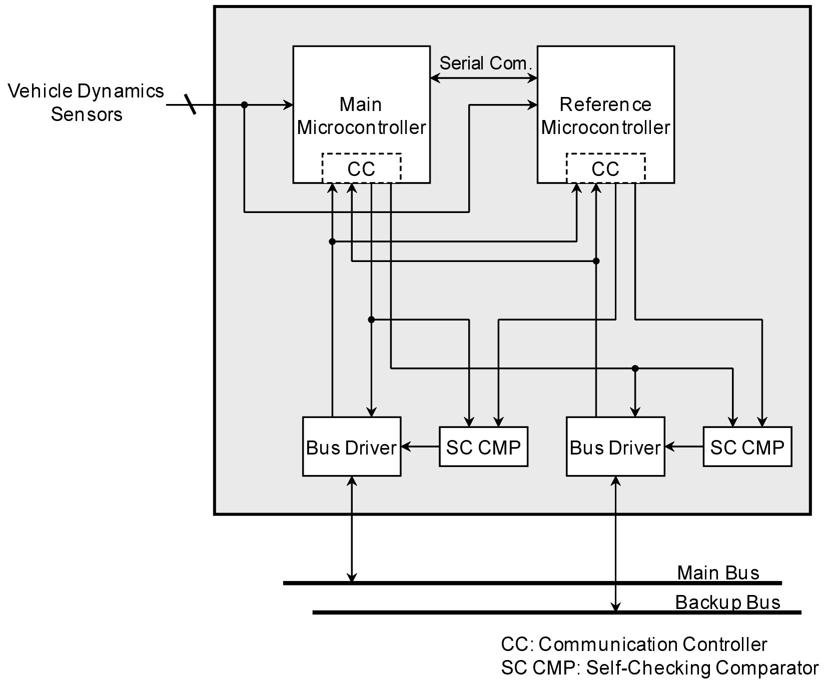


Figure 2.10: Dual redundancy architecture for integrated vehicle ECU

A self-checking comparator consists of the test pattern generator and comparator as shown in Figure 2.11 [24]. The test pattern generator periodically injects a test pattern and “00· · ·00” (all bits are 0) to each bit of the input data. When inputs A and B are equal, and the self-checking comparator is operating correctly, it outputs a rectangular wave with a period that is identical to the test pattern injection period. If the comparator does not output a rectangular wave with this predetermined period, it means either “input A and input B are not equal” or “the self-checking comparator itself is faulty”. Thus, the self-checking comparator can not only detect faults in the input data, but also in the comparator itself, which improves the reliability of the dual redundancy node.

Sensor ECU

The autonomous decentralized architecture requires fail-silent sensor ECUs that can directly broadcast the sensing data to the vehicle control network, as was explained in Sub-

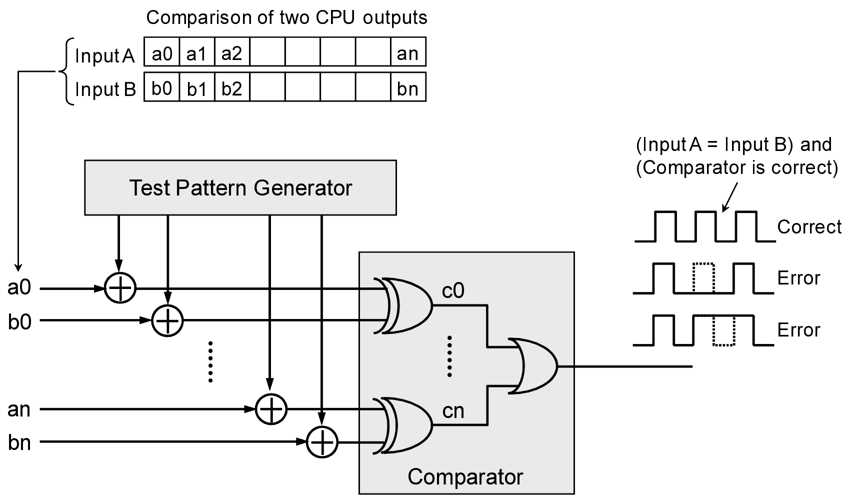


Figure 2.11: Configuration of self-checking comparator

section 2.2.2. The hardware block diagram of a smart sensor node that employs the dual redundancy architecture is shown in Figure 2.12. The sensor node consists of two sensing devices and a dual CPU LSI chip that integrates dual CPU cores, A/D converters, communication controllers, self-checking comparators, and ROMs/RAMs with ECC (Error Correcting Code). This LSI contributes to size reduction in the sensor node. The two CPUs communicate with each other to synchronize the A/D conversion and make the A/D conversion values of the sensor signals coincide. The CPUs execute the appropriate signal processing for each sensing device, such as sensor signal filtering and conversion of voltage to physical values to improve system scalability.

2.4 Estimation of System Cost Reduction

We estimate the effect of the system cost reduction due to the proposed autonomous decentralized architecture and optimal node hardware architecture. A brake-by-wire system with the conventional architecture (Figure 2.6) is compared with one incorporating the autonomous decentralized architecture (Figure 2.7). We focus on electronic components,

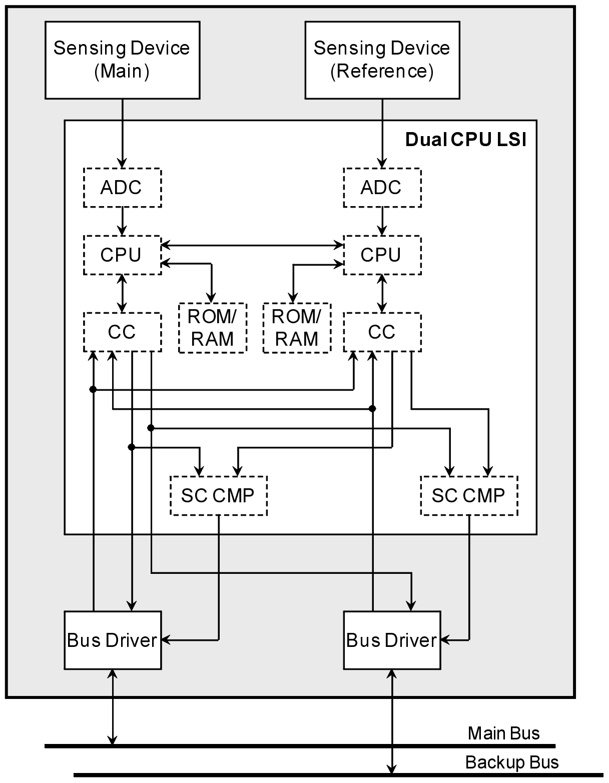


Figure 2.12: Hardware architecture for smart sensor node

i.e., ECUs to estimate the system cost. The system cost for the conventional architecture, $C_{\mathrm{sys}}^{\mathrm{conve}}$, is as follows:

$$C_{\mathrm{sys}}^{\mathrm{conve}} = 2C_{\mathrm{C}} + 4C_{\mathrm{A}} \tag{2.1}$$

where C_{C} is the cost of a fail-silent integrated ECU, and C_{A} is the cost of a fail-silent brake ECU that employs the conventional dual redundancy architecture shown in Table 2.1. Note that the fail-operational integrated ECU consists of two fail-silent integrated ECUs as discussed in Section 2.2. The cost of a sensing device of the brake pedal sensor is assumed to be negligible small compared with that of an ECU.

The increase of the system cost, C_{sys}^{+}, for the autonomous decentralized architecture is as follows:

$$C_{\mathrm{sys}}^{+} = C_{\mathrm{S}} + \frac{C_{\mathrm{dev}}}{N_{\mathrm{pro}}} \tag{2.2}$$

where C_S is the cost of the intelligent brake pedal sensor ECU that can communicate with the other ECUs through the control network, C_dev is the additional software development cost for the node status monitoring and the backup control functions, and N_pro is the production volume of the system. We assume that N_pro is large enough to make the second term in Equation (2.2) negligible because of the mass-production scale in the automotive industry. Although the autonomous decentralized architecture requires the node status monitoring and the backup control functions even for sensor and actuator nodes, a microcontroller with equivalent performance as one equipped with the conventional architecture can be applied. This is because only the basic function is required for the autonomous backup control and the CPU computational overhead for the node status monitoring can be sufficiently reduced, which will be discussed in Chapter 5. Thus, the cost increase of the microcontroller in each ECU is not included in Equation (2.2).

Equation (2.3) expresses the cost decrease, C^-_sys, by the proposed autonomous decentralized architecture and optimal node hardware architecture:

$$C^-_\mathrm{sys} = C_\mathrm{C} + 4\mathit{\Delta} C_\mathrm{A} = C_\mathrm{C} + 2C_\mathrm{mc} \tag{2.3}$$

where $\mathit{\Delta} C_\mathrm{A}$ is the cost reduction per one fail-silent brake ECU. Because we can apply less expensive validity check architecture to fail-silent brake ECUs as indicated in Table 2.1, $\mathit{\Delta} C_\mathrm{A}$ can approximately be estimated as $2C_\mathrm{mc} - 1.5C_\mathrm{mc}$, where C_mc is the cost of a main microcontroller which is assumed to be double compared with the cost of a sub-microcontroller. Furthermore, as discussed in Section 2.2, since the integrated ECU can be fail-silent due to the system-level redundancy in the autonomous decentralized architecture, we can eliminate one fail-silent integrated ECU.

Therefore, with Equations (2.1), (2.2), and (2.3), we can calculate a ratio of the reduced system cost, $\mathit{\Delta} C_\mathrm{sys}$, to the system cost for the conventional architecture as follows:

$$\frac{\mathit{\Delta} C_\mathrm{sys}}{C^\mathrm{conve}_\mathrm{sys}} = \frac{C^-_\mathrm{sys} - C^+_\mathrm{sys}}{2C_\mathrm{C} + 4C_\mathrm{A}} \simeq \frac{C_\mathrm{C} + 2C_\mathrm{mc} - C_\mathrm{S}}{2C_\mathrm{C} + 4C_\mathrm{A}} \simeq \frac{C_\mathrm{C} + 2\alpha C_\mathrm{C} - \beta C_\mathrm{C}}{6C_\mathrm{C}} \tag{2.4}$$

where α is a ratio of the cost of a microcontroller to that of a fail-silent integrated ECU ($\alpha = C_{\mathrm{mc}}/C_{\mathrm{C}} < 1$), and β is a ratio of the cost of a sensor ECU to that of a fail-silent integrated ECU ($\beta = C_{\mathrm{S}}/C_{\mathrm{C}} < 1$). In Equation (2.4), we assume that C_{A} is almost the same as C_{C}.

Figure 2.13 shows the estimation results on the system cost reduction. Although parameters α and β depend on the actual hardware implementation, we conclude that approximately from 20 to 30% cost reduction can be achieved with practically possible combinations of parameters α and β, which contributes to a substantial cost reduction for automotive control systems.

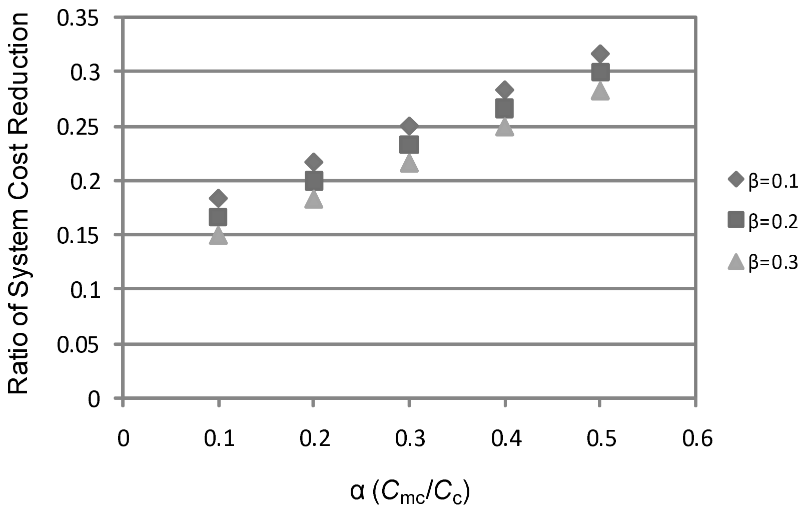


Figure 2.13: Cost reduction in autonomous decentralized brake-by-wire system

CHAPTER 3

AGREEMENT PROTOCOL

3.1 Challenges to Autonomous Decentralized Architecture

The autonomous backup control is an essential feature in the autonomous decentralized architecture as was discussed in the previous chapter. Backup control should be based on the accurate identification of failed nodes. However, as there is no master node for monitoring the status of nodes in the system, the node status monitoring function in each node plays a key role for fault-tolerance.

For example, as shown in Figure 3.1, suppose that the brake ECU 2 has not received braking control target value $B2$ from the integrated ECU because of failure in the receiver, i.e., brake ECU 2. In this case, only the brake ECU 2 autonomously changes the control mode to the backup control mode because it cannot determine by itself whether the sender (integrated ECU) or the receiver (brake ECU 2) is faulty, which might lead to vehicle spin due to braking imbalances.

Therefore, some coordination scheme, i.e., an agreement protocol, is required to accurately identify the failed node and ensure consistency of the information on which nodes

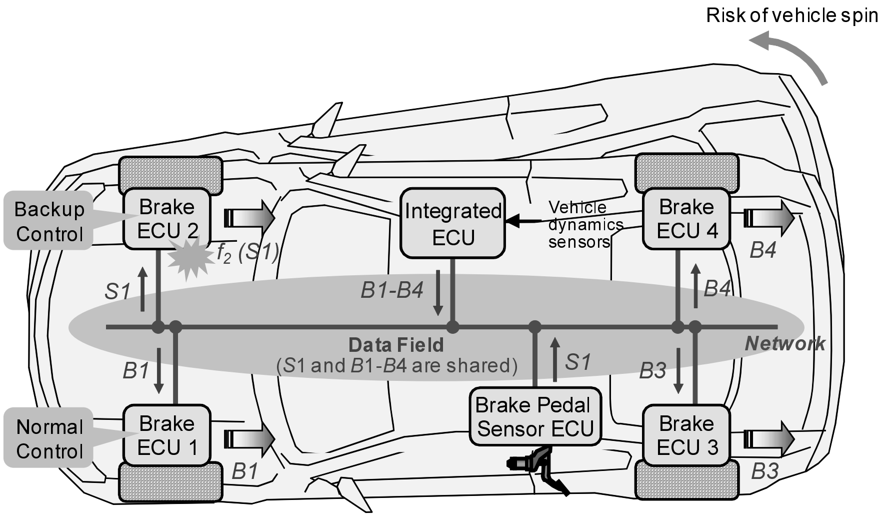


Figure 3.1: Disagreements in control mode in autonomous decentralized architecture

are available among all remaining normal nodes so that disagreements in the control mode can be avoided. We propose a membership protocol as an agreement protocol for safety-critical distributed systems to address this issue, which will be discussed in this chapter.

3.2 Model of Time-Triggered Systems

3.2.1 Communication Model

Before we discuss the proposed membership protocol, let us first define a generic time-triggered (TT) system including a FlexRay communication system, which will be used in X-by-Wire systems.

The system consists of n nodes having unique IDs $1, 2, ..., n$. The communication network is the bus type with TDMA (Time Division Multiple Access). As we can see from Figure 3.2, the system runs by consecutively executing synchronous rounds, starting from round 1. A node is assigned its own sending slots as to where it can send a message frame. A frame sent by a node is received by all the other nodes. All sending slots are statically scheduled in the design time and thus never overlap. Every node sends a frame at least once in one round.

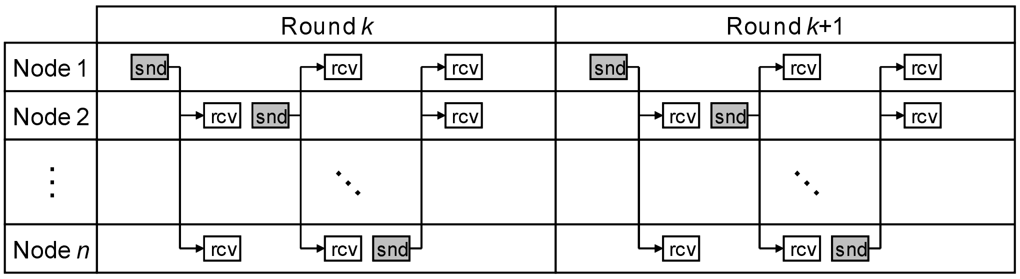


Figure 3.2: Communication in time-triggered systems

3.2.2 Fault Model

Faults in nodes are observed as communication errors. Figure 3.3 illustrates three fault types: *correct*, *benign fault*, and *symmetric Byzantine fault*. If node i suffers neither benign fault nor symmetric Byzantine fault, the node correctly sends the message frame m. If node i suffers a benign fault in round k, then the message frame m sent by the node in the round is lost. Hence, all nodes can locally detect it in round k. A benign fault arises from a transmission error due to e.g., a node crash. If node i suffers a symmetric Byzantine fault in round k, then, all nodes in the same round receive the same erroneous message m' from the faulty node, which does not conform to the protocol specifications. In this dissertation, we assume that a reception error in a certain node can be seen as a symmetric Byzantine fault, as discussed in detail later.

We take account of intermittent faults as well as permanent faults, and also assume that the fault types, either benign or symmetric Byzantine, never change along with communication rounds in one node.

Our fault model does not include faults in terms of timing violation in communication, where a node sends message frames at the sending slots disallowed for the node.

This is because we assume that each node is equipped with so called *bus guardians* that physically prevent a faulty node from accessing the bus at the sending slots assigned to the other nodes [25].

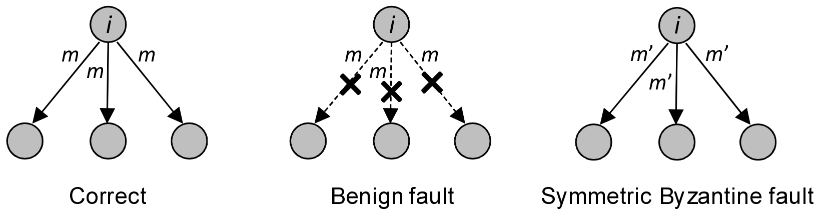


Figure 3.3: Fault types

Nodes are *correct*, *obedient*, or *symmetric Byzantine faulty*. Correct nodes follow the protocol specifications and suffer no faults. Obedient nodes follow the specifications but they may or may not suffer benign faults. Correct nodes are thus also obedient. Symmetric Byzantine faulty nodes do not follow the specifications and suffer symmetric Byzantine faults. We assume:

$$2s + b + 1 < n \tag{3.1}$$

where s is the number of symmetric Byzantine faulty nodes and b is the number of benign faulty nodes.

3.3 Membership Protocol

3.3.1 Protocol Description

We propose a membership protocol for TT systems as an agreement protocol to solve these problems in the autonomous decentralized architecture [16]. It provides information on the availability of all nodes in the system by exchanging a local view of the status of other nodes with all nodes.

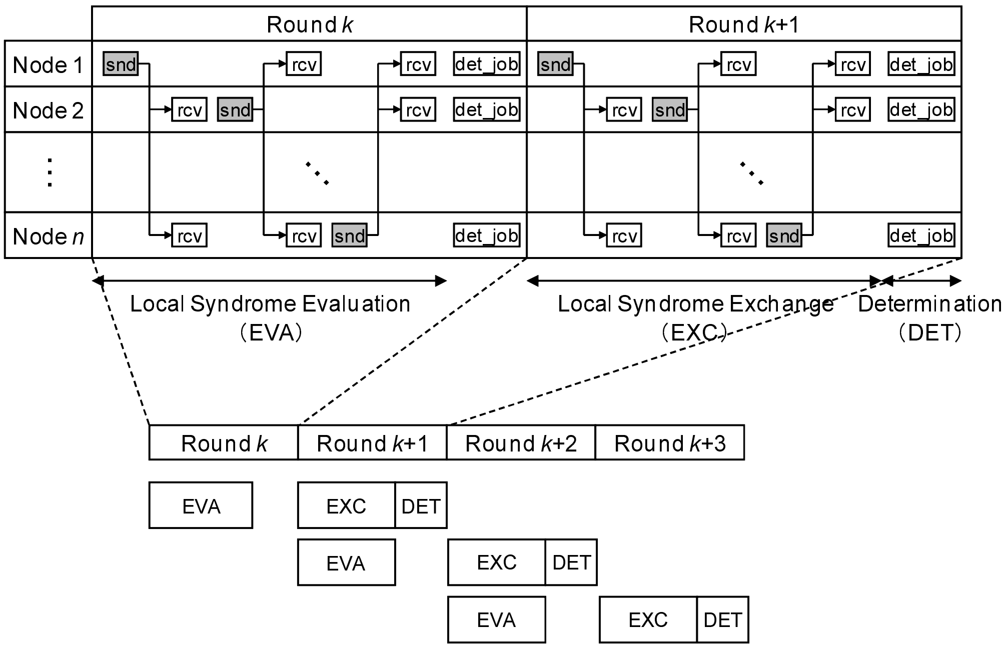


Figure 3.4: Overview of execution sequence in membership protocol

An execution sequence of the protocol in the TT system is outlined in Figure 3.4. In every round k, the membership protocol starts the sequence of three phases, which spans two consecutive rounds, viz., rounds k and $k+1$. The pseudo-code of the membership process of three phases executed in each node p in round k is also presented in Algorithm 1. In the pseudo-code, hereinafter, $information_{k}\_p[i,\ldots,j]$ denotes $information$ on statuses with respect to from node i to node j in round k computed by node p.

In round k, each node evaluates other nodes' statuses locally by receiving frames sent by the other nodes ($receive\_ls_{k-1}\_msg\_i[\,]$) and evaluating them (EVA phase). The evaluation result, which we call a *local syndrome*, represents the local view of other nodes' statuses that was evaluated in round k. A local syndrome evaluated in node p, i.e., $ls_{k}\_p[\,]$, is a binary n-tuple $\langle s_1, s_2, \ldots, s_n\rangle$ where $s_i = 1$ if p evaluates node i as non-faulty and $s_i = 0$ otherwise (Algorithm 1, lines 5-8). In the pseudo-code, Φ denotes no received message (line 5). Note that $ls_{k}\_p[\,]$ has to be buffered in $send\_ls_{k}\_msg\_p[\,]$ because it is not sent in the current round, but in the next round, i.e., round $k+1$ (line 12).

In the following round, round $k+1$, the local syndromes evaluated in round k are exchanged by all nodes and node p constructs a local syndrome matrix, $ls_{k}\_matrix[\,][\,]$, where the ith row is the local syndrome received from node i and the jth column is a vector representing the evaluation for node j from all nodes (EXC phase). An element, $e_{i,j}$, is either 1, 0, ε, or $-$. The case, $e_{i,j} = \varepsilon, i \neq j$, occurs if node i failed to send its local syndrome in round $k+1$ because of its benign fault (lines 17-18). In line 18, $[\varepsilon, \ldots, \varepsilon]_n$ is a vector that has n elements of ε. The opinion of a node about itself, i.e., $e_{i,i}, 1 \leq i \leq n$, is considered unreliable and thus is assigned special value $-$, which specifies that it is to be discarded in voting (line 22).

Each node determines node status in the same round (i.e., round $k+1$) by voting on the local syndrome matrix (DET phase). Each node obtains a binary n-tuple called a *temporal health vector*, $\hat{hv}$, where the ith element represents whether node i is non-faulty or faulty by hybrid voting [26] over each of the columns (line 27). Hybrid voting is specifically defined as:

$$H\!-\!maj(V) = \begin{cases} 0 & N_0(V) > N_1(V) \\ 1 & N_0(V) \leq N_1(V) \end{cases}$$

where V is the column that is voted on, and $N_0(V)$ and $N_1(V)$ are the number of occurrences of 0 and 1 in V.

The temporal health vector is then updated to accuse symmetric Byzantine faulty nodes, which we call a *minority accusation*. If the temporal health vector, $\hat{hv}$, and the local syndrome sent from node i respectively have 0 and 1 or 1 and 0 in the same position, then node i is identified as a faulty node (lines 31-32). The final health vector, hv, is obtained from $\hat{hv}$ by setting the value in the ith position to 0 for all such nodes i[*].

[*]We regard benign and symmetric Byzantine faulty nodes as equally serious because the potential causes of these faults are all physical [27]. This is in contrast to systems deployed in open networks, where intrusion is the main cause of Byzantine faults.

Finally, a node updates counters associated with nodes based on the health vector and possibly eliminates faulty nodes from active ones. For example, one can eliminate a node by counting the number of times when the node is diagnosed as faulty and by deciding to eliminate it when the counter exceeds a predefined threshold. The process of updating the counters can be regarded as executing a stateful function that takes a health vector, hv, as input and produces a set of active nodes as output. We denote this counter updating function by $updateCounter(hv)$. Thus, the output of the membership protocol is obtained by executing the following operation in each node (lines 38 and 40):

$$active\_nodes \leftarrow updateCounter(hv)$$

We emphasize that multiple instances of this sequence of phases are executed concurrently, as shown in Figure 3.4. The node status evaluation process (EVA) can be done concurrently in the status data exchanging phase (EXC) by evaluating the exchanged status data. The node status, correct or faulty, in a certain round can be identified in the next round, which can enhance real-time capabilities for diagnosing faults, due to this pipeline-like process execution.

Algorithm 1: Node p membership process in round k

```
// EVA Phase
// local syndrome evaluation
for $i \leftarrow 1, \dots, n$ do
    $ls_{k-1}\_i[1, \dots, n] \leftarrow receive\_ls_{k-1}\_msg\_i[1, \dots, n]$;
    if $ls_{k-1}\_i[1, \dots, n] = \Phi$ then  // no received message
        $ls_k\_p[i] \leftarrow 0$;
    else
        $ls_k\_p[i] \leftarrow 1$;
    endif

// local syndrome buffering to be sent in round $k+1$
$send\_ls_k\_msg\_p[1, \dots, n] \leftarrow ls_k\_p[1, \dots, n]$;

// EXC Phase
// local syndrome exchange and local syndrome matrix construction
for $i \leftarrow 1, \dots, n$ do
    if $ls_{k-1}\_i[1, \dots, n] = \Phi$ then  // no received message
        $ls_{k-1}\_matrix[i][1, \dots, n] \leftarrow [\varepsilon, \dots, \varepsilon]_n$;
    else
        $ls_{k-1}\_matrix[i][1, \dots, n] \leftarrow ls_{k-1}\_i[1, \dots, n]$;
    endif
    $ls_{k-1}\_matrix[i][i] \leftarrow -$;

// DET Phase
// hybrid voting
for $i \leftarrow 1, \dots, n$ do
    $\hat{hv}_{k-1}[i] \leftarrow H\text{-}maj(ls_{k-1}\_matrix[1, \dots, n][i])$;

// minority accusation
for $i \leftarrow 1, \dots, n$ do
    if $ls_{k-1}\_i[1, \dots, n] \neq \hat{hv}_{k-1}[1, \dots, n]$ then
        $hv_{k-1}[i] \leftarrow 0$;
    else
        $hv_{k-1}[i] \leftarrow \hat{hv}_{k-1}[i]$;
    endif

// counter update
$active\_nodes \leftarrow updateCounter(hv_{k-1}[1, \dots, n])$;

return $active\_nodes$;
```

3.3.2 Example of Membership Protocol

Consider a system consisting of five nodes ($n = 5$) as an example where node 2 is symmetric Byzantine faulty and the other nodes are obedient. Suppose that node 1 suffers benign faults in rounds k and $k + 1$. Also suppose that node 2 sends an erroneous local syndrome in round $k + 1$. Then, each obedient node forms a local syndrome matrix in round $k + 1$ as follows:

$$\begin{array}{l} \left[\begin{array}{ccccc} - & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ 0 & - & 1 & 0 & 1 \\ 0 & 1 & - & 1 & 1 \\ 0 & 1 & 1 & - & 1 \\ 0 & 1 & 1 & 1 & - \end{array}\right] \end{array} \begin{array}{l} \text{from node 1} \\ \text{from node 2} \\ \text{from node 3} \\ \text{from node 4} \\ \text{from node 5} \end{array}$$

Every obedient node carries out hybrid voting on each column after the local syndrome matrix is formed. The ε and $-$ values are first discarded in hybrid voting and then voting is performed on the remaining values. In this case, all obedient nodes obtain the following temporal health vector:

$$\hat{hv} = \langle 0, 1, 1, 1, 1 \rangle$$

Then, symmetric Byzantine faulty nodes are accused. Of the five syndromes, only the one from node 2 has different 0/1 values from $\hat{hv}$. Specifically, the syndrome from node 2 differs from $\hat{hv}$ in the fourth position. The temporal health vector is updated to record that node 2 has been identified as a faulty node by setting the second position to 0:

$$hv = \langle 0, 0, 1, 1, 1 \rangle$$

Finally, using the health vector, each node updates the set of active nodes by executing $updateCounter(hv)$.

Figure 3.5 shows an example of a detailed execution sequence for the protocol. The system configuration and fault conditions are the same as those in the above example. Node 2 becomes symmetric Byzantine faulty in round $k + 1$ because node 2 could not receive node 4's message frame in round k due to, e.g., electrical noise in node 2 or at a network stub to node 2.

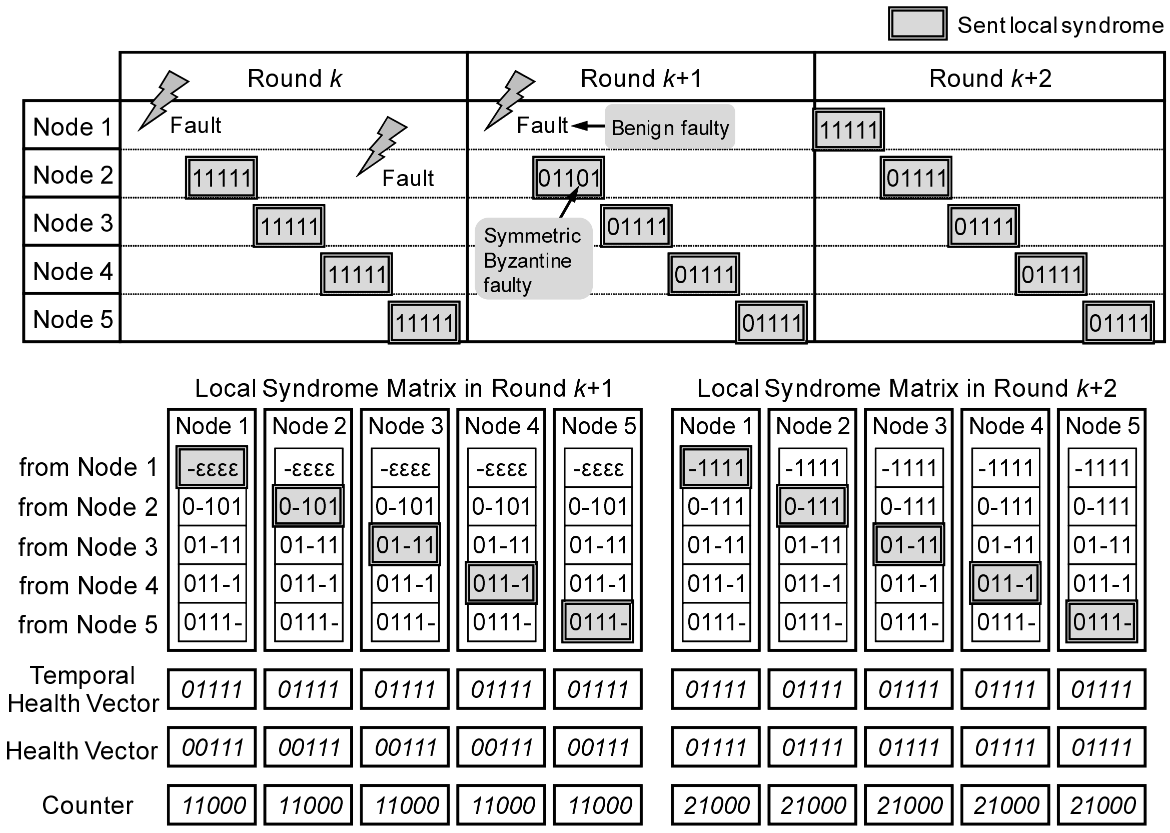


Figure 3.5: Detailed execution sequence for membership protocol

Node 2 evaluates that node 4 was benign faulty in round k and then node 2 sends a local syndrome "01101" in round $k + 1$, which differs from a temporal health vector "01111". Thus, node 2 is identified as symmetric Byzantine faulty by the minority accusation process. Note that the result of the minority accusation in round $k + 1$ is not broadcast in round $k + 2$. Node 1, which suffered benign faults in rounds k and $k + 1$, sends a correct local syndrome in round $k+2$, because it could receive message frames even during benign faulty periods[†].

3.3.3 Properties

Lemma 1 Node i is diagnosed as faulty in temporal health vector $\hat{hv}$ in round $k+1$ by any obedient node if and only if i suffers a benign fault in round k.

Proof: If i suffers no benign fault in round k, then no obedient node sets the ith bit to 0 in its local syndrome in round $k+1$. From Equation (3.1), we have:

$$s < n - s - b - 1 \tag{3.2}$$

Let p be any obedient node (which is possibly i) and let V_i denote the ith column of p's local syndrome matrix. Then, $N_0(V_i) \leq s$ and $N_1(V_i) \geq n - b - s - 1$[‡]. From Equation (3.2), $N_0(V_i) < N_1(V_i)$, thus $H\!-\!maj(V_i) = 1$ and node i is diagnosed as correct in $\hat{hv}$. If i suffers a benign fault in round k, then no obedient node sets the ith bit to 1 in its local syndrome in round $k+1$. Because of the same argument as in the case where i suffers no benign faults, $H\!-\!maj(V_i) = 0$ and node i is diagnosed as faulty in $\hat{hv}$.

Lemma 2 Node i is accused as symmetric Byzantine faulty in the minority accusation by any obedient node in round $k+1$ if and only if i sends an erroneous local syndrome in round $k+1$.

Proof: Let p be any obedient node. If i sends a correct local syndrome in round $k+1$, then by Lemma 1, p's temporal health vector $\hat{hv}$ is the same as the local syndrome received from i, and thus i is not accused. If the local syndrome sent by i is lost, then i

[†]We only assume that a message frame from a benign faulty node cannot be *sent* to the other nodes in the fault model discussed in Section 3.2.

[‡]The term -1 is necessary since p's evaluation of itself is discarded.

is not accused in the minority accusation by p. If i sends an erroneous local syndrome in round $k+1$, then the local syndrome differs from the correct one with respect to the evaluation of some node $j(\neq i)$. Let V_j be the jth column in p's local syndrome matrix and $v \in \{0,1\}$ and $1-v \in \{0,1\}$ be the correct and erroneous values for the evaluation of node j. Because of the same argument as in Lemma 1, $N_v(V_j) > N_{1-v}(V_j)$. Therefore, i is accused as symmetric Byzantine faulty by p in the minority accusation.

Theorem 1 The following two properties hold:

- *Correctness*: a correct node is never diagnosed as faulty in the health vector of any obedient nodes.
- *Completeness*: a faulty node that suffers a benign fault in round k or sends an erroneous local syndrome in round $k+1$ is always diagnosed as faulty in the health vector of all obedient nodes in round $k+1$.

Proof: The theorem directly follows from Lemmas 1 and 2.

Theorem 2 *Consistency*: the health vector is agreed by all obedient nodes in each round.
Proof: Because of Theorem 1, the set of nodes identified as correct or faulty in the health vector is agreed by all obedient nodes. Hence, the theorem follows.

Theorem 3 *Consistent Isolation*: the set of active nodes is agreed by all obedient nodes in each round.
Proof: Let hv denote the health vector obtained by an obedient node. Because of Theorem 2, hv is agreed by all obedient nodes. Since updating function $updateCounter$ is deterministic, its output $updateCounter(hv)$ is also agreed by all obedient nodes.

CHAPTER 4

PROTOTYPE BRAKE-BY-WIRE SYSTEM

4.1 Introduction

The membership protocol was implemented on a hardware and software platform equivalent to one assumed to be used in commercial X-by-Wire systems. Subsequently, we developed a prototype FlexRay-based brake-by-wire system based on the proposed autonomous decentralized architecture to evaluate its function and the required overhead, especially the CPU computational load of the membership middleware. This chapter describes details of implementation and results obtained from evaluating the function of the prototype system and the performance of the membership middleware.

4.2 Software Architecture

Because the membership function is separable from the backup control logic executed by an application program, a natural design choice is to implement it in a middleware layer. We implemented the membership middleware with other software modules required in

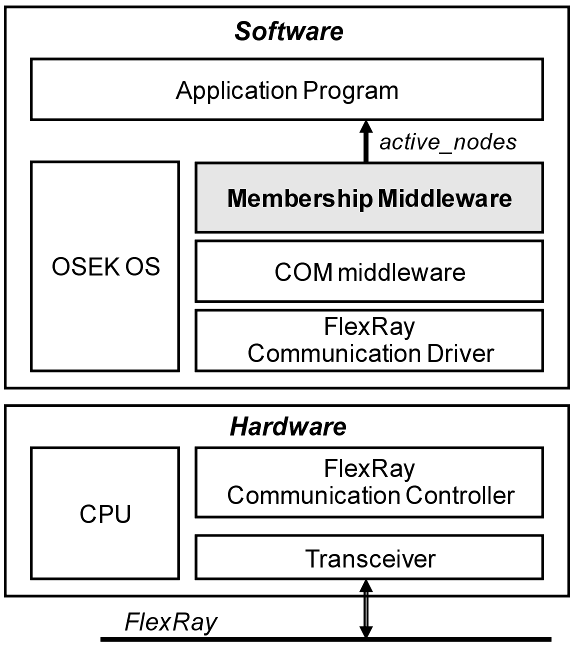


Figure 4.1: Software architecture

actual X-by-Wire systems on a Renesas M32C microcontroller capable of FlexRay communication, and developed a prototype autonomous decentralized brake-by-wire system.

Figure 4.1 shows the software architecture in each node. FlexRay is used for the communication network. The software is composed of basic software including OSEK OS (Operating System) [28], communication (COM) middleware, and a FlexRay communication driver, membership middleware, and an application program to execute the brake control algorithm. A three-phase brushless motor control application was implemented in the brake ECUs.

The COM middleware, which is not a standardized one like OSEK COM or AUTOSAR COM, provides features such as frame packing/unpacking, fault detection above the data link layer by adding and checking the checksum code and sequential number, and handling the redundant frames.

The membership middleware executes the sequence indicated in Figure 3.4, where it evaluates the status of other nodes based on the fault information stored in the COM middleware and communication driver, exchanges the local syndrome with all nodes, and determines other nodes' statuses by voting on the exchanged local syndromes. The middleware offers an API (Application Program Interface) to provide the application programs with the status information of other nodes, i.e., *active_nodes*. The application programs switch the vehicle control logic according to this information. Because the application program itself does not need to monitor the node status, the development efficiency of the application program can be improved, thanks to the membership middleware. We regard the membership function as one of the essential features of the NM (Network Management) layer for safety-critical distributed systems.

Executing the membership middleware has to be synchronized with FlexRay communication, as we discussed in Section 3.3. One solution to achieve this is to use a time-triggered OS such as OSEKTime [29]. However, time-triggered tasks in OSEKTime have higher priority than OSEK event-triggered tasks and interrupts. This feature is suitable for complete time-triggered systems, but automotive control systems generally have some event-triggered tasks and interrupts with the highest priority, e.g., fuel injection and ignition control synchronized to engine revolution. Furthermore, the task scheduling strategy of OSEKTime is stack-based scheduling, where a running task is always preempted by another task regardless of its priority. This strategy is completely different from that of OSEK, which can implement multiple task execution with a priory-based scheduling scheme, i.e., a task with higher priory is not preempted by a lower priority task. The above-mentioned engine control and three-phase brushless motor control require multiple tasks to be executed. Therefore, OSEK was used in our implementation based on this analysis, as shown in Figure 4.1, so that the membership middleware task could coexist with the motor control task.

Figure 4.2 shows how we synchronize the OSEK tasks including the membership task with FlexRay communication. The absolute timer, which counts the time elapsed from the start of the communication round and is reset at the start of every communication round, is implemented in the FlexRay communication controller. We use an interrupt handler activated by this timer. In the interrupt handler, a *SetEvent* call is issued to a task that has to be activated at that time, and the next task activation time is set. The task receiving *SetEvent* makes a transition from the *wait* state to the *ready* state. When multiple tasks receive *SetEvent*, such as at time T_2 in Figure 4.2, the OSEK scheduler determines the order of execution for these tasks based on task priority. The task transitions to the *wait* state again by calling *WaitEvent* at the end of the task, and waits for *SetEvent* to be called in the next communication round.

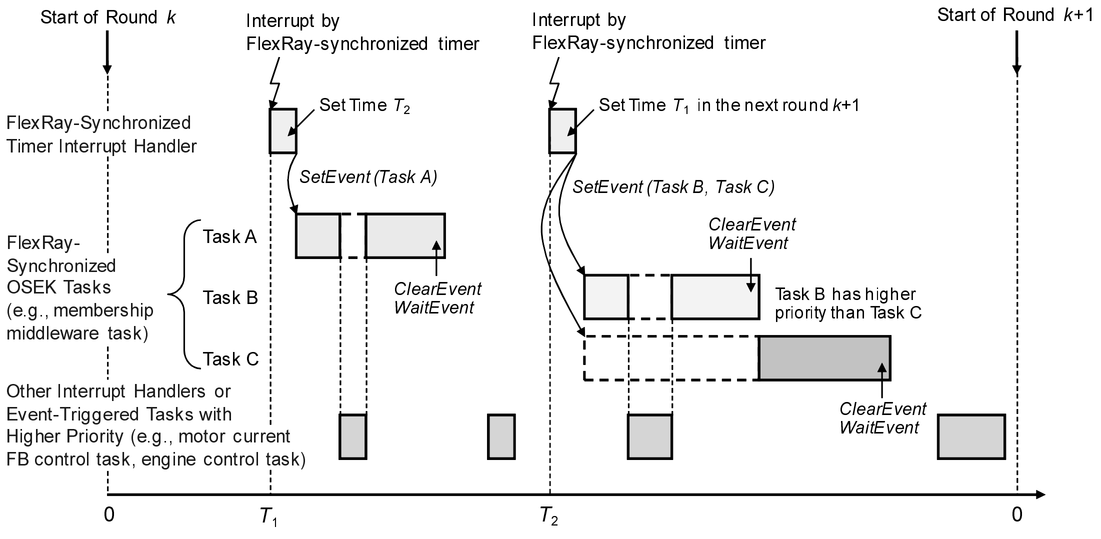

Start of Round k
Interrupt by FlexRay-synchronized timer
Interrupt by FlexRay-synchronized timer
Start of Round k+1
FlexRay-Synchronized Timer Interrupt Handler
Set Time T_2
Set Time T_1 in the next round k+1
SetEvent (Task A)
SetEvent (Task B, Task C)
FlexRay-Synchronized OSEK Tasks (e.g., membership middleware task)
Task A
Task B
Task C
ClearEvent WaitEvent
ClearEvent WaitEvent
Task B has higher priority than Task C
ClearEvent WaitEvent
Other Interrupt Handlers or Event-Triggered Tasks with Higher Priority (e.g., motor current FB control task, engine control task)
0
T_1
T_2
0

Figure 4.2: Synchronization of FlexRay communication and OSEK tasks

OSEK tasks can be synchronized with FlexRay communication in this way. Furthermore, as shown in Figure 4.2, because tasks and interrupt handlers with higher priority than that of the FlexRay-synchronized tasks can preempt the FlexRay-synchronized tasks, event-triggered tasks with the highest priority such as the engine control task and the motor current feedback control task, are not influenced by FlexRay-synchronized tasks, and both kinds of tasks can exist together on the same processor.

One point to note is that FlexRay-synchronized tasks can only be executed when the FlexRay communication controller is in a state where the absolute timer is operating (i.e., "Normal Active" state). Therefore, we have to implement a FlexRay communication-independent task that monitors the state of the communication controller and executes certain exception handling in case the state is other than that state.

4.3 Prototype Implementation and Evaluation

We implemented this software architecture on a Renesas M32C microcontroller integrating a FlexRay communication controller, and developed a prototype autonomous decentralized brake-by-wire system. Figure 4.3 indicates the structure of the prototype system, which consists of six ECUs, i.e., a brake pedal sensor ECU, an integrated ECU that calculates the target braking force values for four wheels based on the brake pedal position, and four brake ECUs (Front-Left, Front-Right, Rear-Left, and Rear-Right) for braking motor control. These ECUs communicate with one another via FlexRay of 5 Mbps baud rate and a communication round of 5 ms.

The rear-left brake ECU executes position servo control of a real three-phase brushless motor with a 150 μs current feedback control loop. The control loop is implemented by a timer interrupt handler that is independent of FlexRay communication and executed at higher priority than that of the membership functionality implemented with the FlexRay-synchronized OSEK task.

Furthermore, as shown in Figure 4.3, the accelerator pedal position signal, steering wheel angle signal, and braking force signals, which the four brake ECUs generate, are input to a real-time vehicle dynamics simulator to observe the vehicle motion during fault injection such as an ECU power supply shutdown or disconnection of the network

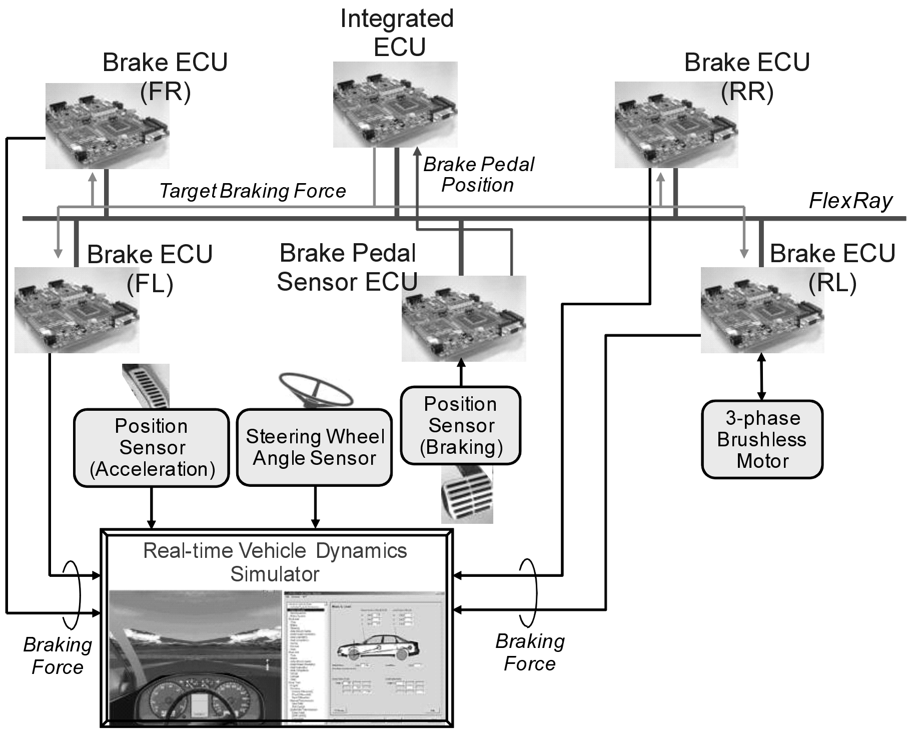


Figure 4.3: Prototype autonomous decentralized brake-by-wire system

channel.

First, functions were evaluated when faults occurred in this prototype system. When the integrated ECU was shut off, the middleware in all the remaining normal ECUs could identify the failure in the integrated ECU. The four brake ECUs shifted to autonomous backup control, where they directly obtained the brake pedal position signal from the network and calculated the target braking force based on the pedal position signal by themselves. Even if the integrated ECU failed, the vehicle maintained the braking function without generating unintended yaw moment, and we successfully demonstrated that the autonomous backup control function worked well in the proposed architecture. Moreover, we found that the vehicle could stably decelerate even when simultaneous faults occurred, e.g., when the power supply of the integrated ECU was shut off and one of the brake ECUs was disconnected from the network.

Table 4.1: Execution time and CPU load in prototype system

Process	Execution time (μs)	CPU load (%)
Frame reception	570	11
Frame transmission	70	1.4
Membership	290	5.8
Motor control	70	47

We also evaluated the performance of the membership middleware. Table 4.1 summarizes the execution time and CPU load for each process on the rear-left brake ECU in which the motor control function was also implemented. These results were based on the following conditions: six nodes, a 5 ms FlexRay communication round, a Renesas M32C microcontroller with a 40 MHz CPU clock and bus of 16 bits, and an IAR EWM32C compiler without any compiler options. The rear-left brake ECU received five message frames that contained the local syndromes from the other five nodes, and two message frames that contained the control application data from the integrated ECU and the brake pedal sensor ECU. The frame reception and transmission execution times in Table 4.1 include those of both the FlexRay communication driver and the COM middleware which provides features such as frame packing/unpacking, fault detection above the data link layer by adding and checking the checksum code and sequential number, and handling of redundant frames. The execution time of the membership process under these conditions was 290 μs, which means it consumed 5.8% (290 μs divided by the communication round of 5 ms) of the whole available CPU time.

Furthermore, we investigated the execution time needed to compute the health vector by the voting process in systems consisting of 4, 5, and 6 nodes. Each node had a microcontroller with a 40 MHz CPU clock. Table 4.2 reveals that the voting execution time

Table 4.2: Voting execution time with respect to number of nodes

Number of nodes	4	5	6
Voting execution time (μs)	78	121	169

increases quadratically with respect to the number of nodes. This is because the local syndrome matrix contains $n \times n$ bits for an n-node system, as seen in Figure 3.5.

Finally, we confirmed that motor position servo control could be adequately executed without interference by the membership service, and that the membership task also satisfied a specified deadline, i.e., the start time of the task in the next communication round, although it took longer to execute due to the interruption by the motor control task. Therefore, we can conclude that the membership middleware can coexist with application programs like three-phase brushless motor control that have to satisfy severe real-time requirements with the proposed method of synchronization between FlexRay communication and OSEK tasks.

4.4 Discussion

The overhead of 5.8% evaluated in the prototype system might sound small, but in practice, this is prohibitively large, because CPU time is already a very scarce resource. It is ideal to assign as much CPU time as possible to control application programs to achieve better driving performance and safety in automotive systems. Thus, it is not acceptable in practice to spend that much CPU time only on the membership service. Furthermore, the execution time for the voting increases with the square of the number of nodes.

Using faster CPUs could mitigate the problem to some extent. In fact, we can predict that the CPU load will be less than that in these evaluation results since we will be able to use much higher performance microcontrollers with around 100 MHz CPU when X-by-Wire systems will actually become commercially available. However, the demands for high levels of responsiveness in driving control will be simultaneously increasing. For example, current automotive control systems often require a much shorter communication round time than that in our prototype system. Such increasing demands for shorter round times easily offset the increase in CPU performance.

Therefore, we have to further develop membership protocols that consume less CPU resources, even in large n-node systems that can have more than 10 or 20 nodes in the actual automotive control systems.

CHAPTER 5

LIGHTWEIGHT MEMBERSHIP PROTOCOL

5.1 Introduction

We propose two membership protocols to reduce the overhead, i.e., *voting sharing* and *clustering*, to address the problem that the protocol requires much CPU resources, especially for systems consisting of large number of nodes, as discussed in Sections 4.3 and 4.4.

Both approaches aim to decrease the computation overhead for voting in the membership protocol. The main idea behind voting sharing is to have each node vote for only one respective node and to share the voting results with all nodes. On the other hand, in the clustering protocol, n nodes are logically divided into n/c clusters, where each cluster consists of c nodes. Node p sends a local syndrome only with respect to nodes within the cluster to which node p belongs to all the other nodes in the system. The clustering protocol can also reduce the communication bandwidth, compared with the original protocol.

This chapter discusses both protocols in detail and presents experimental results on the execution time. We further analyze a tradeoff between the overhead and fault-tolerance in the proposed three membership protocols including the original protocol.

5.2 Voting Sharing

In the voting sharing protocol, each node performs voting to detect a fault in a single node and to share its voting result with all the nodes. Each node has a responsibility to vote on the exchanged local syndromes with respect to one specific node and to broadcast the result in the following round. As a result, the computation cost required for voting is reduced compared with the original approach which votes for each of the n nodes.

5.2.1 Protocol

The execution of the protocol now involves at least three rounds, instead of two rounds, as outlined in Figure 5.1. The pseudo-code of the process in the voting sharing protocol executed in each node p in round k is also presented in Algorithm 2.

A node locally evaluates other nodes' statuses in round k, as is done in the original protocol (EVA phase).

In round $k+1$, the local syndrome representing errors observed in round k is broad-

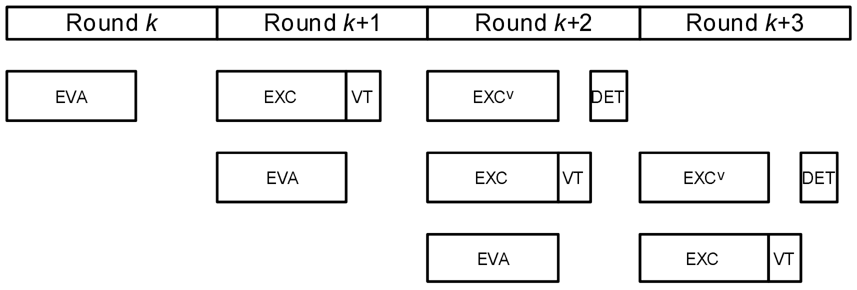


Figure 5.1: Execution sequence for voting sharing protocol

cast, just as is done in the original protocol (EXC phase). Then, in VT phase, each node p only votes for a single node, unlike what occurs in the original protocol. Node p extracts one element that is associated with a node that p is responsible for from each of the local syndromes received. We define node $target(p, k)$ as the node for which p is responsible for diagnosis in the protocol execution that starts from round k. Thus, node p collects $n-1$ values (excluding $target(p, k)$'s own evaluation), each of which is either 1, 0, or ε. Node p carries out hybrid voting on these $n - 1$ values (Algorithm 2, line 26). The result of voting, $result\_p[target(p, k)]$, is either 1 or 0. Following the hybrid voting, node p executes the minority accusation described in Subsection 3.3.1 by comparing the voting result with the $n - 1$ values excluding $target(p, k)$ (lines 29-31). Finally, the result of the diagnosis is encoded as an n-bit vector, $result\_p[\,]$, where 0 on the ith bit means that node i has been diagnosed as benign faulty ($target(p, k) = i$) or symmetric Byzantine faulty ($target(p, k) \neq i$). The result vector $result\_p[\,]$ is buffered in $send\_result\_msg\_p[\,]$ to be exchanged in the next round (line 37).

In round $k + 2$, the results of the voting and the minority accusation are broadcast (EXC^V phase). This phase can be combined with the EXC phase of the next protocol execution by adding the n-bit diagnosis data to the local syndrome. Each node collects diagnosis results from n nodes including itself, to construct a result matrix (lines 46-49).

Finally, in the DET phase, the node obtains the status for node i in the health vector, $hv[i]$, by computing a bitwise AND of all received n-bit results in the ith column of the result matrix, i.e., $result\_matrix[\,][i]$ (line 55). After health vector hv is calculated, the set of active nodes is updated to the return value of function $updateCounter(hv)$, which is deterministic.

Algorithm 2: Node p voting sharing membership process in round k

```
// **EVA Phase**
// local syndrome evaluation
**for** $i \leftarrow 1, \dots, n$ **do**
    $ls_{k-1}\_i[1, \dots, n] \leftarrow receive\_ls_{k-1}\_msg\_i[1, \dots, n]$;
    **if** $ls_{k-1}\_i[1, \dots, n] = \Phi$ **then** // no received message
        $ls_k\_p[i] \leftarrow 0$;
    **else**
        $ls_k\_p[i] \leftarrow 1$;
    **endif**

// local syndrome buffering to be sent in round $k + 1$
$send\_ls_k\_msg\_p[1, \dots, n] \leftarrow ls_k\_p[1, \dots, n]$;

// **EXC Phase**
// local syndrome exchange and local syndrome matrix construction
**for** $i \leftarrow 1, \dots, n$ **do**
    **if** $ls_{k-1}\_i[1, \dots, n] = \Phi$ **then** // no received message
        $ls_{k-1}\_matrix[i][1, \dots, n] \leftarrow [\varepsilon, \dots, \varepsilon]_n$;
    **else**
        $ls_{k-1}\_matrix[i][1, \dots, n] \leftarrow ls_{k-1}\_i[1, \dots, n]$;
    **endif**
    $ls_{k-1}\_matrix[i][i] \leftarrow -$;

// **VT Phase**
// hybrid voting
$result_{k-1}\_p[target(p, k)] \leftarrow H\text{-}maj(ls_{k-1}\_matrix[1, \dots, n][target(p, k)])$;

// minority accusation
**for** $i \leftarrow 1, \dots, target(p, k) - 1, target(p, k) + 1, \dots, n$ **do**
    **if** $ls_{k-1}\_i[target(p, k)] \neq result_{k-1}\_p[target(p, k)]$ **then**
        $result_{k-1}\_p[i] \leftarrow 0$;
    **else**
        $result_{k-1}\_p[i] \leftarrow 1$;
    **endif**

// diagnosis result buffering to be sent in round $k + 1$
$send\_result_{k-1}\_msg\_p[1, \dots, n] \leftarrow result_{k-1}\_p[1, \dots, n]$;

// target node rotation for voting in round $k + 1$
$target(p, k + 1) \leftarrow rotate(target(p, k))$;

// **EXC$^{\text{V}}$ phase**
// exchange of diagnosis results with respect to round $k - 2$ computed in round $k - 1$
**for** $i \leftarrow 1, \dots, n$ **do**
    $result_{k-2}\_i[1, \dots, n] \leftarrow receive\_result_{k-2}\_msg\_i[1, \dots, n]$;
    **if** $result_{k-2}\_i[1, \dots, n] = \Phi$ **then** // no received message
        $result_{k-2}\_matrix[i][1, \dots, n] \leftarrow [\varepsilon, \dots, \varepsilon]_n$;
    **else**
        $result_{k-2}\_matrix[i][1, \dots, n] \leftarrow result_{k-2}\_i[1, \dots, n]$;
    **endif**
    $result_{k-2}\_matrix[i][i] \leftarrow -$;

// **DET phase**
**for** $i \leftarrow 1, \dots, n$ **do**
    $hv_{k-2}[i] \leftarrow result_{k-2}\_matrix[1][i] \,\&\dots\&\, result_{k-2}\_matrix[n][i]$;

// counter update
$active\_nodes \leftarrow updateCounter(hv_{k-2}[1, \dots, n])$;

**return** $active\_nodes$;
```

The $target(p, k)$ must meet two properties for any round k:

$$\forall p, q : p \neq q \Rightarrow target(p, k) \neq target(q, k)$$

$$\forall p : p \neq target(p, k)$$

The first property states that two different nodes are responsible for voting for two different nodes. This property ensures that every node in every round has its own unique node that is responsible for voting for it. The second property signifies that such a voter node is different from the node diagnosed by that voter node.

Note that in an execution of the proposed protocol that starts from round k, node i is diagnosed twice: in round $k + 1$ by the voter node p, such that $target(p, k + 1) = i$, and in round $k + 2$ by all nodes in the health vector. When necessary, we say that a node is diagnosed (as faulty or non-faulty) *in the health vector* in the latter case.

5.2.2 Properties

Here, we prove some key properties of voting sharing. These properties hold when there are no symmetric Byzantine faulty voters. We will discuss how we resolve a problem of symmetric Byzantine faulty voters in Subsection 5.2.3.

First, consistency and consistent isolation hold straightforwardly.

Theorem 4 *Consistency*: The health vector is agreed by all obedient nodes in each round.
Proof: When a node sends its diagnosis result, the message is either received correctly by all nodes, lost due to a benign fault, or received incorrectly by all nodes due to a symmetric Byzantine fault. Hence, the health vector finally obtained is identical in all obedient nodes.

Theorem 5 *Consistent isolation*: The set of active nodes is agreed by all obedient nodes in each round.

Proof: The counter updating function, $counterUpdate(hv)$, is deterministic. Because of Theorem 4, health vector hv is identical for all obedient nodes. Hence, the active nodes, which are updated to the output of the function in every round, are always identical for all obedient nodes.

Correctness and completeness hold in a somewhat weaker form than those in the original protocol.

Theorem 6 *Correctness*: Correct node q is never diagnosed as faulty in the health vector in round $k+2$ if all nodes are obedient.
Proof: Suppose that q is a correct node and all nodes are obedient. From Lemmas 1 and 2, q is always diagnosed as correct by any node in round $k+1$. The diagnosis result sent by each node in round $k+2$ is either correctly broadcast or simply lost, because the node is obedient. Hence, q is never diagnosed as faulty in round $k+2$ by any nodes in their health vector.

Theorem 7 *Completeness w.r.t benign faults*: If node q suffers a benign fault in round k, then q is diagnosed as faulty in round $k+2$ by all obedient nodes if q's voter node p (i.e., the node such that $target(p,k) = q$) is obedient and suffers no faults in round $k+2$.
Proof: From Lemma 1, if node q suffers a benign fault in round k, then q is always diagnosed as faulty by voter node p in round $k+1$ if p is obedient. If no faults occur in the voter node in round $k+2$, then the voting result 0 is correctly broadcast and occurs in the health vector of all obedient nodes in the round.

Theorem 8 *Completeness w.r.t symmetric Byzantine faults*: Suppose that symmetric Byzantine faulty node q sends an erroneous local syndrome in round $k+1$ and that the erroneous syndrome differs from the correct one with respect to the evaluation of node i. Then, q is diagnosed as faulty in round $k+2$ by all obedient nodes in their health vector if node p, such that $target(p, k) = i$, is obedient and suffers no faults in round $k + 2$.

Proof: By using the argument in Lemma 2, if node p, such that $target(p, k) = i$, is obedient, then q is always diagnosed as faulty by p in the minority accusation in round $k + 1$. If no faults occur in p in round $k + 2$, then the diagnosis result is correctly broadcast and reflected in the health vector of all obedient nodes in the round.

These weaker guarantees pose the following two problems:

- False negatives: diagnosing faulty nodes as non-faulty.
- False positives: diagnosing correct nodes as faulty.

These problems are mainly caused by symmetric Byzantine faulty voters. The next subsection explains how these problems can be addressed.

5.2.3 Rotating Voters

We propose the use of *rotating voters* to mitigate these two problems. The main idea is to change the voter node for each node in every round so that a node becomes the voter node for any other node in any consecutive $n - 1$ rounds. More concretely, node p becomes the voter of node $p + 1$ in round 1 and then changes the node it is responsible for (i.e., $target(p, k)$) to $p + 2, p + 3, ..., n, 1, 2, ..., p - 1$. This rotation is repeated every $n - 1$ rounds because of the condition, $\forall p : p \neq target(p, k)$, defined in Subsection 5.2.1. Note that the rotating voter scheme ensures the two conditions on $target(p, k)$. Function $rotate(target(p, k))$ (Algorithm 2, line 40) calculates the target node in the next round for node p ($target(p, k + 1)$)) based on these conditions.

By means of rotating voters and the design of the algorithm for the counter updating function, $counterUpdate(hv)$, the two problems can be mitigated as follows.

A false negative with respect to node q occurs if all nodes that should be able to diagnose q as faulty happen to be simultaneously faulty in the same round[*]. Rotating voters ensure that any node always becomes a voter for any other node in every consecutive $n-1$ rounds. Thus, if q suffers faults intermittently or permanently, it is safely diagnosed as faulty by correct nodes[†].

The case of false positives is trickier than that of false negatives because a symmetric Byzantine voter can repeatedly produce incorrect diagnosis results for any correct nodes. Therefore, if the counter updating function simply counted the times when each node was diagnosed as faulty, this would lead to a rapid and undesirable shrink of the set of active nodes. A possible solution to this is to offset the effects of incorrect diagnosis with those of correct ones. This solution can be implemented, e.g., by decreasing the counter if the node is diagnosed as correct. Another approach could be to use two counters for each node that represent penalties and rewards in the p-r algorithm proposed in [15].

Figure 5.2 shows the voting sharing protocol with rotating voters for 4 nodes. If $target(p, k+1)$ is node $p+1$ for node p ($1 \leq p \leq 3$) and node 1 for node 4 in round $k+1$, node p ($1 \leq p \leq 3$) votes on the local syndromes with respect to node $p+1$ and node 4 votes on node 1 in round $k+1$. In round $k+2$, $target(p, k+2)$ changes to node $p+2$ for node p ($p = 1, 2$) and node (($p+2$) mod 4) for node p ($p = 3, 4$). Node p broadcasts n-bit results consisting of one bit voting result on $target(p, k+1)$ and $n-1$-bit result of the minority accusation executed in round $k+1$ (Algorithm 2, lines 26-33). Note that in round $k+2$, along with these n-bit results, every node sends the n-bit local syndrome evaluated in round $k+1$. Therefore, the required communication bandwidth in the voting sharing protocol is $2n \times n$ bits.

[*]A node responsible for diagnosis of node q and nodes that diagnose node q as symmetric Byzantine faulty by using the minority accusation can diagnose node q as faulty.

[†]Even though there might be cases where q does not suffer faults only during voting by correct voters, we assume this probability to be quite low.

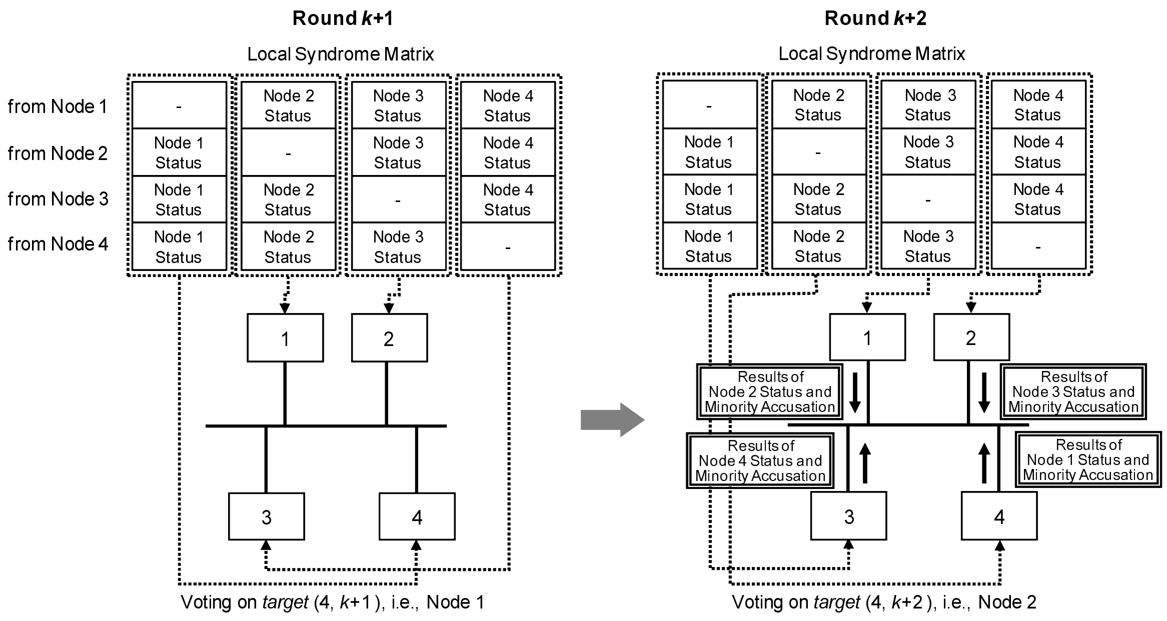

Round k+1
Local Syndrome Matrix
from Node 1
from Node 2
from Node 3
from Node 4
Node 1 Status
Node 2 Status
Node 3 Status
Node 4 Status
Voting on target (4, k+1), i.e., Node 1
Round k+2
Local Syndrome Matrix
Results of Node 2 Status and Minority Accusation
Results of Node 3 Status and Minority Accusation
Results of Node 4 Status and Minority Accusation
Results of Node 1 Status and Minority Accusation
Voting on target (4, k+2), i.e., Node 2

Figure 5.2: Voting sharing protocol with rotating voters

5.2.4 Experiment Results

This section presents the results we obtained from our experiment. We developed a prototype system that was equipped with four to eight nodes. Each node was equipped with a 60 MHz CPU.

The results are summarized in Table 5.1. The row “Voting” indicates the time required for 1) calculating a health vector (DET phase) in the original membership protocol, and that for 2) hybrid voting of the target node (VT phase) and calculating a health vector (DET phase) in the voting sharing protocol. The row “All” indicates the total time used for the membership service including the execution time to receive and send message frames.

Although various processing overheads such as task switching are added to the pure voting calculation, which is theoretically reduced by a factor of the number of nodes, the time to execute the voting process in the voting sharing is reduced by approximately 60%

Table 5.1: Comparison of execution time in original and voting sharing protocols

	Original			Voting sharing		
Number of nodes	4	6	8	4	6	8
Voting (μs)	16.06	26.72	34.60	10.22	12.85	14.60
All (μs)	86.42	133.43	181.17	82.19	121.46	161.31

compared with the original protocol for eight nodes. The total execution time for the membership service is decreased by approximately 10%, which is substantially effective for industrial embedded systems with extremely limited hardware resources.

5.3 Clustering

The n nodes in the clustering protocol are logically divided into n/c clusters, where each cluster consists of c nodes, as shown in Figure 5.3. Even though we assume that n can be divided by c for simplicity in this example, the following discussion can also be applied to systems that have n nodes not dividable by c.

The underlying concept of this protocol is that node p transmits a local syndrome *only* with respect to nodes *within the cluster to which node p belongs* to all the other nodes

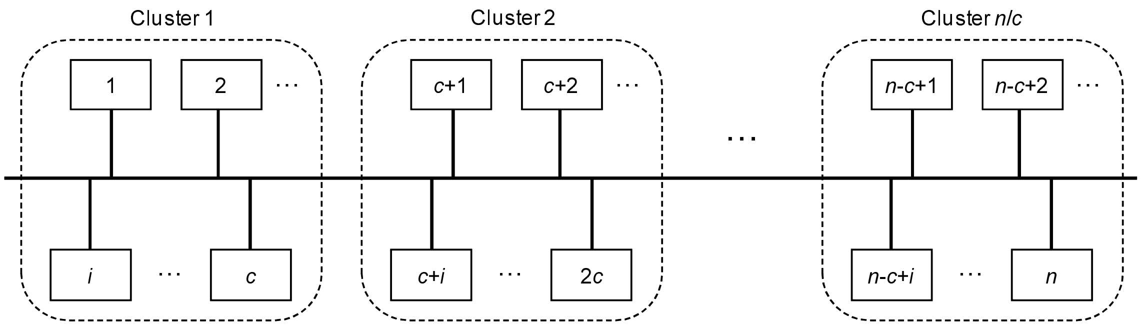


Figure 5.3: Clustering of nodes

in the system. The data size of the transmitted local syndrome is thus reduced from n to c bits, which reduces the communication bandwidth as well as the computation cost compared with the original protocol.

5.3.1 Protocol

The execution of the clustering protocol requires two rounds to identify the node status, which is the same as the original protocol. Figure 5.4 indicates a local syndrome matrix in each node. The pseudo-code of the process in the clustering protocol executed in each node p in round k is also described in Algorithm 3. As explained in Subsection 3.3.1, we define $information_{k\_}p[i, \ldots, j]$ as $information$ on statuses with respect to from node i to node j in round k computed by node p. $\mathrm{INT}(a/b)$ denotes a quotient of a/b.

In round k, a node locally evaluates all the other nodes' statuses, as is done in the original protocol (EVA phase).

Then, the local syndrome expressing errors observed in round k are broadcast in round $k + 1$, just as is done in the original protocol (EXC phase). Unlike what is done in the previous two approaches, if node p is a member of cluster m, it sends a local syndrome with respect to nodes within cluster m (Algorithm 3, lines 15-16). In the clustering protocol, the bit for node p itself in the local syndrome has a dedicated function, which will be discussed in Subsection 5.3.3.

In the DET phase, node p calculates the status of node i ($1 \leq i \leq n$) by hybrid voting on the local syndromes sent from the nodes in the same cluster to which node i belongs (line 35). The minority accusation is also performed for each cluster (lines 42-43), as is done in the original protocol. Thus, each node generates health vector hv for all nodes in the system by the end of round $k + 1$. Finally, a counter is handled by computing $updateCounter(hv)$ and the set of active nodes is updated like the other two protocols.

Every node in this protocol has $c \times c$ bits of local syndromes for cluster m. Since there

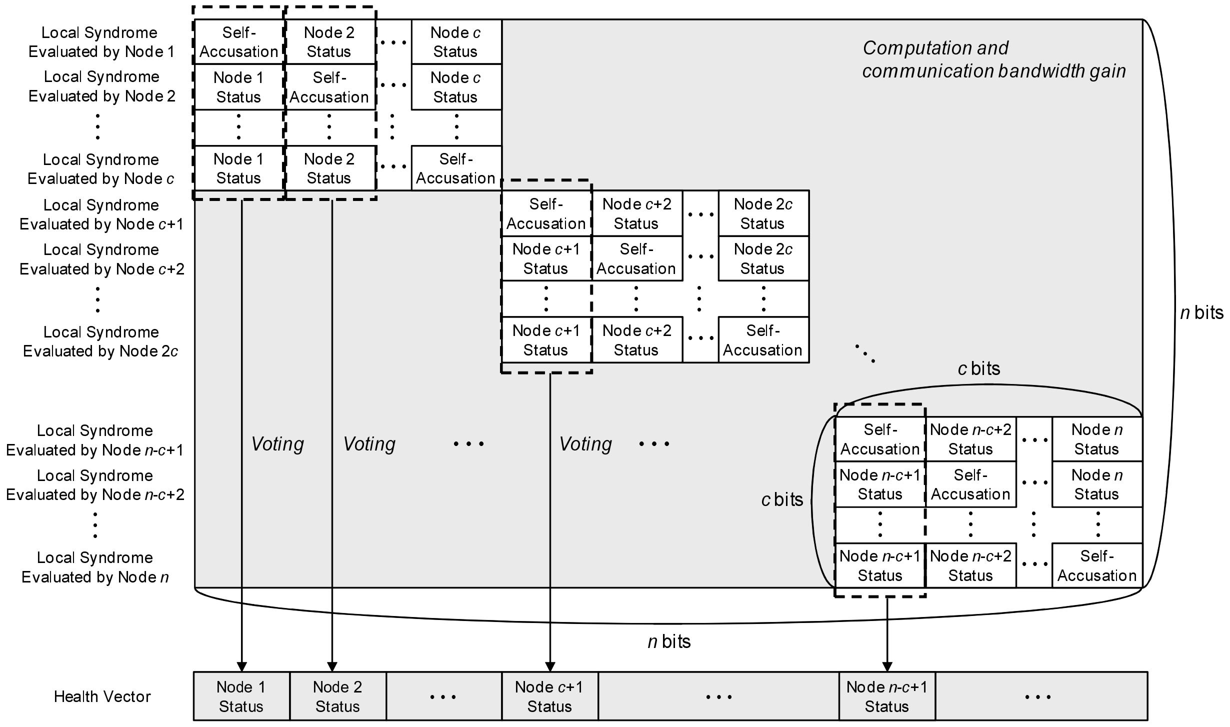

Local Syndrome Evaluated by Node 1
Local Syndrome Evaluated by Node 2
Local Syndrome Evaluated by Node *c*
Local Syndrome Evaluated by Node *c*+1
Local Syndrome Evaluated by Node *c*+2
Local Syndrome Evaluated by Node 2*c*
Local Syndrome Evaluated by Node *n*-*c*+1
Local Syndrome Evaluated by Node *n*-*c*+2
Local Syndrome Evaluated by Node *n*
Self-Accusation
Node 2 Status
Node *c* Status
Node 1 Status
Node *c*+1 Status
Node *c*+2 Status
Node 2*c* Status
Node *n*-*c*+1 Status
Node *n*-*c*+2 Status
Node *n* Status
Computation and communication bandwidth gain
n bits
c bits
c bits
n bits
Voting
Voting
Voting
Health Vector

Figure 5.4: Local syndrome matrix in clustering protocol

are n/c clusters in the system, we can reduce the local syndrome matrix to only $n \times c$ bits, which reduces the communication bandwidth and the execution time for the voting process by a factor of c/n compared with the original membership protocol. The shadowed area in Figure 5.4 represents the gain in computation and communication bandwidth.

5.3.2 Properties

Here, we prove four key properties of the clustering protocol. All the four properties are straightforwardly derived from the lemmas of the original protocol if the fault condition (3.1) is modified as follows:

$$2s + b + 1 < c \tag{5.1}$$

where c is the number of nodes in a cluster.

Theorem 9 *Correctness*: Correct node q is never diagnosed as faulty in the health vector in round $k+1$ if all nodes are obedient.

Proof: Suppose that q is a correct node and all nodes are obedient. From Lemmas 1 and 2, q is always diagnosed as correct by any node inside and outside q's cluster in round $k+1$ if n is substituted with c in the discussion of Lemma 1.

Theorem 10 *Completeness*: A faulty node that suffers a benign fault in round k or sends an erroneous local syndrome in round $k+1$ is always diagnosed as faulty in the health vector of all obedient nodes in round $k+1$.

Proof: Whenever faulty node q is benign faulty or symmetric Byzantine faulty, q is diagnosed as faulty by any node inside and outside q's cluster because of Lemmas 1 and 2 if n is substituted with c in the discussion of Lemma 1.

Theorem 11 *Consistency*: The health vector is agreed by all obedient nodes in each round.

Proof: Because of Theorems 9 and 10, the set of nodes identified as correct or faulty in the health vector is agreed by all obedient nodes. Hence, the theorem holds.

Theorem 12 *Consistent isolation*: The set of active nodes is agreed by all obedient nodes in each round.

Proof: Let hv denote the health vector obtained by an obedient node. Because of Theorem 11, hv is agreed by all obedient nodes. Since updating function $updateCounter$ is deterministic, its output $updateCounter(hv)$ is also agreed by all obedient nodes.

Algorithm 3: Node p clustering membership process in round k

```
const $u = \mathrm{INT}(p/c)$;

// EVA Phase
// local syndrome evaluation
for $i \leftarrow 1, \ldots, n$ do
    $v = \mathrm{INT}(i/c)$;
    $ls_{k-1}\_i[c \cdot v+1, \ldots, c \cdot (v+1)] \leftarrow receive\_ls_{k-1}\_msg\_i[c \cdot v+1, \ldots, c \cdot (v+1)]$;
    if $ls_{k-1}\_i[c \cdot v+1, \ldots, c \cdot (v+1)] = \Phi$ then // no received message
        $ls_k\_p[i] \leftarrow 0$;
    else
        $ls_k\_p[i] \leftarrow 1$;
    endif

// local syndrome buffering to be sent in round $k+1$
for $m \leftarrow c \cdot u+1, \ldots, p-1, p+1, \ldots, c \cdot (u+1)$ do // wrt. nodes within node $p$'s cluster
    $send\_ls_k\_msg\_p[m] \leftarrow ls_k\_p[m]$;

// EXC Phase
// local syndrome exchange and local syndrome matrix construction
for $i \leftarrow 1, \ldots, n$ do
    $v = \mathrm{INT}(i/c)$;
    if $ls_{k-1}\_i[c \cdot v+1, \ldots, c \cdot (v+1)] = \Phi$ then // no received message
        $ls_{k-1}\_matrix[i][c \cdot v+1, \ldots, c \cdot (v+1)] \leftarrow [\varepsilon, \ldots, \varepsilon]_c$;
    else
        $ls_{k-1}\_matrix[i][c \cdot v+1, \ldots, c \cdot (v+1)] \leftarrow ls_{k-1}\_i[c \cdot v+1, \ldots, c \cdot (v+1)]$;
    endif

// DET Phase
// hybrid voting
for $i \leftarrow 1, \ldots, n$ do
    $v = \mathrm{INT}(i/c)$;
    if $ls_{k-1}\_matrix[i][i] = 0$ then
        $hv_{k-1}[i] \leftarrow 0$; $sa \leftarrow i$ // result of self-accusation
    else
        $ls_{k-1}\_matrix[i][i] \leftarrow -$; $\hat{h}v_{k-1}[i] \leftarrow H\text{-}maj(ls_{k-1}\_matrix[c \cdot v+1, \ldots, c \cdot (v+1)][i])$;
    endif

// minority accusation
for $i \leftarrow 1, \ldots, sa-1, sa+1, \ldots, n$ do
    $v = \mathrm{INT}(i/c)$;
    for $j \leftarrow c \cdot v+1, \ldots, c \cdot (v+1) \wedge j \neq sa$ do
        if $ls_{k-1}\_i[j] \neq \hat{h}v_{k-1}[j]$ then
            $hv_{k-1}[i] \leftarrow 0$;
        else
            $hv_{k-1}[i] \leftarrow \hat{h}v_{k-1}[i]$;
        endif

// self-accusation
$send\_ls_k\_msg\_p[p] \leftarrow 1$;
for $i \leftarrow 1, \ldots, c \cdot u, c \cdot (u+1)+1, \ldots, n$ do // wrt. nodes outside node $p$'s cluster
    if $hv_{k-1}[i] \neq 0 \wedge (ls_{k-1}\_p[i] \neq hv_{k-1}[i])$ then
        $send\_ls_k\_msg\_p[p] \leftarrow 0$;
    endif

// counter update
$active\_nodes \leftarrow updateCounter(hv_{k-1}[1, \ldots, n])$;

return $active\_nodes$;
```

5.3.3 Self-Accusation

Although all the properties of the original protocol also hold in the clustering protocol, a latent symmetric Byzantine faulty node cannot be diagnosed in some cases. This is because a node sends a local syndrome with respect to nodes only within its cluster. Suppose that latent symmetric Byzantine faulty node p and correct node q belong to cluster i and j ($i \neq j$) respectively. Even if the local syndrome of node p differs from health vector hv in the qth position, which should be 1, the other nodes except for node p can never accuse node p since node p cannot broadcast the local syndrome with respect to node q.

Figure 5.5 indicates this case, where eight nodes are divided into two clusters consisting of four nodes. We assume that node 7 belonging to cluster 2 could not receive a message sent from node 3 in cluster 1 because of a receiver fault, while the other nodes in cluster 1 could correctly receive it. Node 7 in this case receives local syndromes “1111” from all the nodes in cluster 1 and calculates the health vector of the nodes in cluster 1 as “1111”. However, as node 7 could not receive a message from node 3, the local syndrome on cluster 1 evaluated by node 7 is “1101”, which is different from that of the health vector.

Node 7 is not a symmetric Byzantine faulty node by definition because it does not send erroneous messages. However, this case might cause adverse situations in practical

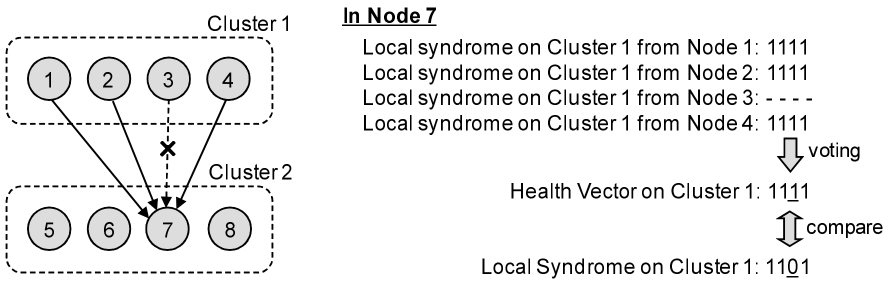

Cluster 1
1
2
3
4
Cluster 2
5
6
7
8
In Node 7
Local syndrome on Cluster 1 from Node 1: 1111
Local syndrome on Cluster 1 from Node 2: 1111
Local syndrome on Cluster 1 from Node 3: - - - -
Local syndrome on Cluster 1 from Node 4: 1111
voting
Health Vector on Cluster 1: 11$\underline{1}$1
compare
Local Syndrome on Cluster 1: 11$\underline{0}$1

Figure 5.5: Example of self-accusation

control systems. Suppose that node 7 controls some actuator based on control target data calculated in node 3's control logic. Node 3 then continues to execute control logic without noticing that the control target data have not been received by node 7, and node 3 cannot switch to the backup control function.

A diagonal element of the local syndrome matrix is used as a self-accusation bit to solve this problem, as shown in Figure 5.4. In the DET phase in round $k + 1$, with respect to each node q, node p is responsible for checking whether the local syndrome evaluated by p agrees with the health vector calculated by using the local syndromes from the nodes of q's cluster (Algorithm 3, lines 50-51). If not, node p accuses itself by setting the self-accusation bit in its local syndrome to "faulty", i.e., 0 (line 52).

The self-accusation result is broadcast in round $k + 2$ to all nodes in the system. Thus, all nodes recognize that node p suffered a latent symmetric Byzantine fault in round k and set the pth bit in the health vector to 0 (lines 32-33)[‡]. Note that hybrid voting is executed only with respect to nodes which are not self-accused (line 35). Moreover, node p has to execute self-accusation only with respect to nodes whose bits in the health vector are correct in this extended protocol (line 51), otherwise node p may have to set the self-accusation bit to "faulty" even though node p is correct, which will further be discussed in Section 6.5.

Due to the self-accusation mechanism, node 3 in Figure 5.5 can switch the normal control mode to the backup control mode in which node 7 cannot control the actuator.

[‡]Although the actual Byzantine faulty node may not set the self-accusation bit to 0, it will immediately be accused by nodes within its cluster with the original minority accusation mechanism, because such a Byzantine faulty node usually sends a local syndrome with random values.

5.4 Discussion

This section summarizes advantages and drawbacks of the three proposed membership protocols of original, voting sharing, and clustering. Table 5.2 compares the three protocols in terms of five features: diagnosis accuracy, tolerable number of faults, computational overhead, required communication bandwidth, and diagnosis latency. We assume that the computational overhead mostly stems from voting calculations and estimate this from the data size of the local syndromes to be voted on although the actual execution time depends on the software implementation.

The general characteristics indicate that there is a tradeoff between the computational overhead and the diagnosis accuracy. Although the cost of the lightweight protocols involves small degradation in diagnosis accuracy, the computational overhead can be substantially reduced. However, degradation in diagnosis accuracy can be mitigated with additional mechanisms such as rotating voters, counter update algorithms, and self-accusation, as discussed in Subsections 5.2.3 and 5.3.3. The drawbacks of the lightweight protocols also include the increase in the required communication bandwidth and diagnosis latency in the voting sharing protocol, and the decrease in the tolerable number of faults in the clustering protocol.

The required communication bandwidth increases in the voting sharing because every node sends the n-bit voting and minority accusation results in addition to the n-bit local syndrome.

We further analyze the degradation of diagnosis latency in the voting sharing. As shown in Figure 5.1, the diagnosis latency is three communication rounds if voters are non-faulty. However, if voter p suffers either benign or symmetric Byzantine fault in round k and the fault lasts permanently, the correct diagnostic result of $target(p, k)$ cannot be sent in round $k + 2$, and thus the diagnosis latency increases. The latency increases by

Table 5.2: Comparison of three proposed membership protocols

Features \ Protocols	Original	Voting sharing	Clustering
Diagnosis accuracy	+++ Correctness, completeness, and consistency properties hold	+ Correctness and completeness degrade	++ In consideration of latent Byzantine
		+++ w/ rotating voters and counter update algorithm	+++ w/ self-accusation for latent Byzantine
Tolerable number of faults	+++ $n > 2s + b + 1$	+++ $n > 2s + b + 1$ (w/ additional mechanisms)	+ $c > 2s + b + 1$
Computational overhead	+ $O(n^2)$	+++ $O(n) + hv$ calc. in DET phase	++ $O(cn)$
Required communication bandwidth (bits)	++ n^2	+ $2n^2$ (results of voting and minority accusation added)	+++ cn
Diagnosis latency (rounds)	+++ 2	+ max: $s + b + 3$ min: 3	+++ 2

one round per one faulty voter, while once the correct voter appears, the correct voting result of the target node is guaranteed to be sent within three rounds from Theorems 6, 7, and 8. Therefore, the worst case happens when voters responsible for voting on the same target node continuously fail in multiple rounds.

From the above discussion, if $target(p, k)$ is correct, $s + b$ faulty nodes can continuously be voters for $target(p, k)$, where s is the number of symmetric Byzantine faulty nodes and b is the number of benign faulty nodes, and thus the maximum latency to identify the status of $target(p, k)$ as correct is $s + b + 3$. On the other hand, if $target(p, k)$ is either benign or symmetric Byzantine faulty, $s + b - 1$ faulty nodes can continuously be voters for $target(p, k)$, and thus the latency is $s + b + 2$.

In the clustering protocol, the tolerable number of faults does not depend on n but c, which means that fault tolerance cannot be improved if we increase the number of nodes in the system, in contrast to the original and the voting sharing protocols. In other words, the original and the voting sharing protocols can tolerate more faults than the clustering protocol if n is the same and $n \geq 2c$. However, the clustering protocol is a well-balanced protocol when the system consists of large number of nodes, e.g., equal to or more than 8 or 10 nodes and the condition of the number of faults is not so severe, e.g., $s = 0, b = 2$ or $s = 1, b = 1$.

CHAPTER 6

FORMAL MODEL OF TIME-TRIGGERED SYSTEMS

6.1 Introduction

The design of safety-critical systems entails ensuring the predictability of their behavior and their overall correctness. The *time-triggered* (TT) paradigm has emerged as a viable concept to implement safety-critical systems, with implementations such as TTP/C [6], TT-Ethernet [7], FlexRay [5], or SAFEbus [8] actually deployed in the avionic and automotive fields. The use of *formal methods* is increasingly being advocated to verify general safety-critical systems (e.g., [17]). However, it has been reported in previous work [18,19] that the correctness of high-level applications for TT systems does not directly imply correctness in their implementation. Consequently, formal techniques dedicated to time-triggered systems are needed especially given their increasing deployment.

Although results on successful formal analyses of TT systems do exist, they present specific solutions (e.g., [18–20]) where modeling patterns can only partially be re-used in new projects. It is generally difficult in industry to apply formal methods to the develop-

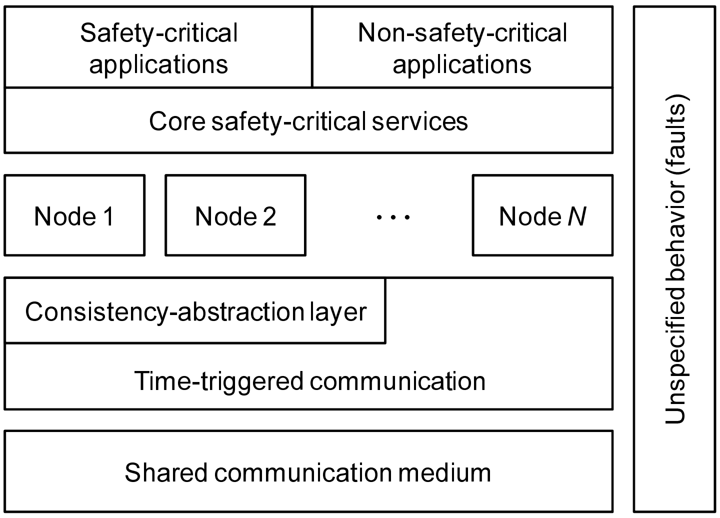

Safety-critical applications
Non-safety-critical applications
Core safety-critical services
Node 1
Node 2
· · ·
Node *N*
Consistency-abstraction layer
Time-triggered communication
Shared communication medium
Unspecified behavior (faults)

Figure 6.1: Scheme of TT systems

ment processes of mass production. Software engineers rarely design formal models of their system or software from scratch. Therefore, we propose a generalized customizable template model for TT applications that is not restricted to any dedicated implementation and that can be easily used also by engineers who are not specialists in formal methods.

Our model is an executable specification of a system that (a) not only enables verification but, (b) through simulating the system, it can also guide the deployment of applications, and (c) the effective generation of test suites.

To establish the context and contributions in this chapter, we provide a brief overview of TT systems, discuss the motivation, and highlight our proposed solutions.

Overview of system

Figure 6.1 outlines our model of TT architecture that follows the general view of TT systems [45] and augments it with a consistency abstraction layer [18, 47]. This model is generic and encompasses the one described in Section 3.2. User applications and core services are implemented by jobs with each of them running on one or more nodes. The actions of the system are triggered as time passes. The execution of jobs in the host nodes is scheduled at design time to guarantee predictability. Nodes communicate with one another using a shared bus where a communication controller grants write access to the nodes in a round-robin manner and where receiver nodes can read the bus when messages are sent.

Because the system executes safety-critical applications, our model provides core safety-critical services such as diagnosis or a membership protocol [15, 46]. In this protocol, one of the replicated jobs is executed in every node. Each job exchanges a local view on the status of nodes within the system and determines whether there are failed nodes within the system based on a majority vote of the exchanged local views. Inconsistency in the freshness of the sent and received messages should be avoided in such a protocol. This mismatch can be tackled by buffering the messages to delay operations [18, 47], whose functionality is implemented as a consistency-abstraction layer that will be discussed later in more detail. Our model also has to incorporate faults at different levels of the architecture to verify fault-tolerance of the system.

Motivation

We seek a unified and formal treatment of general TT systems to guide key system design tasks such as *task scheduling*, *test case generation*, and *verification* over user-guided applications and system configuration scenarios. Consequently, users should be able to customize the general model to describe specific applications while ensuring the required levels of system assurance.

Solutions

Deductive reasoning (e.g., theorem proving [17]) is a powerful tool to verify the complex properties of even infinite systems; however, it cannot directly be applied to simulate the system to find certain execution paths (e.g., counterexamples and test cases). Therefore, we propose the use of an executable system specification to provide further features besides verification. Since our approach uses the same model to perform different tasks of design and verification, we do not need to prove conformance between different representations. Our model is easy-to-understand as it maps each component of the high-level model (see Figure 6.1) into a syntactic module of the applied formal language. Users only need to tailor corresponding modules to customize the general model.

Our three overall contributions in this chapter are:

- We present an *executable* formal model of general TT systems. The model is intuitive and easily *customizable* for TT operations and the required classes of faults due to its *modular* structure. We use the abstraction of discrete time scales that directly stems from the assumption that the system is synchronous.
- We present a *prototype implementation* of the general model by using SAL language.
- We demonstrate the usability of our prototype by utilizing the SAL tool suite to carry out *verification*, effective *deployment*, and *test generation* based on the same model in a case study.

6.2 Overview of TT Systems

This section defines TT systems [6, 45] and a general class of faults to expand the TT communication and fault models discussed in Section 3.2 to more generic ones.

6.2.1 Basic Concepts and Definitions

A system consists of N *nodes* with unique IDs $\{1, ..., N\}$. Each node hosts one or more *jobs* that use the local resources of the node when executed. Jobs communicate with

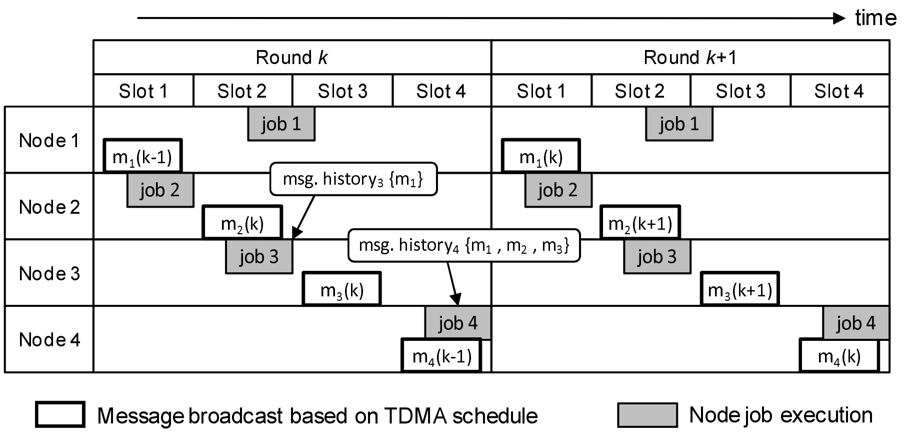


Figure 6.2: TDMA communication and internal node schedules

one another following a synchronous schedule called TDMA (Time Division Multiple Access), as can be seen in Figure 6.2. The main idea is that nodes share a communication bus in a round-robin manner*. Each node is assigned a time window, called a sending *slot*, in each TDMA *round*. Node i sends a message at sending slot i and other nodes can receive this message by identifying the sender from the time it is sent. The communication is time-triggered because the action of sending the message and receiving it is launched by the time of local clocks. Collision on the bus is avoided by assuming that clocks are synchronized. Disallowed access to the bus is avoided by so called bus guardians that physically prevent a faulty node from accessing the bus.

Besides the TDMA communication schedule, each node has its own *internal schedule* which determines when jobs are executed, as shown in Figure 6.2. Both the TDMA and the internal node schedules are independent of each other in a general model of TT systems. Both schedules are statically defined when the system is designed.

*Other TT systems, like *frame-based* systems [39], use dedicated channels and can be treated as a special case of our general model.

6.2.2 Consistency-Abstraction Layer

Fault-tolerance is often achieved by replicating application jobs on different nodes. A convenient abstraction layer is provided by a mechanism called a read/send alignment [18, 47], which enables nodes to exchange and compute messages as if there were dedicated links between every pair of nodes and the replicated jobs were executed parallel in time. This facilitates the development of applications where replicated jobs are assumed to maintain a common consistent state (e.g., diagnosis [47]). The solution is flexible since it is able to provide the same abstraction independent of how a replicated job is scheduled within the host node. Consequently, even core (transparent) services that should not assume constrained scheduling can be implemented with this technique. For example, a low-level diagnostic service can be designed without posing any assumptions about *when* the code of the service is executed within a particular node.

Example Inconsistency

Let us demonstrate the need for abstraction through a simple example of a replicated service. Assume that replication is achieved through jobs that execute the same deterministic operation in every round using the same inputs from other jobs sent via messages. Consistency is defined by requiring every replica job to have the same local state in every round after execution. The main problem in TT systems is that the *freshness* of data sent or processed by different replicas might be different. For example, in Figure 6.2, replicated job 4 reads messages m_1, m_2, and m_3 sent in the current round from nodes 1, 2, and 3 respectively. However, job 3 can read only one fresh message, m_1. Inconsistency can arise in this way in round k since jobs 3 and 4 update their local states based on different message histories. Node scheduling also determines when the message calculated by a job can actually be sent. Freshness now means whether a message can be sent in the TDMA round when it is calculated. For example, messages m_1 and m_4 contain the results calculated in the previous round as the job execution of the respective node occurs after or during the assigned sending slot. Nodes 2 and 3, on the other hand, can send fresh messages m_2 and m_3 in the current round since both jobs are completed before the sending slots of the hosting nodes. As a result, messages that are sent in the same TDMA round by different nodes might refer to different TDMA rounds. This can cause inconsistency, if, e.g., nodes want to agree on a view of the system regarding the time period of a round (e.g., diagnosis or membership [46, 47]).

Read and Send Alignment

We use *read and send alignment* [18, 47] to rectify the previous inconsistencies and provide it as a layer between jobs and the host nodes communicating via TDMA. The read alignment layer buffers the messages read from the shared bus and computes a consistent message history in the following way. Assume that a job can read the messages sent by nodes $1, ..., i$ in the current TDMA round. A consistent message history now contains $\{m_1, ..., m_i\}$ as read in the previous TDMA round, and $\{m_{i+1}, ..., m_N\}$ as read in the current round. The send alignment layer buffers the message calculated by the job and sends the old message if there is at least one other job that cannot send the newly computed message. Note that the alignment mechanisms are based on *a priori* known schedules.

6.2.3 Characterization of Faults

Faults can be manifested in any component of the system. The most obvious classification of faults in TT systems distinguishes between communication and application faults. Application faults happen during the execution of a job and are usually specific to the application logic. Faults in communication mean that a message other than intended is sent or received. We define communication faults independent of the application. Note that the number of faults tolerated by the application is generally based on their degree of severity [50]. The three different classes of communication faults are defined with ascending severity:

- *benign fault*: A fault can be detected locally by every receiver other than the sender, e.g., missing messages due to sender crashes.
- *symmetric fault*: All receivers read the same semantically incorrect but locally undetectable message, e.g., a sender processor fails and improper messages are sent.
- *asymmetric fault* (or Byzantine [41]): The most severe fault where no assumptions are made about what message is sent by a faulty sender. We categorize every fault that is neither benign nor symmetric as Byzantine, e.g., some receivers receive message m and others receive m' or do not receive any message due to sender faults or malicious intrusions.

Note that asymmetric faults are added to the fault model defined in Section 3.2. Application faults depend on the application itself. The rationale of simultaneously modeling communication and application faults is the ability to analyze their interplay with respect to the high-level specifications of the system.

6.3 Customizable Formal Model

This section proposes a template for the formal model of general TT systems, based on the previous description of TT systems. The model consists of five types of high-level elements (called modules): *controller*, *node*, *alignment*, *TDMA*, and *faults*. The modules and their interconnections are shown in Figure 6.3. We then detail the module operations and describe their interfaces. A prototype implementation of the model will be presented in Section 6.4.

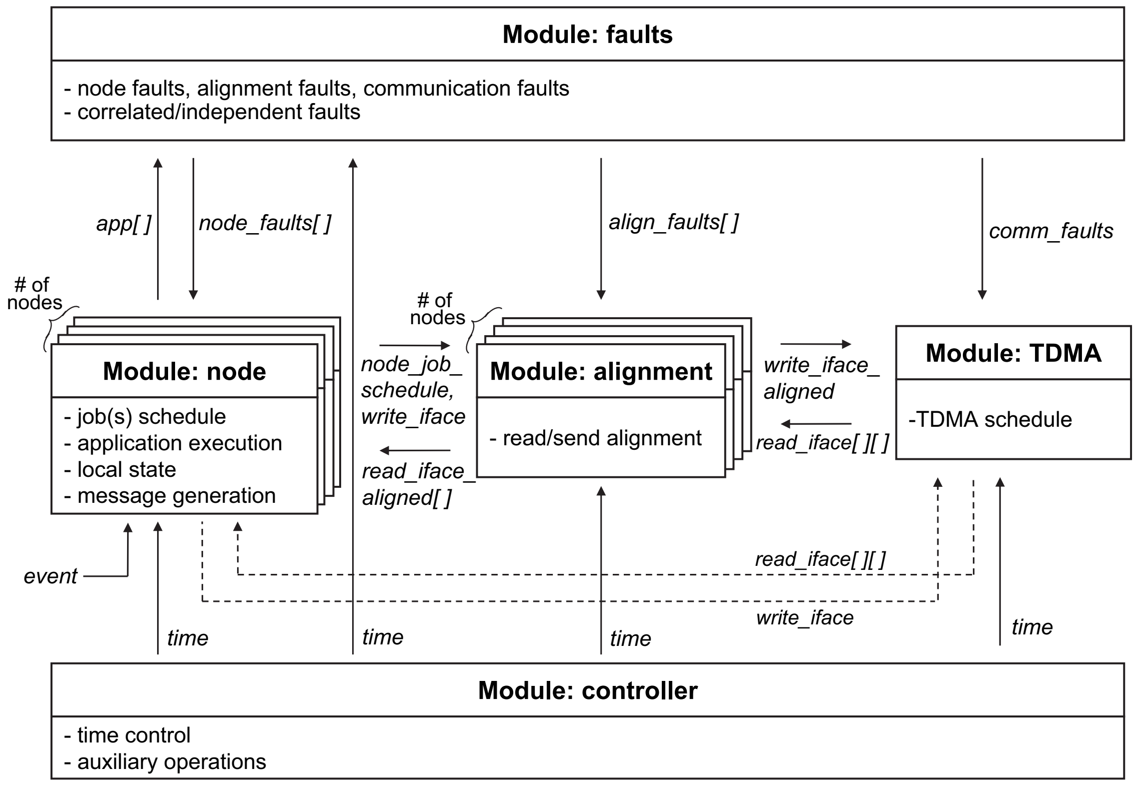


Figure 6.3: General modular structure of proposed customizable formal model

Controller Module The controller module implements the notion of *time* and distributes it to the other modules to organize synchronized execution for the system. The controller adjusts the time and triggers operations in other modules. Such a centralized treatment of time corresponds to the assumption of synchronized clocks in TT systems.

Node Module The node module has multiple instances, i.e., one for each node. This module is in charge of executing the hosted jobs when they are scheduled by updating the local state of each job and handling external events, e.g., reading sensors. We assume that every node hosts a single job to simplify further discussion. The input interface defines events (*event*), the current time, and messages received from other nodes. The messages are accessed by reading a buffer, called a read interface, which stores messages read from the bus. If a job uses the consistency abstraction, it reads consistently aligned messages (*read_iface_aligned*[]); otherwise it simply reads data from the TDMA bus (*read_iface*[]). Jobs send messages by writing them into the write interface, which are then copied and sent on the bus by the communication controller. The output interface of the node module contains the job schedule (*node_job_schedule*) and the messages sent by a job (*write_iface*). The former is required by the alignment layer, and the latter is sent directly on the bus or via alignment.

Alignment Module The alignment module is a consistency abstraction that implements read and send alignment at every node. The input interface contains the current time, the node's internal schedule, the message to be sent, and the messages read on the TDMA bus. Based on this information, the module outputs the aligned message of the node (*write_iface_aligned*) to the TDMA module, and outputs the consistently aligned incoming messages (*read_iface_aligned*[]) to the node module.

TDMA Module The TDMA module simulates the TDMA communication bus that nodes use to send and receive messages. The input interface consists of the current time and the message to be sent by each node, and the output interface returns the values of delivered messages at each node (*read_iface*[][]). The returned variable is a matrix of messages where the i^{th} message in the j^{th} row is the message that node i receives from node j. Messages are passed between the node and TDMA modules either directly or via the abstraction layer.

Fault Module The fault module extends the input and output interfaces of normal operation. As the modeling of correlated faults needs coordination between different system elements, we assume that faults are implemented by a single module. The fault module takes the current time as input and injects faults into nodes (*node_faults*[]), the alignment layer (*align_faults*[]), and the communication bus (*comm_faults*). Since the application logic is system specific, the list of jobs (*app*[]) is passed to the fault module to derive application faults.

The proposed overall model supports reusability and customization for varied TT functionalities and applications. The implementation of the modules can be tailored to the characteristics of an actual TT system. For example, when we verify fault-tolerance under different fault conditions within the same TT application, only the fault module needs to be modified. Note that more simplistic models can be proposed by assuming services like alignment or membership. For example, it has been demonstrated that the frame-based model can be used to model general TT systems if alignment is used [18]. In contrast, we propose a realistic model in this dissertation that can also be used for the design of new algorithms that exploit the characteristics of the TT architecture.

6.4 Implementation with SAL Language

This section provides a detailed walkthrough of our prototype implementation of the general formal model described in Section 6.3. Although we concentrate on safety-critical applications where jobs are replicated and each job executes the same program, our implementation can easily be customized to describe any TT application.

We used the SAL (Symbolic Analysis Laboratory) language in our prototype implementation. The SAL language is a formal description language to specify concurrent systems. The SAL model checker offers various tools based on BDD (Binary Decision Diagrams) based symbolic and SAT (Satisfiability) based bounded model checkers, and it also has auxiliary tools including a simulator, a deadlock checker, and an automated test generator [21]. We focused on SAL language because of its expressiveness and high-level constructs and also because a powerful execution environment is attached to the language [33] that enables direct analysis, which will be explained in Section 6.5. Here, we describe the implementation of each module and the composition of the modules. We use a convention where the SAL code in the explanatory text is written in *italics*.

Snippet 1: Type declarations, auxiliary functions

```
1  tt{;N: natural, A: natural}: CONTEXT =
2  BEGIN
3    node: TYPE = [1..N];
4    discr_time: TYPE = [0..N];
5    fault: TYPE = {nonfaulty, benign, symmetric, asymmetric};
6    fault_vector: TYPE = ARRAY node OF fault;
7    error: natural = 2;
8    message: TYPE = [0..error];
9    message_array: TYPE = ARRAY node OF message;
10   function:TYPE =[[message_array,message,message]–> message];
11
12   %auxiliary definitions
13   _fault_counter(v: fault_vector,e: fault,sum: [0..N],i: node):[0..N]=
14     IF i = 0 THEN sum ELSE
15     _fault_counter(v,e,sum+IF v[i]=e THEN 1 ELSE 0 ENDIF,i–1)
16     ENDIF;
```

SAL is a typed language and every SAL model begins with the definitions of types and functions (Snippet 1). Our general system model contains N nodes (*node* at line 3), the usual communication faults (*fault* at line 5), and a binary message domain that is augmented with an error value (*message* at line 8). Time is modeled on a discrete scale such that each clock tick corresponds to a slot (*discr_time* at line 4). The main idea is that the same clock tick triggers all operations in every module that are supposed to be performed in a corresponding slot, i.e., scheduled jobs are executed (in node), messages are written/read to/from the bus (in TDMA), messages are buffered (in alignment) and faults are generated (in faults). A virtual slot is defined (with time value zero) to model jobs that can read every message from the previous TDMA round and that are ready to send in the current round from slot 1 on. The correctness of our discrete-time abstraction stems directly from the assumption that the precision of the applied clock synchronization algorithm allows agreement in the time slots. Otherwise, the discrete-time model needs to be justified by users (e.g., [20]). Jobs execute applications (*function* at line 10) in our model, which are functions taking a message received from each node, the current local state, and an external event as inputs and returning the new value of the local state. For simplicity, messages, states, and events share the same type. In addition, array types define a value for each node (e.g., *message_array* at line 9). These basic definitions can be customized by the user. For example, a simple counter function is defined (lines 13-16), which returns the number of faults in an array.

Snippet 2: Controller module

```
17 controller: MODULE =
18 BEGIN
19   INPUT
20     inp_ev_vec: ARRAY node OF message
21   OUTPUT
22     time: discr_time,
23     fun: function,
24     ev_vec: ARRAY node OF message
25   INITIALIZATION
26     time = 0;
27   DEFINITION
28     ev_vec = inp_ev_vec;
29   TRANSITION
30     [
31       time < N --> time' = time + 1;
32     []
33       time = N --> time' = 0;
34     []
35       ELSE -->
36     ]
37 END;
```

SAL is able to directly map the modules of the general model into modules of the language. SAL modules define local variables, communicate with other modules via input and output variables, initialize local and output variables, and define invariants and guarded transitions. The controller module (Snippet 2) periodically adjusts the discrete time, thus modeling an infinite sequence of TDMA rounds. The module also maintains auxiliary operations in the SAL implementation, like the definition of the replicated application logic and the distribution of node events. Assuming that a job is a replicated instance of a safety-critical application, the application logic (*fun* at line 23) can be defined only once and passed on to each node. Since *fun* is not initialized, SAL will arbitrarily assign a function to it. Events are defined as external input variables (*inp_ev_vec* at line 20), which are passed to the nodes (*ev_vec* at lines 24,28) for processing.

Snippet 3: Alignment module

```
38 alignment[id: node]: MODULE =
39 BEGIN
40   LOCAL
41     read_iface_buffered: message_array,
42     write_iface_buffered: message
43   INPUT
44     time: discr_time,
45     job_sched: discr_time,
46     read_iface: message_array,
47     write_iface: message,
48     send_curr_round_vec: ARRAY node OF BOOLEAN
49   OUTPUT
50     read_iface_aligned: message_array,
51     write_iface_aligned: message
52   DEFINITION
53     write_iface_aligned =    %send alignment
54       IF FORALL (n: node): send_curr_round_vec[n]
55         THEN write_iface ELSE
56           IF time > job_sched   %job exec. is modeled as atomic event
57             THEN write_iface_buffered ELSE write_iface
58           ENDIF
59       ENDIF;
60     read_iface_aligned = [[n:node]    %read alignment
61       IF n < job_sched
62         THEN read_iface_buffered[n] ELSE read_iface[n]
63       ENDIF]
64   INITIALIZATION
65     read_iface_buffered = [[n:node] 0];
66     write_iface_buffered = 0;
67   TRANSITION
68     [
69       time > 0 -->
70         read_iface_buffered'[time] = read_iface[time];
71         write_iface_buffered' =
72           IF time = job_sched
73             THEN write_iface ELSE write_iface_buffered
74           ENDIF;
75     []
76       ELSE -->
77     ]
78 END;
```

SAL allows the parameterized definition of modules. The parameters need to be defined when the module definitions are used to compose the system. An alignment module instance (Snippet 3) is defined for each node (*id* at line 38). We abstract the job schedule of a node by defining at which discrete time instant the job starts executing (*job_sched* at line 45) and whether the nodes of the system are able to send the latest message in the current TDMA round (*send_curr_round_vec* at line 48). Local buffers are defined to store values that were sent by other nodes (*read_iface_buffered* at line 41) and that were computed by the corresponding node (*write_iface_buffered* at line 42) in the previous TDMA round. The former is updated every time a remote node sends a message on the bus (line 70), and the latter only changes when the local node updates its local state and generates a new message (lines 71-74). Send (lines 53-59) and read alignment (lines 60-63) can be defined as invariants based on the current and buffered values. Definitions in SAL (labeled DEFINITION) can be thought of as macros that use the values of other state variables. Consequently, using SAL definitions is not only a logical way of modeling send and read alignment but it can also save state space during analysis because definitions do not affect the state transition relation of the system.

Note that we do not need read and send alignment in a particular solution, when we design the system such that every node executes a job after the last communication slot in a communication round, e.g., Slot 4 in Figure 6.2. In the proposed membership protocol as indicated in Figure 3.4 in Section 3.3, process of the node status determination, det_job, is also executed at the end of each communication round, which can simplify the model of TT systems.

The node module (Snippet 4) is also parameterized by the identifier of the node (*id* at line 79). The job schedule is not initialized, which corresponds to an arbitrary schedule. This is according to the premise that safety-critical services are not prioritized. Unrealistic cases are ruled out such that a node is only able to send a fresh message in the same TDMA round if it finishes execution before its sending slot (lines 93-94)[†]. We abstract that an application is executed instantaneously when a job is scheduled (99-100). For

[†]We use SAL's `IN` operator to non-deterministically assign a value from a set of constrained candidates. The empty constraint is denoted by the Boolean `TRUE`.

Snippet 4: Node module

```
79  node[id: node]: MODULE =
80  BEGIN
81    INPUT
82      time: discr_time,
83      fun: function,
84      read_iface_aligned: message_array,
85      ev: message
86    OUTPUT
87      write_iface: message,
88      job_sched: discr_time,
89      send_curr_round: BOOLEAN,
90      local_state: message
91    INITIALIZATION
92      local_state = 0;
93      send_curr_round IN {v: BOOLEAN |
94        IF job_sched >= id THEN v = FALSE ELSE TRUE ENDIF};
95    DEFINITION
96      write_iface = local_state; %assumption: node sends local state
97    TRANSITION
98      [
99        time = job_sched -->
100         local_state' = fun(read_iface_aligned, local_state, ev);
101     []
102       ELSE -->
103     ]
104 END;
```

simplicity, it is assumed that the local state is sent as the node's message (line 96). However, the message can generally be a function of the local state. Note that the domain of discrete time instants $[0..N]$ cannot cover the full generality of job scheduling. It cannot be modeled that a node reads everything up to time i (including the message sent in slot i) and immediately sends a message at time $(i+1)$. For that, the model needs to be extended such that an intermediate time instant is defined between i and $(i+1)$ where the application computes the message to be sent. As such extensions can affect the complexity of analysis, their use is only recommended if needed.

The TDMA module is responsible for modeling the communication of messages on the bus (Snippet 5). Faults are generally defined and propagated by the fault module. However, our SAL implementation reduces the number of transitions by directly injecting communication faults into the TDMA module. The output of the module is a matrix that indicates the value received by node i from node j after each tick. If node j is

Snippet 5: TDMA module

```
105 TDMA: MODULE =
106 BEGIN
107   INPUT
108     time: discr_time,
109     write_iface_aligned_vec: message_array,
110     fv_comm: fault_vector
111   OUTPUT
112     read_iface_vec: ARRAY node OF message_array
113   INITIALIZATION
114     read_iface_vec = [[n:node] [[m:node] 0]];
115   TRANSITION
116     [
117       time > 0 -->
118         read_iface_vec' IN {v: ARRAY node OF message_array |
119           FORALL (i, j: node):
120             IF j /= time THEN v[i][j]=read_iface_vec[i][j] ELSE (
121             IF fv_comm[j] = nonfaulty
122              THEN v[i][j] = write_iface_aligned_vec[j] ELSE (
123             IF fv_comm[j] = benign
124              THEN v[i][j] = error ELSE (
125             IF fv_comm[j] = symmetric
126              THEN FORALL(k: node):
127                 v[i][j] = v[k][j] AND v[i][j] /= error ELSE
128             TRUE
129             ENDIF) ENDIF) ENDIF) ENDIF};
130     []
131       ELSE -->
132     ]
133 END;
```

not the sender in the slot then the value remains unchanged (line 120). Otherwise, non-faulty and faulty cases are distinguished. If the sender is non-faulty, the correct value that is determined by the alignment layer is sent (lines 121-122). In case of a benign sender, every recipient receives *error* (lines 123-124)[‡], while symmetric senders distribute arbitrary but consistent and valid data (lines 125-127). Asymmetric senders can send any value to any node (line 128).

Our prototype currently implements communication faults (Snippet 6), and other required system specific faults can be added by users. The number of faults that the application is able to tolerate is usually limited. We explain how the number of communication faults can be tuned in our model using the example of asymmetric faults. The number of asymmetric faults is initially limited by A (lines 142-143), which is an input parameter

[‡]The definition can be modified such that the faulty sender can read its own message.

Snippet 6: Comm_faults module

```
134 comm_faults: MODULE =
135 BEGIN
136   INPUT
137     time: discr_time
138   OUTPUT
139     fv_comm: fault_vector
140   INITIALIZATION
141     %at most A asymmetric faults
142     fv_comm IN {v: fault_vector |
143           _fault_counter(v, asymmetric, 0, N) <= A};
144   TRANSITION
145     [
146       time = 0 -->
147         fv_comm' IN {v: fault_vector |
148               _fault_counter(v, asymmetric, 0, N) <= A};
149     []
150       ELSE -->
151     ]
152 END;
```

of the model (line 1). Transient faults, in addition to permanent faults, can be modeled by periodically re-defining the fault vector at the beginning of each TDMA round (lines 147-148).

Note that model checking is a powerful method to verify the fault-tolerance of the system as it exhaustively generates all fault combinations within the given condition. Hence, we can find design bugs even though the system consists of a large number of nodes and suffers from complex fault injection.

The previous modules can easily be composed together by wiring the corresponding input and output variables (see Snippet 7). We use the synchronous composition operator (||) so that modules can execute transitions in parallel. This means, e.g., that the simulated execution of an application occurs in parallel when a message is sent on the bus. Recall that the use of the alignment module is generally optional, although it is "hard-wired" in this prototype implementation.

Snippet 7: Synchronized composition of N-node system

```
system: MODULE =
  controller
  | | (WITH INPUT read_iface_aligned_vec:
                              ARRAY node OF message_array
      WITH INPUT ev_vec: ARRAY node OF message
      WITH OUTPUT write_iface_vec: message_array
      WITH OUTPUT job_sched_vec: ARRAY node OF discr_time
      WITH OUTPUT send_curr_round_vec:
                              ARRAY node OF BOOLEAN
      WITH OUTPUT local_state_vec: ARRAY node OF message
      (| | (n: node): RENAME
           read_iface_aligned TO read_iface_aligned_vec[n],
           ev TO ev_vec[n],
           write_iface TO write_iface_vec[n],
           job_sched TO job_sched_vec[n],
           send_curr_round TO send_curr_round_vec[n],
           local_state TO local_state_vec[n]
      IN node[n]))
  | | (WITH INPUT job_sched_vec: ARRAY node OF discr_time
      WITH INPUT read_iface_vec:
                              ARRAY node OF message_array
      WITH INPUT write_iface_vec: message_array
      WITH OUTPUT read_iface_aligned_vec:
                              ARRAY node OF message_array
      WITH OUTPUT write_iface_aligned_vec: message_array
      (| | (n: node): RENAME
           job_sched TO job_sched_vec[n],
           read_iface TO read_iface_vec[n],
           write_iface TO write_iface_vec[n],
           read_iface_aligned TO read_iface_aligned_vec[n],
           write_iface_aligned TO write_iface_aligned_vec[n]
      IN alignment[n]))
  | | comm_faults
  | | TDMA;
```

6.5 Example Use Cases: Design and Verification

We will now explain how the SAL model of TT systems can be used to verify the properties of the system (Subsection 6.5.1) to find appropriate scheduling of jobs (Subsection 6.5.2) and to automatically generate test cases for specific test goals (Subsection 6.5.3). The different tools we use are all part of the SAL environment; thus, they can directly work on the model described in the previous section. In every case, the execution engine is a *model checker* that performs exhaustive simulation of the system model [32]. We only assume that the model checker is able to explore all executions of the system independently of the actual model checking algorithm. Therefore, we can safely state that a property is true in a system if the model checker cannot find a counterexample. For simplicity, we assume that the system contains four nodes ($N = 4$) if not specified. Note that setting N to a constant value is necessary in classical model checking as it is impossible to explore infinitely numerous states.

6.5.1 Verification

Task 1: Consistent Replica States Suppose we prove that send and read alignment indeed implements the abstraction of dedicated communication paths and parallel job execution. We consider an abstract application that is implemented by an *arbitrary* function taking N messages and the local state as inputs and returning the new local state. All values are ternary (0, 1 and *error*) similarly to the type of *fun* (see Snippet 1). The correctness of the abstraction can be shown by proving consistency, i.e., replica jobs have the same state at the end of each round[§]:

[§]Auxiliary variables `local_state_vec[i]` (and `local_state_prev_vec[i]`) are introduced to denote the local state of job i in the current and previous rounds.

```
consistency: THEOREM
 system |- G(time=0 => FORALL(i,j:node):
  local_state_vec[i]=local_state_vec[j]);
```

Result 1: Consistency in Symmetric Systems In fact, the SAL model checker could prove the property unless asymmetric faults were allowed in the system. Since *fun* is never explicitly initialized in the model, SAL assumes that it can be any function and tries all possibilities. This corresponds to checking that `consistency` is true for any application expressed by *fun*. Note that this is a special case of the general theorem that states that consistency holds for any application with any number of replica nodes [18]. The model checker finds a counterexample if an asymmetric sender distributes different messages to different nodes that then compute inconsistent local states. This means that `consistency` can be proven for $A = 0$ and a counterexample is found if $A > 0$, where A is the number of asymmetric faulty nodes. In the latter case, consistency can only be obtained through the use of a Byzantine agreement protocol [41].

Task 2: Proposed Membership Protocol We applied our customizable formal models to our proposed membership protocols in the next step, including the clustering protocol explained in Sections 3.3 and 5.3. We redesigned *fun* such that it could execute the membership protocol in the node module. We impose a fault condition, $N > 2S + B + 1$, as given in Equation (3.1) in Section 3.2 in the comm_faults module, where N, S, and B correspond to the number of nodes in the system, symmetric faulty nodes, and benign faulty nodes¶. The fault condition in the clustering protocol is $C > 2S + B + 1$, where C is the number of nodes in a cluster.

¶In this chapter, capital letters are used to represent the numbers of nodes in the system and faulty nodes.

Result 2 With SAL BMC (Bounded Model Checker), we could prove the *correctness*, *completeness*, and *consistency* properties where $N = 4, 5, 6$ in the original membership protocol even though symmetric faults occurred. The BDD-based SAL SMC (Symbolic Model Checker) could not prove the properties for $N > 4$ and we could not obtain results even with the SAL BMC for more than six nodes because of state explosion.

The SAL BMC could prove these three properties for the clustering protocol for up to $N = 16$ and $C = 4$, and $N = 10$ and $C = 5$ within a few tens of minutes on an average laptop PC, while the SAL SMC could not prove them for such a large number of nodes. Even the SAL BMC could not prove the property when C was increased to six, i.e., $N = 12$ and $C = 6$, in our current model.

Before the protocol design was fixed, the SAL BMC actually exhibited a counterexample for the correctness property, i.e., a correct node is never diagnosed as faulty by any obedient nodes. After the counterexample was analyzed, we found a design bug in the self-accusation mechanism discussed in Subsection 5.3.3.

In the original design, a node resets a self-accusation bit to 0 if its local syndrome differs from the calculated health vector, as explained in Figure 5.5. This can be specified in the SAL language as follows, where we define `hv[i]` and `ls_prev[i]` as the health vector on node i and the local syndrome denoting whether the message from node i can be correctly received in the previous communication round respectively:

```
IF (EXISTS(i: nodes): hv[i] /= ls_prev[i]) THEN 0 ELSE 1 ENDIF;
```

The model checker found a counterexample, where node j is a symmetric faulty. All nodes in this case receive the message from node j correctly, i.e., `ls_prev[j]=1`, although the message is semantically incorrect. In the next round, as discussed in Section 3.3, node j is determined as faulty by the minority accusation because node j's message is different from that of the other nodes in node j's cluster. Thus, `hv[j]` becomes 0.

However, all nodes outside node j's cluster also have to accuse themselves by the self-accusation since `hv[j] /= ls_prev[j]`, although they are non-faulty nodes. This does not satisfy the correctness property. Therefore, we added a precondition such that `hv[j] /= 0` to execute the self-accusation, and then the correctness property was finally proven:

```
IF (EXISTS(i: nodes): hv[i] /= 0 AND hv[i] /= ls_prev[i]) THEN 0 ELSE 1 ENDIF;
```

Discussion Although we observed that it was computationally very expensive to consider all possible fault conditions, we could successfully prove the properties within a few tens of minutes with the SAL BMC. The proof took approximately 20 minutes running on a single processor of a dual-core Intel Xeon 5130 at 2 GHz with 4 GBytes of memory. SAL was installed on a Linux system with kernel version 2.6.17. We found that the BDD-based SAL SMC turned out to be ineffective to prove the properties as it took a long time ($>$ 1 hour) to compute the BDD-representation of the model.

The next applications of the model described in the following subsections are based on finding counterexamples that are in general computationally less complex than complete verification. This is because only a portion of the state space needs to be explored. In fact, proving consistency took the most time in our set of experiments.

We found that model checking was helpful especially for communication systems with a large number of nodes and complex fault conditions, e.g., different kinds of faults occurred simultaneously because it was very difficult to find bugs in these systems only through manual verification. However, model abstraction techniques may become necessary for more complex systems to avoid state explosion.

On a final note, we only modified the node and the comm_faults modules in Task 2, which means that our customizable generic formal model can effectively be tailored for various TT applications and easily be handled by software engineers.

6.5.2 Scheduling of Jobs

Task: Schedule for Reduced Abstraction Delay We now present a proof-of-concept example of how to use the model checker to find effective scheduling of replicated jobs within the hosting nodes. We see that the delay induced by alignment abstraction can be mitigated if all nodes are able to send a message in the same round when the message is computed. The delay in this case is caused by read alignment and is one TDMA round. In fact, the model checker can find an appropriate schedule that minimizes these delays. We achieve this by stating that such a schedule does not exist and the model checker finds a counterexample that is a required solution. To track the delay between when a message is sent and when it is processed at a remote node, we extend the domain of messages and define that *fun* returns a special value `SPECV` if all input messages are 1. The following property states that it is never true that the local state in a round is `SPECV` and the local state in the previous round is 1 in all nodes:

```
exist_schedule:
 THEOREM system |-
  G(NOT(time=0 AND FORALL(i:node):
   local_state_prev_vec[i]=1 =>
   EXISTS(j:node):local_state_vec[j]=SPECV));
```

Result: Early Scheduled Jobs The property cannot be proven and a counterexample is provided where four jobs are executed “early”, at slots 0, 0, 1, and 1 respectively. We can see that this schedule indeed allows every job to send a fresh message (value 1). Therefore, the overall delay can be reduced in this proof-of-concept example. Note that the same technique can be used to find appropriate mapping to deploy jobs on nodes even if more complex constraints are specified.

6.5.3 Test Generation

The idea of automated test generation is to construct a sequence of inputs (called *test cases*) that will cause the system under test to exhibit some behavior of interest, called a *test goal*. Model checkers can be naturally used to generate test cases such that Boolean *trap variables* are defined that are initially false and set to true by the program when the corresponding test goal is reached. The model checker is instructed to prove that the trap variables cannot become true; therefore, every counterexample is a valid test case. However, the straightforward way of doing that can be ineffective. For example, the strategy of generating one test case for each test goal might be redundant. More sophisticated techniques leverage the model checker, e.g., by checking for paths that are extensions (i.e., continuations) of already explored executions [36]. This technique is also implemented by SAL's ATG (Automated Test Generation) tool that integrates the BMC and BDD model checkers of SAL by augmenting them with clever techniques required for effective test generation.

Task: Tests for Specific Message Patterns We used the ATG tool of SAL to generate a sequence of events observed by a node that drives the system to a certain state. We re-define *fun* such that it returns `SPECV` if a node observes an external event of one. Let the test goal be to reach a state where value `SPECV` is sent on the bus as the message of the 1st node. Therefore, we place the following assignment of trap variable `atg_trap` (newly added) in the TDMA module:

```
atg_trap'=IF EXISTS(n:node):
 read_iface_vec[n][1]=SPECV THEN TRUE
 ELSE FALSE ENDIF;
```

Result: Test Case Found By running SAL ATG, we find that node 1 has observed event 1 and generated `SPECV` but it could not send it immediately because of the node schedule. Therefore, the trap variable only becomes true in the second TDMA round of the execution path that constitutes the test case. A sequence of events in the SAL example, containing one event for every node, is returned by the tool for each slot until the trap variable becomes true.

We have shown how our executable formal specification and trap variables can be used to support model-based test generation. An important issue concerning testing is the coverage of the obtained test cases, i.e., whether it is able to exercise the system to the required extent, which is determined by the coverage metrics. However, a discussion on test coverage in TT systems is beyond the scope of this dissertation.

6.6 Related Work

Formal methods have been successfully used to verify various TT applications. For example, the membership protocol in the TTP/C time-triggered protocol suite was found to be correct through manual proofs in [11], or automated theorem proving was applied to analyze an agreement protocol in another time-triggered environment [19]. Other work has used executable formal specifications to model check the startup protocol in TTP/C [20]. It is also possible to specify the system in an intermediate, preferably understandable and easy-to-read, notation and translate it into the input language of the verification engine (e.g., [42]). Our prototype implementation omits this intermediate step and specifies the system directly in the input language of analysis. We argue that the modularity of the proposed model of TT systems and the resemblance of its structure to a real system fulfill the role of a precise though intuitive specification. Our approach mainly differs from previous work in that it proposes a skeleton model of the target systems that can be used as a template for customized solutions and the specifications need not be created from scratch.

As part of the system integration process, deployment, allocation, and scheduling tasks can be uniformly thought of as restrictions with respect to the unconstrained space of solutions. Different techniques such as constraint propagation, branch and bound, backtracking, or mixed integer programming have been proposed (e.g., [31, 38, 40]). However, they require either the development of new computation engines (e.g., written in C [38] or Java [40]) or the use of existing dedicated engines (e.g., [31]) to solve (or even optimize) the constraint problem. Although our method, when used for scheduling, might be outperformed by other techniques, it uses the same system representation that is used for other tasks. Therefore, the overall cost of design and analysis can be reduced. If the constraint problem entails infeasible complexity with our executable formal model, the application of other techniques is inevitable. However, it is possible that the combination of our approach with other techniques will enhance quality and performance in finding the best solution. Investigations into using model checking to integrate the system is part of our ongoing work.

Testing of distributed systems deals with *what* to test and *how* to test it. Our approach is related to the former question because we generate test cases as opposed to actually testing the system. The latter question is nontrivial in distributed systems due to issues such as interoperability, synchronization, timing, and concurrency. Model-based testing can support existing solutions to testing distributed systems like [30, 49].

We applied the method of model-based test generation using a model checker [34, 44] to generate tests to validate TT systems. The main idea was to challenge the model checker to find execution paths in the model of the system that reached specific test goals. This was done by stating that no such paths exist and the counterexample returned by the model checker could be directly used as a test case. As an alternative or a supplement to the model-based approach, requirement-based test-case generation [43, 51] can be used, where tests are generated by only analyzing the requirements. This can be useful if tests are required to be independent of the unit under test.

6.7 Summary

We have proposed a formal but intuitive method to support the development of safety-critical TT systems from design to verification and testing. The solution is general and the core elements of the method (i.e., system characteristics, application logic, and fault modes) can be customized by application developers. We have implemented a prototype model of general safety-critical TT applications in our case study. This model has been analyzed by model checkers, and simple verification, task scheduling, and test generation examples have been shown. We have also applied our customizable generic formal model to real membership protocols for automotive brake-wire systems and demonstrated that the model can effectively be reused for various TT applications. We used the SAL language in our implementation, but other executable specification languages such as NuSMV [48], SystemC [37] could also be applied equally well. However, additional transformation steps might be needed to translate from the specifications to the language of the execution engine (e.g., model checking of SystemC codes [35]). The main strength of our method compared to the previous work is that it provides an all-in-one solution that can be used for tasks that usually require multiple representations of the system.

CHAPTER 7

CONCLUSION

7.1 Achievements

This dissertation has focused on safety-critical embedded distributed control systems and taken emerging automotive X-by-Wire systems, where driving, steering, and braking are electrically and electronically controlled synthetically, as an example application. Automotive control systems, out of the various embedded control systems, impose particularly severe restrictions on cost and available hardware resources because of the scale of mass production. Therefore, the goal of this dissertation is to propose suitable solutions that can fulfill not only high dependability but also cost-effectiveness for automotive safety-critical systems.

We have first proposed a novel architecture that incorporates the concept of autonomous decentralized systems to achieve this goal. This architecture allows all nodes in the system including sensor and actuator nodes to obtain the shared information required for vehicle control through the data field implemented with the communication network and to autonomously execute backup control if some node in the system fails. Therefore, the proposed architecture can tolerate the existence of failed nodes and thus does not need expensive fail-operational nodes with triple or more redundant hardware. This approach with reduced-redundancy dependability has also been applied to the node level by taking into consideration the node function. We have proposed a validity check method instead of dual redundancy to detect faults in fail-silent actuator nodes. Our estimation revealed that the system cost could be reduced by approximately from 20 to 30% due to the proposed autonomous decentralized architecture and optimal node hardware architecture, which contributes to a substantial cost reduction for automotive control systems.

Although the autonomous decentralized architecture satisfies competing demands, a coordination scheme, i.e., an agreement protocol, is required in this architecture to accurately identify failed nodes so that disagreements in the control mode can be avoided. To resolve this challenge, we have proposed a membership protocol as an agreement protocol for safety-critical distributed systems.

Each node in the proposed membership protocol locally evaluates the status of other nodes in the system and exchanges a local view, which we call a local syndrome, with all nodes. Then, every node identifies the failed node by voting on the exchanged local syndromes. In contrast to related work, our membership protocol tolerates simultaneous and non-fail-silent (Byzantine) faults and can flexibly be implemented in time-triggered systems as a middleware component. In addition, a pipeline-like execution of the protocol has been proposed to improve real-time capabilities, where a fault detected in a certain communication round can be identified in the next round. Important properties such as correctness, completeness, and consistency have been defined and proved by hand for the proposed membership protocol.

With a widely used time-triggered communication network, i.e., FlexRay, in the automotive industry, we developed a prototype Brake-by-Wire system employing the proposed autonomous decentralized architecture and membership protocol in a realistic hardware and software environment for automotive control systems. Although we demonstrated that the prototype system could persevere in practical use, the results obtained from evaluating the performance of this prototype system revealed that the computation overhead for the membership middleware was prohibitively large. The overhead was 5.8% under conditions where there were six nodes, a communication round of 5 ms, and a 40 MHz CPU. It is ideal to assign as much CPU time as possible to control application programs to achieve better driving performance and safety in automotive systems. Thus, it is not practically acceptable to spend that much CPU time only for the membership service. Furthermore, the execution time required for the voting process increases along with the number of nodes in the system.

We have further proposed novel lightweight membership protocols, i.e., voting sharing and clustering protocols, to address this problem. The main idea behind voting sharing is to have each node vote for only one respective node and to share the voting results with all nodes. In the clustering concept, n nodes are logically divided into n/c clusters, where each cluster consists of c nodes. Node p sends a local syndrome only with respect to nodes within the cluster to which node p belongs to all the other nodes in the system. Proofs of the same properties as the original protocol have been also done. Both approaches can reduce the computation overhead for membership, and the clustering protocol can also decrease the communication bandwidth, compared with the original protocol. Our experiments revealed that the execution time for the voting process in the voting sharing protocol was reduced by approximately 60% compared with the original protocol for eight nodes.

We have investigated advantages and drawbacks of the three proposed membership protocols in terms of computation and communication overhead, diagnosis latency, and fault tolerance. Our analysis showed a tradeoff between the overhead and fault tolerance. The lightweight protocols incur degradation in diagnosis accuracy in exchange for the reduction of the computational overhead. However, it can be mitigated with additional mechanisms such as rotating voters, counter update algorithm, and self-accusation. Drawbacks of the lightweight protocols also include the increase in the required communication bandwidth and diagnosis latency in the voting sharing protocol, and the decrease in the tolerable number of faults in the clustering protocol. Despite these drawbacks, the clustering protocol is well-balanced among these three protocols when the system consists of large number of nodes, e.g., equal to or more than 8 or 10 nodes and the fault condition is not so severe, e.g., $s = 0, b = 2$ or $s = 1, b = 1$, where s is the number of symmetric Byzantine faulty nodes, and b is the number of benign faulty nodes.

Finally, we have proposed a customizable formal model of generic time-triggered systems to support essential system design processes such as task scheduling, test case generation, and verification. Our model is not restricted to any dedicated implementation and can easily be used also by industrial practitioners who are not specialists in formal methods. Because the proposed formal model has a modular architecture, it can be reused and easily customized, which can reduce the model development costs. Users only need to tailor corresponding modules to customize the general model. The prototype implementation of the formal model was carried out with the SAL language. We have demonstrated the usability of our prototype with the SAL tool suite by presenting use cases of verification, task scheduling, and test case generation based on one identical formal model. The proposed membership protocols were model-checked in a use case of verification, and we confirmed the design correctness of the protocols.

In conclusion, the overall achievement of the goal in this dissertation is to propose an autonomous decentralized architecture and membership protocols that can be applied to actual safety-critical automotive control systems. The key is that the proposed architecture and protocols allow the system to be fail-operational even though it consists of only inexpensive fail-silent components with limited computational capability.

7.2 Future Work

Future research directions include implementation and evaluation of the proposed approaches in actual products. Additional improvements to the lightweight membership protocols to mitigate degradation in diagnosis accuracy remains as important outstanding work. One of the most promising directions is to enhance a counter updating algorithm for the voting sharing. It is worth considering a system-oriented method to cluster nodes in the clustering protocol, where, for example, nodes in the same subsystem belong to an identical cluster, to avoid latent symmetric Byzantine faults. We are also interested in how we can solve the state explosion problem in model checking. Appropriate solutions like abstraction techniques will be required to verify fault-tolerance in distributed control systems with more nodes and more complex fault conditions.

BIBLIOGRAPHY

[1] K. H. Gaubatz, Progress of Automotive Electronics – Trends at Electronic Systems for Chassis Control, In *Proceedings of the 9th International Conference on Automobil Elektronik*, 2005.

[2] R. C. Hammett and P. S. Babcock, Achieving 10^{-9} Dependability with Drive-by-Wire Systems, In *Proceedings of SAE 2003 World Congress,* No. 2003-01-1290, 2003.

[3] C. Wilwert, Y. Song, F. Simonot-Lion, A. Charlois, and A. Gilberg, Impact of Fault Tolerance Mechanisms on X-by-Wire System Dependability, In *Proceedings of SAE 2004 World Congress*, No. 2004-01-0705, 2004.

[4] K. Mori, Autonomous Decentralized Systems: Concept, Data Field Architecture and Future Trends, In *Proceedings of IEEE International Symposium on Autonomous Decentralized Systems*, pp. 28–34 (1993).

[5] FlexRay Consortium, FlexRay Communications System, Protocol Specification Version 3.0.1, 2010.

[6] H. Kopetz and G. Grunsteidl, TTP – A Protocol for Fault Tolerant Real Time Systems, *Computer*, vol. 27, no. 1, pp. 14–23, 1994.

[7] H. Kopetz et al., The Time-Triggered Ethernet (TTE) Design, In *Proceedings of International Symposium on Object-Oriented Real-Time Distributed Computing*, pp. 22–23, 2005.

[8] K. Hoyme and K. Driscoll, SAFEbus, *IEEE Aerospace and Electronic Systems Magazine*, vol. 8, no. 3, pp. 34–39, 1993.

[9] AUTOSAR, http://www.autosar.org/, AUTOSAR Specification R4.1.2, 2013.

[10] P. D. Ezhilchelvan and R. Lemos, A Robust Group Membership Algorithm for Distributed Real-Time Systems, In *Proceedings of Real-Time Systems Symposium*, pp. 173–179, 1990.

[11] G. Bauer and M. Paulitsch, An Investigation of Membership and Clique Avoidance in TTP/C, In *Proceedings of Symposium on Reliable Distributed Systems*, pp. 118–124, 2000.

[12] C. Bergenhem and J. Karlsson, A Process Health Status Service for Safety Related Systems Using TT/ET Communication Scheduling, In *Proceedings of the 14th IEEE Pacific Rim International Symposium on Dependable Computing*, pp. 122–131, 2008.

[13] H. Pfeifer, Formal Verification of the TTP Group Membership Algorithm, In *Proceedings of Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols and Protocol Specification, Testing and Verification*, pp. 3–18, 2000.

[14] A. Bouajjani and A. Merceron, Parametric Verification of a Group Membership Algorithm, *Theory and Practice of Logic Programming*, vol. 6, no. 3, pp. 321–353, 2006.

[15] M. Serafini, P. Bokor, N. Suri, J. Vinter, A. Ademaj, W. Brandstätter, F. Tagliabò, and J. Koch, Application-Level Diagnostic and Membership Protocols for Generic Time-Triggered Systems, *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 2, pp. 177–193, 2011.

[16] K. Sakurai, M. Matsubara, and M. Hoshino, Membership Middleware for Dependable and Cost-effective X-by-wire systems, *SAE International Journal of Passenger Cars - Electronic and Electrical Systems*, vol. 1, no. 1, pp. 180–186, 2009.

[17] J. Rushby, Formal Methods and their Role in the Certification of Critical Systems, TR SRI-CSL-95-1, SRI International, 1995.

[18] P. Bokor et al., Sustaining Property Verification of Synchronous Dependable Protocols over Implementation, In *Proceedings of the 10th IEEE High Assurance Systems Engineering Symposium*, pp. 169–178, 2007.

[19] J. Rushby, Systematic Formal Verification for Fault-Tolerant Time-Triggered Algorithms, *IEEE Transactions on Software Engineering*, vol. 25, no. 5, pp. 651–660, 1999.

[20] W. Steiner et al., Model Checking a Fault-Tolerant Startup Algorithm: From Design Exploration to Exhaustive Fault Simulation, In *Proceedings of IEEE Dependable Systems and Networks*, pp. 189–198, 2004.

[21] SAL (Symbolic Analysis Laboratory), http://sal.csl.sri.com/.

[22] F. Kitahara, et al., The ATOS Tokyo Metropolitan Area Train Traffic Control System, *HITACHI REVIEW*, vol. 46, no. 2, 1997.

[23] International Standard, ISO 26262-5, Road vehicles – Functional safety –, Part 5: Product development at the hardware level, Annex D, 2011.

[24] N. Kanekawa, M. Nohmi, Y. Satoh, and H. Satoh, Self-Checking and Fail-Safe LSIs by Intra-Chip Redundancy, In *Proceedings of the 26th International Symposium on Fault-Tolerant Computing*, pp. 426–430, 1996.

[25] FlexRay Consortium, FlexRay Communications System, Electrical Physical Layer Specification, Version 2.1 Revision B, 2006.

[26] P. Lincoln and J. Rushby, A Formally Verified Algorithm for Interactive Consistency under Hybrid Fault Models, In *Proceedings of the 23rd International Symposium on Fault-Tolerant Computing*, pp. 402–411, 1993.

[27] K. Driscoll, B. Hall, H. Sivencrona, and P. Zumsteg, Byzantine Fault Tolerance, from Theory to Reality, In *Proceedings of the 22nd International Conference on Computer Safety, Reliability, and Security*, pp. 235–248, 2003.

[28] OSEK VDX, Operating System (Version 2.2.3), 2005.

[29] OSEK VDX, Time-Triggered Operating System (Version 1.0), 2001.

[30] G. A. Alvarez and F. Cristian, Simulation-Based Testing of Communication Protocols for Dependable Embedded Systems, *Journal of Supercomputing*, 16(1-2), pp. 93–116, 2000.

[31] A. Balogh, A. Pataricza, and J. Rácz, Scheduling of Embedded Time-Triggered Systems, In *Proceedings of Workshop on Engineering Fault Tolerant Systems*, pp. 44–49, 2007.

[32] E. Clarke, O. Grumberg, and D. Peled, *Model Checking*, MIT Press, 2000.

[33] L. de Moura et al., SAL 2, In *Proceedings of Computer Aided Verification*, pp. 496–500, 2004.

[34] A. Gargantini and C. L. Heitmeyer, Using Model Checking to Generate Tests from Requirements Specifications, In *Proceedings of European Software Engineering Conference*, pp. 146–162, 1999.

[35] A. Habibi and S. Tahar, An Approach for the Verification of SystemC Designs using AsmL, In *Proceedings of Automated Technology for Verification and Analysis*, pp. 69–83, 2005.

[36] G. Hamon, L. de Moura, and J. Rushby, Generating Efficient Test Sets with a Model Checker, In *Proceedings of Software Engineering and Formal Methods*, pp. 261–270, 2004.

[37] Open SystemC Initiative, http://www.systemc.org.

[38] S. Islam and N. Suri, A Multi Variable Optimization Approach for the Design of Integrated Dependable Real-Time Embedded Systems, In *Proceedings of Embedded and Ubiquitous Computing*, pp. 517–530, 2007.

[39] R. M. Kieckhafer et al., The MAFT Architecture for Distributed Fault Tolerance, *IEEE Transactions on Computers*, 37(4), pp. 398–405, 1988.

[40] K. Kuchcinski, Constraints-Driven Scheduling and Resource Assignment, *ACM Transactions on Design Automation of Electronic Systems*, 8(3), pp. 355–383, 2003.

[41] L. Lamport, R. Shostak, and M. Pease, The Byzantine Generals Problem, *ACM Transactions on Programming Languages and Systems*, 4(3), 1982.

[42] S. P. Miller et al., Proving the Shalls, In *Proceedings of Formal Methods Europe*, pp. 75–93, 2003.

[43] A. Rajan, M. W. Whalen, and M. P. Heimdahl, Model Validation using Automatically Generated Requirements-Based Tests, In *Proceedings of IEEE High Assurance Systems Engineering Symposium*, pp. 95–104, 2007.

[44] S. Rayadurgam and M. Heimdahl, Coverage based Test-Case Generation using Model Checkers, In *Proceedings of Workshop on Engineering of Computer Based Systems*, pp. 83–91, 2001.

[45] J. Rushby, Bus Architectures for Safety-Critical Embedded Systems, In *Proceedings of Embedded Software*, pp. 306–323, 2001.

[46] K. Sakurai, M. Hoshino, Y. Morita, and Y. Takahashi, Design and Implementation of Middleware for Network Centric X-by-Wire Systems, In *Proceedings of SAE 2006 World Congress*, No. 2006-01-1326, 2006.

[47] M. Serafini et al., A Tunable Add-On Diagnostic Protocol for Time Triggered Systems, In *Proceedings of IEEE Dependable Systems and Networks*, pp. 164–174, 2007.

[48] NuSMV Toolset, http://nusmv.irst.itc.it/.

[49] W. Tsai et al., Scenario-based Object-Oriented Test Frameworks for Testing Distributed Systems, In *Proceedings of Future Trends of Distributed Computing Systems*, pp. 288–294, 2003.

[50] C. J. Walter, M. Hugue, and N. Suri, Continual On-Line Diagnosis of Hybrid Faults, In *Proceedings of the 4th Conference on Dependable Computing for Critical Applications*, pp. 233–249, 1994.

[51] M. W. Whalen, A. Rajan, M. P. Heimdahl, and S. P. Miller, Coverage Metrics for Requirements-based Testing, In *Proceedings of Software Testing and Analysis*, pp. 25–36, 2006.